Mémoire d'habilitation à diriger des recherches

Université Pierre et Marie Curie, Paris 6

Cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs

Nicolas Sendrier

Institut National de Recherche en Informatique et Automatique, Rocquencourt

Soutenue le 5 mars 2002 devant un jury composé de

Rapporteurs :	Thierry Berger
	Gregory Kabatianski
	Daniel Lazard
Examinateurs :	Pascale Charpin (*Directeur*)
	Marc Girault
	James Massey (*Président*)
	François Morain
	Alexander Vardy

Table des matières

Introduction générale

Dans ce travail, nous allons nous intéresser aux cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs. Deux systèmes retiendront plus particulièrement notre attention, ceux proposés respectivement en 1978 par Robert McEliece [McE78] et en 1986 par Harald Niederreiter [Nie86]. Après plus de vingt années d'effort [BMvT78, AM87, LB88, Leo88, Ste89, BN90, vT90, Gib91, CC93, CC94, Can95, CC98, CS98], aucune cryptanalyse n'a pu venir à bout de ces systèmes lorsque les codes de Goppa sont utilisés pour engendrer la clé publique. L'usage des codes de Goppa fut proposée par McEliece dans son papier originel. Dans sa version, Niederreiter suggérait en revanche d'autres possibilités pour la famille de codes secrets (codes concaténés, codes de Reed-Solomon, ...) qui se révélèrent fragiles [SS92, Sen98].

Les systèmes de chiffrement proposés par McEliece et Niederreiter présentent l'avantage de permettre, à niveau de sécurité identique, une mise en œuvre significativement plus rapide que la plupart des systèmes utilisés en pratique (RSA, courbes elliptiques, ...). Les points noirs sont la taille relativement importante de la clé publique et l'absence, jusqu'à récemment [CFS01], de schéma de signature digitale efficace.

Il existe principalement deux approches pour analyser la sécurité de ces systèmes. La première, qui est aussi celle qui a fait l'objet de la plus grande attention, consiste à tenter de retrouver le texte clair correspondant à un chiffré donné, c'est-à-dire en pratique à effectuer un décodage dans un code dont on connaît une matrice génératrice mais dont la structure algébrique a été masquée. La seconde approche consiste justement à essayer de retrouver cette structure algébrique. Les problèmes soulevés par cette seconde approche, ou cryptanalyse structurelle, font l'objet de la première partie de ce document dans laquelle nous aborderons divers problèmes liés à l'équivalence de codes. Dans la seconde partie nous aborderons les cryptosystèmes proprement dits, fonctionnalités, implémentation, sécurité ...

Dans les cryptosystèmes que nous étudions, la clé secrète est un code correcteur d'erreurs binaire muni d'un algorithme de décodage rapide (le code caché). La clé publique est une matrice génératrice (ou de parité) d'un code (le code public) obtenu en permutant aléatoirement les positions du code caché. Ainsi code public et code secret sont équivalents et le problème de la cryptanalyse structurelle se ramène à trouver dans la classe d'équivalence du code public l'un des codes appartenant à une famille déterminée, les codes de Goppa par exemple. Dans la première partie de ce document nous précisons la notion d'équivalence d'un point de vue algébrique, puis d'un point de vue algorithmique. Déterminer si deux codes linéaires sont équivalents est difficile dans le pire cas [PR97]. Nous montrerons ici qu'il est facile en moyenne et qu'il peut être résolu à l'aide de l'algorithme de séparation du support [Sen00] en temps polynomial lorsque le hull (intersection d'un code avec son orthogonal) a une faible dimension, ce qui est le cas pour presque tous les codes linéaires [Sen97b].

Dans la seconde partie nous nous intéresserons à des aspects plus concrets de ces cryp-

tosystèmes, comme la taille des paramètres pour résister aux cryptanalyses et l'évolution de ces paramètres à sécurité constante ; les problèmes liés à la mise en œuvre, le décodage des codes de Goppa, la réduction de la taille des clés, le codage des mots de poids constants ; les moyens d'obtenir une signature digitale. Enfin, dans un dernier chapitre nous jetterons les bases d'une réduction de la sécurité des cryptosystèmes basés sur les codes de Goppa, qui donne l'espoir d'atteindre un jour une preuve de sécurité formelle.

Première partie

Structure des codes linéaires

Chapitre 1

Codes équivalents

Soit un alphabet A et n un entier strictement positif, la *distance de Hamming* entre deux éléments de A^n est le nombre de coordonnées où ces éléments diffèrent. L'espace métrique A^n muni de cette distance est appelé *espace de Hamming* et ses éléments seront appelés *mots*. L'alphabet A sera généralement un corps ou un anneau. Si ce n'est pas le cas, nous distinguerons par commodité un de ses éléments que nous noterons 0. Le *poids de Hamming* d'un mot sera son nombre de coordonnées non nulles, c'est-à-dire sa distance de Hamming au mot $\mathbf{0} = (0, \ldots, 0)$. Enfin pour indexer les positions des mots nous utiliserons un ensemble de cardinal n que nous noterons I_n, typiquement $I_n = \{1, \ldots, n\}$. Le *support* d'un mot de A^n est le sous-ensemble de I_n indexant ses coordonnées non nulles. Le lecteur intéressé pourra trouver dans [Ber96] un complément d'information sur les isométries de l'espace de Hamming.

1.1 Isométries de l'espace de Hamming

Une isométrie d'un espace métrique est une application de cet espace dans lui-même telle que la distance entre les images de deux mots est égale à la distance entre ces mots. L'ensemble des isométries muni de la composition forme un groupe. Dans le cas d'un espace de Hamming, un résultat de Pierre Bonneau caractérise les isométries.

Proposition 1 *Les isométries de l'espace de Hamming A^n sont les applications de la forme*

$$\begin{array}{rccc} \Phi_{\Pi,\sigma} : & A^n & \to & A^n \\ & (x_i)_{i\in I_n} & \mapsto & (\pi_i(x_{\sigma^{-1}(i)}))_{i\in I_n} \end{array} \tag{1.1}$$

où σ est une permutation de I_n et $\Pi = (\pi_i)_{i\in I_n}$ est une famille de permutations de A.

Preuve : Pour tout x dans A^n, tout i dans I_n et tout a dans A, nous notons $\Lambda_{a,i}(x)$ le mot de A^n obtenu en remplaçant la i-ème coordonnée de x par a.

Remarquons d'abord que toute application de la forme (1.1) est une isométrie.

Réciproquement. Soit f une isométrie de A^n. Pour tout i dans I_n et tout a dans A, le mot $f(\Lambda_{a,i}(\mathbf{0}))$ est à distance 1 de $f(\mathbf{0})$, donc il existe un unique (b, j) dans $A \times I_n$ tel que $f(\Lambda_{a,i}(\mathbf{0})) = \Lambda_{b,j}(f(\mathbf{0}))$, de plus $b \neq f(\mathbf{0})_j$. Pour i donné, l'indice j est le même pour tout $a \neq 0$, dans le cas contraire, on pourrait trouver a' dans A tel que $f(\Lambda_{a',i}(\mathbf{0}))$ soit à distance 2 de $f(\Lambda_{a,i}(\mathbf{0}))$, ce qui contredirait la propriété d'isométrie de f. Cela nous permet de poser $j = \sigma(i)$, $\pi_j(a) = b$ et $\pi_j(0) = f(\mathbf{0})_j$. Enfin, chaque π_i est une permutation car toute isométrie est bijective. Soit $\Pi = (\pi_i)_{i\in I_n}$ et $g = \Phi^{-1}_{\Pi,\sigma} \circ f$.

