Cours de Cryptologie

Guilhem Castagnos

Janvier – Avril 2024

version du 24 mars 2024

Table des matières

Chapitre I

Introduction

1. Bibliographie

- Gilles Zémor, *Cours de cryptographie*, Cassini, 2000.
- Nigel Smart, *Cryptography Made Simple*, Springer, 2016
- Dan Boneh et Victor Shoup, *A Graduate Course in Applied Cryptography*, en ligne `https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_6.pdf`

2. Tentative de plan du cours :

- Chiffrement parfait et chiffrement par flot
- Chiffrement par bloc
- Fonction de hachage
- Cryptographie sur le problème du logarithme discret
- Cryptographie sur le problème de la factorisation
- Signatures numériques
- Calcul multipartite sécurisé

3. Courte introduction à la cryptologie

Cryptographie : ensemble de méthodes pour sécuriser l'information et les communications numériques contre des adversaires.

Cryptanalyse : consiste à casser ces méthodes (attaques), alors que la cryptographie consiste à les concevoir.

Cryptologie = Cryptographie + Cryptanalyse. En pratique, on emploie souvent le terme cryptographie à la place de cryptologie.

Adversaire : Principe de Kerckhoffs (fin 19e siècle), l'adversaire connaît la méthode utilisée. Seule une donnée lui est inconnue : la clef secrète. On définit plusieurs niveaux de sécurité en définissant les buts de l'adversaire : retrouver la clef secrète (bris total), retrouver l'information d'origine...On définit également les moyens à la disposition de l'adversaire pour faire son attaque.

Chapitre II

Chiffrement parfait, Chiffrement par flot

1. Le chiffrement symétrique

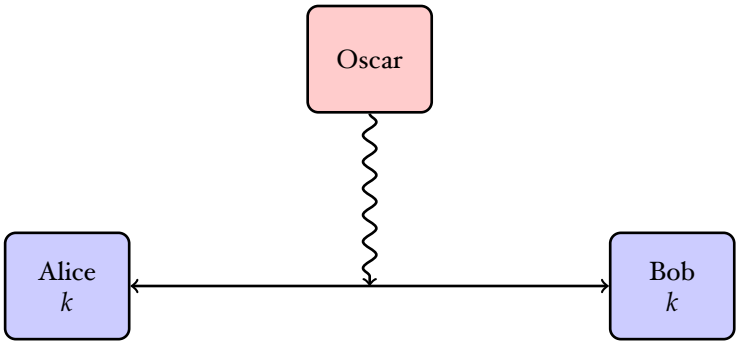
Notations et vocabulaire :

- $\mathscr{M}$: l'espace des messages clairs (*plaintexts*)
- $\mathscr{C}$: l'espace des messages chiffrés (*ciphertexts*)
- $\mathscr{K}$: l'espace des clefs secrètes (*secret keys*)
- $\mathrm{E} : \mathscr{K} \times \mathscr{M} \to \mathscr{C}$ la fonction de chiffrement (*encryption*)
- $\mathrm{D} : \mathscr{K} \times \mathscr{C} \to \mathscr{M}$ la fonction de déchiffrement (*decryption*)

Typiquement $\mathscr{M} = \{0,1\}^\ell$ ou $\{0,1\}^*$. On suppose que l'information est codée sous forme d'une chaîne de bits.

On veut une notion de correction : pour tout $k \in \mathscr{K}$, $m \in \mathscr{M}$, $\mathrm{D}(k, \mathrm{E}(m,k)) = m$.

On suppose qu'Alice et Bob disposent d'une même clef k, connue d'eux seuls, et utilisent E et D pour communiquer. Le but est d'assurer la **confidentialité** des messages clairs échangés.



$$\mathrm{E}(k, m_\mathrm{A}) = c_\mathrm{A} \longrightarrow \mathrm{D}(k, c_\mathrm{A}) = m_\mathrm{A}$$

$$\mathrm{D}(k, c_\mathrm{B}) = m_\mathrm{B} \longleftarrow \mathrm{E}(k, m_\mathrm{B}) = c_\mathrm{B}$$

2. Le chiffrement parfait

Le chiffrement de Vernam (1917)...

Aussi appelé masque jetable ou *One Time Pad* en anglais.

- $\mathscr{M} = \mathscr{C} = \mathscr{K} = \{0,1\}^\ell$
- $\mathrm{E}(k,m) = k \oplus m$
- $\mathrm{D}(k,c) = k \oplus c$

On va voir que si la clef k est utilisée une seule fois et choisie au hasard avec équiprobabilité, alors le chiffrement de Vernam est un chiffrement parfait, inconditionnellement sûr même pour un adversaire tout puissant. Ce chiffrement est avant tout théorique, mais a été tout de même utilisé en pratique par exemple pour le téléphone rouge ou encore dans les *Numbers stations*, une très longue clef secrète étant transmise préalablement.

Pourquoi une utilisation **unique** de la clef est nécessaire :

- **attaque à clair(s) connu(s)** : l'adversaire connaît un ou plusieurs (éventuellement un très grand nombre) couples clairs chiffrés. Souvent le moyen minimal en cryptographie symétrique : entête de fichier connu (mail, image jpeg...), début de protocole.

  Sur le chiffrement de Vernam : $c_1 \oplus m_1 \to k$. Bris total ! Si on réutilise k pour chiffrer m_2 par $c_2 = m_2 \oplus k$. On peut retrouver m_2 à partir de c_2.
- **attaque à chiffré(s) seul(s)** : l'adversaire ne connaît que le chiffré (moyen le plus faible) : $c_1 \oplus c_2 = m_1 \oplus m_2$. Information sur les clairs : on attaque la **sécurité sémantique**.

...est un chiffrement parfait

On voit message, chiffré, et clef, comme des variables aléatoires discrètes M, C et K, ainsi on attache une probabilité au choix d'un message particulier m, au choix d'une clef k et à celui d'obtenir un chiffre c. On supposera toujours que le choix de la clef se fait indépendamment de celui du message, c'est à dire que M et K sont indépendantes.

Rappel : cela signifie que pour tout $m \in \mathscr{M}, k \in \mathscr{K}$, $\mathrm{P}(\mathrm{M} = m, \mathrm{K} = k) = \mathrm{P}(\mathrm{M} = m)\mathrm{P}(\mathrm{K} = k)$.

Autre rappel, si $\mathrm{P}(\mathrm{C} = c) > 0$, on définit la probabilité conditionnelle

$$\mathrm{P}(\mathrm{M} = m \,|\, \mathrm{C} = c) := \frac{\mathrm{P}(\mathrm{M} = m, \mathrm{C} = c)}{\mathrm{P}(\mathrm{C} = c)}.$$

Dans la suite, on supposera également toujours que pour tout m, k, c,

$$\mathrm{P}(\mathrm{M} = m) > 0, \mathrm{P}(\mathrm{C} = c) > 0, \mathrm{P}(\mathrm{K} = k) > 0.$$

Définition II – 1. Un système de chiffrement symétrique est parfaitement sûr si pour tout $m \in \mathscr{M}, c \in \mathscr{C}$, $\mathrm{P}(\mathrm{M} = m \,|\, \mathrm{C} = c) = \mathrm{P}(\mathrm{M} = m)$.

Autrement dit le chiffré n'apporte pas d'information supplémentaire sur le clair, même pour un adversaire tout puissant. Cela signifie également que M et C sont indépendantes.

Théorème II – 2. Le système de Vernam où la clef est choisie de manière équiprobable est parfaitement sûr.

Démonstration. On a d'une part

$$\mathrm{P}(\mathrm{M} = m, \mathrm{C} = \mathrm{C}) = \mathrm{P}(\mathrm{M} = m, \mathrm{K} = m \oplus c) = \mathrm{P}(\mathrm{M} = m) \times \mathrm{P}(\mathrm{K} = m \oplus c) = \frac{\mathrm{P}(\mathrm{M} = m)}{|\mathscr{K}|}.$$

D'autre part,

$$\mathrm{P}(\mathrm{C} = c) = \sum_{m \in \mathscr{M}} \mathrm{P}(\mathrm{C} = c, \mathrm{M} = m) = \sum_{m \in \mathscr{M}} \mathrm{P}(\mathrm{K} = m \oplus c, \mathrm{M} = m) = \frac{1}{|\mathscr{K}|} \sum_{m \in \mathscr{M}} \mathrm{P}(\mathrm{M} = m) = \frac{1}{|\mathscr{K}|}.$$

□

Proposition II – 3. Si un système de chiffrement symétrique est parfaitement sûr alors $|\mathscr{M}| \leqslant |\mathscr{C}| \leqslant |\mathscr{K}|$.

Démonstration. Nécessairement $|\mathscr{M}| \leqslant |\mathscr{C}|$: la fonction de chiffrement doit être injective pour pouvoir déchiffrer sans ambiguïté.

Si le chiffrement est parfait on a $P(C = c \,|\, M = m) = P(C = c)$ par indépendance. D'autre part, on a supposé que $P(C = c) > 0$. Donc pour un m fixé et pour chaque c il existe au moins un k tel que $c = E(k, m)$. Ainsi $|\mathscr{C}| \leqslant |\mathscr{K}|$. □

L'espace des clefs doit donc au moins être aussi grand que celui des clairs et des chiffrés. Pour transmettre un message de ℓ bits de manière confidentielle il faut donc avoir préalablement transmis un clef secrète d'au moins ℓ bits. Cela est peu pratique ! Remarque : la notion d'entropie (cf. le cours de théorie de l'information) permet de mieux formaliser cela : si le chiffrement est parfait $H(K) \geqslant H(M)$.

Les chiffrements symétriques actuels utilisent une même clef courte (typiquement 128 à 256 bits) que l'on réutilise pour chiffrer des gigas octets de données. On perd donc la sécurité parfaite.

Tous ces algorithmes de chiffrement à clef secrète sont sensibles à la **recherche exhaustive**. C'est une attaque à clair connu. Connaissant un message clair m et son chiffré correspondant c, on calcule le chiffrement de m par toutes les clefs possibles jusqu'à trouver c. Si la clef secrète fait n bits, on doit donc faire 2^n chiffrements dans le cas le pire. Le but du cryptographe symétrique est de concevoir un système dont la meilleure attaque connue soit la recherche exhaustive.

3. Chiffrement par flot

Cadre

C'est une version pratique du chiffrement de Vernam. À partir d'une clef secrète, une chaîne de bits aléatoire courte, on crée de manière déterministe une suite (pseudo-aléatoire) de bits $(z_k)_{k\in\mathbf{N}}$, dite suite chiffrante. Puis on chiffre une suite de bits $(m_k)_{k\in\mathbf{N}}$ par $c_k = m_k \oplus z_k$ pour tout $k \in \mathbf{N}$ (on parle plus exactement de chiffrement par flot synchrone additif).

Ce chiffrement est plus rapide que le chiffrement par bloc que l'on verra ensuite, surtout en implantation matérielle car la complexité matérielle est plus faible. Il est basé sur des primitives simples (LFSR cf. suite du cours). Ils peuvent être exécutés avec une mémoire limitée, ils traitent le message clair bit par bit à la volée, et sont donc utilisés pour chiffrer des communications.

Exemples : RC4 (1987, obsolète), SNOW3G (2002, 3G→), ChaCha20 (2008, TLS, OpenSSH), ZUC (2010, 3G→).

Ces systèmes peuvent être vus comme des automates, dont l'état interne est initialisé par une clef secrète à l'instant $t = 0$. Au temps $t > 0$, l'état interne S_t est fonction de l'état au temps précédent, S_{t-1}, et un bit de suite chiffrante fonction de l'état est produit, z_t.

Si l'état interne de l'automate est un registre ℓ bits, alors au plus 2^ℓ états différents sont possibles. Au bout de 2^ℓ itérations, on retombe donc sur un état déjà rencontré, et la suite chiffrante est donc périodique de période au plus 2^ℓ. Or répéter une même suite chiffrante conduit aux mêmes attaques que sur le chiffrement de Vernam. Il faut donc pouvoir créer différentes suites chiffrantes pour un même choix de clef secrète.

Dans les chiffrements par flot modernes, on utilise une entrée auxiliaire, un vecteur d'initialisation public, IV, afin de pouvoir produire plusieurs suites chiffrantes à partir de la même clef secrète. Au bout de n bits produits avec une même clef secrète, un nouvel IV (public) est choisi par Alice et Bob et l'état interne du chiffrement par flot est réinitialisé avec la clef secrète et ce nouvel IV.

Pour résumer, on a

- Initialisation : choix d'un IV et $S_0 = h(k, \mathrm{IV})$
- Pour $t = 1, \ldots, n$
  - Mise à jour de l'état : $S_t = f(S_{t-1})$

- Extraction du terme de suite chiffrante : $z_t = g(S_t)$

• Retour à l'initialisation.

LFSR

C'est une brique de base intervenant avec d'autres composants dans de nombreuses constructions de chiffrements par flot afin de mettre à jour l'état. Ils permettent de construire à moindre coût des suites avec des périodes élevées et des bonnes propriétés statistiques.

Définition II – 4. Un registre à décalage à rétroaction linéaire (LFSR, *Linear Feedback Shift Register*) binaire de longueur ℓ est un automate composé d'un registre S à décalage de ℓ bits. Au temps $t \geqslant 0$, on note $S^{(t)} = (S_0^{(t)}, \dots, S_{\ell-1}^{(t)})$ l'état du registre.

Son **polynôme de rétroaction** (parfois appelé polynôme de connexion) est un polynôme de $\mathbf{F}_2[X]$ de degré ℓ

$$f(X) := 1 \oplus c_1 X \oplus c_2 X^2 \oplus \cdots \oplus c_\ell X^\ell.$$

Son **état initial** est $S^{(0)} := (z_0, \dots, z_{\ell-1}) \in \mathbf{F}_2^\ell$.

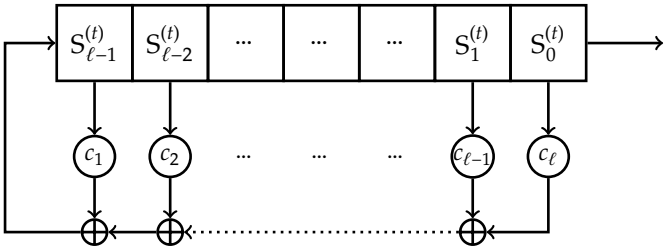
À l'instant t, on sort le bit d'indice 0, $S_0^{(t)}$, et on met à jour pour donner le registre $S^{(t+1)}$ de la façon suivante : les bits d'indices 1 à ℓ sont décalées :

$$S_i^{(t+1)} = S_{i+1}^{(t)}, \text{ pour } 0 \leqslant i \leqslant \ell - 2,$$

et le bit d'indice $\ell - 1$ est mis à jour par une fonction linéaire :

$$S_{\ell-1}^{(t+1)} = c_1 S_{\ell-1}^{(t)} \oplus c_2 S_{\ell-2}^{(t)} \oplus \cdots \oplus c_{\ell-1} S_1^{(t)} \oplus c_\ell S_0^{(t)},$$

où les calculs sont dans $\mathbf{F}_2$. On représente un LFSR ainsi :



Un LFSR est donc un automate qui calcule les termes d'une suite à récurrence linéaire d'ordre ℓ initialisée par $z_0, \dots, z_{\ell-1}$ (le contenu de $S^{(0)}$) et de récurrence $z_t = c_1 z_{t-1} \oplus c_2 z_{t-2} \oplus \cdots \oplus c_{\ell-1} z_{t-\ell+1} \oplus c_\ell z_{t-\ell}$ pour tout $t \geqslant \ell$. À chaque instant $t \geqslant 0$, $S^{(t)}$ contient t termes consécutifs de la suite $S^{(t)} = (z_t, z_{t+1}, \dots, z_{t+\ell-1})$.

Proposition II – 5. La suite z produite par un LFSR de longueur ℓ est périodique de période $T \leqslant 2^\ell - 1$. D'autre part, la période est maximale $T = 2^\ell - 1$ si et seulement si le polynôme de rétroaction est primitif. On parle de m-**suite** ou m-**séquence**.

Démonstration. Un registre peut prendre au plus 2^ℓ états. S'il vaut $(0, \dots, 0)$ alors les registres successifs sont tous nuls et la suite de sortie est elle-même nulle à partir de ce rang, elle est donc périodique de période 1.

Si les registres ne sont jamais nuls alors parmi les 2^ℓ registres $S^{(0)}, S^{(1)}, \dots, S^{(2^\ell-1)}$, au moins deux registres sont identiques. Supposons $S^{(t_0)} = S^{(t_0+T)}$, alors la suite des registres $S^{(t_0)}, S^{(t_0+1)}, \dots, S^{(t_0+T-1)}$ se répète indéfiniment. On a donc $z_{t+T} = z_t$ pour tout $t \geq t_0$ avec $T \leq 2^\ell - 1$.

