Chapitre 12
Logarithme discret et cryptographie : Texte

§1. Introduction

Il y a à peine quelques années, le problème de la sécurité des transmissions de données semblait être l'apanage des seuls militaires. Ce n'est plus le cas, avec l'essor des techniques numériques dans le commerce (Internet, les cartes de crédit, les télécommunications, les décodeurs de télévision, etc.)

Les méthodes empiriques traditionnelles se sont révélées trop fragiles ; elles reposaient souvent sur le schéma suivant : on choisit un livre, une fois pour toutes, et on considère la permutation des vingt-six lettres de l'alphabet définie par les vingt-six premières lettres de ce livre.

Le codage d'un texte consiste alors à appliquer cette permutation σ au texte à coder, et le décodage à appliquer la permutation réciproque σ^{-1} au texte à décoder.

En numérique, si par exemple le message à transmettre est codé sur 64 bits, on peut employer cette technique en considérant une permutation $\sigma \in S_{64}$. C'est ce genre d'idées qui est sous-jacent au procédé DES, dû à IBM, et qui est encore le plus utilisé en informatique.

Le talon d'Achille de ce genre de cryptosystème est le suivant : si quelqu'un sait coder, il sait aussi décoder, car il est très facile de trouver la permutation réciproque d'une permutation sur 26, 64, 128 ou même 256 lettres.

C'est pourquoi l'apparition de cryptosystèmes dits *à clé publique*, à la fin du siècle dernier, a fait sensation.

Ces cryptosystèmes, comme le nom l'indique, sont tels que le *codage* est public : tout le monde connaît l'algorithme pour coder le message. Mais on ne peut pas en déduire le décodage.

En fait, cela revient à construire une permutation σ d'un ensemble à N éléments, mais ici N est gigantesque (de l'ordre de 10^{500}, par exemple.) On ne peut même pas écrire la liste de ces éléments, et la méthode habituelle pour trouver la permutation réciproque d'une permutation donnée ne peut plus s'appliquer.

Dans le paragraphe suivant, nous décrivons le cryptosystème le plus célèbre de ce genre : la méthode RSA, du nom des ses inventeurs Rivest, Shamir et Adelman.

§2. RSA

Cette méthode est basée sur le théorème suivant :

Théorème.- *Soit G un groupe fini d'ordre N et m un entier premier à N. L'application $f_m : G \to G$ définie par $x \mapsto x^m$ est bijective, et son application réciproque est l'application f_k définie par $x \mapsto x^k$, où k est l'inverse de m dans le groupe multiplicatif $(\mathbf{Z}/N\mathbf{Z})^*$.*

Si l'on connaît N et m, on en déduit donc rapidement k, par exemple par l'algorithme d'Euclide étendu, qui permet d'obtenir en $O(\log N)$ opérations élémentaires des entiers k et l tels que $mk + lN = 1$.

Par ailleurs, pour $x \in G$, le calcul de x^k se fait lui aussi rapidement, en $O(\log k)$ opérations dans le groupe G, en écrivant k en binaire. L'algorithme RSA consiste à fournir au codeur un groupe G dans lequel il sait calculer, mais dont il ne connaît pas l'ordre N, et un certain entier m. Le codage consiste alors, à partir d'un message x identifié à un élément de G, à calculer x^m. Le décodage nécessite de connaître k, inverse de m dans $(\mathbf{Z}/N\mathbf{Z})^*$. Dans les cas utilisés dans la pratique, on ne sait pas comment trouver k sans connaître N.

Donnons l'exemple habituel de groupe G utilisé : soient deux nombres premiers distincts p et q, et soient $M = pq$, et $G = (\mathbf{Z}/M\mathbf{Z})^*$. Il suffit de connaître M pour savoir calculer dans G (et rapidement : les additions et multiplications se font en $O(\log^2 M)$ opérations), mais on ne connaît l'ordre de G, à savoir $(p-1)(q-1)$ que si l'on connaît p et q, c'est-à-dire la décomposition en facteurs premiers de M.

On ne peut bien sûr pas choisir $m = 2$ pour coder. Par contre, si $m = 3$, et si l'on a choisi p et q tels que $p \equiv q \equiv 2 \mod 3$, $x \mapsto x^m$ est bien une permutation de G, et on trouve immédiatement $k = \dfrac{2(p-1)(q-1)+1}{3}$.