Il reste à montrer que l'isométrie g est égale à l'identité. D'après ce qui précède, tout mot de poids 0 ou 1 est un point fixe de g. Supposons que tout mot de poids $< i$ soit un point fixe de g. Soit $x = (x_1, \ldots, x_n)$ un mot de poids $i > 1$. Quitte à modifier l'ordre des coordonnées, nous pouvons supposer que $x_1 \neq 0$ et $x_2 \neq 0$. Soient y, z et u les mots définis par $y = (x_1, 0, x_3, \ldots, x_n)$, $z = (0, x_2, x_3, \ldots, x_n)$ et $u = (0, 0, x_3, \ldots, x_n)$. Ces trois mots sont de poids $< i$ et sont donc des points fixes de g. Enfin, puisque g est une isométrie, $g(x)$ doit être à distance 1 de $g(y) = y$ et de $g(z) = z$ et à distance 2 de $g(u) = u$. L'unique mot vérifiant ces propriétés est x, donc $g(x) = x$. On en déduit aisément que $f = \Phi_{\Pi,\sigma}$. ⋄

Dans le cas où l'alphabet est un corps fini $\mathbf{F}_q$, nous pouvons définir les isométries semi-linéaires comme étant les isométries de la forme

$$\begin{array}{rccc} \Psi_{V,\pi,\sigma} : & \mathbf{F}_q^n & \to & \mathbf{F}_q^n \\ & (x_i)_{i\in I_n} & \mapsto & (v_i \pi(x_{\sigma^{-1}(i)}))_{i\in I_n} \end{array} \tag{1.2}$$

où $V = (v_i)_{i\in I_n}$ est une famille d'éléments non nuls de $\mathbf{F}_q$, π est un automorphisme du corps $\mathbf{F}_q$ et σ une permutation de I_n.

Proposition 2 *Une isométrie f de $\mathbf{F}_q^n$ qui transforme tout espace vectoriel en un espace vectoriel est semi-linéaire.*

Preuve : Remarquons tout d'abord que puisque $f(\mathbf{0}) = \mathbf{0}$ nous avons $\pi_i(0) = 0$ pour tout i. D'autre part, l'image de la droite engendrée par le mot $\mathbf{1}$ est une droite, donc, pour tout α dans $\mathbf{F}_q$, le mot $f(\alpha\mathbf{1}) = (\pi_i(\alpha))_{i\in I_n}$ est proportionnel au mot $f(\mathbf{1}) = (\pi_i(1))_{i\in I_n}$. Donc, pour α donné, $\pi_i(\alpha)/\pi_i(1)$ a la même valeur pour tout i, nous noterons $\pi(\alpha)$ cette valeur. Soit $V = (\pi_i(1))_{i\in I_n}$, nous avons $f = \Psi_{V,\pi,\sigma}$.

Il reste à montrer que π est un automorphisme du corps $\mathbf{F}_q$. Soit $g : (x_i)_{i\in I_n} \mapsto (\pi(x_i))_{i\in I_n}$. Nous avons $g = f \circ \Psi_{V,1,\sigma}^{-1}$ et donc l'image par g de tout espace vectoriel est un espace vectoriel. En particulier, soient $x = (1, \alpha, 0, \ldots)$ et $y = (0, \beta, 0, \ldots)$, les mots $g(x) = (1, \pi(\alpha), 0, \ldots)$, $g(y) = (0, \pi(\beta), 0, \ldots)$ et $g(x+y) = (1, \pi(\alpha+\beta), 0, \ldots)$ sont liés et donc $\pi(\alpha+\beta) = \pi(\alpha)+\pi(\beta)$. De même les mots $x = (1, \alpha, 0, \ldots)$ et $y = (\beta, \alpha\beta, 0, \ldots)$ étant liés c'est également le cas de leurs images $g(x) = (1, \pi(\alpha), 0, \ldots)$ et $g(y) = (\pi(\beta), \pi(\alpha\beta), 0, \ldots)$. On en déduit que $\pi(\alpha\beta) = \pi(\alpha)\pi(\beta)$ et donc que π est un automorphisme du corps $\mathbf{F}_q$. ⋄

1.2 Équivalence

Deux codes sont équivalents s'ils appartiennent à la même orbite sous l'action du groupe des isométries de l'espace de Hamming. Autrement dit

Définition 3 (Équivalence par isométrie) *Deux codes de longueur n sur A sont* équivalents par isométrie *s'il existe une permutation σ de I_n et une famille $\Pi = (\pi_i)_{i\in I_n}$ de permutations de A tels que $C' = \Phi_{\Pi,\sigma}(C)$.*

Cette définition est parfois restreinte aux permutations du support, nous parlerons alors d'équivalence par permutation.

Définition 4 (Équivalence par permutation) *Pour tout code C de longueur n et toute permutation σ de I_n, nous notons*

$$\sigma(C) = \{(x_{\sigma^{-1}(i)})_{i\in I_n} \mid (x_i)_{i\in I_n}\}.$$

Les codes C et C' sont dits équivalents par permutation.

Dans le cas des codes linéaires, on se limite à l'action des isométries semi-linéaires. Cette définition coïncide avec la première pour la plupart des codes grâce à au résultat suivant, du à Montpetit [Mon87], et qui généralise un résultat de MacWilliams [1]

Proposition 5 *Soit f une isométrie de $\mathbf{F}_q^n$ et C un code linéaire non décomposable*[2] *de longueur n sur $\mathbf{F}_q$. Il existe une isométrie semi-linéaire g telle que $g(C) = f(C)$.*

Ce résultat est d'une grande importance, il montre en effet que si deux codes linéaires non décomposables sont dans la même orbite sous l'action du groupe des isométries semi-linéaires, alors ils sont également dans la même orbite sous l'action du groupe des isométries. Nous pouvons à présent donner une définition de l'équivalence pour les codes linéaires.

Définition 6 (Équivalence) *Deux codes linéaires C et C' de longueur n sur $\mathbf{F}_q$ sont* équivalent *s'il existe une permutation σ de I_n, une famille $V = (v_i)_{i \in I_n}$ d'éléments non nuls de $\mathbf{F}_q$ et un automorphisme π du corps $\mathbf{F}_q$ tels que $C' = \Psi_{V,\pi,\sigma}(C)$.*

Dans le cas du corps à deux éléments, la définition se simplifie, puisqu'il ne reste plus que la permutation σ du support I_n. Dans le cas d'un corps premier (*i.e.* q est un nombre premier), l'automorphisme π disparaît.

Enfin, il faut noter que les définitions 3 et 6 sont incompatibles, en effet, on peut trouver des exemples de codes linéaires décomposables non équivalents mais néanmoins équivalents par isométrie.

1. MacWilliams suppose que la restriction de f à C est semi-linéaire, mais son résultat est vrai pour tout code.

2. il n'est pas somme directe de deux codes non triviaux à supports disjoints.

Chapitre 2

Détermination de l'équivalence entre deux codes

Il s'agit ici de répondre à la question de l'équivalence entre deux codes linéaires à l'aide d'un algorithme qui prendra en argument, par exemple, leurs matrices génératrices. Dans le cas de deux codes linéaires binaires, Petrank et Roth [PR97] ont précisé la classe de complexité du problème de décision correspondant (appelé « CODE EQUIVALENCE PROBLEM ») ; décider de l'équivalence entre deux codes linéaires binaires est au moins aussi difficile que de décider de l'existence d'un isomorphisme entre deux graphes, et n'est pas NP-complet si P $\neq$ NP. En clair, il faut s'attendre à ce qu'il existe des instances difficiles du problème. Par contre nous verrons dans le chapitre 3 que, comme pour l'isomorphisme de graphe, le problème peut être résolu en temps polynomial dans presque tous les cas.