On note

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ & & & & 0 \\ 0 & \cdots & & 0 & 1 \\ c_\ell & c_{\ell-1} & \cdots & c_2 & c_1 \end{pmatrix}$$

En considérant les registres comme des vecteurs colonnes, cette matrice permet d'exprimer la rétroaction linéaire :

$$S^{(t+1)} = AS^{(t)}.$$

Ainsi, on a $S^{(t)} = A^t S^{(0)}$. En développant par rapport à la première colonne, on remarque que le déterminant de A est égal à $c_\ell = 1$. La matrice A est donc inversible et le LFSR ne passe jamais par le registre nul si l'état initial $S^{(0)}$ est non nul. La condition $S^{(t_0)} = S^{(t_0+T)}$ devient $A^{t_0}S^{(0)} = A^{t_0+T}S^{(0)}$ mais comme A est inversible, on en déduit $S^{(0)} = A^T S^{(0)} = S^{(T)}$ et la suite s est périodique.

On montre ensuite que le polynôme caractéristique de A est le polynôme réciproque du polynôme de rétroaction dont les racines ont le même ordre. En diagonalisant A, on voit que la période de z est l'ordre de A qui est lui même l'ordre des racines du polynôme caractéristique. La période est donc maximale si le polynôme est primitif.

On admet la réciproque. □

Une m-suite sera souhaitable pour des applications cryptographiques. Remarquons que, dans ce cas, dans une période, les registres prennent tous les états possibles de $\mathbf{F}_2^\ell$ sauf le registre nul. En particulier, deux telles suites associées aux mêmes coefficients de récurrence mais pas au même état initial sont en fait décalées l'une de l'autre. D'autre part une m-suite a de bons critères statistiques.

Proposition II – 6. Une m-suite est équilibrée, c'est à dire que dans une période, les nombres de 0 et de 1 diffèrent au plus de 1 (premier critère de Golomb).

Démonstration. On a vu que dans une période, le registre prend toutes les valeurs possibles de $\mathbf{F}_2^\ell$ sauf le registre nul. Le premier élément de chaque registre prend donc $2^{\ell-1}$ fois la valeur 1 et $2^{\ell-1} - 1$ la valeur 0. Ce premier élément étant à chaque tour le bit de sortie, on en déduit que dans une période, dans la m-suite produite, le nombre de 1 et de 0 diffèrent de 1. □

On associe à la suite $z = (z_t)_{t\geqslant 0}$ la série génératrice

$$Z(X) = \sum_{t\geqslant 0} z_t X^t.$$

Proposition II – 7. Une suite z strictement périodique est produite par un LFSR de longueur ℓ dont le polynôme de rétroaction est $f(X) = 1 \oplus c_1 X \oplus c_2 X_2 \oplus \cdots \oplus c_\ell X^\ell$ si et seulement si son développement en série formelle vérifie

$$Z(X) = g(X)/f(X),$$

où g est un polynôme de $\mathbf{F}_2[X]$ tel que $\deg(g) < \deg(f)$. En outre, le polynôme g est entièrement déterminé par l'état initial du registre :

$$g(X) = \sum_{i=0}^{\ell-1} X^i \sum_{j=0}^{i} c_j z_{i-j}.$$

Démonstration. Supposons z produite par un LFSR de polynôme de rétroaction $c_0 \oplus c_1 X \oplus \cdots \oplus c_\ell X^\ell$ avec $c_0 = c_\ell = 1$. On pose

$$g(X) = Z(X)f(X) = (z_0 \oplus z_1 X \oplus \cdots)(c_0 \oplus c_1 X \oplus \cdots \oplus c_\ell X^\ell).$$

On vérifie que g est bien un polynôme. On note g_i le coefficient de g de degré i. On a $g_0 = z_0 c_0$, $g_1 = z_0 c_1 \oplus z_1 c_0$ et pour tout i avec $0 \leqslant i < \ell$, $g_i = \sum_{j=0}^{i} c_j z_{i-j}$. Ensuite, pour tout $i \geqslant 0$,

$$g_{\ell+i} = \sum_{j=0}^{\ell} c_j z_{\ell+i-j} = c_0 z_{\ell+i} \oplus c_1 z_{\ell+i-1} \oplus \cdots \oplus c_\ell z_i = 0,$$

car on retrouve l'équation de rétroaction. Donc g est bien un polynôme de degré inférieur à ℓ. Réciproquement, si $Z(X) = g(X)/f(X)$, alors la suite z satisfait une récurrence linéaire d'ordre ℓ, donnée par la formule précédente. □

Afin d'obtenir une forme canonique de la série génératrice Z, on définit le polynôme de rétroaction minimal : c'est un diviseur de $f(X)$, qui de plus est le polynôme de plus bas degré parmi les polynômes de rétroaction de tous les LFSR possibles qui génèrent la suite z.

Définition II – 8. Soit un LFSR de longueur ℓ d'initialisation non nulle et z sa suite de sortie supposée strictement périodique. Son **polynôme de rétroaction minimal** est l'unique polynôme unitaire f de $\mathbf{F}_2[X]$ tel qu'il existe $g \in \mathbf{F}_2[X]$, avec $\deg(g) < \deg(f)$ et $\operatorname{pgcd}(f, g) = 1$, vérifiant $Z(X) = g(X)/f(X)$. La **complexité linéaire** du LFSR produisant la suite z, notée $\Lambda(z)$, est alors égale au degré de f : c'est la longueur du plus petit LFSR permettant d'engendrer z.

Si le polynôme de rétroaction est irréductible de degré ℓ, on a $\Lambda(z) = \ell$.

Un LFSR de polynôme de rétroaction primitif est donc un bon candidat pour construire un chiffrement par flot : on a une implantation logicielle et matérielle très rapide, de bonnes propriétés statistiques et une grande période possible (un LFSR de ℓ bits peut produire une suite de $2^\ell - 1$ bits). Cependant, on ne peut pas les utiliser directement : ℓ bits consécutifs de la suite chiffrante (obtenue par une attaque à clairs connus avec ℓ bits de clairs) fournissent directement l'état interne. Une solution pourrait être de garder comme clef secrète la fonction de rétroaction, donc la valeur de $(c_1, \ldots, c_\ell)$: même si on récupère l'état interne, on ne pourrait « dérouler » ou « rembobiner » le LFSR. Cependant...

Proposition II – 9. Soit z une suite produite par un LFSR de longueur ℓ et de polynôme de rétroaction f irréductible de degré ℓ. Si on connaît 2ℓ bits consécutifs de z alors on peut retrouver les coefficients de rétroaction en inversant un système linéaire $\ell \times \ell$.

Démonstration. Comme z est une suite produite par un LFSR de longueur ℓ et de polynôme de rétroaction f irréductible de degré ℓ, on a $\Lambda(z) = \ell$. On suppose que l'on connaît 2ℓ bits à partir du temps t : $z_t, z_{t+1}, z_{t+2}, \ldots, z_{t+2\ell-1}$. On construit le système linéaire suivant :

$$\begin{pmatrix} z_t & z_{t+1} & z_{t+2} & \cdots & z_{t+\ell-1} \\ z_{t+1} & z_{t+2} & \cdots & \cdots & z_{t+\ell} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ z_{t+\ell-2} & \cdots & \cdots & \cdots & z_{t+2\ell-3} \\ z_{t+\ell-1} & z_{t+\ell} & \cdots & \cdots & z_{t+2\ell-2} \end{pmatrix} \begin{pmatrix} c_\ell \\ c_{\ell-1} \\ \vdots \\ c_2 \\ c_1 \end{pmatrix} = \begin{pmatrix} z_{t+\ell} \\ z_{t+\ell+1} \\ \vdots \\ z_{t+2\ell-2} \\ z_{t+2\ell-1} \end{pmatrix}.$$

Les lignes de la matrice sont les registres $S^{(t)}, S^{(t+1)}, \ldots, S^{(t+\ell-1)}$. Si le système a une solution, les coefficients de rétroaction seront donc bien solutions. On montre en fait que le système a une unique solution en montrant que les lignes sont indépendantes (par l'absurde, on montre que sinon cela donnerait une récurrence linéaire sur la suite d'ordre $< \ell$, ce qui contredit le fait que $\Lambda(z) = \ell$). □

Si la longueur du LFSR est ℓ, sa complexité linéaire est au plus ℓ. Donc en la devinant, la proposition montre que l'on peut retrouver le LFSR en temps au plus $\mathcal{O}(\ell^4)$ avec seulement 2ℓ bits consécutifs de suite chiffrante. Cette attaque polynomiale empêche d'utiliser tel quel un LFSR. De plus, on peut faire mieux (en temps $\mathcal{O}(\ell^2)$ avec un algorithme du à Berlekamp et Massey).

En pratique, la mise à jour des LFSRs modernes fait également intervenir au moins un composant non linéaire (des fonctions booléennes), par exemple SNOW3G et ZUC sont des constructions de type LFSR filtré : l'état interne d'un LFSR est l'entrée d'une fonction non linéaire qui produit le bit de suite chiffrante.

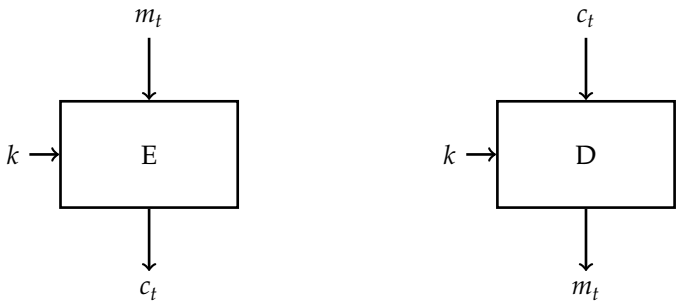
Chapitre III

Chiffrement par bloc

1. Introduction, modes opératoires

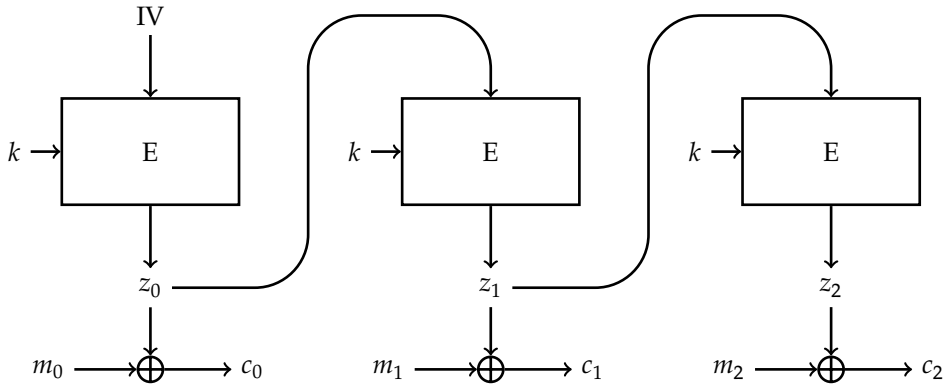
Un algorithme de **chiffrement par bloc** est un algorithme de cryptographie symétrique. Il prend en entrée un message clair de n bits (que l'on peut voir comme un élément de $\mathbf{F}_2^n$) et donne en sortie un chiffré, qui est un autre bloc de bits (en général également n), en utilisant une clef secrète k. Si le message clair est de taille plus grande que n on le découpe en des blocs, $m_0, m_1, \ldots$ de taille n. Puis pour chaque $t = 0, 1, 2, \ldots$, on applique le chiffrement. Le déchiffrement se fait de manière similaire, avec la même clef k.

On peut voir la fonction de chiffrement comme une permutation de $\mathbf{F}_2^n$ sélectionnée par la clef (2^ℓ choix pour une clef de ℓ bits) parmi les $2^n!$ possibles. Tout l'enjeu est de produire une permutation la plus « aléatoire » possible tout en gardant une certaine structure pour avoir un algorithme de longueur succincte (décrire une permutation sans structure quelconque se fait en donnant la liste de ses images ici $2^n \times n$ bits et $n = 128$ en pratique). Contrairement au chiffrement par flot, les chiffrements par blocs n'ont pas d'état interne, et sont en général plus complexes à évaluer.



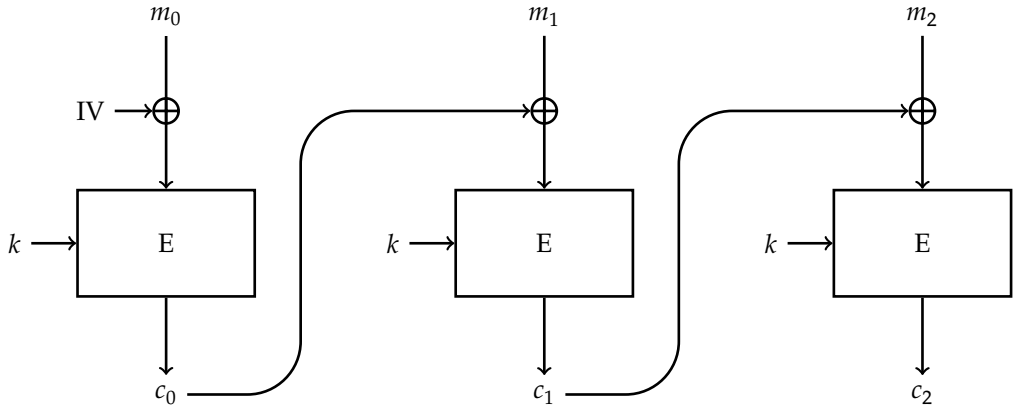
Ci-dessus le mode ECB, *Electronic CodeBook*. Ce mode ne cache pas les redondances éventuelles du texte clair, (par exemple si $m_i = m_j$ avec $i \neq j$ alors $c_i = c_j$). On n'a donc pas de sécurité sémantique.

D'autres modes opératoires remédient à ce problème. Par exemple, le mode OFB (*Output FeedBack*) permet d'obtenir un chiffrement par flot :



Il en est de même du mode CTR (compteur) où $c_t = m_t \oplus \mathrm{E}(k, \mathrm{IV} \oplus \langle t \rangle_n)$ où $\langle t \rangle_n$ est la représentation de l'entier t sur n bits. Comparé au mode OFB, ce mode, très populaire, permet de procéder directement au chiffrement ou déchiffrement à n'importe quel temps sans devoir générer toute une suite chiffrante.

Autre exemple, le mode CBC (*Cipher Block Chaining*) utilise le schéma suivant :



Tous ces modes (sauf ECB) permettent d'avoir une sécurité sémantique pour des attaques à chiffrés seuls, en idéalisant le chiffrement par bloc utilisé, quel que soit le choix d'IV (fixe, *number used once* (*nonce*), aléatoire). Si l'IV est pris aléatoirement, on peut également montrer que tous ces modes (sauf ECB) donnent une sécurité sémantique pour des attaques à clairs choisis. Ainsi on assure bien la confidentialité des données échangées. Cependant ces modes ne protègent pas contre un attaquant qui modifierait les chiffrés ou des attaques à chiffrés choisis. Nous verrons comment assurer l'intégrité des données et les authentifier au chapitre suivant.

La *construction* des chiffrements par bloc utilise la plupart du temps un schéma itératif. Les itérations, appelées tours ou rondes, sont en général identiques (à part la première et la dernière), seule la clef de tour, créée à partir de la clef secrète k au moyen d'un algorithme dit de cadencement de clef, change.

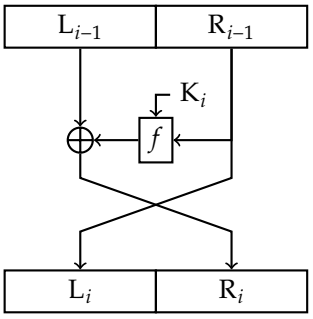
$$m = x_0 \xrightarrow{\mathrm{F}_{\mathrm{K}_0}} x_1 \xrightarrow{\mathrm{F}_{\mathrm{K}_1}} x_2 \longrightarrow \cdots \xrightarrow{\mathrm{F}_{\mathrm{K}_{r-1}}} x_r = c$$

Cette construction itérative permet une description concise de l'algorithme de chiffrement. On doit juste décrire la permutation F et l'algorithme de cadencement de clefs. Pour obtenir cette permutation F, deux classes générales de constructions ont été proposées : les schémas de Feistel et les schémas substitution permutation (SPN).