La force de la méthode RSA réside dans le fait qu'à l'heure actuelle on ne connaît pas de méthode rapide de factorisation d'un entier : il faut actuellement des mois et plusieurs centaines de stations de travail travaillant en parallèle pour factoriser un entier de l'ordre de 150 chiffres.

Quelques remarques.

- Supposons que le mot à coder soit un entier $x < \sqrt[3]{M}$. On retrouve alors rapidement x à partir de x^3, sans rien savoir de la factorisation de M.
- Supposons que l'on ait un même entier x à coder pour, par exemple, trois destinataires différents. On pourrait penser qu'il est plus prudent de considérer trois entiers M_1, M_2 et M_3 distincts (et $> x$) pour le codage RSA. En fait, il n'en est rien : une personne mal intentionnée qui connaît les trois messages codés $x^3 \mod M_1, x^3 \mod M_2$ et $x^3 \mod M_3$ reconstitue immédiatement x par le théorème chinois !
- Au lieu de prendre $m = 3$, on peut prendre tout m premier à $(p-1)(q-1)$. L'intérêt de prendre m petit est que le codage est d'autant plus rapide, mais k est alors grand, et le décodage est plus lent. Par ailleurs, comme les deux remarques précédentes le montrent, m petit a des effets pervers. C'est pourquoi il a été proposé de prendre d'autres valeurs de m, même très grandes, mais pour lesquelles le codage reste rapide, par exemple $m = 65537$.

§3. Signatures sécurisées

Un problème voisin du précédent consiste à s'assurer de l'identité d'un interlocuteur : par exemple, si A introduit une carte bancaire dans un lecteur de cartes, celui-ci doit s'assurer que A est bien le possesseur de la carte en lui demandant son code secret. Ou encore : si A veut se connecter à un ordinateur, celui-ci s'assure de son identité en lui demandant son mot de passe. De même, un décodeur de télévision satellite ou câble s'assure de l'identité d'un utilisateur à l'aide d'un code secret, etc.

La méthode la plus utilisée est la suivante : sur la carte, figure le code secret *codé* par un algorithme à clé publique.

Lorsque le possesseur de la carte tape le nombre à quatre chiffres correspondant à son code sur le clavier du lecteur, le lecteur code ce nombre, et vérifie que le résultat est identique à celui qui est sur la carte.

De même, si, depuis chez lui, un utilisateur d'un réseau informatique veut se connecter sur l'un des ordinateurs du réseau, il envoie son mot de passe, l'ordinateur le code, et vérifie qu'il est bien égal au mot de passe codé qu'il possède.

Une alternative à ce type de codage est la suivante : supposons que A connaisse la clé publique P d'un algorithme à clé publique, et que B soit le seul à en connaître la clé privée S.

Bien sûr, $S \circ P$ est l'application identique, mais il en est de même de $P \circ S$. Par suite, A a une méthode simple pour s'assurer de l'identité de B : supposons par exemple que B s'appelle Bernard ; B envoie alors à A le message $X = S(Bernard)$ Pour vérifier que B est bien Bernard, A calcule alors $P(X) = P \circ S(Bernard) = Bernard$.

Comme on le voit, il n'est pas du tout nécessaire que A connaisse la clé privée P pour vérifier la signature de B !

§4. Le logarithme discret

Soit G un groupe cyclique d'ordre N , et soit g l'un de ses générateurs. Le *logarithme discret* de G en base g est l'application $\log_g : G \to \mathbf{Z}/N\mathbf{Z}$ définie par $\log_g(g^m) = m$. C'est donc un isomorphisme de groupes, d'application réciproque $m \mapsto g^m$.

Il y a des cas où le calcul du logarithme discret est trivial, par exemple si $G = \mathbf{Z}/n\mathbf{Z}$ et $g = 1$. Par contre, si $G = (\mathbf{Z}/p\mathbf{Z})^*$, où p est un nombre premier, trouver un générateur de G n'est déjà pas facile, et le calcul du logarithme discret semble très difficile.

Ceci a donné l'idée d'algorithmes sécurisés dits symétriques : supposons que A et B veuillent s'assurer de leur identité respective. Ils se donnent un groupe G cyclique de générateur g ; A (resp. B) choisit au hasard un entier

a (resp. b), et envoie à B (resp. A) l'élément g^a (resp. g^b) ; A et B s'assurent chacun de l'identité de l'autre en calculant $g^{ab} = (g^a)^b = (g^b)^a$.