2.1 Préambule

Lorsque deux codes ne sont pas équivalents, il peut exister des propriétés invariantes par isométrie permettant de les différencier ; un code à poids pair ne pourra jamais être équivalent à un code possédant des mots de poids impair. De même deux codes possédant des énumérateurs des poids différents ne pourront pas être équivalents. Par contre, ces propriétés invariantes, si elles sont identiques pour deux codes, ne permettent en général pas de conclure sur leur équivalence ; il existe de nombreux codes non équivalents et qui possèdent des énumérateurs des poids identiques. Il est même possible de trouver des codes binaires non équivalents et possédant le même ensemble d'énumérateurs de poids des cosets.

Si l'on dispose d'un invariant, il est possible de pousser le raisonnement un peu plus loin. Si deux codes de longueur n sont équivalents, que nous les poinçonnons en chaque position et que nous calculons pour chacun les n valeurs correspondantes de l'invariant, nous devons obtenir le même ensemble (et la même multiplicité lorsqu'une valeur est obtenue plusieurs fois). Il est intéressant de noter que si les ensembles sont identiques et contiennent n valeurs distinctes, alors la permutation de l'éventuelle isométrie semi-linéaire entre ces deux codes est entièrement déterminée. En fait, sauf dans le cas où les n valeurs de l'invariant sont identiques, nous obtenons toujours une information sur cette permutation.

Ceci nous a amené à définir la notion de signature. En général, une signature sera obtenue à partir d'un invariant. Cela va nous permettre de décrire un algorithme qui non seulement pourra décider, mais également expliciter l'équivalence entre deux codes.

Le problème difficile qui demeure est de trouver un invariant dont le calcul puisse se faire dans un temps raisonnable. Le plus naturel semble être d'utiliser l'énumérateur des poids, malheureusement son calcul nécessite un temps qui dépend exponentiellement de la dimension.

2.2 Invariants et signatures

La notion d'invariant que nous utiliserons ici sera liée à celle d'équivalence. Il s'agit de toute propriété d'un code qui ne changera pas lorsque l'on appliquera une permutation ou plus généralement une isométrie semi-linéaire.

Définition 7 *Un* invariant *est une fonction à valeurs dans un ensemble quelconque, prenant en argument un code et telle que deux codes équivalents par permutation prennent la même valeur.*

Sont des invariants, des paramètres comme la longueur, la dimension ou la distance minimale. Plus discriminant, nous avons des invariants tels que l'énumérateur des poids ou l'ensemble des énumérateurs des poids des cosets.

Définition 8 *Une* signature *est une fonction à valeurs dans un ensemble quelconque, prenant en argument un code et un élément du support de ce code, et telle que* $S(\sigma(C), \sigma(i)) = S(C, i)$ *pour tout code* C *de longueur* n*, tout* i *dans* I_n *et toute permutation* σ *de* I_n*.*

On peut définir de la même manière un *invariant général* et une *signature générale* à partir de l'équivalence par isométrie semi-linéaire.

Définition 9

1. *Un* invariant général *est une fonction à valeurs dans un ensemble quelconque, prenant en argument un code et telle que deux codes équivalents prennent la même valeur.*
2. *Une* signature générale *est une fonction à valeurs dans un ensemble quelconque, prenant en argument un code et un élément du support de ce code, et telle que* $S(\Psi_{V,\pi,\sigma}(C), \sigma(i)) = S(C, i)$ *pour tout code* C *de longueur* n*, tout* i *dans* I_n *et toute isométrie semi-linéaire* $\Psi_{V,\pi,\sigma}$ *(cf* (1.2)*).*

Définition 10 *Une signature est* discriminante *pour un code* C *donné de longueur* n *s'il existe* i *et* j *dans* I_n *tels que* $S(C, i) \neq S(C, j)$*.*

Une signature est totalement discriminante *pour un code* C *donné de longueur* n *si* $S(C, i) \neq S(C, j)$ *pour tout* i *et tout* j *distincts dans* I_n*.*

2.2.1 Exemples

La notion d'invariant est liée à celle d'équivalence (donc d'isométrie). Il s'agira donc la plupart du temps de propriétés métriques des codes : distance minimale, énumérateur des poids du code, énumérateurs des poids des cosets.

$$d(C) = \min(w_H(c) \mid c \in C)$$

$$W(C) = \sum_{c \in C} X^{w_H(c)}$$

$$CW(C) = \{\{W(u + C) \mid u \in \mathbf{F}_q^n\}\}$$

A - Invariants par permutation

Il existe un certain nombre d'invariants (par permutation) qui ne sont pas des invariants généraux, par exemple l'énumérateur des poids complets :

$$WW(C) = \sum_{(c_1,\ldots,c_n)\in C} \prod_{i=1}^{n} X_{c_i}$$

qui est un polynôme à q indéterminées X_α, $\alpha \in \mathbf{F}_q$. Plus intéressant, nous avons le cas des invariants du hull ($\mathcal{H}(C)$, intersection d'un code linéaire avec son dual). Pour tout invariant ν, l'application $C \mapsto \nu(\mathcal{H}(C))$ est un invariant. Par contre, sauf pour $q = 2$, $q = 3$ et $q = 4$ avec le produit scalaire hermitien, il est possible de trouver des codes équivalents ayant des hulls de dimension différentes.

B - Signatures

On peut construire une signature à partir de tout invariant. Soit ν un invariant, pour tout code C de longueur n, et pour tout $i \in I_n$ nous notons C_i le code obtenu en poinçonnant la i-ème position de C. L'application définie par $S(C, i) = \nu(C_i)$ est une signature. La difficulté sera de trouver des invariants suffisamment simples à calculer et qui produisent des signatures discriminantes.

Deux propriétés des signatures sont importantes, leur caractère discriminant et leur coût algorithmique. Les signatures les plus discriminantes sont malheureusement le plus souvent difficiles à calculer, par exemple l'énumérateur des poids d'un code.

2.3 Existence d'invariants et de signatures discriminants

Il existe un invariant trivial qui permet de déterminer à coup sûr si deux codes sont équivalents par permutation :

$$\begin{array}{rccc} \mathcal{V} & : & \mathcal{L} & \to & 2^{\mathcal{L}} \\ & & C & \mapsto & \{\sigma(C) \mid \sigma \in S_n\}, \end{array}$$

où $\mathcal{L}$ est l'ensemble des codes linéaires et $2^{\mathcal{L}}$ est l'ensemble des parties de $\mathcal{L}$. Par définition $\mathcal{V}(C) = \mathcal{V}(C')$ si et seulement si C et C' sont équivalents par permutation. Bien entendu, la complexité algorithmique du calcul de $\mathcal{V}(C)$ rend cet invariant inutilisable en pratique même pour des petites valeurs de n. Il ne peut pas en revanche exister de signature totalement discriminante pour tout code puisque dès que le groupe de permutations d'un code C contiendra un élément non trivial σ nous aurons $S(C, i) = S(C, \sigma(i))$ pour tout i. De manière générale, deux positions appartenant à la même orbite sous l'action du groupe de permutations ne pourront jamais être discriminées. Moyennant ces limitations, il existe une signature qui réalise la meilleure discrimination possible pour tout code.

Proposition 11 *Soit $\mathcal{M}$ la signature définie pour tout code C de longueur n et tout i dans I_n par*

$$\mathcal{M}(C, i) = \{(\sigma(C), \sigma(C_i)) \mid \sigma \in S_n\}.$$

Pour tout code C de longueur n, on a :

1. *Deux éléments i et j de I_n vérifient $\mathcal{M}(C,i) = \mathcal{M}(C,j)$ si et seulement si ils appartiennent à la même orbite sous l'action du groupe de permutations de C.*
2. *Toute signature S vérifie :*

$$\forall i,j \in I_n, \quad \mathcal{M}(C,i) = \mathcal{M}(C,j) \Rightarrow S(C,i) = S(C,j).$$

Le preuve de ce résultat peut être trouvée dans [Sen00]. Le résultat peut être adapté sans trop de difficultés aux signatures générales.