2. Schéma de Feistel, le DES

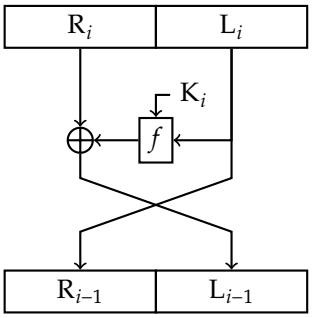
Cette construction a été introduite par Feistel dans les années 70. On suppose que le bloc de message clair est de longueur paire n et on le découpe en deux blocs de longueur $\frac{n}{2}$, notés L_0 et R_0. On note $m = L_0 || R_0$. À chaque tour $i = 1, 2, \dots, r$, on prend en entrée un bloc (L_{i-1}, R_{i-1}) et on le transforme en un bloc (L_i, R_i) en faisant intervenir la clef de tour K_i. On note f une fonction prenant en entrée et en sortie des blocs de $n/2$ bits. La transformation se fait par les formules :

$$L_i = R_{i-1}, \qquad R_i = L_{i-1} \oplus f(K_i, R_{i-1})$$



Au bout de r tours, le chiffré est $c = R_r || L_r$ (on ne « croise » pas les flèches au dernier tour). Ce schéma est inversible, (si l'on connaît les clefs de tours), que f soit une bijection ou non. En effet, on a

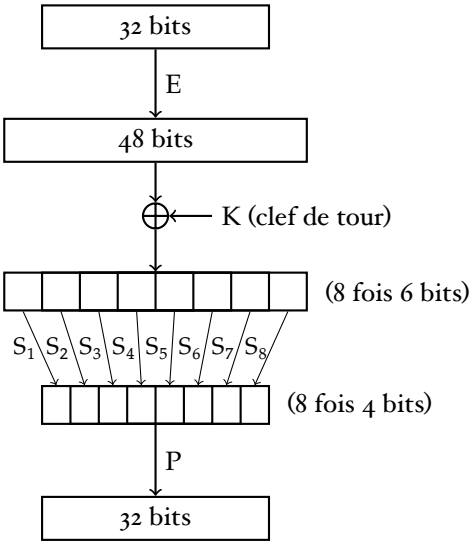
$$R_{i-1} = L_i, \qquad L_{i-1} = R_i \oplus f(K_i, R_{i-1}).$$



Le déchiffrement de $c = R_r || L_r$ se fait donc avec exactement le même procédé que le chiffrement en appliquant les clefs de tours dans l'ordre inverse. On retrouve bien, à la dernière étape, (toujours sans croiser) $L_0 || R_0$.

Le **DES**, ***Data Encryption Standard***, standard de chiffrement par bloc de 1977 à 2000 utilise un schéma de Feistel. Les blocs de clair et chiffré sont de 64 bits, la clef secrète de 56 bits et les clefs de tour de 48 bits. On effectue 16 tours de schéma de Feistel avec des permutations initiale et finale.

La fonction de tour opère sur des mots de 32 bits. Elle est la composition de plusieurs fonctions, suivant le schéma suivant.



La fonction d'expansion E est linéaire de $\mathbf{F}_2^{32} \to \mathbf{F}_2^{48}$ (on répète certains bits). La fonction P est une permutation des 32 bits, donc aussi une fonction linéaire de $\mathbf{F}_2^{32} \to \mathbf{F}_2^{32}$. Ces deux fonctions, E et P apportent de la **diffusion** c'est-à-dire que si l'on change un bit en entrée, cette modification va se propager sur l'ensemble de l'état interne.

Les fonctions $S_1, \dots, S_8$ sont appelées **boîtes S** ou *S-box*. Ce sont des fonctions dites booléennes vectorielles, ici de $\mathbf{F}_2^6 \to \mathbf{F}_2^4$, non linéaires. Les boites S sont décrites par la table des 2^6 sorties possibles. Elles permettent d'apporter de la **confusion** : le but est de rendre complexe les relations entre bits de chiffré et bits de clef.

Les concepts de diffusion et confusion ont été introduits par Shannon en 1949. Les chiffrements par blocs modernes sont bâtis sur l'alternance de ces étapes de confusion et de diffusion.

Voir `https://en.wikipedia.org/wiki/DES_supplementary_material` pour une description précise de ses fonctions.

Pour palier à la faiblesse du DES due à sa clef de 56 bits trop courte, on utilise encore couramment aujourd'hui (dans le monde bancaire) une variante utilisant 3 clefs DES k_1, k_2, k_3, appelée **Triple DES**. Cela consiste à composer trois fois le DES (donc trois fois plus lent que le DES) de la manière suivante :

$$c = \mathrm{E}\Big(k_3, \mathrm{D}\big(k_2, \mathrm{E}(k_1, m)\big)\Big),$$

avec trois options sur le choix des clefs :

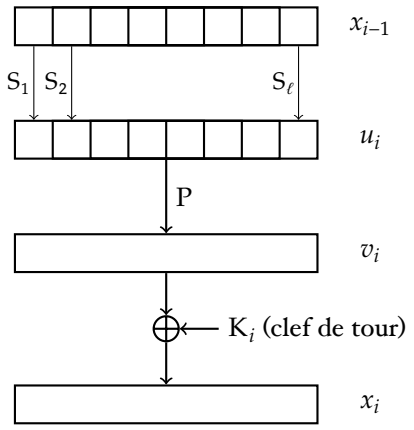
Option 1 : trois clefs distinctes, ce qui revient à 168 bits de clef ;

Option 2 : $k_1 = k_3$ et $k_2 \neq k_1$, ce qui revient à 112 bits de clef ;

Option 3 : $k_1 = k_2 = k_3$, une seule clef de 56 bits, et $c = \mathrm{E}(k_1, m)$ on a un chiffrement classique du DES pour garantir la compatibilité (ce qui explique pourquoi on alterne chiffrement et déchiffrement).

3. Schéma substitution permutation (SPN), l'AES

C'est une autre construction itérative de chiffrement par bloc. On utilise maintenant une fonction de tour bijective en alternant les opérations de diffusion et de confusion. On effectue une étape initiale d'ajout bit à bit de la première clef de tour : $x_0 = m + \mathrm{K}_0$. Puis pour $i = 1, \dots, r$, on calcule $x_i = \mathrm{F}_{\mathrm{K}_i}(x_{i-1})$ pour obtenir $c = x_r$. La fonction f commence par l'application de ℓ boîtes S bijectives (étape de confusion) sur x_{i-1} découpé en ℓ sous blocs, pour donner un nouveau bloc u_i. Puis on applique une permutation P (étape de diffusion) sur les bits de u_i, on note v_i le résultat. Enfin, on ajoute la clef de tour : $x_i = v_i + \mathrm{K}_i$.



L'étape initiale évite qu'un attaquant puisse calculer le début du chiffrement jusqu'à l'ajout de clef K_1. Le déchiffrement se fait en « remontant » tout le chiffrement, toutes les opérations étant inversibles.

Un exemple de tel schéma est l'**AES**, ***Advanced Encryption Standard***. Ce standard pour remplacé le DES est issu d'un concours qui s'est déroulé de 1997 à 2001. Le vainqueur a été l'algorithme Rijndael conçu par Joan Daemen et Vincent Rijmen. L'AES utilise une clef du 128, 192 ou 256 bits avec des blocs de 128 bits. Suivant la taille de la clef, le nombre de tours (en plus de l'étape initiale d'ajout de clef k_0) est respectivement 10, 12 et 14. On détaille le fonctionnement de l'AES dans le cas 128 bits.

L'état interne est vu comme un tableau de 4×4 octets. Chaque octet étant identifié avec un élément de $\mathbf{F}_{2^8}$ par le choix du polynôme irréductible (non primitif)

$$T(X) = X^8 + X^4 + X^3 + X + 1.$$

En notant $\alpha = X \mod T(X)$, on a

$$\mathbf{F}_{256} = \left\{ \sum_{i=0}^{7} b_i \alpha^i, b_i \in \mathbf{F}_2 \right\}.$$

Ainsi comme les éléments de $\mathbf{F}_2$ sont les bits 0 et 1, un élément de $\mathbf{F}_{256}$ peut être représenté par 8 bits, c'est à dire un octet. Plus précisément, dans l'AES, chaque élément $b_0 + b_1\alpha + \cdots + b_7\alpha^7$ est identifié avec l'octet $(b_7, b_6, \ldots, b_0) \in \mathbf{F}_2$ (Attention au sens d'écriture !).

L'état est ainsi une matrice de $\mathscr{M}_4(F_{2^8})$, l'ensemble des matrices 4×4 à coefficients dans $\mathbf{F}_{2^8}$.

La fonction de diffusion est la composée de deux fonctions linéaires, `ShiftRows` et `MixColumns`. La fonction `ShiftRows` consiste en une permutation circulaire des lignes de la matrice tandis que `MixColumns` est une multiplication par une matrice fixe inversible de $\mathscr{M}_4(\mathbf{F}_{2^8})$. Ces fonctions linéaires n'agissent que sur les octets, et pas sur les bits de l'état.

La fonction de substitution, `SubBytes`, consiste en une seule boîte S appliquée 16 fois, sur chaque octet de l'état. Cette fonction consiste à composer l'inversion dans $\mathbf{F}_{2^8}$ (complétée par $0 \mapsto 0$) avec une transformation affine dans $\mathbf{F}_2^8$, du type $Ax + b$ avec A une matrice inversible fixe de $\mathscr{M}_8(\mathbf{F}_2)$ et b un vecteur fixe de $\mathbf{F}_2^8$. Cette transformation permet de casser le caractère algébrique de l'inversion.

Tout le déroulement de l'AES suit le schéma général d'un SPN, hormis le tour final qui n'inclut pas la fonction *MixColumns*. Au final, le fonctionnement à haut niveau de l'AES est le suivant. On part d'un bloc de 128 bits de texte clair et on applique successivement sur ce bloc les opérations suivantes :

1. `AddRoundKey`(K_0) ;
2. Pour $i = 0, \ldots, r-2$, on effectue le tour comportant les 4 opérations `SubBytes`, `ShiftRows`, `MixColumns`, `AddRoundKey`(K_{i+1}) dans cet ordre ;
3. Un dernier tour ne comporte plus que 3 étapes : `SubBytes`, `ShiftRows`, `AddRoundKey`(K_r) ;
4. Le contenu actuel du bloc donne les 128 bits du texte chiffré.

De nombreuses attaques ont été proposées sur des versions réduites de l'AES (en réduisant le nombre de tours). À ce jour la meilleure attaque connue sur l'AES complet (Bogdanov, Khovratovich, Rechberger, 2011) ne gagne qu'un facteur 4 sur la recherche exhaustive (donc 2^{126} opérations pour une clef de 128 bits).

On détaille maintenant le fonctionnement plus précis des 4 opérations de l'AES.

Représentation des éléments

Tout d'abord, le message clair, puis chaque résultat intermédiaire est un bloc de 128 bits. On le voit comme 16×8 bits c'est à dire 16 octets. On range ces octets dans un tableau 4×4 :

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

D'autre part, comme vu dans la section précédente, chaque octet est identifié avec un élément du corps fini $\mathbf{F}_{256}$.

`SubBytes` (boîte S)

Elle opère indépendamment sur chacun des 16 octets. C'est la composée $\mathrm{S} = f \circ \mathrm{I}$ des applications

$$\begin{aligned} \mathrm{I} : \mathbf{F}_{256} &\to \mathbf{F}_{256} \\ x &\mapsto \begin{cases} x^{-1} & \text{si } x \neq 0, \\ 0 & \text{si } x = 0, \end{cases} \end{aligned}$$

et

$$\begin{aligned} f : (\mathbf{F}_2)^8 &\to (\mathbf{F}_2)^8 \\ y &\mapsto \mathrm{A}y + \mathrm{B} \end{aligned}$$

où A est une matrice 8×8 à coefficients dans $\mathbf{F}_2$, B est un vecteur de $(\mathbf{F}_2)^8$, explicités en dessous. Plus précisément, on identifie comme on l'a vu l'élément $\sum_{i=0}^{7} y_i \alpha^i$ de $\mathbf{F}_{256}$ calculé par I avec l'octet $y = (y_7, y_6, \dots, y_0)$ puis on calcule $\mathrm{A}y + \mathrm{B}$ ainsi :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

En pratique, lorsque l'on implante l'AES, cette boîte S est donnée par une table décrivant toutes les sorties[1], on ne fait pas les calculs dans le corps $\mathbf{F}_{256}$ et dans l'espace vectoriel $(\mathbf{F}_2)^8$.

La fonction I d'inversion dans le corps $\mathbf{F}_{256}$ a été choisie car on peut montrer qu'elle a de très bonnes propriétés pour se protéger contre certaines attaques avancées. Ensuite, on compose par la fonction affine f pour casser le caractère algébrique de I et enlever le point fixe $0 \mapsto 0$.

`ShiftRows`

Elle fait subir une permutation circulaire vers la gauche aux lignes du tableau, respectivement de 0, 1, 2, 3 cases :

[1] On peut la trouver ici : `https://en.wikipedia.org/wiki/Rijndael_S-box`

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$\longrightarrow$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

`MixColumns`

Elle s'interprète comme une multiplication matricielle :

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$\longrightarrow$

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

où

$$\begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} = \begin{pmatrix} \alpha & \alpha+1 & 1 & 1 \\ 1 & \alpha & \alpha+1 & 1 \\ 1 & 1 & \alpha & \alpha+1 \\ \alpha+1 & 1 & 1 & \alpha \end{pmatrix} \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}.$$

est le produit des matrices à coefficients dans $\mathbf{F}_{256}$. La matrice de cette opération a été choisie pour ses propriétés de diffusion.

`AddRoundKey`(k_{i+1})

C'est l'addition bit à bit (le xor) de la clef de tour k_{i+1}, case par case.

L'algorithme de cadencement de clefs, voir par exemple `https://en.wikipedia.org/wiki/AES_key_schedule` pour une description, calcule à partir de la clef secrète k une suite de $r+1$ clefs de tour $(k_0, \dots, k_r)$, comportant toutes 128 bits.

Chapitre IV

Fonctions de hachages, MAC

1. Fonctions de hachages

Une fonction de hachage h est une application prenant en entrée des messages, des suites de bits de longueurs quelconques et retournant un **haché** ou une empreinte de longueur fixé, par exemple une suite binaire de longueur n :

$$h : \{0,1\}^* \longrightarrow \{0,1\}^n.$$

On veut que h soit rapide à évaluer.

Originellement, ces fonctions ont été introduites dans le contexte des bases de données afin d'y « ranger » des objets de natures diverses. On cherche donc en général à éviter les collisions (le fait que $h(x) = h(y)$ pour $x \neq y$) pour éviter que deux objets ne se retrouvent au même endroit, ce qui allonge la recherche dans la base de données. En cryptographie, on veut des propriétés plus restrictives (on parle parfois de **fonctions de hachage cryptographiques**).

Propriétés de sécurité attendues

- À **sens-unique** : étant donné un haché $y \in \{0,1\}^n$ il est difficile de trouver $m \in \{0,1\}^*$ tel que $h(m) = y$.
- Résistance à la **seconde pré-image** : étant donné un message $m \in \{0,1\}^*$, il est difficile de trouver $m' \neq m$ avec $m' \in \{0,1\}^*$ tel que $h(m) = h(m')$.
- Résistance aux **collisions** : il est difficile de trouver $m \neq m'$ avec $m, m' \in \{0,1\}^*$ tels que $h(m) = h(m')$.

Si h n'est pas injective, il existe des collisions. Par exemple, si h désigne l'application qui a une personne associe le jour de son anniversaire (à valeurs dans $\{1, \dots, 365\}$), alors le paradoxe des anniversaires nous dit qu'avec 23 évaluations différentes de h on a plus de 50% de chances d'avoir une collision. De manière générale, si h est à valeurs dans $\{0,1\}^n$, en $\mathcal{O}(2^{n/2})$ évaluations on a une bonne probabilité d'obtenir une collision. En pratique, on prend aujourd'hui $n \geqslant 256$ pour avoir une sécurité de 128 bits : la meilleure attaque prend plus de 2^{128} opérations.

Plus généralement, on aimerait qu'une fonction de hachage se comporte comme un oracle aléatoire : pour obtenir le haché de m, on soumet m à un oracle modélisant h qui répond par une valeur aléatoire de $\{0,1\}^n$, $h(m)$, obtenue avec équiprobabilité. Par contre, si on a déjà soumis m à l'oracle, alors on obtient la même valeur $h(m)$.

Quelques Applications

Une application directe est d'assurer l'**intégrité** de messages ou de fichiers. Modifier un fichier donné afin qu'il ait toujours un haché donné correspond à trouver une seconde pré-image (mais cette application pose le problème de l'intégrité du haché).

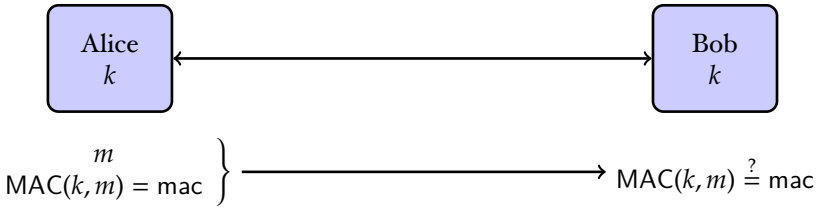
Une autre application est la **vérification de mot de passe**. Plutôt que de stocker des mots de passe sur un serveur, on stocke leur haché. Lors d'une authentification, le serveur hache le mot de passe reçu et compare les hachés. Cela évite la divulgation des mots de passe en cas de compromission du serveur. Retrouver les mots de passe à partir du haché revient à attaquer la notion de sens-unique (en général on fait une attaque par dictionnaire, le mot de passe appartenant à un ensemble de cardinal petit).

D'autres applications comme extracteur d'aléa, pour la génération de mot de passe à usage unique (*One-Time Password*), ou la dérivation de clefs de chiffrement symétrique (à partir d'un mot de passe par exemple).

On verra d'autres applications pour les **signatures numériques**. On ne signe pas directement un message mais un haché du message. Le **chiffrement asymétrique** utilise aussi souvent des fonctions de hachage (par exemple le standard RSA-OAEP) pour résister aux attaques à chiffrés choisis.