§5. Preuves sans apport d'information

Le principal talon d'Achille de la méthode de signature sécurisée décrite dans le troisième paragraphe réside dans le fait qu'à un moment donné, la personne qui doit prouver son identité doit transmettre son code en clair au lecteur de cartes ou à l'ordinateur, par exemple, afin que ce dernier puisse vérifier l'identité de son interlocuteur. Il est donc possible à quelqu'un de mal intentionné d'intercepter ce code (par exemple, dans le cas d'un distributeur de billets, si quelqu'un, derrière vous, regarde les chiffres que vous tapez). Plus généralement, supposons que A possède un secret (ici un code secret), et que B veuille s'assurer que A possède ce secret. Est-ce possible sans que A donne à B une quelconque information sur ce secret ? Il existe des algorithmes (de type probabiliste : B saura avec une très forte probabilité que A possède bien le secret en question) résolvant ce problème. Nous en donnons deux ci-dessous :

I.- Soit $(G, \times)$ un groupe dans lequel on sait calculer rapidement, mais dans lequel le calcul des racines carrées est difficile (par exemple $(\mathbf{Z}/pq\mathbf{Z})^*$, où p et q sont de grands nombres premiers distincts).

Le secret de A est une racine carrée x d'un élément $a \in G$; B connaît a, et veut s'assurer que A connaît x.

Il demande alors à A de choisir $y \in G$, et de lui envoyer $b = y^2$. Il aura alors le choix entre deux questions qu'il pourra poser à A :

1) Lui demander y (Il vérifie alors que $b = y^2$)

2) Lui demander $c = xy$ (Il vérifie alors que $ab = c^2$).

Supposons que A ne connaisse pas x.

a) S'il parie que B va lui poser la question 2), il choisit un élément quelconque $z \in G$, et envoie $b = a/z^2$ à B. Si B lui pose effectivement la question 2), le bluff aura marché, car A enverra $c = a/z$. Par contre, si B pose la question 1), B sera piégé, car il ne connaît pas de racine carrée de a/z^2 (sinon il connaîtrait une racine carrée de a).

b) S'il parie que B va poser la question 1), il joue le jeu en choisissant bien un $y \in G$ et en envoyant y^2 à B.

Mais il se fait piéger si B pose la question 2) !

Dans le cas par exemple du groupe $(\mathbf{Z}/pq\mathbf{Z})^*$, il est facile de voir qu'en $O(\log pq)$ questions, on a une probabilité de $1/(pq)$ d'être trompé par A. Or, les informations recueillies par B pendant son questionnaire ne lui donnent pas d'information susceptible de trouver x.

II.- Une méthode assez voisine repose sur le logarithme discret.

Soit G un groupe cyclique de générateur g, dans lequel le problème du logarithme discret est difficile.

Ici, le secret de A est un entier x, et B connaît $a = g^x$. Il doit s'assurer que A connaît bien x.

Il demande à A de choisir un entier y, et de lui envoyer $b = g^y$. Il a alors le choix entre deux questions :

1) demander y à A (Il vérifie alors que $b = g^y$)

2) demander $z = x + y$ à A (Il vérifie alors que $g^z = ab$.)

Comme précédemment, la probabilité que A ne connaisse pas x au bout de $O(\log N)$ questions est de l'ordre de $1/N$, et les informations recueillies par B durant son questionnaire ne lui donnent pas d'informations sur x.

Indications pour le candidat

Le candidat pourra choisir de développer les points qu'il désire sur le texte. En ce qui concerne le passage sur machine, il pourra programmer la méthode RSA, pour M et m quelconque.

Plusieurs points du texte sont affirmés sans démonstration. Le candidat est invité à les expliciter.

Le candidat pourra par exemple démontrer le théorème de la section 2, et détailler les remarques concluant cette section. Par ailleurs, pourquoi le connaissance de l'ordre de G implique-t-elle la connaissance de p et q ? Pourquoi ne peut-on pas choisir $m = 2$?

Le candidat pourra également programmer le calcul du logarithme discret dans le cas du groupe $(\mathbf{Z}/n\mathbf{Z}, +)$, puis dans le cas du groupe $(\mathbf{Z}/p\mathbf{Z})^*, \times)$, où p est un nombre premier, avec l'algorithme de son choix, ou encore programmer une méthode de preuve sans apport d'information proposée dans le texte.