L'énumérateur des poids, bien qu'il puisse prendre la même valeur pour des codes non équivalents, permet de construire des signatures qui sont souvent très discriminantes voire totalement discriminantes. Par exemple pour des codes binaires aléatoires de longueur 40 et de dimension 20, il est difficile de trouver des exemples dans lesquels deux codes poinçonnés ont le même énumérateur. Le calcul de l'énumérateur des poids est un problème NP-dur, exponentiel en pratique, mais reste accessible pour des codes de petite dimension ou codimension (jusqu'à 40 dans le cas binaire). Pour des codes de taille importante, il faudra trouver des invariants plus faciles à calculer et qui produisent néanmoins des signatures discriminantes. Les invariants construits à partir du hull, utilisés dans l'algorithme de séparation du support, possèdent de telles caractéristiques pour la plupart des codes.

Chapitre 3

L'algorithme de séparation du support

Le résultat de Petrank et Roth [PR97] permet de supposer raisonnablement que décider si deux codes linéaires binaires sont équivalents est difficile dans le pire cas. Nous présentons dans ce chapitre un algorithme permettant de résoudre ce problème, dont la complexité dépend polynomiale-ment de la longueur et de la dimension et exponentiellement de la dimension du hull. Nous verrons au chapitre 4 que le hull d'un code linéaire est en moyenne égal à une petite constante lorsque sa longueur tend vers l'infini. L'algorithme de séparation du support [Sen97a, Sen00] peut être appliqué à la cryptanalyse du système de McEliece (§6.2 et [LS01]) ainsi qu'au calcul du groupe d'automorphisme d'un code [SS99, SS01].

3.1 Calcul de la permutation entre deux codes équivalents

L'algorithme présenté ici permet de déterminer si deux codes donnés en arguments sont équivalents par permutation, et, dans l'affirmative de déterminer la permutation entre leurs supports. Supposons que nous disposons d'une signature totalement discriminante S pour un code C de longueur n. La procédure suivante permet de déterminer la permutation entre C et C'.

```
input C, C'
    for i in I_n do T[S(C,i)] ← i
    for i in I_n do σ[i] ← T[S(C',i)]
output σ
```

La notation $T[S(C,i)]$ implique l'utilisation d'une table de hachage. Il s'agit en outre d'une version simplifiée qui ne fonctionne que si les codes sont bien équivalents par permutation. Il faudrait ajouter des tests un peu partout pour inclure le cas où C et C' ne sont pas équivalents.

La difficulté essentielle de l'algorithme présenté dans [Sen00] a été de construire pour un code donné une signature totalement discriminante qui puisse être calculée en temps polynomial. La signature utilisée est la suivante :

$$(C,i) \mapsto \Big(\mathcal{W}(\mathcal{H}(C_i)), \mathcal{W}(\mathcal{H}(C_i^{\perp}))\Big)$$

où $\mathcal{W}(C)$ est l'énumérateur des poids, $\mathcal{H}(C)$ le hull et C_i le code obtenu en poinçonnant la i-ème coordonnée de C. Cette signature possède des propriétés remarquables, d'une part elle a

une complexité algorithmique raisonnable (polynomiale) car le hull de presque tous les codes linéaires est de petite dimension (cf. chapitre 4 et [Sen98]), et de plus elle est discriminante, c'est-à-dire que les énumérateurs des poids calculés prendront souvent des valeurs différentes (cf. [Sen00, §V])

Cette signature n'est en général pas totalement discriminante, mais elle permet de partitionner le support du code en un nombre raisonnable de cellules, chacune étant caractérisée par une certaine valeur de la signature. La signature est ensuite calculée pour chaque position mais en considérant le code C_J (code poinçonné en J) où J est l'une des cellules de la partition, si possible de petit cardinal.

Bien qu'il soit difficile d'en apporter une preuve, l'algorithme de séparation du support permet toujours en pratique d'obtenir la plus petite partition possible du support, c'est-à-dire les orbites sous l'action du groupe de permutations du code.

3.2 Calcul du groupe de permutations

Il est possible à l'aide de l'algorithme de séparation du support de calculer le groupe de permutations d'un code pour des paramètres qui n'étaient jusqu'alors pas accessibles. L'algorithme de Leon [Leo82], le seul connu, nécessite le pré-calcul d'un ensemble W de mots du code globalement invariants par permutation (tous les mots de poids minimum du code, ou bien tous les mots du code jusqu'à un certain poids, le hull, ...). Cet ensemble devra être suffisamment grand pour être significatif (c'est le groupe de permutations de cet ensemble qui est en fait calculé), suffisamment petit pour pouvoir être manipulé et, bien sûr, suffisamment facile à calculer.

La primitive de base de l'algorithme de Leon consiste à placer l'ensemble W dans une table, les lignes et les colonnes sont partitionnées en fonction de leur poids, et le processus est répété en tenant compte à chaque fois de la partition obtenue au rang précédent. On doit obtenir ainsi les orbites sous l'action du groupe de permutations, ou quelque chose qui s'en approche.

Cette primitive peut être remplacée par l'algorithme de séparation du support (la mise en œuvre nécessite en pratique des changements beaucoup plus importants). De cette manière, il est possible de calculer le groupe de permutations de tout code dont la dimension serait importante mais possédant un hull de petite taille[1].

Par exemple, il est possible de calculer le groupe de permutations de certains codes résidus quadratiques pour des dimensions allant jusqu'à plusieurs milliers [SS99, Ske99, SS01].

3.3 Applications à la cryptanalyse

L'algorithme de séparation du support permet d'obtenir la meilleure cryptanalyse structurelle connue des systèmes de McEliece et de Niederreiter (cf. §6.2). Il restreint également beaucoup l'espace des clés d'un protocole d'identification proposé par Marc Girault [Gir90] et qui repose, entre autres, sur la difficulté de déterminer l'équivalence entre deux codes linéaires. En pratique il faudra utiliser des codes ayant un hull de dimension élevée afin de se placer dans le cas où l'algorithme $\mathcal{SSA}$ est impraticable. De tels codes sont facile à générer et le protocole n'empêche en rien leur utilisation.

1. Si le hull est trop petit, par exemple $\{0\}$, il ne peut pas convenir à l'algorithme de Leon alors qu'il ne constitue pas, au contraire, un handicap pour $\mathcal{SSA}$.

Chapitre 4

Hull des codes linéaires

La recherche d'invariants dont le calcul est de faible complexité algorithmique est l'un des points clé de l'algorithme de séparation du support. Un premier algorithme pour reconstituer la permutation entre deux codes équivalents par permutation utilisant l'énumérateur des poids d'un code avait été mis au point pour l'une des étapes de la reconstruction d'un code concaténé [Sen94, Sen98]. À partir de cette première ébauche et grâce aux suggestions de Ed Assmus, nous nous sommes rapidement intéressé au hull et à ses invariants. Il est apparu lors des premières simulations que cet objet répondait à tous les critères car il permettait de construire des signatures à la fois discriminantes et de faible complexité algorithmique. Rien n'étant connu sur le hull, il nous a fallu étudier ses principales propriétés (cf. [Sen97b] et [Sen00]).

La notion de hull d'un code linéaire a été introduite par Assmus et Key [AK90, AK92].

Définition 12 *Le* hull *d'un code linéaire est égal à son intersection avec son code dual. Nous noterons* $\mathcal{H}(C) = C \cap C^{\perp}$.

4.1 La dimension du hull

Grâce à des formules d'inversion appropriées [Com74] [1] et aux travaux de dénombrement des codes faiblement auto-duaux de Vera Pless [Ple65], il a été possible de démontrer le résultat qui suit ainsi que son corollaire.