2. MAC

C'est une primitive proche des fonctions de hachage : *Message authentication code* (**MAC**). Lors d'une communication, un MAC permet de garantir **l'intégrité** du message et **authentifie** l'expéditeur par l'utilisation d'une clef secrète.



La notion de sécurité la plus forte pour un MAC consiste à ce qu'un attaquant ne puisse créer un nouveau MAC valide étant connus des couples $(m_i, \mathsf{MAC}(k, m_i))$ pour des m_i de son choix.

On peut construire des MAC directement, par des chiffrements par blocs, ou à partir de fonctions de hachages. Une construction populaire, HMAC, proposée par Bellare, Canetti, et Krawczyk en 1996, est standardisée et utilisée dans de nombreux protocoles. On a $\mathsf{HMAC}(k, m) = h(k_2 \| h(k_1 \| m))$ avec $k_2 = k \oplus \mathsf{opad}$ et $k_1 = k \oplus \mathsf{ipad}$, avec opad et ipad des constantes. La notation $\|$ désigne la concaténation des chaînes binaires. On verra en TD pourquoi on évite des constructions plus simples du type $\mathsf{MAC}(k, m) = h(k \| m)$ avec certaines fonctions de hachages.

Retour sur les mode opératoires

Pour assurer confidentialité, intégrité et authentification d'une communication on combine un mode opératoire d'un chiffrement par bloc avec un MAC ou un procédé plus léger apportant ces propriétés. On parle de chiffrement authentifié (*Authenticated Encryption*).

Exemples

Une méthode générique (*Encrypt-then-MAC*) : on chiffre m en $c = \mathrm{E}(k_1, m)$ et on l'envoie au destinataire avec $\mathsf{mac} = \mathsf{MAC}(k_2, c)$. Celui-ci connaissant (k_1, k_2) peut déchiffrer et vérifier le mac afin de prendre en compte ou non le message. Ceci permet d'obtenir une sécurité sémantique contre des attaques à chiffrés choisis où l'attaquant peut obtenir le déchiffrement de certains messages.

Le mode opératoire GCM (*Galois/counter mode*) combine le mode compteur avec un procédé d'authentification adapté. Voici une présentation simplifiée[1] : on utilise tout d'abord le mode CTR pour obtenir

[1] On peut trouver une description complète ici `https://en.wikipedia.org/wiki/Galois/Counter_Mode`, incluant le traitement de la longueur du message et de données additionnelles non chiffrées.

une suite de chiffré $c_0, c_1, c_2, \ldots$ en utilisant un chiffrement par bloc de 128 bits, E avec une clef k. On désigne par $\mathrm{H} = \mathrm{E}(k, 00\cdots00)$ le chiffrement du bloc nul. On voit les blocs de 128 bits comme éléments du corps $\mathbf{F}_{2^{128}}$ pour un certain choix de polynôme minimal. On calcule ensuite successivement $x_0 := c_0 \times \mathrm{H}$ dans ce corps puis $x_1 = (x_0 + c_1) \times \mathrm{H}$, $x_2 = (x_1 + c_2) \times \mathrm{H}$ ... La valeur finale sera une sorte de mac, qui pourra être vérifiée par le même procédé par le destinataire. Ce mode est un standard très performant et populaire couplé avec l'AES (WPA-3, SSH, TLS...).

Un concours (*Competition for Authenticated Encryption : Security, Applicability, and Robustness*, CAESAR) a été organisé (2013-2019), plusieurs nouveaux schémas et modes opératoires ont été sélectionnés suivant les cas d'applications (Ascon, AEGIS-128, Deoxys-II, ACORN, OCB, COLM).

3. Constructions de fonctions de hachages

Exemples

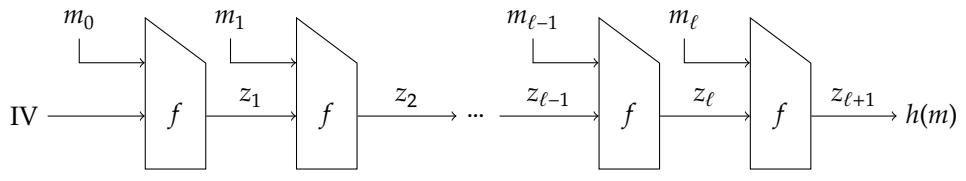
- MD5 128 bits, Rivest 1992, collision en quelques secondes, obsolète.
- SHA1, 160 bits, NIST 1993, collision trouvée en $2^{63.1}$ opérations par Stevens, Bursztein, Karpman, Albertini, Markov en 2017, obsolète
- SHA2, plusieurs variantes de 224 à 512 bits, NIST 2001, pas d'attaque sur la fonction complète
- SHA3, vainqueur de la compétition du NIST (2007-2012), Keccak, 224 à 512 bits, Bertoni, Daemen, Peeters, Van Assche, 2008

Les attaques par collisions sont des attaques dites différentielles : on considère une paire de messages avec une petite différence et on cherche à contrôler la propagation des différences.

Construction de Merkle-Damgård

Cette construction est suivie par les fonctions MD5, SHA1 et SHA2 avec des variations sur les itérations initiale et finale. Elle permet de transformer une fonction de hachage admettant une entrée de longueur fixée (dite fonction de compression) en une fonction de hachage admettant une entrée de longueur (presque) quelconque. On note f une fonction de compression de $\{0,1\}^{n+k}$ dans $\{0,1\}^n$, avec $k > 0$. Soit IV un élément fixé de $\{0,1\}^n$. Soit m message à hacher. On commence par découper m en ℓ blocs de k bits, $m_0, \ldots, m_{\ell-1}$ en « paddant » $m_{\ell-1}$ par $10000\ldots$ pour obtenir un bloc de k bits. Dans le bloc m_ℓ, on code sur exactement k bits le nombre de bits du message m. Il faut donc que m ait strictement moins de 2^k bits. Ce rajout de la longueur de m est parfois appelé *Merkle–Damgård strengthening*. Il permet d'éviter que le haché de m soit le même que celui d'une sous-chaîne de m en ajustant l'IV, ou alors de construire des collisions à l'aide d'une pré-image de l'IV.

On note $z_0 = \mathrm{IV}$ et pour $i = 0, \ldots, \ell$, $z_{i+1} = f(m_i \| z_i)$. Le haché $h(m)$ de m est alors $h(m) = z_{\ell+1}$.



Théorème IV – 1 (Merkle, Damgård 1989)**.** Soit f une fonction de compression résistante aux collisions. La construction précédente appliquée à f donne une fonction de hachage h résistante aux collisions.

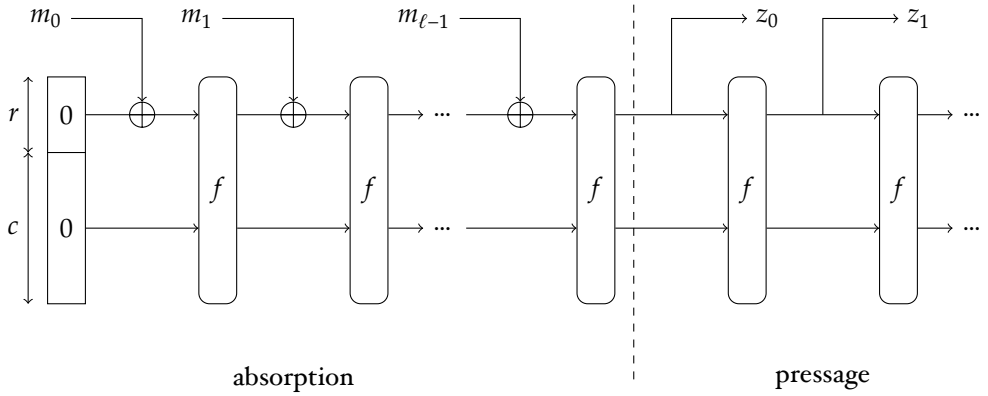
Démonstration. Soit $m \neq m'$ tel que $h(m) = h(m')$. Supposons que m et m' aient des longueurs différentes. La dernière itération donne $h(m) = f(m_\ell \| z_\ell) = h(m') = f(m'_{\ell'} \| z'_{\ell'})$. On a alors $m_\ell \neq m'_\ell$ puisque ce bloc code la longueur. Donc on a trouvé une collision sur f.

Supposons maintenant m et m' de même longueur, ce qui implique que le nombre d'itérations est le même pour m et m' : $\ell = \ell'$. On a $f(m_\ell \| z_\ell) = f(m'_\ell \| z'_\ell)$, avec $m_\ell = m'_\ell$. Soit $z_\ell \neq z'_\ell$ et on a trouvé une collision sur f, soit $z_\ell = z'_\ell$. Dans ce cas, on a, à l'itération précédente, $f(m_{\ell-1} \| z_{\ell-1}) = f(m'_{\ell-1} \| z'_{\ell-1})$, soit $m_{\ell-1} \| z_{\ell-1} \neq m'_{\ell-1} \| z'_{\ell-1}$ et on a trouvé une collision, soit $m_{\ell-1} \| z_{\ell-1} = m'_{\ell-1} \| z'_{\ell-1}$. On remonte ainsi jusqu'à trouver une collision sur f. S'il n'y a pas de collision, $m_i \| z_i = m'_i \| z'_i$ pour $i = 0$ à ℓ, ce qui implique que $m = m'$ et on a une contradiction. □

Pour SHA2, SHA256 (et SHA224 qui est une version tronquée) on utilise Merkle Damgård avec une fonction f de compression $\{0,1\}^{n+k}$ dans $\{0,1\}^n$ avec $n = 256$ et $k = 512$. Cette fonction f consiste à effectuer 64 tours opérant sur un état interne de 8×32 bits en composant des permutations, des additions modulo 2^{32}, des rotations, et l'application de fonctions booléennes simples. De même, SHA512 (et SHA384) utilise $n = 512$ et $k = 1024$ et une fonction f de 80 tour opérant sur un état interne de 8×64 bits.

Fonctions éponges

Une autre construction de fonction de hachage a été proposée et utilisée par les auteurs de Keccak (Bertoni, Daemen, Peeters, Van Assche), le vainqueur du concours SHA3 : les fonctions éponges. Un état interne de $r + c$ bits est utilisé. On note $m_0, \ldots, m_{\ell-1}$ le message avec *padding*, découpé en blocs de r bits. À chacun des ℓ tours, le bloc m_i est ajouté bit à bit aux r premiers bit de l'état, puis une permutation f est appliqué à l'état (c'est la partie d'absorption). Une fois que tout le message a été traité, le haché est produit sur plusieurs tours : à chaque tour les r premiers bits de l'état donnent une partie du haché ($z_0, z_1, \ldots$ sur le dessin ci-dessous), et la fonction f est appliquée sur l'état (c'est la partie où l'on presse l'éponge). Quand f est une permutation aléatoire, les sorties de cette construction sont indistinguables d'un oracle aléatoire.



SHA3 a été standardisé par le NIST en 2015[2]. Comme pour SHA2, plusieurs versions de SHA3 sont spécifiées : SHA3-224, SHA3-256, SHA3-384, SHA3-512, Ces versions correspondent à fixer $r + c = 1600$ bits dans la construction de Keccak. Les tailles de hachés n sont respectivement $224, 256, 384$ et 512 bits, et $c = 2n$, on a donc r égal respectivement à $1152, 1088, 832$ ou 576 suivant la taille du haché. L'état interne, de 1600 bits est donc beaucoup plus gros que pour les fonctions basées sur Merkle Damgård. Un seul tour de pressage d'éponge est effectuée comme r est plus grand que n (les n bits de poids fort sont utilisés).

La permutation f est construite en appliquant 24 fois une fonction de tour. L'état interne est vu comme un pavé de $5 \times 5 \times 64$ bits. Cette fonction de tour est la composition de 5 fonctions. La première, θ, consiste à rajouter à chaque bit de l'état la parité de deux colonnes. La deuxième ρ, est une rotation circulaire sur chaque ligne (dans le sens de l'axe des z). La fonction π, est une permutation des bits de chaque tranche du plan (x, y). La fonction χ est la seule fonction non linéaire, elle consiste à appliquer la fonction booléenne $x_1 \oplus x_2 x_3 \oplus x_3$ sur 3 bits d'une même ligne (dans le sens de l'axe des x). Enfin la fonction ι, change uniquement une ligne (dans le sens de l'axe des z) en lui ajoutant une constante ne dépendant que du numéro de tour, calculée à l'aide d'un LFSR.

[2] cf. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`

Chapitre V

Cryptographie fondée sur le problème du logarithme discret

1. Le problème du logarithme discret

Définition

Soit $(G, \times)$ un groupe cyclique d'ordre n et g un générateur. $G = \langle g \rangle = \{g, g^2, \dots, g^{n-1}, g^n = 1\}$. Étant donné $h \in G$, le **problème du logarithme discret** consiste à retrouver x défini modulo n tel que $h = g^x$. On note $\log_g(h) = x$. En général, n sera un nombre premier.

Si on considère le morphisme de groupe $exp_g : (\mathbf{Z}/n\mathbf{Z}, +) \to (G, \times)$, $a \mapsto g^a$, $\log_g$ est le morphisme inverse de $(G, \times)$ dans $(\mathbf{Z}/n\mathbf{Z}, +)$.

Si ce problème est difficile dans G alors la fonction exp_g est à sens unique. Ce problème n'est pas toujours difficile par exemple si $G = (\mathbf{Z}/n\mathbf{Z}, +)$. Dans ce cas un entier g engendre G si $\text{pgcd}(g, n) = 1$. Si $h \in G$, en notation additive, on a $h \equiv xg \pmod{n}$ et on peut trouver x en calculant $x = hg^{-1} \pmod{n}$.

Cependant il existe des groupes dans lequel ce problème est supposé difficile notamment dans des sous groupes de $\mathbf{Z}/p\mathbf{Z}$ pour p premier et des groupes utilisant des courbes elliptiques sur des corps finis. Ces deux groupes permettent de construire de nombreux systèmes cryptographiques basés sur le problème du logarithme discret.

Propriétés

Si g et g' sont deux générateurs alors $g' = g^a$ avec a inversible modulo n : en effet il existe aussi b tel que $g = g'^b = g^{ab}$, donc $ab \equiv 1 \mod n$. Réciproquement, si $h = g^a$ avec a inversible alors h est un générateur : si $h^b = 1$ alors $g^{ab} = 1$ donc $ab \equiv 0$ donc $b \equiv 0$.

Changement de générateur : si g, g' sont deux générateurs, $h = g^a$ et $h = g'^b$, c'est à dire $\log_g(h) = a$ et $\log_{g'}(h) = b$. On pose $c = \log_g g'$. Alors $g' = g^c$ donc $h = g^{bc} = g^a$ et $a \equiv bc$. Ainsi, $\log_g(g) = \log_{g'}(h) \times \log_g g'$, ou $\log_{g'}(h) \equiv \log_g(g) \log_g g'^{-1}$ où le dernier terme est inversible car g' est un générateur.

Par conséquent, si on sait calculer des logarithmes discrets dans un base g alors on peut en calculer en base g'. La difficulté du calcul de logarithme discret dépend donc seulement du groupe et non pas du choix du générateur.

2. Quelques applications cryptographiques

L'échange de clefs de Diffie-Hellman (76)

Ce protocole interactif permet à Alice et Bob d'échanger publiquement des informations et à la fin du protocole de connaître une quantité qui pourra servir comme clef secrète pour faire du chiffrement symétrique : on souhaite qu'un adversaire Oscar qui écoute la conversation entre Alice et Bob n'ait aucune information sur cette clef.

Pour cela Alice et Bob se mettent tout d'abord publiquement d'accord sur un groupe $(G, \times)$ cyclique d'ordre n et g un générateur.

- Ensuite Alice choisit un a aléatoire $1 < a < n$ et calcule $A := g^a$ et envoie cette quantité à Bob sur un canal public.
- Parallèlement Bob choisit un b aléatoire $1 < b < n$ et calcule $B := g^b$ et envoie cette quantité à Alice sur ce canal public.
- Alice calcule $B^a = g^{ab}$. Bob de son côté calcule $A^b = g^{ab}$. Cette quantité $C = g^{ab}$ sera leur secret commun (où un haché de C).

Pour retrouver cette quantité, Oscar qui écoute les échanges entre Alice et Bob doit résoudre le problème suivant : étant donné $A, B \in G$ calculer $C \in G$ tel que $C = g^{ab}$ où a et b sont tels que $A = g^a$ et $B = g^b$. Ce problème est appelé **problème calculatoire de Diffie-Hellman**, et (A, B, C) est appelé un triplet Diffie-Hellman.

Si on sait calculer des logarithmes discrets dans G alors on peut résoudre ce problème (en calculant a ou b). Cependant on ne sait pas s'il est possible de résoudre ce problème sans savoir calculer de logarithme discret.

Un but moins fort pour Oscar serait d'obtenir une information sur C à partir de A et B. Ceci est équivalent à résoudre le problème décisionnel suivant : étant donné $A, B, C \in G$, décider si (A, B, C) est un triplet Diffie-Hellman ou non. On parle de **problème décisionnel de Diffie-Hellman**. La seule manière connue de résoudre ce problème est de résoudre le problème calculatoire associé en calculant un logarithme discret.