Proposition 13 *Soit* $A_{n,k,l}$ *le nombre de codes linéaires* q*-aires de longueur* n *et de dimension* k *dont le hull est de dimension* l*. Nous avons :*

$$R_l = \lim_{n,k\to\infty} \frac{A_{n,k,l}}{\begin{bmatrix} n \\ k \end{bmatrix}} = \frac{R_0}{(q-1)(q^2-1)\cdots(q^l-1)}$$

où $R_0 = \prod_{i\geq 1}(1+q^{-i})$ *et* $\begin{bmatrix} n \\ k \end{bmatrix} = \prod_{i=0}^{k-1}(q^{n-i}-1)/(q^{k-i}-1)$ *est le coefficient binomial gaussien, correspondant au le nombre de sous-espaces de dimension* k *d'un espace vectoriel de dimension* n *sur* $\mathbf{F}_q$.

Corollaire 14 *La dimension moyenne du hull d'un code linéaire* q*-aire est asymptotiquement égale à :*

$$\sum_{l\geq 1} lR_l = \sum_{i\geq 1} \frac{1}{q^i+1}.$$

1. Merci à Philippe Flajolet pour ses précieuses indications

Il est remarquable que la proportion de codes ayant un hull de dimension donnée ne dépende ni de la longueur ni de la dimension des codes considérés, mais uniquement de la taille de l'alphabet. La table 4.1 donne une idée des dimensions auxquelles il faut s'attendre selon la taille de l'alphabet.

q	R_0	R_1	R_2	R_3	moyenne
2	0.4194	0.4194	0.1398	0.0200	0.7645
3	0.6390	0.3195	0.0399	$1.5\,10^{-3}$	0.4041
4	0.7375	0.2458	0.0164	$2.6\,10^{-4}$	0.2794
16	0.9373	0.0628	$2.5\,10^{-4}$	$6.0\,10^{-8}$	0.0630
256	0.9961	0.0039	$6.0\,10^{-8}$	$3.6\,10^{-15}$	0.0039

TAB. 4.1: *Hull des codes linéaires q-aires*

4.2 Les signatures basées sur le hull

Une autre question d'importance, maintenant que nous savons que le hull sera facile à énumérer pour la plupart des codes, est de savoir si les signatures que nous pourrons construire seront suffisamment discriminantes. Rien n'autorise à penser *a priori* que le hull d'un code poinçonné en une position varie de façon significative pour chaque position.

La signature que nous utilisons dans le cas binaire est

$$(C, i) \mapsto \Big(\mathcal{W}(\mathcal{H}(C_i)), \mathcal{W}(\mathcal{H}(C_i^{\perp}))\Big)$$

où $\mathcal{W}(C)$ est l'énumérateur des poids, $\mathcal{H}(C)$ le hull et C_i le code obtenu en poinçonnant la i-ème coordonnée de C. Nous montrons dans [Sen00] que cet invariant est discriminant, et également que les codes $\mathcal{H}(C_i)$ et $\mathcal{H}(C_i^{\perp})$ peuvent être obtenus pour tout i à l'aide d'une seule élimination de Gauss de la matrice

$$M = \left(\begin{array}{c|c} Id & R \\ \hline {}^tR & -Id \end{array}\right)$$

où $G = (Id \mid R)$ est une matrice génératrice de C (Id représentant la matrice identité). Ce résultat est généralisé dans [Sen00] au cas q-aire tant d'un point de vue algébrique que d'un point de vue algorithmique.

Deuxième partie

Cryptosystèmes basés sur les codes correcteurs d'erreurs

Chapitre 5

Le cryptosystème de McEliece et ses variantes

L'existence de systèmes de chiffrement à clé publique est liée à celle de *fonctions à sens unique à trappe* ; il s'agit d'une fonction, qui peut être publique, dont le calcul est facile mais dont l'inverse est difficile à calculer, à moins de posséder un secret supplémentaire (la trappe). Les codes correcteurs d'erreurs peuvent être utilisés pour la réalisation de telles fonctions, il est en effet très facile de produire une instance d'un problème de décodage (on choisit un mot dans un code et on modifie certaines de ses coordonnées) et difficile en général de réaliser le décodage de cette instance, à moins de connaître une structure algébrique masquée, qui constitue la trappe et permet le décodage. Il faudra ainsi d'une part choisir des paramètres suffisants pour que le problème général du décodage soit difficile, et d'autre part masquer la structure algébrique du code, ce qui se fera tout simplement en permutant ses coordonnées.

Suivant de peu la publication en 1978 par Rivest, Shamir et Adelman du premier système de chiffrement asymétrique [RSA78] dont la sécurité est liée à la difficulté de la factorisation des entiers, Robert McEliece a proposé un système [McE78] lié à la difficulté du décodage d'un code linéaire. Dans ce système, le texte clair est un mot de code et le texte chiffré est un mot de l'espace de Hamming obtenu en modifiant quelques coordonnées. En 1986, Harald Niederreiter en a proposé une variante [Nie86] de sécurité équivalente [LDW94], dans laquelle le texte clair est un motif d'erreur corrigible (c'est-à-dire un mot de l'espace de Hamming de poids inférieur à la moitié de la distance minimale du code) et le texte chiffré est le syndrome correspondant à ce motif. Dans ces deux systèmes les positions d'un code linéaire constituant la clé secrète (McEliece utilise les codes de Goppa irréductibles) sont permutées aléatoirement ; le code équivalent ainsi obtenu est rendu public (sous la forme d'une matrice génératrice ou de parité). Pour casser le système, la première approche consiste à décoder dans le code public, elle a été examinée par de nombreux auteurs [LB88, Leo88, Ste89, Can95, CC98] et est liée au problème difficile de la recherche de mots de petit poids dans un code linéaire [BMvT78]. L'autre approche consister à essayer de retrouver le code secret à partir du code public, ou du moins, à déduire suffisamment d'information pour obtenir un algorithme de décodage en temps polynomial du code public. De telles attaques ont été couronnées de succès pour les codes de Reed-Solomon généralisés [RS85, SS92] et pour les codes concaténés [Sen98]. Pour les codes de Goppa, aucun résultat général n'est connu à ce jour, il est toutefois possible de détecter et de reconstruire un code de Goppa si son polynôme générateur est à coefficients binaires [LS01].

Un système utilisant les codes de Reed-Muller a été proposé par Sidel'nikov en 1994 [Sid94]. Ce système essaie de tirer avantage d'un algorithme de décodage algébrique décodant au delà de la moitié de la distance minimale [SP92]. Bien que le « code secret » soit unique (ce qui le rend moins secret !), la cryptanalyse structurelle échoue (cf. §6.2) car les codes de Reed-Muller sont faiblement auto-duaux et déterminer la permutation entre un tel code et le code public constitue précisément une instance (supposée) difficile du problème de l'équivalence de codes.

Signalons enfin qu'il existe d'autres systèmes de chiffrement très voisins, dont nous ne parlerons pas ici, reposant sur des techniques similaires, par exemple, une solution utilisant la métrique du rang plutôt que la métrique de Hamming a été proposée par Gabidulin, Paramonov et Tretjakov [GPT91].

5.1 Définitions

Soit C un code linéaire q-aire t-correcteur de longueur n de dimension k. Nous noterons $CW_q(n,t)$ l'ensemble des mots de $\mathbf{F}_q^n$ de poids de Hamming t. Soient G et H respectivement une matrice génératrice et une matrice de parité de C.

	McEliece	**Niederreiter**
clé publique :	G	H
texte clair :	$x \in \mathbf{F}_q^k$	$x \in CW_q(n,t)$
cryptogramme :	$y = xG + e,\ w_H(e) = t$	$y = Hx^T$

L'erreur e de poids t sera choisie aléatoirement au moment du chiffrement. Dans les deux cas la *clé secrète* sera un algorithme de décodage corrigeant t erreurs dans C et constitue la *trappe* du système.

En pratique un *code secret* C_0 (de matrice génératrice G_0) pour lequel une procédure de décodage rapide est connue (un code alternant par exemple) est choisi. La structure de ce code sera masquée à l'aide d'une permutation du support et finalement une matrice génératrice quelconque de ce code permutée sera utilisée comme clé publique :

$$G = UG_0P, \text{ où } \begin{cases} U \text{ est inversible} \\ P \text{ est une matrice de permutation} \end{cases}$$

Dans le cas du système de Niederreiter, on partira d'une matrice de parité H_0 de C_0 et la clé publique sera :

$$H = UH_0P, \text{ où } \begin{cases} U \text{ est inversible} \\ P \text{ est une matrice de permutation} \end{cases}$$

Bien entendu, pour garantir la sécurité du système, la famille de codes dans laquelle le code secret sera tiré doit être choisie avec soin et les paramètres des codes devront être suffisamment importants (cf. §6.1).

5.1.1 Un système hybride

Une question se pose très naturellement lorsque l'on observe le système de McEliece : que se passe-t-il si l'on décide de mettre de l'information dans l'erreur ? Ceci permettrait d'augmenter

significativement le taux de transmission sans rendre la mise en œuvre significativement plus complexe.

Soit G la matrice génératrice d'un code linéaire q-aire t-correcteur de longueur n et de dimension k. Dans le cryptosystème hybride de clé publique G, le texte clair (x, e) est choisi dans $\mathbf{F}_q^k \times CW_q(n,t)$ (rappelons que $CW_q(n,t)$ désigne l'ensemble des mots q-aire de longueur n et de poids t) et le texte chiffré correspondant est égal à $y = xG + e \in \mathbf{F}_q^n$.

Dans le cas où le texte clair est choisi uniformément dans $\mathbf{F}_q^k \times CW_q(n,t)$, le coût moyen d'une attaque de ce système sera égal à

$$\chi_{Hyb}(G) = \sum_{e \in CW_q(n,t), x \in \mathbf{F}_q^k} \frac{\chi(G, xG+e)}{M} = \sum_{y \in \mathbf{F}_q^n, d_H(y,C)=t} \frac{\chi(G,y)}{M}.$$

où $M = q^k \binom{n}{t}(q-1)^t$ et $\chi(G,y)$ est le coût pour décoder un vecteur y de $\mathbf{F}_q^n$ dans le code défini par G.

Le même calcul pour une mise en œuvre du cryptosystème de McEliece dans lequel la loi d'émission est uniforme et l'erreur e est choisie uniformément dans $CW_q(n,t)$ donne un coût moyen identique.

$$\chi_{McE}(G) = \sum_{x \in \mathbf{F}_q^k} \frac{1}{q^k} \sum_{e \in CW_q(n,t)} \frac{1}{\binom{n}{t}(q-1)^t} \chi(G, xG+e) = \sum_{y \in \mathbf{F}_q^n, d_H(y,C)=t} \frac{\chi(G,y)}{M}.$$

Donc en faisant en sorte lors de sa mise en œuvre que la loi d'émission soit uniforme, en compressant les données par exemple, le système hybride n'est pas moins sûr que le système de McEliece possédant la même clé publique.

5.2 Mise en œuvre

En pratique, l'utilisation de codes de Goppa binaires comme clé secrète (c'est la famille initialement proposée par McEliece) semble constituer un choix judicieux. En effet ils existent en grand nombre, pour une grande variété de paramètres, possèdent une bonne capacité de correction et un algorithme de décodage très efficace.

L'un des plus gros inconvénients de ces systèmes est la grande taille de la clé publique. Cette taille peut être réduite en utilisant des matrices sous forme systématique. Cette opération est sans danger dans la variante de Niederreiter mais doit s'accompagner de précaution dans le cas du système de McEliece (cf. §7.2). Dans toute la suite nous considérerons des matrices publiques sous forme systématique.

Dans la suite de ce chapitre nous considérons un code de Goppa binaire dont le support est égal à $\mathbf{F}_{2^m}$ et dont le générateur est un polynôme de degré t, irréductible dans $\mathbf{F}_{2^m}[z]$. La longueur de ce code est $n = 2^m$ et sa dimension $k = n - tm$. Nous noterons $R = k/n$ le taux de transmission.

5.2.1 Chiffrement

Dans la variante de McEliece, le chiffrement consiste à multiplier un vecteur de $\mathbf{F}_2^k$ par une matrice binaire $k \times (n-k)$, puis à modifier t bits choisis au hasard dans le mot de code correspondant. Bien que cela puisse constituer une difficulté dans la mise en œuvre effective, le coût de l'ajout de l'erreur peut être négligé. En nombre d'opérations binaires, la complexité

du chiffrement sera de $\approx \frac{1}{2}k(n-k)$. Dans la variante de Niederreiter nous devons calculer le syndrome correspondant à un mot de poids t c'est-à-dire faire la somme de t colonnes de H. En moyenne, seulement Rt de ces colonnes seront effectivement additionnées, les autres seront dans la partie « identité » de la matrice H et auront pour seul effet de modifier un bit dans le résultat, soit environ $Rt(n-k) + (1-R)t \approx Rt(n-k)$ opérations binaires.

5.2.2 Déchiffrement

Le décodage d'un code de Goppa binaire comporte trois étapes :

1. le calcul du syndrome,
2. la résolution de l'équation clé pour obtenir le polynôme localisateur,
3. la recherche des racines du polynôme localisateur.

La complexité de l'algorithme de Patterson est détaillée dans la section 7.1. Nous obtenons en nombre d'opérations binaires :

1. pour le calcul du syndrome : $tmn/2$ (McEliece) et $t^2m^2/2$ (Niederreiter),
2. pour le calcul du polynôme localisateur : $6t^2m$,
3. pour le calcul des racines : $2t^2m^2$,

Ces chiffres tiennent compte de pré-calculs pour accélérer les calculs dans le corps $\mathbf{F}_{2^m}$ et celui du syndrome.

	McEliece	**Niederreiter**
Taille en bits du texte chiffré	n	tm
Nombre de bits d'information	$n - tm$	$\log_2 \binom{n}{t} \approx tm - t\log_2(t/\mathrm{e})$
Taux d'information	$R = 1 - \dfrac{tm}{n}$	$\tau \approx 1 - \dfrac{\log_2(t/\mathrm{e})}{m}$
Coût du chiffrement (par bloc)	$\dfrac{R}{2}tmn$	Rt^2m
Coût du chiffrement (par bit d'information)	$\dfrac{1}{2}tm$	$\dfrac{R}{\tau}t$
Coût du déchiffrement (par bloc)	$c(tm)^2 + Rtmn$	$(c+1)\,(tm)^2$
Coût du déchiffrement (par bit d'information)	$\left(\dfrac{c(1-R)}{R} + 1\right) tm$	$\dfrac{c+1}{\tau}tm$
Taille de la clé publique	$tm(n-tm)$	$tm(n-tm)$

La constante c est de l'ordre de 2. Les valeurs utiles de R sont comprises entre $1 - \mathrm{e}^{-1}$ et 1 (cf. remarque 1). Les valeurs de τ correspondantes sont comprises entre $\log_2(m\mathrm{e}^2)/m$ et 1.

Chapitre 6

Cryptanalyse du système de McEliece

La sécurité des deux variantes est identique (pour un code donné) et repose sur l'impossibilité de déduire un algorithme de décodage efficace de la seule clé publique. Si tel est le cas, le décryptage d'une instance du système se ramènera à la résolution d'une instance d'un problème de décodage dans un code linéaire. Il en découle que toute attaque du système devra appartenir à l'une des catégories suivantes :

- les attaques par décodage : il s'agit de décoder une instance du cryptosystème à l'aide d'un algorithme général (i.e. le code public est considéré comme un code linéaire aléatoire).
- les attaques structurelles : il s'agit, à l'aide de la clé publique, de retrouver tout ou partie de la structure du code secret.