Le chiffrement d'Elgamal (85)

Principe

C'est un **chiffrement à clef publique** qui peut se déduire de l'échange de clef Diffie-Hellman. Dans un chiffrement à clef publique ou **chiffrement asymétrique**, Bob possède un couple **clef publique, clef privée**. Cette dernière est connue de lui seul, alors que sa clef publique est connue de tous. Pour envoyer un message m à Bob, Alice utilise la clef publique de Bob et un algorithme de chiffrement pour obtenir le chiffré c. Pour retrouver c, Bob lui appliquera un algorithme de déchiffrement en utilisant sa clef privée.

Pour recevoir des messages Bob choisit un groupe $(G, \times)$ cyclique d'ordre n et g un générateur. Il choisit ensuite un x aléatoire $1 < x < n$ et calcule $h = g^x$.

Le triplet $K^B_{pub} = (n, g, h)$ constitue la clef publique de Bob, et $K^B_{priv} = x$ est sa clef privée.

Pour envoyer un message $m \in G$ à Bob, Alice choisit r aléatoire $1 < r < n$, et calcule $\mathsf{Encrypt}_{K^B_{pub}}(m) := c := (c_1, c_2) = (g^r, mh^r) \in G \times G$.

Pour déchiffrer, Bob calcule $\mathsf{Decrypt}_{K^B_{priv}}(c_1, c_2) := c_2 c_1^{-x}$.

Ce système de chiffrement est correct car si $(c_1, c_2) = (g^r, mh^r)$, $c_1^x = g^{rx} = h^r$ comme dans l'échange de clef Diffie-Hellman. Donc $c_2 c_1^{-x} = mh^r (h^r)^{-1} = m$.

Sécurité

On se place dans le contexte minimal pour le chiffrement asymétrique : une attaque à clairs choisis, la clef publique étant connue, un adversaire peut obtenir les chiffrés des messages clairs de son choix.

Le bris total (retrouver la clef secrète à partir de la clef publique) est équivalent au problème du logarithme discret dans G.

On peut montrer que casser la notion de sens unique est équivalent à résoudre un problème calculatoire Diffie-Hellman dans G, en effet (h, c_1, h^r) est un triplet Diffie-Hellman.

De même ce système est sémantiquement sûr (étant donné un chiffré c il est difficile de retrouver une information sur le message clair m) si et seulement si le problème décisionnel de Diffie-Hellman est difficile dans G.

Remarquons que les algorithmes de chiffrement à clef publique sont beaucoup plus lents (exponentiations dans un groupe) que ceux à clef secrète (manipulations simples sur les bits) mais ne nécessitent pas d'échange de clef. Leur sécurité est reliée à un problème algorithmique réputé difficile (ici le problème du logarithme discret) et non à la recherche exhaustive de la clef secrète comme en symétrique.

3. Algorithmes de calcul du logarithme discret

Nous alors d'abord voir quelques algorithmes génériques qui fonctionnent pour tous les groupes cycliques. Par un théorème de Shoup (1997), un algorithme qui résout le logarithme discret dans G d'ordre n doit faire au moins $\mathcal{O}(\sqrt{n})$ opérations dans G.

L'algorithme naïf

$h = g^x$, x ?

Calculer $g, g^2, g^3, \ldots$. Complexité, $\mathcal{O}(n)$ multiplications dans G

En utilisant de la mémoire : On pré-calcule tous les (g^i, i) et on les stocke dans une liste triée par rapport au g^i (en utilisant la représentation binaire des éléments de G par exemple). Complexité : $\mathcal{O}(n)$ multiplications dans G, $\mathcal{O}(n)$ éléments de G en mémoire. L'algorithme de tri à pour complexité $\mathcal{O}(n \log(n))$ comparaisons. On calcule ensuite x tel que $h = g^x$. Pour cela, on cherche h dans la liste, complexité : $\mathcal{O}(\log(n))$ comparaisons.

Baby Step/Giant Step

Shanks 1971. C'est un compromis temps mémoire.

Soit $m = \lceil \sqrt{n} \rceil$ on décompose en base m : $x = i + mj$ avec $0 \leqslant i, j < m$. On a donc $h = g^x = (g^m)^j g^i$, et

$$h(g^{-1})^i = (g^m)^j.$$

Pré-calculs : une liste des $((g^m)^j, j)$ avec $j < m$ triée par rapport à la première coordonnée : $\mathcal{O}(\sqrt{n})$ éléments en mémoire, et $\mathcal{O}(\sqrt{n}\log(n))$ pour l'algorithme de tri.

Dans la phase active, on calcule $h, hg^{-1}, h(g^{-1})^2, \ldots$ en cherchant chaque élément dans la liste. Si on le trouve on a $h(g^{-1})^i = (g^m)^j$ et on obtient donc $x = i + mj$. Dans le cas le pire : $\mathcal{O}(\sqrt{n})$ multiplications et $\mathcal{O}(\sqrt{n}\log(n))$ comparaisons.

Pollard ρ

Pollard 1978. Voir TD, similaire à la recherche de collision de fonction de hachage. On obtient un algorithme avec une complexité calculatoire heuristique similaire en utilisant quasiment pas de mémoire. Contrairement à Baby Step/Giant Step, c'est un algorithme probabiliste. Parfois aucun résultat n'est trouvé (on relance alors l'algorithme avec d'autres choix aléatoires). C'est le meilleur algorithme qui fonctionne dans n'importe quel groupe, en particulier c'est le meilleur algorithme pour les courbes elliptiques.

Pohlig–Hellman

La méthode de Pohlig–Hellman réduit le problème du calcul de logarithme discret dans un groupe d'ordre n à celui du calcul dans des sous groupes d'ordre p où p est premier et p divise n.

Supposons connue la factorisation de $n = p_1^{e_1} \dots p_r^{e_r}$. On calcule d'abord les valeurs du logarithme discret modulo chaque $p_i^{e_i}$ et on en déduit la valeur modulo n par le théorème des restes chinois.

Pour un premier $p = p_i$ donné, comment calculer $x \mod p^e$? On note $x \mod p^e = a_0 + a_1 p + \dots + a_{e-1} p^{e-1}$, où $0 \leqslant a_i \leqslant p-1$ la décomposition en base p. Alors de $h = g^x$, on a $h^{n/p} = (g^{n/p})^{a_0}$ et $g^{n/p}$ est d'ordre p (g est un générateur). On obtient ainsi $a_0 \mod p = a_0$ par un algorithme qui calcule des logarithmes discrets modulo p en base $g^{n/p}$. On continue ensuite en remarquant que $h^{n/p^2} = (g^{n/p^2})^{a_0 + a_1 p}$, donc $(h/g^{a_0})^{n/p^2} = (g^{n/p^2})^{a_0 + a_1 p - a_0} = (g^{n/p^2})^{a_1 p} = (g^{n/p})^{a_1}$. On peut trouver ainsi a_1 et on itérer la méthode pour trouver $x \mod p^e$ avec e calculs de logarithme discret $\mod p$.

En pratique, pour les applications cryptographiques, on prend toujours (presque) n premier, sinon si on peut le factoriser, alors on peut réduire à un calcul plus petit de logarithmes discrets modulo $p|n$.

Algorithmes de type calcul d'indice

Les algorithmes génériques précédents ont une complexité exponentielle. Si le groupe G est particulier, on peut parfois faire mieux. Supposons qu'il existe un ensemble $S = \{p_1, p_2, \dots, p_t\} \subset G$, appelé base de facteurs tels qu'une grande proportion d'éléments de G peuvent s'écrire de manière efficace comme un produit de p_is.

Phase de pré-calculs : on détermine les $\log_g(p_i)$: pour cela on prend des k dans $\mathbf{Z}$ et par la propriété supposée sur G on a de bonnes chances de pouvoir écrire $g^k = \prod p_i^{e_i}$ (facilement parallélisable)

En appliquant la fonction log on obtient : $k = \sum e_i \log_g(p_i)$. Avec au moins t équations linéaires indépendantes, on peut résoudre et trouver les $\log_g(p_i)$.

Ensuite dans une phase active, si $h = g^x$, on calcule hg^k pour des k aléatoires. Si on factorise, en appliquant le log, on obtient $x + k = \sum e_i \log_g(p_i)$.

Il faut faire un compromis sur t : petit on a besoin de moins d'équations. Grand, il y a une meilleure probabilité qu'un élément aléatoire de G se factorise dans S.

Rappel : complexité sous exponentielle : $L_n(\alpha, c) = \mathcal{O}(exp(c(\log n)^\alpha (\log\log n)^{1-\alpha}))$.

$L(0, c) = (\log n)^c$: complexité polynomiale (en la taille de n) et $L(1, c) = n^c$: complexité exponentielle.

Dans $\mathbf{Z}/p\mathbf{Z}$ avec $S = \{\text{premiers} < B\}$ on obtient un algorithme en $L_p(1/2, \sqrt{2})$ (Kraitchik, 1922, redécouvert à la fin des années 1970). Le crible sur corps de nombre (***number field sieve***, NFS, 1993) est une amélioration de cette idée en cherchant des éléments dits lisses dans des corps de nombres (des extensions de $\mathbf{Q}$). Pour le calcul de logarithmes discrets dans un corps finis $\mathbf{F}_q$, cet algorithme a une complexité de $L_q(1/3, c)$ pous un certain $c > 0$. Il y a également des algorithmes plus rapides dans le cas de corps fini $\mathbf{F}_{p^k}$, suivant la taille et la forme de p et k.

Pour $k = 1$, c'est à dire dans $\mathbf{Z}/p\mathbf{Z}$, le meilleur algorithme est NFS. Le record est pour un p de 795 bits (2019, calculé en plus d'une année, avec un coût approximatif de 3100 ans sur un seul cœur).

Mise en place cryptographique dans $\mathbf{Z}/p\mathbf{Z}$

En pratique on utilise généralement pour implanter les algorithmes cryptographiques fondés sur le logarithme discret des groupes cycliques G d'ordre premier q. Comme les algorithmes génériques ont une complexité $\mathcal{O}(\sqrt{q})$ opérations, pour avoir une sécurité de k bits (c'est à dire que les meilleures attaques connues fonctionnent en 2^k opérations), il faut prendre q de $2k$ bits (par exemple 256 bits pour une sécurité classique de 128 bits).

Un choix est d'utiliser pour G un sous groupe des inversibles de $\mathbf{Z}/p\mathbf{Z}$ avec p premier et q divisant $p-1$. Les algorithmes de type calcul d'indice s'appliquant dans $\mathbf{Z}/p\mathbf{Z}$, on doit prendre p de telle manière que l'algorithme NFS, prenne 2^k opérations. Le tableau suivant donne les tables estimées pour obtenir cela.

Niveau de sécurité	taille de q (bits)	taille de p (bits)
112	224	2048
128	**256**	**3072**
192	384	7680
256	512	15360

Pour l'échange de clefs Diffie-Hellman et le chiffrement Elgamal prendre q relativement petit comparé à p augmente les performances : quasiment toutes les opérations sont des exponentiations modulaires dont la complexité dépend essentiellement de la taille des exposants.

Pour générer de tels groupes, on choisit un nombre premier q aléatoire suivant les tailles ci-dessus (en utilisant des tests de pseudo-primalité comme Rabin-Miller), puis on tire des nombres A aléatoires, jusqu'à trouver $p := q\mathrm{A} + 1$ premier (par le théorème des nombres premiers pour les progressions arithmétiques, on s'attend à faire de l'ordre de $\log(p)$ essais). On est ainsi assuré que $\left(\mathbf{Z}/p\mathbf{Z}\right)^\times$ contient un sous groupe cyclique d'ordre q premier. Pour obtenir un générateur, on prend un entier x au hasard et on calcule x^{A} modulo p. Si l'on ne trouve pas 1, on est assuré par le petit théorème de Fermat que $g := x^{\mathrm{A}}$ est d'ordre q. Comme il y a A éléments d'ordre divisant A, on a une probabilité négligeable $1/q$ que $x^{\mathrm{A}} = 1$. On peut aussi utiliser des groupes standardisés.

Un choix plus efficace de nos jours est d'utiliser des groupes issus des courbes elliptiques que l'on va voir dans la section suivante.

4. Introduction aux courbes elliptiques

Définition

Les courbes elliptiques sont des objets aux propriétés très riches ayant de nombreuses applications mathématiques et en particulier en cryptographie. On va ici les définir dans un cas très particulier qui correspond aux applications que l'on va voir.

Soit $p > 3$ un nombre premier et a et b deux éléments de $\mathbf{F}_p$. On considère l'ensemble des points (X, Y) à coordonnées dans $\mathbf{F}_p$ satisfaisant l'équation

$$\mathrm{E} : \mathrm{Y}^2 = \mathrm{X}^3 + a\mathrm{X} + b,$$

avec la condition de non singularité : $4a^3 + 27b^2 \neq 0$ dans $\mathbf{F}_p$.

On rajoute à cet ensemble de points un point particulier dit point à l'infini.

Loi de groupe

On obtient ainsi un groupe fini noté $(\mathrm{E}(\mathbf{F}_p), +)$ dont le neutre O_E est le point à l'infini. La loi de groupe a une interprétation géométrique.

L'opposé d'un point $(x, y) \in \mathrm{E}(\mathbf{F}_p)$ est le point $(x, -y)$. L'addition de deux points distincts et non opposés, $\mathrm{P}_1 = (x_1, y_1)$ et $\mathrm{P}_2 = (x_2, y_2)$, est l'opposé du troisième point d'intersection de la droite $(\mathrm{P}_1, \mathrm{P}_2)$ avec E. C'est le point de coordonnée (x_3, y_3) avec

$$x_3 = \lambda^2 - x_1 - x_2 \quad \text{et} \quad y_3 = \lambda(x_1 - x_3) - y_1,$$

où $\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$. Le double d'un point $\mathrm{P} = (x_1, y_1)$ avec $\mathrm{P} \neq -\mathrm{P}$ (c'est à dire $y_1 \neq 0$) est le point de coordonnées (x_3, y_3) avec

$$x_3 = \lambda^2 - 2x_1 \quad \text{et} \quad y_3 = \lambda(x_1 - x_3) - y_1,$$

avec $\lambda = \frac{3x_1^2+a}{2y_1}$. C'est l'autre point d'intersection de la tangente en P avec E.

Il existe d'autres manières de représenter les points de $\mathrm{E}(\mathbf{F}_p)$ et d'autres formules plus efficaces pour la loi de groupe.

Structure

L'ordre de $\mathrm{E}(\mathbf{F}_p)$ est proche de p : le théorème de Hasse donne l'encadrement

$$p + 1 - 2\sqrt{p} \leqslant \#\mathrm{E}(\mathbf{F}_p) \leqslant p + 1 + 2\sqrt{p}.$$

De plus, $\mathrm{E}(\mathbf{F}_p)$ peut entre engendrée par au plus deux générateurs et en général $\mathrm{E}(\mathbf{F}_p)$ est cyclique ou proche d'être cyclique. Il est possible de calculer l'ordre et la structure de $\mathrm{E}(\mathbf{F}_p)$ en temps polynomial.

Cependant en général, pour les applications cryptographiques, on utilise des courbes standardisées comme la courbe « P − 256 » définie sur $\mathbf{F}_p$ avec p de 256 bits qui est cyclique d'ordre premier q (aussi de 256 bits) ou la courbe « Curve25519 » défini sur $\mathbf{F}_p$ avec $p = 2^{255} - 19$ qui est également cyclique avec un sous-groupe d'ordre premier q de 253 bits. On travaille ensuite dans le groupe cyclique d'ordre premier q.

Compression de points

Les éléments de $\mathrm{E}(\mathbf{F}_p)$ peuvent être représentés de manière compacte, ce qui donnera des chiffrés et des signatures plus courts. En effet, un point $\mathrm{P} = (x_\mathrm{P}, y_p)$ satisfaisant l'équation de E, peut se « compresser ». Étant donné x_p, on peut retrouver $\mathrm{S} := y_p^2 = x_p^3 + ax_p + b$. Comme p est premier, S a deux racines carrés opposées $\pm a$. Si on représente a comme un entier entre 1 et $p - 1$, $p - a$ représentera l'autre racine carré dans cet intervalle. Ainsi les deux représentations ont des parités différentes car p est impair.

Par conséquent, on représente P par (x_P, b) où b est un bit représentant la parité de $y_\mathrm{P} \mod p$, en utilisant donc taille$(p) + 1$ bits.

Applications cryptographiques

Comme vu précédemment, le problème du logarithme est plus difficile dans les courbes elliptiques que dans les corps finis. La meilleure attaque est Pollard ρ avec une complexité exponentielle contre une complexité sous-exponentielle pour NFS dans $\mathbf{F}_p$. Le record de calcul actuel a été réalisé en 2020 en cherchant un logarithme discret dans un intervalle de taille 2^{114} sur une courbe de 256 bits en 13 jours sur 256 GPU avec une version parallèle de Pollard ρ[1].

Pour avoir k bits de sécurité, il suffira de prendre une courbe définie sur $\mathbf{F}_p$ avec p de $2k$ bits : cela permettra de définir un groupe cyclique d'ordre premier qui aura aussi $2k$ bits (ou pas loin) comme dans les exemples ci-dessus. Cela donne des implantations de systèmes cryptographiques plus rapides (un facteur 5 à 10 sur le temps de calcul des exponentiations comparé aux sous groupes de $\mathbf{Z}/p\mathbf{Z}$ pour la même sécurité). En effet, même si la loi de groupe des courbes elliptiques est plus complexe que la multiplication dans les corps finis, elle est évaluée sur des entiers beaucoup plus petits.

De nos jours les courbes elliptiques ont donc massivement remplacé les corps finis pour les implantations de la cryptographie fondée sur le logarithme discret. Ainsi l'échange de clefs Diffie-Hellman se fait en utilisant le standard ECDH (*Elliptic Curve Diffie Hellman*).

Couplages

Les courbes elliptiques ont un autre intérêt pour la cryptographie. Il est possible d'y définir un application bilinéaire, appelé couplage (*pairing*) ayant de nombreuses applications cryptographiques.

On considère toujours une courbe E définie sur le corps $\mathbf{F}_p$ tel que $\mathrm{E}(\mathbf{F}_p)$ contienne un grand sous groupe cyclique d'ordre premier q. Ce couplage prend deux points en entrée et retourne un élément d'une

[1] cf. `https://en.wikipedia.org/wiki/Discrete_logarithm_records`

extension $\mathbf{F}_{p^k}$ de $\mathbf{F}_p$. L'entier k est appelé degré de plongement, c'est le plus petit entier tel que $(\mathbf{F}_{p^k})^\times$ contiennent les racines q–ièmes de l'unité, c'est à dire tel que q divise $p^k - 1$. Autrement dit, c'est l'ordre de p modulo q.

Pour une courbe elliptique quelconque, ainsi que pour les exemples mentionnés au dessus, k va être de l'ordre de q. Un élément de $\mathbf{F}_{p^k}$ ne pourra être stocké sur machine et, *a fortiori*, le couplage ne peut être calculé. Cependant il est possible de construire des courbes elliptiques où k est petit (< 20). On parle de *pairing friendly elliptic curve*. On peut alors utiliser de tels couplages. Ils sont définis en général à partir de deux objets mathématiques, le couplage de Weil ou le couplage de Tate et prennent en entrée un point d'ordre q de $\mathrm{E}(\mathbf{F}_p)$ et un point d'ordre q de $\mathrm{E}(\mathbf{F}_{p^k})$ et retourne un élément d'ordre q de $\mathbf{F}_{p^k}$. Si k est petit, ces couplages peuvent être calculés en temps polynomial pour un algorithme du à Miller.

Dans la suite nous allons utiliser de tels couplages en « boîte noire ». Ainsi, nous considérerons la notion de couplage cryptographique.

Soient $(\mathrm{G}_1, +)$, $(\mathrm{G}_2, +)$, $(\mathrm{G}_t, \times)$ trois groupes cycliques d'ordre premier q. On note P et Q des générateurs de G_1 et G_2. Un couplage cryptographique $e : \mathrm{G}_1 \times \mathrm{G}_2 \to \mathrm{G}_t$ est :

- une application bilinéaire : $e(a\mathrm{P}, b\mathrm{Q}) = e(\mathrm{P}, \mathrm{Q})^{ab}$ pour tout $a, b \in \mathbf{Z}/q\mathbf{Z}$;
- non dégénérée : $e(\mathrm{P}, \mathrm{Q}) \neq 1$;
- calculable efficacement.

Une telle application permet de transporter le problème du logarithme discret de G_1 dans G_t (attaque MOV) : Si $\mathrm{H} = x\mathrm{P}$, $e(\mathrm{H}, \mathrm{Q}) = e(\mathrm{P}, \mathrm{Q})^x$. Ainsi $\log_{e(\mathrm{P},\mathrm{Q})} e(\mathrm{H}, \mathrm{Q})$ donne x. Or pour les applications cryptographiques, on veut que le problème du logarithme discret soit dur dans les trois groupes. Cela impose une analyse fine des paramètres quand on construit une courbe elliptique *pairing friendly* : comme $\mathrm{G}_1 \subset \mathrm{E}(\mathbf{F}_p)$ et $\mathrm{G}_t \subset \mathbf{F}_{p^k}$ où le problème du logarithme discret est plus facile. Il existe de nombreuses constructions de telles courbes pairing friendly. Un choix populaire est la courbe BLS12 – 381, définie sur $\mathbf{F}_p$ avec p de 381 bits, définissant des groupes d'ordre premier q de 255 bits, avec un degré de plongement 12, c'est à dire $\mathrm{G}_t \subset \mathrm{F}^*_{p^{12}}$.

Cryptographie fondée sur les couplages

Les couplages ont fait leur entrée en cryptographie avec l'attaque MOV. Mais bien vite, au début des années 2000, un nouveau domaine s'est développé en utilisant les couplages pour proposer des applications cryptographiques qui n'étaient pas réalisable auparavant.

Dans la suite on va utiliser, pour simplifier l'exposition, un couplage cryptographique où $\mathrm{G}_1 = \mathrm{G}_2$ (dit symétrique). En pratique on utilise plutôt un couplage où $\mathrm{G}_1 \neq \mathrm{G}_2$ dont les instantiations permettent d'avoir de meilleures performances.

Diffie-Hellman Tripartite en un tour

La première application, proposée par Joux en 2000, permet de faire un échange de clef entre Alice, Bob et Charlie en un seul tour de communication. Pour cela, les trois participants se mettent tout d'abord publiquement d'accord sur un couplage cryptographique symétrique $e : \mathrm{G} \times \mathrm{G} \to \mathrm{G}_t$, avec P un générateur de G d'ordre q.

- Alice choisit un $1 < a < n$ aléatoire , et calcule $a\mathrm{P}$ qu'elle envoie à Bob et Carl.
- Bob choisit un $1 < b < n$ aléatoire, et calcule $b\mathrm{P}$ qu'il envoie à Alice et Carl.
- Carl choisit un $1 < c < n$ aléatoire, et calcule $c\mathrm{P}$ qu'il envoie à Alice et Bob.

Alice calcule $e(b\mathrm{P}, c\mathrm{P})^a = e(\mathrm{P}, \mathrm{P})^{abc}$. Bob calcule $e(a\mathrm{P}, c\mathrm{P})^b = e(\mathrm{P}, \mathrm{P})^{abc}$; Carl calcule $e(a\mathrm{P}, b\mathrm{P})^c = e(\mathrm{P}, \mathrm{P})^{abc}$.

La sécurité est fondée sur l'hypothèse suivante : étant donnés $a\mathrm{P}, b\mathrm{P}, b\mathrm{Q}, c\mathrm{Q}$ il est difficile de calculer $e(\mathrm{P}, \mathrm{P})^{abc}$ appelé problème bilinéaire Diffie-Hellman (BDH). On peut également définir une variante décisionnelle (DBDH).

Chiffrement fondé sur l'identité

Le chiffrement à clef publique traditionnel est vulnérable aux attaques dites de l'homme du milieu. Lorsqu'Alice envoie un message à Bob, elle doit récupérer la clé publique de Bob. Un adversaire actif peut intercepter cette connexion et envoyer à Alice sa propre clef publique dont il connaît la clef secrète correspondante. Cet attaquant peut ainsi déchiffrer chaque message qu'Alice souhaite envoyer à Bob. Pour corriger ce problème, Alice doit s'assurer que la clé publique reçue appartient bien à Bob.

Dans une infrastructure à clé publique (PKI), une autorité de certification (CA) fournit cette confiance dans les clefs publiques. Bob s'adresse à l'autorité qui signe sa clef publique en délivrant un certificat. Grâce à ce certificat, Alice peut vérifier si la clef publique est légitime, et la confiance dans la clef publique repose sur la confiance dans la CA.

En 1984, Shamir a introduit un concept connu sous le nom de chiffrement fondé sur l'identité (*Identity-Based Encryption*, IBE). Ici toute information publique d'un utilisateur (par exemple une adresse email) peut être utilisée comme clef publique. Ceci conduit à une infrastructure moins complexe, il n'y a plus lieu de certifier les clefs publiques.

Plus précisément, un **schéma de chiffrement fondé sur l'identité** (IBE) est composé deux quatre algorithmes probabilistes :

- Setup : prend en entrée un paramètre de sécurité et retourne des paramètres publics qui seront une entrée commune aux autres algorithmes, ainsi qu'une clef publique maître mpk et une clef secrète maître msk ;
- Dérivation de clef : prend en entrée msk et une identité $\mathsf{id} \in \{0,1\}^*$ et retourne une clef privée sk_{id} ;
- Chiffrement : prend en entrée id et un message clair m et retourne un chiffré c_{id} ;
- Déchiffrement : prend en entrée une clef privée sk_{id} et un chiffré c_{id} et retourne un message clair.

Ce schéma doit être correct, c'est à dire pour tout paramètre de sécurité, pour toute clef secrète sk_{id} dérivée pour une certaine identité id, et pour tout message clair m, le déchiffrement avec la clef sk_{id} de c_{id} généré par l'algorithme de chiffrement sur l'entrée (id, m) retourne bien m.

En pratique, la dérivation de clef est faite par une autorité qui connaît la clef secrète maître msk. Alice récupère mpk et les paramètres du système et peut envoyer à n'importe quel autre utilisateur, Bob, un message chiffré avec son identité (par exemple bob@gmail.com). Bob pour pouvoir déchiffrer des messages envoyés pour cette identité, devra demander la clef secrète correspondante à l'autorité. Notons que l'autorité a beaucoup de pouvoir, elle peut déchiffrer tout message chiffré !

Une première réalisation efficace de ce concept a été proposé par Boneh et Franklin en 2001, en utilisant un couplage cryptographique. On utilise comme pour l'échange de clef un couplage symétrique $e : \mathrm{G}\times\mathrm{G} \to \mathrm{G}_t$, avec P un générateur de G d'ordre q.

- La clef secrète maître msk est un entier pris au hasard entre 1 et q. On pose $mpk = msk\,\mathrm{P} \in \mathrm{G}$. On utilisera aussi une fonction de hachage cryptographique , $\mathrm{H} : \{0,1\}^* \to \mathrm{G}$;
- Dérivation de clef : $\mathrm{Q}_{\mathsf{id}} = \mathrm{H}(\mathsf{id})$ et $sk_{\mathsf{id}} = msk\,\mathrm{Q}_{\mathsf{id}}$;
- Chiffrement d'un message $m \in \mathrm{G}_t$: $\mathrm{Q}_{\mathsf{id}} = \mathrm{H}(\mathsf{id})$, on prend r un entier aléatoire entre 1 et q, le chiffré est $c = (r\mathrm{P}, m\,e(mpk, \mathrm{Q}_{\mathsf{id}})^r)$;
- Déchiffrement de (c_1, c_2) : $c_2\,e(c_1, sk_{\mathsf{id}})^{-1}$.

Le déchiffrement est bien correct car

$$e(c_1, sk_{\mathsf{id}}) = e(r\mathrm{P}, msk\,\mathrm{Q}_{\mathsf{id}}) = e(\mathrm{P}, \mathrm{Q}_{\mathsf{id}})^{r\,msk} = e(msk\,\mathrm{P}, \mathrm{Q}_{\mathsf{id}})^r = e(mpk, \mathrm{Q}_{\mathsf{id}})^r.$$

On peut montrer que la sécurité repose sur les mêmes hypothèses que l'échange de clef tripartite. Si on pose x l'entier inconnu tel que $\mathrm{Q}_{\mathsf{id}} = x\,\mathrm{P}$, le masque pour le chiffrement est $e(\mathrm{P}, \mathrm{P})^{r\,msk\,x}$, étant connu $c_1 = r\,\mathrm{P}$, et $mpk = msk\,\mathrm{P}$ et $\mathrm{Q}_{\mathsf{id}} = x\,\mathrm{P}$. C'est donc un problème BDH.

Les couplages cryptographiques ont de nombreuses autres applications en chiffrement avancé. Le chiffrement fondé sur l'identité est un cas particulier de chiffrement fondé sur les attributs où le chiffré dépend d'un certain nombre d'attributs (par exemple le pays de résidence, l'âge...). Le déchiffrement est possible si l'ensemble des attributs de l'utilisateur correspond à ceux du chiffré.

Un domaine proche est le chiffrement fonctionnel où les clefs secrètes permettent de récupérer une fonction du message chiffré et non pas le chiffré tout entier comme dans le chiffrement asymétrique classique.

Chapitre VI

Cryptographie fondée sur la factorisation

Dans le chapitre précédent, on a utilisé le problème du logarithme discret pour construire une fonction à sens unique, $x \mapsto g^x$. Le problème de la factorisation des entiers va nous permettre de construire des groupes dont l'ordre est difficile à calculer. Ceci va nous permettre de construire d'autres fonctions à sens unique. De plus, la connaissance de la factorisation permettra de calculer l'ordre des groupes, ce qui donnera une trappe pour inverser ces fonctions.

Soient p et q deux grands nombres premiers distincts, on pose $\mathrm{N} = pq$. Dans la suite on appellera un tel entier un **entier RSA** ou un module RSA du nom de Rivest, Shamir, Adleman, qui ont proposé le système du même nom en 1977.

1. Rappels sur $\mathbf{Z}/\mathrm{N}\mathbf{Z}$

Structure

On note $(\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times$ l'ensemble des éléments inversibles modulo N. C'est un groupe pour la multiplication. Si $\mathrm{N} = pq$ est un entier RSA, l'ordre du groupe est $\varphi(\mathrm{N}) = (p-1)(q-1)$. De plus, par le théorème des restes chinois $(\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times$ est isomorphe à $(\mathbf{Z}/p\mathbf{Z})^\times \times (\mathbf{Z}/q\mathbf{Z})^\times$:

$$\begin{array}{ccc} (\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times & \xrightarrow{\sim} & (\mathbf{Z}/p\mathbf{Z})^\times \times (\mathbf{Z}/q\mathbf{Z})^\times \\ a & \longmapsto & (a \pmod p, a \pmod q) \end{array}$$

Ce groupe n'est donc pas cyclique, c'est le produit de deux groupes cycliques, d'ordre $p-1$ et $q-1$. L'ordre maximal d'un élément est $\mathrm{ppcm}(p-1, q-1) = (p-1)(q-1)/\operatorname{pgcd}(p-1, q-1)$ et ce pgcd est au moins égal à 2. Il n'y a donc pas d'élément d'ordre $\varphi(\mathrm{N})$.

Carrés modulo p

Soit p un nombre premier **impair**. Dans $(\mathbf{Z}/p\mathbf{Z})^\times$ il y a $(p-1)/2$ carrés (ou résidus quadratiques) et $(p-1)/2$ non carrés. En effet, le morphisme $a \mapsto a^2$ a pour noyau ± 1 car $\mathbf{Z}/p\mathbf{Z}$ est un corps. Son image correspond aux carrés et a pour cardinal $(p-1)/2$.

Le **symbole de Legendre** permet de déterminer si un élément est un carré ou pas. Par définition, si a est un entier, on a $\left(\frac{a}{p}\right) = 0$ si $p \mid a$, $\left(\frac{a}{p}\right) = -1$ si a n'est pas un carré modulo p et $\left(\frac{a}{p}\right) = 1$ si a est un carré modulo p. On a en fait

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod p.$$

En effet, si $p \mid a$, c'est immédiat. Sinon comme $a^{p-1} = 1$, on a $a^{(p-1)/2} = \pm 1$ comme vu précédemment. Si $a = b^2$ est un carré, on a $a^{(p-1)/2} = b^{p-1} = 1$. Les carrés sont donc les racines du polynôme $\mathrm{X}^{(p-1)/2} - 1$

qui a plus $(p-1)/2$ racines. On a donc égalité entre l'ensemble des carrés et l'ensemble des racines de ce polynôme. On a donc bien $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} = 1$ dans le cas où a est un carré. Si a n'est pas un carré, on a donc forcement, $a^{\frac{p-1}{2}} = -1 = \left(\frac{a}{p}\right)$.

Les conséquences sont que

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

et

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

c'est à dire que -1 est un carré modulo p si $p \equiv 1 \pmod 4$ et n'est pas un carré si $p \equiv 3 \pmod 4$.

On peut également montrer que

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

c'est à dire que 2 est un carré modulo p si et seulement si p est congru à ± 1 modulo 8.

Loi de réciprocité quadratique : si p et q sont premiers impairs distincts, alors

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

Autrement dit, $\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right)$ sauf si $p \equiv q \equiv -1 \pmod 4$, auquel cas $\left(\frac{p}{q}\right) = -\left(\frac{q}{p}\right)$.

Si a est un carré modulo p, on peut calculer en temps polynomial une des ses deux racines carrées. Si $p \equiv 3 \pmod 4$, c'est facile : $a^{\frac{p+1}{4}}$. En effet, $a^{\frac{p+1}{2}} = a^{\frac{p-1}{2}} a = \left(\frac{a}{p}\right) a = a$. Si $p \equiv 1 \pmod 4$ on peut toujours calculer des racines carrés (algorithme de Tonelli-Shanks). Plus généralement on peut trouver les racines éventuelles de n'importe quel polynôme dans $\mathbf{Z}/p\mathbf{Z}$ en temps polynomial (algorithme de Berlekamp).