6.1 Cryptanalyse par décodage

6.1.1 Coût

Si l'on décode par ensemble d'information, une erreur sera décodée si son support est en totalité hors de l'ensemble d'information. La probabilité pour qu'une erreur de poids t soit décodable est donc $P_{dec} = \binom{n-k}{t}/\binom{n}{t}$. En utilisant une l'estimation suivante des coefficients binomiaux

$$\frac{2^{nH_2(t/n)}}{\sqrt{8t(1-t/n)}} \leq \binom{n}{t} \leq \frac{2^{nH_2(t/n)}}{\sqrt{2\pi t(1-t/n)}},$$

où $H_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$, nous obtenons

$$P_{dec} = O(1) \cdot 2^{-n(H_2(t/n)-(1-k/n)H_2(t/(n-k)))}$$

Pour un code de Goppa, nous avons $n = 2^m$ et $k = n - tm$. On pose $R = k/n$, et on considère une famille de Goppa de taux de transmission R constant. Le développement en série de l'exposant dans l'expression ci-dessus donne $-n\left(-(1-R)\log_2(1-R)/m + O(1/m^2)\right)$.

Finalement, si l'on considère l'algorithme consistant à tirer au hasard des ensembles de positions d'information jusqu'à en obtenir un convenable, nous devrons effectuer un pivot de

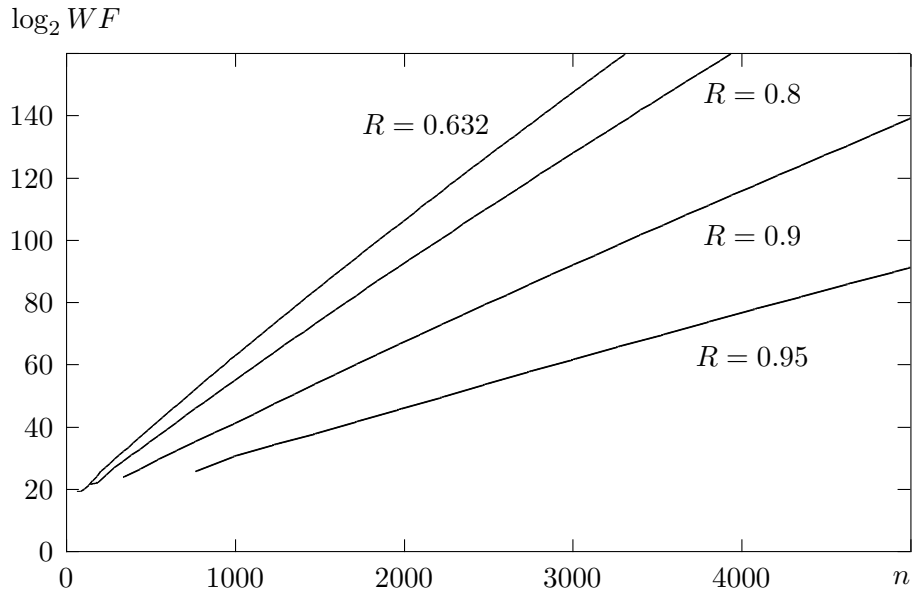

FIG. 6.1: *Coût de l'attaque de Canteaut-Chabaud pour les codes de Goppa (logarithme en base 2 du facteur de travail en fonction de la longueur pour diverses valeurs du taux de transmission)*

Gauss (de complexité $O(n^3)$) à chaque itération. La complexité totale devient pour un taux de transmission donné :

$$P(n) \exp\left(c \frac{n}{\log_2 n}(1 + o(1))\right)$$

où $c = -(1 - R)\ln(1 - R)$ et $P(n)$ est un polynôme de degré 3. On peut remplacer le terme polynomial par un polynôme de degré 1 en testant si l'ensemble de positions d'information retenu contient une ou deux erreurs [LB88], on peut même le baisser encore, probablement jusqu'à une constante en utilisant des algorithmes de recherche plus complexes [Ste89, Leo88, vT90, Can96, CC98].

Remarque 1 *Le coût maximal du décodage par ensemble d'information est atteint pour un taux de transmission $R \approx 1 - \mathrm{e}^{-1} = 0.632$. Il n'y a donc pas grand intérêt à choisir des taux inférieurs pour les sytèmes de McEliece et Niederreiter car les coûts du chiffrement, du déchiffrement et la taille de la clé publique augmentent alors que le taux d'information et la sécurité diminuent.*

6.1.2 Choix des paramètres

La meilleure attaque connue pour les paramètres d'origine de McEliece ($n = 1024$ et $t = 50$) permet de résoudre une instance du cryptosystème à un coût d'environ 2^{64} opérations binaires [Can96, CS98]. Pour cette même longueur, il est possible d'arriver à un facteur de travail binaire de 2^{66} en choisissant $t = 39$ comme degré du polynôme de Goppa. Comme

l'on peut estimer à 2^{60} le nombre d'opérations binaires effectuées aujourd'hui par un PC fonctionnant à 1 Ghz [1], il est clair que ces valeurs sont insuffisantes.

Le facteur de travail binaire minimal aujourd'hui devra donc être de l'ordre de 2^{80}. Cette valeur est dépassée par l'algorithme de Canteaut-Chabaud pour un code de Goppa de longueur $n = 2048$ corrigeant $t = 26$ erreurs. Pour cette même longueur, le maximum est atteint pour $t = 70$, correspondant à un facteur de travail de 2^{108}. Enfin, signalons qu'avec un code de Goppa raccourci de longueur $n = 1401$ corrigeant $t = 47$ erreurs nous atteignons un facteur de travail de 2^{80}.

A - Et la loi de Moore?

La loi de Moore prévoit (et elle est vérifiée depuis plus de 20 ans) que la capacité de calcul et la mémoire des ordinateurs sera multipliée par 2 tous les 18 mois. Donc, dans la mesure où la loi continue de se vérifier et en l'absence de progrès algorithmique ou technologique significatif, les codes de Goppa de longueur 2048 pourront être utilisés pour encore une quarantaine d'années. Au-delà, il apparaît clairement d'après la Figure 6.2 [2] que le système pourra sans difficulté augmenter en sécurité. Ce comportement extrêmement avantageux est lié à la complexité exponentielle de tous les algorithmes de décodage généraux connus [3].

6.2 Cryptanalyse structurelle

6.2.1 Attaques algébriques

Plusieurs classes de codes sont à écarter pour l'usage dans les cryptosystèmes de type McEliece.

A - Codes GRS

La structure d'un code de Reed-Solomon généralisé dont le support a été permuté peut être facilement retrouvée en résolvant un système linéaire (l'attaque est décrite en détail dans [SS92] dans le contexte de la cryptographie mais certaines des propriétés permettant de faire aboutir l'attaque étaient déjà connues [RS85]).

L'attaque repose sur la propriété suivante qui relie la matrice de parité d'un code GRS à

1. Cette valeur est à considérer avec précaution si on la compare aux 2^{64} opérations binaires nécessaires pour casser McEliece. En pratique une implémentation va demander de l'ordre de quelques siècles de temps de calcul et non simplement 16 ans. Les facteurs de travail, et plus généralement les quantités d'opérations binaires, que nous indiquerons tout au long de ce travail devront être compris comme une mesure de la complexité. Les temps de calcul effectifs dépendront de chaque implémentation particulière et devront prendre en compte des opérations comme la gestion de la mémoire ou le dialogue entre les différents sous-processus du calcul dont le coût est difficile à évaluer *a priori*.

2. Les courbes des figures 6.1 et 6.2 ont pu être obtenues grâce à un programme aimablement fourni par Anne Canteaut.

3. Contrairement par exemple à RSA dont la meilleure cryptanalyse est sous-exponentielle ($\exp(cn^{1/3}(\log n)^{2/3})$, où n est la taille d'un bloc).