Carrés modulo N

Si n est un entier **impair** on définit le **symbole de Jacobi**. La factorisation de n étant $n = \prod_{i=1}^{r} p_i$ (premiers non nécessairement distincts), on pose pour tout entier a :

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

On a alors pour tout m et n impairs et entier a et b :

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right) \quad \text{et} \quad \left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right).$$

D'autre part, $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$ et $\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}$. Enfin la loi de réciprocité quadratique s'étend : si m et n sont impairs premiers entre eux

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

Ceci donne un algorithme efficace pour calculer le symbole de Jacobi (essentiellement l'algorithme d'Euclide), même sans connaître la factorisation de n.

Si a est un carré modulo n, $\left(\frac{a}{n}\right) = 1$. Cependant la réciproque est généralement fausse. Plaçons nous dans le cas $N = pq$, un entier RSA et considérons les éléments inversibles. En utilisant le théorème des restes chinois, on voit qu'il y a $\varphi(N)/4$ carrés modulo N : les carrés modulo N sont les éléments qui sont

des carrés modulo p et modulo q. Il y a aussi $\varphi(N)/4$ éléments qui sont ni des carrés modulo p, ni des carrés modulo q. Ainsi, il y a $\varphi(N)/2$ éléments de symbole de Jacobi 1. Distinguer parmi eux ce qui sont des carrés est un problème difficile (le **problème de la résidualité quadratique**) sans connaître la factorisation de N.

De même, il est difficile de calculer une des 4 racines carrés d'un carré modulo $N = pq$. Si on sait calculer des racines carrés efficacement, alors on peut factoriser N ! En effet, prenons x un entier inversible modulo N, qui correspond à (x_p, x_q) dans $\mathbf{Z}/p\mathbf{Z} \times \mathbf{Z}/q\mathbf{Z}$ via les restes chinois. Calculons $y = x^2$ dans $\mathbf{Z}/N\mathbf{Z}$. Alors les 4 racines carrées de y sont $x = (x_p, x_q)$, $-x = (-x_p, -x_q)$, $z := (-x_p, x_q)$ et $-z := (x_p, -x_q)$. Si un oracle nous donne une racine carrée de y, alors il y a une chance sur 2 que l'on obtienne z ou $-z$. On a alors $\text{pgcd}(x - z, N) = q$ et $\text{pgcd}(x + z, N) = p$. C'est l'idée de l'algorithme de Fermat pour factoriser, que l'on retrouve dans les algorithmes sous exponentiel de factorisation.

2. Cryptographie avec les carrés de $\mathbf{Z}/N\mathbf{Z}$

Fonction de Rabin (1978)

On prend $N = pq$ un entier RSA avec $p \equiv q \equiv 3 \pmod 4$. On considère alors la fonction

$$S : (\mathbf{Z}/N\mathbf{Z})^\times \to (\mathbf{Z}/N\mathbf{Z})^\times, m \mapsto m^2.$$

Cette fonction est à sens-unique si factoriser N est difficile. En effet, inverser la fonction revient à calculer des racines carrés, ce qui est équivalent à factoriser comme on vient de le voir.

La factorisation de N donne une trappe pour inverser cette fonction (on calcule les racines carrées modulo p et q et on recombine avec les restes chinois). Par contre, si on utilise cette fonction pour créer un algorithme de chiffrement, il y aura une ambiguïté dans le déchiffrement : il faut distinguer le bon message clair parmi les 4 racines carrées. On verra en TD un moyen de lever cette ambiguïté. Ce système est donc à sens-unique pour des attaques à clairs choisis sous l'hypothèse que factoriser est difficile. Par contre on n'a pas de sécurité sémantique car le système est déterministe.

Le **générateur de Blum-Blum-Shub (1986)** est un générateur pseudo-aléatoire de suite binaire construit en itérant la fonction de Rabin et en prenant le bit de parité. Ceci est extrêmement lent comparé aux algorithmes de chiffrement par flot que l'on a vu précédemment mais on peut prouver ici que l'on ne peut pas distinguer la sortie du générateur d'une suite aléatoire si factoriser N est difficile.

Chiffrement de Goldwasser Micali (1982)

On prend $N = pq$ un entier RSA avec $p \equiv q \equiv 3 \pmod 4$. La clef publique est N, la clef privée est (p, q).

Pour chiffrer un message $m \in \{0, 1\}$, on choisit r aléatoire entre 1 et N et on calcule $c = (-1)^m r^2$ dans $(\mathbf{Z}/N\mathbf{Z})^\times$. Remarquons que r peut être considéré comme inversible sinon, on aurait factorisé N !

On a un chiffrement probabiliste comme Elgamal. Le bit 0 est chiffré par un carré aléatoire de $(\mathbf{Z}/N\mathbf{Z})^\times$ tandis que le bit 1 va donner un élément aléatoire qui est ni un carré modulo p ni un carré modulo q (car $\left(\frac{-1}{p}\right) = \left(\frac{-1}{q}\right) = -1$). L'espace des chiffrés est donc l'ensemble des éléments de $(\mathbf{Z}/N\mathbf{Z})^\times$ de symbole de Jacobi 1.

Ainsi distinguer les chiffrés de 0 de ceux de 1 est exactement le problème de la résidualité quadratique. Ce schéma est donc sémantiquement sûr pour des attaques à clairs choisis sous l'hypothèse que ce problème est difficile. Il est à sens-unique sous la même hypothèse. Le bris total repose sur la factorisation de N.

Le déchiffrement se fait par calcul de symbole de Legendre en utilisant la factorisation de N.

Ce schéma a des propriétés homomorphes. Si c_1 (resp. c_2) est un chiffré de m_1 (resp. m_2), alors $c_1 c_2$ est un chiffré de $m_1 + m_2$ modulo 2.

3. Le chiffrement RSA (1977)

Principe

Pour recevoir des messages Bob choisit deux grands nombres premiers p et q distincts et pose $N = pq$. Bob choisit e, un entier premier avec $\varphi(N)$ et pose d tel que $ed \equiv 1 \pmod{\varphi(N)}$.

Le couple (N, e) constitue la clef publique de Bob, et d est sa clef privée.

Pour envoyer un message $m \in (\mathbf{Z}/N\mathbf{Z})^\times$ à Bob, Alice calcule $c := m^e \pmod N$.

Pour déchiffrer, Bob calcule $c^d \pmod N$.

Ce système de chiffrement est correct car si $c \equiv m^e \pmod N$, $c^d \equiv m^{ed} \equiv m^{1+k\varphi(N)} \equiv m \pmod N$ par le théorème d'Euler.

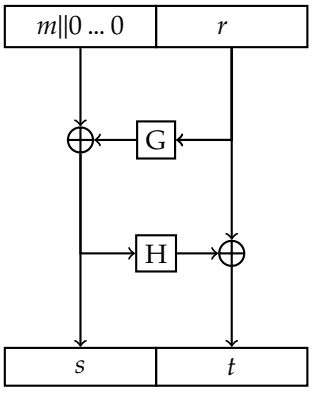
Sécurité

On se place toujours dans le cadre minimal d'attaque à clairs choisis.

Le bris total consiste à retrouver d à partir de (N, e). On peut montrer que cela est équivalent à factoriser N.

Pour la notion de sens-unique, retrouver m à partir de $c \equiv m^e \pmod N$ connaissant N et e est appelé le problème RSA. On ne connaît pas d'autres méthodes que de factoriser N (pour retrouver d) afin de résoudre le problème RSA.
La permutation $m \mapsto m^e \pmod N$ de $(\mathbf{Z}/N\mathbf{Z})^\times$ est donc à sens-unique sous l'hypothèse que le problème RSA est dur. La factorisation et donc la connaissance de d est une trappe pour inverser cette permutation.

Notez que RSA tel quel n'est pas sémantiquement sûr car il est déterministe. Pour résoudre ce problème on utilise des versions de RSA qui rajoutent de l'aléa au message. La version standardisée, RSA-OAEP, est prouvée sémantiquement sûre même pour des attaques à chiffrés choisis sous l'hypothèse que le problème RSA est dur (prouvé en 2001). Elle consiste à effectuer deux tours de schémas de Feistel au moyen de fonctions de hachages. Notons qu'il existe d'autres transformations permettant de construire un schéma de chiffrement sûr contre des attaques à chiffrés choisis à partir d'un schéma seulement sûr contre des attaques à clairs choisis.



Soit ℓ la taille en bits du module RSA N. Soit k_0, k_1 deux entiers tels que $k_1 + k_0 < \ell$. On considère deux fonctions de hachages $G : \{0,1\}^{k_0} \to \{0,1\}^{\ell - k_0}$ et $H : \{0,1\}^{\ell - k_0} \to \{0,1\}^{k_0}$. Pour chiffrer un message m, un chaîne binaire de longueur $\ell - k_1 - k_0$, on prend un aléa r de k_0 bits, et on pose $s = (m||0 \dots 0) \oplus G(r)$ (en concaténant k_1 zéros à m), et $t = r \oplus H(s)$. On considère ensuite $s||t$ comme un élément x de $(\mathbf{Z}/N\mathbf{Z})^\times$. On pose ensuite $c = x^e \mod N$.

Le déchiffrement commence par calculer $c^d \mod \mathrm{N}$ et à considérer le résultat comme une chaîne de bits $s||t$ avec t de longueur k_0. On inverse ensuite le schéma de Feistel en calculant $r = t \oplus \mathrm{H}(s)$ puis $\mathrm{G}(r) \oplus s$. Si les k_1 bits de poids faible du résultat ne sont pas tous nuls, on retourne une erreur. Sinon, le déchiffrement est donné par les $\ell - k_1 - k_0$ bits de poids fort.

4. Le chiffrement de Paillier (1999)

Nous avons vu le chiffrement de Goldwasser Micali qui est sémantiquement sûr sous l'hypothèse de la résidualité quadratique et qui est homomorphe modulo 2. Cependant ce schéma utilise pour espaces des chiffrés un sous-groupe de $(\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times$ où N est un entier RSA pour chiffrer un seul bit. L'expansion (le rapport taille du chiffré sur taille du clair) est donc de $\log_2(\mathrm{N})$, ce qui est loin d'être optimal !

De nombreux systèmes de chiffrements avec des propriétés homomorphes ont été proposés pour améliorer cette expansion. Un des systèmes les plus aboutis en ce sens est le chiffrement de Paillier qui atteint une expansion de 2. Pour cela Paillier utilise le groupe $(\mathbf{Z}/\mathrm{N}^2\mathbf{Z})^\times$ où N est un entier RSA.

Si $\mathrm{N} = pq$, on a

$$\varphi(\mathrm{N}^2) = \varphi(p^2q^2) = \varphi(p^2)\varphi(q^2) = p(p-1)q(q-1) = \mathrm{N}\varphi(\mathrm{N}).$$

Dans la suite on suppose que N est $\varphi(\mathrm{N})$ sont premiers entre eux. On va voir que dans ce cas, chaque élément z de $(\mathbf{Z}/\mathrm{N}^2\mathbf{Z})^\times$ peut s'écrire de manière unique xy où x est dans un sous-groupe d'ordre N et y dans un sous-groupe d'ordre $\varphi(\mathrm{N})$. Le système de Paillier utilise cette décomposition.

Sous-groupe d'ordre N

On note $f = 1 + \mathrm{N}$ dans $\mathbf{Z}/\mathrm{N}^2\mathbf{Z}$. Cet élément est bien inversible car premier avec N et donc N^2. De plus, par la formule du binôme on a, modulo N^2,

$$f^k = (1+\mathrm{N})^k = \sum_{i=0}^{k} \binom{k}{i} \mathrm{N}^i = 1 + k\mathrm{N}.$$

Ainsi, f est d'ordre N. De plus, si on prend un élément x du groupe engendré par f alors il est facile de calculer son logarithme discret en base f : on peut représenter x par un entier modulo N^2 que l'on note toujours x. On a alors $x = f^k = 1 + k\mathrm{N}$ avec $k = (x-1)/\mathrm{N}$.

Sous-groupe d'ordre $\varphi(\mathrm{N})$

On considère la fonction

$$\begin{array}{rccc} s: & (\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times & \longrightarrow & \left(\mathbf{Z}/\mathrm{N}^2\mathbf{Z}\right)^\times \\ & r \mod \mathrm{N} & \longrightarrow & r^{\mathrm{N}} \mod \mathrm{N}^2 \end{array}$$

Montrons que cette fonction est bien définie. Soit $a \equiv b \pmod{\mathrm{N}}$ avec a, b premiers avec N. Il existe $k \in \mathbf{Z}$ tel que $a = b + k\mathrm{N}$. Soit c l'inverse de b modulo N. On a

$$a \equiv b + k\mathrm{N} \equiv b(1 + kc\mathrm{N}) = b(1+\mathrm{N})^{kc} \pmod{\mathrm{N}^2}.$$

Donc :

$$s(a) \equiv a^{\mathrm{N}} \equiv b^{\mathrm{N}}(1+\mathrm{N})^{\mathrm{N}kc} \equiv b^{\mathrm{N}} \pmod{\mathrm{N}^2}.$$

De plus s est un morphisme de groupe ($s(ab) = s(a)s(b)$ pour tout a, b). D'autre part s est injective : si $s(x) \equiv 1 \pmod{\mathrm{N}^2}$, on a aussi $x^{\mathrm{N}} \equiv 1 \pmod{\mathrm{N}}$. Comme N est premier avec $\varphi(\mathrm{N})$ il existe un inverse d de N modulo $\varphi(\mathrm{N})$ et $x \equiv x^{\mathrm{N}d} \equiv 1^d \equiv 1 \pmod{\mathrm{N}}$. Ainsi $\ker s = \{1\}$ et s est un morphisme injectif. En particulier, l'image de s qui est le sous-groupe de $\left(\mathbf{Z}/\mathrm{N}^2\mathbf{Z}\right)^\times$ constitué des puissances N–ièmes est d'ordre $\varphi(\mathrm{N})$.

Fonction de chiffrement de Paillier

Au final, on a l'isomorphisme

$$\begin{array}{rccc} \mathscr{E} : & \mathbf{Z}/\mathrm{N}\mathbf{Z} \times (\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times & \longrightarrow & \left(\mathbf{Z}/\mathrm{N}^2\mathbf{Z}\right)^\times \\ & (m, r) & \longrightarrow & (1+\mathrm{N})^m r^\mathrm{N} \end{array}$$

En effet, on voit facilement que $\mathscr{E}$ est bien définie et que $\mathscr{E}(m_1 + m_2, r_1 r_2) = \mathscr{E}(m_1, r_1)\mathscr{E}(m_2, r_2)$. De même l'injectivité de s entraîne celle de $\mathscr{E}$ et on en déduit que $\mathscr{E}$ est bien bijective par égalité des cardinaux.

Le chiffrement de Paillier utilise cet isomorphisme. La clef publique est N, la clef privée est $\varphi(\mathrm{N})$. Pour chiffrer $m \in \mathbf{Z}/\mathrm{N}\mathbf{Z}$, on prend r aléatoire, $1 < r < \mathrm{N}$ et le chiffré $c = \mathrm{E}(m, r) = (1 + m\mathrm{N})r^\mathrm{N}$ dans $\mathbf{Z}/\mathrm{N}^2\mathbf{Z}$.

Pour déchiffrer, on pourrait inverser s comme précédemment pour retrouver r puis m. Plus directement, on calcule

$$\begin{aligned} c^{\varphi(\mathrm{N})} &= (1+\mathrm{N})^{m\varphi(\mathrm{N})} r^{\mathrm{N}\varphi(\mathrm{N})} \\ &= (1+\mathrm{N})^{m\varphi(\mathrm{N})} r^{\varphi(\mathrm{N}^2)} \\ &= 1 + m\varphi(\mathrm{N})\mathrm{N} \end{aligned}$$

On en déduit :

$$\frac{c^{\varphi(\mathrm{N})} - 1}{\mathrm{N}} \varphi(\mathrm{N})^{-1} \equiv m \pmod{\mathrm{N}},$$

où la division est effectuée dans $\mathbf{Z}$, en utilisant aussi que N et $\varphi(\mathrm{N})$ sont premiers entre eux donc $\varphi(\mathrm{N})$ est bien inversible modulo N.

Propriétés

On peut montrer que le chiffrement de Paillier est sémantiquement sûr pour des attaques à clairs choisis si le problème suivant, dit de la résidualité composite est difficile : étant donné $x \in \left(\mathbf{Z}/\mathrm{N}^2\mathbf{Z}\right)^\times$, existe-t-il $r \in (\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times$ tel que $x = r^\mathrm{N} = s(r)$. C'est une généralisation de l'hypothèse de résidualité quadratique.

Comme la fonction $\mathscr{E}$ est un morphisme, le chiffrement de Paillier est homomorphe modulo N : si c_1 et c_2 chiffrent respectivement m_1 et m_2, $c_1 c_2$ chiffre $m_1 + m_2$ modulo N. De plus l'expansion est maintenant constante égale à 2.

Une application

Supposons que ℓ électeurs veuillent voter à un référendum. Une autorité a un couple (pk, sk) pour Paillier, la clef publique étant un module RSA, N. On suppose que $\ell < \mathrm{N}$ ce qui n'est pas une restriction en pratique (N fait au moins 2048 bits, soit plus de 600 chiffres décimaux).

On désigne par $m_i \in \{0, 1\}$ le vote en clair de l'électeur i, avec 0 pour « non » et 1 pour « oui » (on suppose pour simplifier qu'il n'y a pas de vote blanc). Chaque électeur envoie c_i un chiffré de m_i avec la clef pk à l'autorité. Ces chiffrés sont publiés en ligne. À partir de ces chiffrés, il est possible pour chacun de calculer c un chiffré de $\sum_{i=1}^{\ell} m_i$ en faisant le produit de tous les chiffrés c_i.

En déchiffrant c avec sk, l'autorité retrouvera $\sum_{i=1}^{\ell} m_i \mod \mathrm{N} = \sum_{i=1}^{\ell} m_i$ dans $\mathbf{Z}$ car ce nombre est inférieur à $\ell < \mathrm{N}$. Ainsi on trouvera le résultat du vote : cela donne le nombre de votes pour « oui », donc si ce nombre est supérieur à $\ell/2$ le « oui » l'emporte.

Quelques avantages et inconvénients de ce protocole :

- La sécurité sémantique du chiffrement assure qu'un adversaire extérieur ne pourra briser la confidentialité du vote, il ne sera pas distinguer les chiffrés de 0 de ce de 1.

- La sécurité du chiffrement ne protège pas contre les attaques actives. L'adversaire pourrait modifier un c_i.
- L'autorité, disposant de la clef privée sk, pourrait déchiffrer n'importe quel c_i au lieu de seulement c et retrouver le vote en clair de chaque électeur.
- Un électeur malhonnête pourrait voter 10 ou –10, ce qui correspondrait à voter 10 fois.

Il existe des solutions cryptographiques à tous ces problèmes ! Pour protéger l'intégrité des chiffrés, on peut rajouter une signature numérique qui de plus permet d'authentifier les électeurs inscrits au vote. On verra les signatures numériques au prochain chapitre. Pour protéger contre la toute puissance de l'autorité, on peut partager le procédé de déchiffrement entre plusieurs autorités. Pour finir, on peut vérifier qu'un électeur a bien voté 0 ou 1 en lui demandant de prouver que c_i est un chiffré de 0 ou de 1, par un procédé appelé preuve à divulgation nulle de connaissance.

On peut généraliser simplement ce protocole pour voter pour k candidats. Le vote en clair pour le candidat j, avec $0 \leqslant j \leqslant k-1$, sera A^j où $\mathrm{A} > \ell$ majore strictement le nombre d'électeurs ($\ell + 1$ suffit). La somme des votes décomposée en base A donnera $v_0 + v_1\mathrm{A} + \cdots + v_{k-1}\mathrm{A}^{k-1}$, où $v_j \leqslant \ell < \mathrm{A}$ est le nombre de votes pour le candidat j. Avec le protocole de Paillier, comme le déchiffrement retrouve ce nombre modulo N, pour avoir le résultat dans $\mathbf{Z}$ il faut que $\mathrm{A}^k < \mathrm{N}$. Donc si N fait 2048 bits, soit plus de 600 chiffres décimaux, on peut avoir 10^{10} personnes qui votent pour 60 candidats, donc couvrir une élection mondiale.

5. Mise en oeuvre

Génération des clefs

Pour les schémas cryptographiques fondés sur le logarithme discret, nous avons vu que l'on peut utiliser des groupes standardisés et seulement générer des clefs secrètes qui sont en général des entiers aléatoires modulo l'ordre du groupe.

Pour les schémas fondés sur la factorisation, la génération est plus critique, il faut générer $\mathrm{N} = pq$ difficile à factoriser. Il ne faut donc pas qu'il y ait de biais sur la génération de p et q. Pour avoir N de k bits, il faut donc générer deux nombres premiers de $k/2$ bits. Pour cela on prend des nombres aléatoires de $k/2$ bits et on teste s'ils sont premiers, en général avec un test de pseudo-primalité comme le test de Rabin Miller.

Pour générer e dans RSA on peut prendre n'importe quel entier $2 < e < \varphi(\mathrm{N})$ premier avec $\varphi(\mathrm{N})$. On veut parfois pouvoir accélérer le calcul de $x^e \mod \mathrm{N}$. Dans ce cas, on prend e petit. On évite en général $e = 3$ pour éviter certaines attaques. La valeur $2^{2^4} + 1 = 65537$ est populaire. C'est F_4 le plus grand nombre de Fermat premier connu. Comme c'est un nombre premier, il a de bonnes chances d'être premier avec $\varphi(\mathrm{N})$. D'autre part sa forme particulière fait que le calcul de $x^e \mod \mathrm{N}$ se fait en seulement 17 multiplications modulo N. On calcule ensuite d par l'algorithme d'Euclide étendu. On évite de choisir un d petit, ce qui conduirait à des attaques astucieuses permettant de factoriser N en temps polynomial (si $d < \mathrm{N}^{0.292}$).

Taille des clefs

La taille de N est paramétrée par la complexité des meilleurs algorithmes de factorisation connus. Une classe d'algorithmes de complexité sous-exponentielle vise à trouver x et y dans $(\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times$ tels que $x \neq \pm y$ et $x^2 = y^2$ selon une idée de Fermat comme vu plus haut. Ces algorithmes passent par une phase de collecte de relations puis une phase d'algèbre linéaire pour trouver ces deux entiers, de manière similaire aux algorithmes de calcul d'indice pour le logarithme discret. Le meilleur algorithme est le crible sur corps de nombres (NFS, *Number Field Sieve*) avec une complexité, $\mathrm{L}_\mathrm{N}[1/3, c]$. Le record actuel date de février 2020 : factorisation d'un entier RSA de 829 bits (250 chiffres décimaux, avec un coût de 2450 ans sur un seul cœur pour la phase de crible et 250 ans pour la phase d'algèbre linéaire)[1].

[1] cf. `https://en.wikipedia.org/wiki/Integer_factorization_records`

D'autres algorithmes de factorisation sont adaptés à trouver des facteurs relativement petits d'un entier N : méthode ρ de Pollard (1975), de complexité exponentielle $\mathcal{O}(\sqrt{p})$ où p est le plus petit facteur premier de N. De même un algorithme utilise les courbes elliptiques (ECM) avec une complexité sous-exponentielle en $L_p[1/2, \sqrt{2}]$. Il est moins intéressant que NFS pour attaquer des entiers RSA, mais en fait utilisé comme sous-procédure dans NFS. Au final les tailles d'entiers RSA, N, recommandées sont les mêmes que pour le logarithme discret dans $(\mathbf{Z}/p\mathbf{Z})^\times$:

Niveau de sécurité	taille de N (bits)
112	**2048**
128	**3072**
192	7680
256	15360

Chapitre VII

Signatures numériques

Comme le chiffrement, les signatures numériques sont une application cruciale et extrêmement répandue en cryptographie à clef publique. Alice dispose toujours d'un couple clef publique (clef de vérification), clef secrète (clef de signature). La clef secrète va lui permettre, à l'aide d'un algorithme de signature, de pouvoir signer un document numérique représenté par une chaîne de bits m (un fichier pdf, un email, une transaction bancaire...) en produisant une signature σ.

Bob ayant à sa disposition la clef publique d'Alice, le document m et la signature σ de m émise par Alice, va utiliser un algorithme de vérification qui lui permettra de vérifier si cette signature est bien valide.

Les signatures sont largement utilisées (signature de certificats pour l'authentification des sites web, de logiciels, de transactions financières, authentification forte...).

1. Propriétés

On veut pour ces schémas de signatures des propriétés analogues à la signature manuscrite classique :

- la signature doit engager la responsabilité du signataire : seul lui connaît la clef privée permettant de signer (notion de non-répudiation) ;
- la signature ne peut être imitée, cela garantit qu'elle provient d'un utilisateur donné (notion d'authentification) ;
- le message signé n'a pas été modifié (notion d'intégrité) ;
- la signature peut-être vérifiée par tout le monde en utilisant la clef publique (notion de vérification universelle).

On définit plusieurs niveaux de sécurité. L'attaquant peut avoir accès seulement à la clef publique de vérification, ou connaître des couples (m, σ) valides pour une certaine clef publique (attaque à messages connus), ou pouvoir interroger un oracle de signature sur des messages m (attaque à messages choisis). On définit également plusieurs buts : retrouver la clef de signature, être capable de produire une signature d'un message donné (contrefaçon universelle), ou d'un message de son choix (contrefaçon sélective), ou de savoir produire un nouveau couple $(m^\star, \sigma^\star)$ valide (contrefaçon existentielle).

Le plus haut niveau de sécurité pour un schéma de signature est la résistance aux contrefaçons existentielles pour des attaques à messages choisis.

Pour signer une chaîne de bits m, en général on commence par appliquer sur m une fonction de hachage cryptographique h et l'algorithme traite ensuite $h(m)$ et la signature ne dépend donc que de $h(m)$. Des attaques sur la fonction de hachage donnent donc des attaques sur la signature. Connaissant un couple valide (m, σ), trouver un seconde pré-image va donner $m' \neq m$ avec $h(m) = h(m')$, donc (m', σ) est aussi une

signature valide. Si l'attaquant trouve une collision sur h, il pourra également réaliser une contrefaçon par une attaque à message choisi. Une collision permet aussi à un utilisateur légitime de répudier des signatures (il signe m et dit plus tard qu'il avait en fait signé m'). La non résistance à la notion de sens-unique permet aussi des attaques pour certaines constructions comme RSA-FDH, défini plus bas.

2. Signature RSA-FDH

On utilise les mêmes notations que pour le chiffrement. On pose $N = pq$ avec p, q deux grands nombres premiers distincts . On note e et d tels que $ed \equiv 1 \pmod{\varphi(N)}$. La clef publique d'Alice est toujours (N, e) et d sa clef privée. On note h une fonction de hachage cryptographique de $\{0,1\}^*$ dans $(\mathbf{Z}/N\mathbf{Z})^\times$. On présente ici la signature RSA-FDH (*Full Domain Hash*). Il existe d'autres constructions de signatures utilisant la permutation à trappe RSA, comme RSA-PSS (*Probabilistic Signature Scheme*).

Pour une chaîne de bits m, Alice calcule

$$\sigma :\equiv h(m)^d \pmod{N}.$$

Bob disposant de m, (N, e) et σ pourra vérifier que σ est bien la signature d'Alice du message m en testant si

$$\sigma^e \stackrel{?}{\equiv} h(m) \pmod{N}.$$

En idéalisant les propriétés de la fonction de hachage (modèle de l'oracle aléatoire), on montre que si le problème RSA est difficile alors il est aussi difficile de produire une contrefaçon existentielle pour des attaques à messages choisis pour la signature RSA-FDH.

3. Signature de Schnorr

Cette signature est construire à partir d'un schéma d'authentification du à Schnorr (1989), qui est une preuve à divulgation nulle de connaissance de la connaissance d'un logarithme discret.

Pour cela Alice et Bob se mettent tout d'abord publiquement d'accord sur un groupe $(G, \times)$ cyclique d'ordre premier q et g un générateur.

Alice (une carte à puce) a une clef privée $1 < x < q$ et une clef publique $h = g^x$. Elle souhaite s'authentifier auprès de Bob (un lecteur de carte), avec sa clef publique h, en prouvant qu'elle connaît x, mais sans donner d'information sur x.

Le protocole se déroule ainsi :

- Alice choisit aléatoirement de manière uniforme $r \in \mathbf{Z}/q\mathbf{Z}$, calcule $t = g^r$ et envoie t à Bob ;
- Bob lui envoie un défi aléatoire uniforme $c \in \mathbf{Z}/q\mathbf{Z}$;
- Alice répond $s = r + xc$, calculé dans $\mathbf{Z}/q\mathbf{Z}$;
- Bob vérifie que $g^s = th^c$, si c'est le cas, il accepte l'authentification d'Alice.

Le protocole est bien correct : si Alice connaît bien x et suit le protocole, on aura $th^c = g^r(g^x)^c = g^{r+xc} = g^s$. D'autre part, r étant aléatoire, on montre qu'il masque la valeur de x dans s, ainsi Bob n'apprend rien sur x. Enfin, on peut montrer que si Alice réussit à s'authentifier, elle connaît forcément. En effet, on montre qu'elle est capable de le faire pour une même valeur de $t = g^r$ mais pour deux valeurs c_1, c_2 différentes. On a donc deux réponses s_1, s_2 différentes et

$$g^{s_1} = th^{c_1} \quad \text{et} \quad g^{s_2} = th^{c_2}.$$

On a alors en combinant ces deux équations

$$g^{s_1 - s_2} = h^{c_1 - c_2},$$

et comme $c_1 - c_2$ est inversible modulo q,

$$g^{(s_1-s_2)(c_1-c_2)^{-1}} = h.$$

À partir de deux réponses d'Alice pour une même valeur t, on peut donc calculer $x = (s_1 - s_2)(c_1 - c_2)^{-1}$ prouvant donc qu'elle connaît bien cette valeur.

On construit un algorithme de signature à partir de ce protocole. Alice a toujours la clef privée x et la clef publique $h = g^x$. On considère une fonction de hachage cryptographique h de $\{0,1\}^*$ à valeurs dans $\mathbf{Z}/q\mathbf{Z}$. Pour signer m, Alice choisit aléatoirement de manière uniforme $r \in \mathbf{Z}/q\mathbf{Z}$, calcule $t = g^r$, et pose $c = \mathrm{H}(m\|t)$, et calcule $s = r + xc$ dans $\mathbf{Z}/q\mathbf{Z}$. La signature est (c, s).

La vérification consiste à recalculer t en utilisant l'équation $t = g^s h^{-c}$, puis à vérifier $c : c \stackrel{?}{=} \mathrm{H}(m\|t)$.

En utilisant les propriétés du schéma d'authentification, on peut montrer que cette signature est sûre contre des contrefaçons existentielles pour des attaques à messages choisis si le problème du logarithme discret est difficile dans G.

On peut instancier les signatures de Schnorr en utilisant un sous-groupe de $(\mathbf{Z}/p\mathbf{Z})^\times$ ou avec des courbes elliptiques. Comme déjà vu, les courbes elliptiques vont fournir des instantiations plus efficaces en terme de taille de signature et de coût calculatoire.

Il existe d'autres schémas de signatures fondés sur le problème du logarithme discret dérivés des signatures d'Elgamal. Ces schémas sont standardisés : DSA utilisant un sous groupe d'ordre premier q de $\left(\mathbf{Z}/p\mathbf{Z}\right)^\times$ et ECDSA utilisant des courbes elliptiques. Ce dernier schéma est largement utilisé à l'heure actuelle, il sera vu en TD.

4. Signatures utilisant des couplages

À partir d'un schéma de chiffrement fondé sur l'identité, on peut construire de manière générique un schéma de signature dont la sécurité sera reliée à celle du chiffrement fondé sur l'identité.

La clef secrète maître msk sera la clef privée du schéma de signature, et la clef publique maître sera la clef publique. Une identité id sera un message pour le schéma de signature et l'algorithme de dérivation de clef sera celui de signature : la clef privée sk_{id} est la signature d'id.
Pour vérifier une signature de id, on chiffre un message aléatoire pour l'identité id (le message) et on le déchiffre avec sk_{id} (la signature). Si le résultat correspond au message aléatoire, la signature est acceptée.

Pour des instantiations concrètes, l'algorithme de vérification peut-être simplifié (on chiffre l'élément neutre, et on prend l'aléa de chiffrement égal à 1). Ainsi avec le chiffrement de Boneh et Franklin, on obtient l'algorithme de signature suivant.

On considère un couplage symétrique $e : \mathrm{G} \times \mathrm{G} \to \mathrm{G}_t$, avec P un générateur de G d'ordre q. La clef secrète sk est un entier pris au hasard entre 1 et q. On pose $pk = sk\,\mathrm{P} \in \mathrm{G}$. On utilise aussi une fonction de hachage, $h : \{0,1\}^* \to \mathrm{G}$.
Pour signer m, Alice calcule $\mathrm{Q} = h(m) \in \mathrm{G}$ puis $\sigma = sk\,\mathrm{Q} \in \mathrm{G}$. Pour vérifier, le chiffrement de l'élément neutre avec $r = 1$ correspond essentiellement à calculer $e(pk, \mathrm{Q})$ et le déchiffrement à calculer $e(\mathrm{P}, \sigma)$. Ainsi l'algorithme de vérification teste l'égalité

$$e(pk, h(m)) \stackrel{?}{=} e(\mathrm{P}, \sigma).$$

Ce test correspond donc à tester si le triplet $(pk, h(m), \sigma)$ est un triplet Diffie-Hellman. En effet, si σ est une signature de m, ce triplet est égal à $(sk\,\mathrm{P}, h(m), sk\,h(m))$.

Ce schéma de signature du à Boneh, Lynn and Shacham (BLS, 2001) fournit des signatures très courtes, avec un seul élément de groupe. On montre que ce schéma est sûr contre des contrefaçons existentielles pour des attaques à messages choisis si le problème calculatoire Diffie-Hellman est difficile.