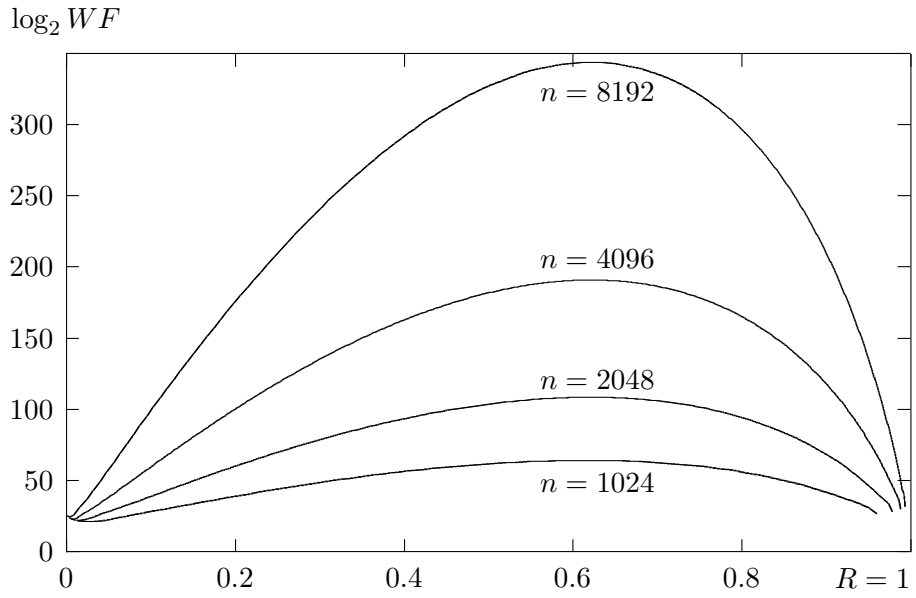

FIG. 6.2: *Coût de l'attaque de Canteaut-Chabaud pour les codes de Goppa (logarithme en base 2 du facteur de travail en fonction du taux de transmission pour diverses valeurs de la longueur)*

sa forme systématique.

Proposition 15 *Pour tous entiers $r \leq n \leq q$, la matrice*

$$H = \begin{pmatrix} y_1 & \cdots & y_n \\ y_1\alpha_1 & \cdots & y_n\alpha_n \\ \vdots & \ddots & \vdots \\ y_1\alpha_1^{r-1} & \cdots & y_n\alpha_n^{r-1} \end{pmatrix}$$

où les α_j, $1 \leq j \leq n$, sont des éléments distincts de $\mathbf{F}_q$ et les y_j, $1 \leq j \leq n$, sont des éléments non nuls de $\mathbf{F}_q$ admet pour forme systématique la matrice

$$\left(\begin{array}{ccc|ccc} 1 & & & R_{1,r+1} & \cdots & R_{1,n} \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & R_{r,r+1} & \cdots & R_{r,n} \end{array}\right) \tag{6.1}$$

dont le terme général en i-ème ligne, $1 \leq i \leq r$, et j-ème colonne, $r < j \leq n$, vaut

$$R_{i,j} = \frac{y_j}{y_i} \prod_{l=1, l\neq i}^{r} \frac{\alpha_j - \alpha_l}{\alpha_i - \alpha_l}.$$

L'unicité de la forme systématique permet d'écrire, après réduction de la clé publique par un pivot de Gauss, un système d'équations dont les inconnues sont les y_i et les α_i. Les symétries que l'on constate dans l'expression des $R_{i,j}$ permettent de rendre ce système linéaire.

B - Codes concaténés

Utiliser des codes concaténés ou des codes produits dans un système de chiffrement de type McEliece peut paraître avantageux car, d'une part ils possèdent des algorithmes de décodage très rapides et d'autre part ils permettent de décoder « en pratique » (c'est-à-dire avec une probabilité très proche de 1) des motifs d'erreurs de poids nettement supérieur à la moitié de leur distance minimale (cf. [Sen91]). Malheureusement, nous avons pu exhiber une attaque structurelle [Sen94, Sen98] qui rend leur usage périlleux. Cette attaque repose sur l'existence de nombreux mots de très petit poids dans le code dual du code concaténé considéré. La recherche de ces mots permet de retrouver une structure « par bloc » dans la matrice génératrice du code public :

$$\left(\begin{array}{ccc|ccc} G_{1,1} & & 0 & G_{1,K+1} & \cdots & G_{1,N} \\ & \ddots & & \vdots & \ddots & \vdots \\ 0 & & G_{K,K} & G_{K,K+1} & \cdots & G_{K,N} \end{array}\right) \tag{6.2}$$

où N et K sont la longueur et la dimension du code externe et chaque $G_{i,j}$ est une matrice génératrice d'un code (n, k) équivalent au code interne. Dans une seconde étape chaque bloc de n colonnes est réordonné à l'aide d'une version simplifiée de l'algorithme de séparation du support (cf. Chapitre 3). Finalement le calcul du polynôme minimal de chaque $G_{i,j}$ permet de retrouver à une transformation monomiale près la matrice génératrice du code externe.

L'algorithme complet permet de retrouver en temps polynomial une structure concaténée équivalente à celle de la structure initiale. Ceci n'est pas encore suffisant pour obtenir un décodeur mais rend caduc l'utilisation des codes concaténés ; en effet une attaque, structurelle ou par décodage, sur chacun des codes constituants devient suffisante.

Codes produits. Cette même attaque peut très probablement être généralisée aux codes produits, en effet, ces codes possèdent exactement la même faiblesse, à savoir un grand nombre de mots de très petit poids dans le dual. Cette propriété révèle beaucoup trop d'information sur la structure interne du code.

Code concaténés généralisés. Dans le cas des codes concaténés généralisés, il est possible de construire des codes dont la distance dualle est élevée. Ceci rend *a priori* l'attaque ci-dessus inefficace. La construction de tels codes est possible, par contre ils seront, d'une certaine manière, dégénérés et ne permettront peut-être pas d'obtenir le gain espéré en performance. Il convient toutefois de noter que certains des meilleurs codes connus sont des codes concaténés généralisés. La question de leur usage en cryptographie reste à mon avis un problème ouvert.

6.2.2 Attaques exhaustives

On peut ranger dans cette catégorie toutes les attaques qui vont parcourir tout ou partie de l'espace des clés. Rappelons pour commencer quelques faits sur la clé secrète d'une instance du cryptosystème de McEliece. Elle est constituée de trois éléments :

- un code C_0 appartenant une famille $\mathcal{F}$ de codes linéaires, on notera G_0 une de ses matrices génératrices,
- une matrice de permutation P (de même taille que la longueur du code),

– une matrice inversible U (de même taille que la dimension du code).

La clé publique sera la matrice $G = UG_0P$.

1. Tout d'abord la matrice U n'a aucun rôle combinatoire sur l'espace des clés, en effet on peut par exemple décider de n'opérer les attaques que sur une forme systématique de G. Il est vain d'espérer que la taille de l'espace des clés sera multipliée par le nombre de matrices inversibles. Le rôle de U est toutefois essentiel pour masquer une éventuelle structure de G_0. En effet, si la permutation P mélange bien les colonnes, elle a très peu d'impact sur les lignes. Dans le cas par exemple d'un code de Goppa et si la clé publique est une matrice de parité, il serait possible de se ramener immédiatement à une attaque structurelle du sur-code (qui est un GRS et est donc vulnérable).

2. Ensuite, si l'on parcourt l'ensemble des matrices de permutations que l'on multiplie à gauche par la clé publique, on finira par retomber sur le code secret (en fait la matrice UG_0). En pratique pour la plupart des classes de code on pourra détecter ce fait sans difficulté et en déduire le code secret. Keith Gibson a mis en œuvre une telle attaque pour les codes de Goppa [Gib91].

3. Enfin, si l'on parcourt la famille des codes secrets, l'algorithme de séparation du support (cf. 3) fournira dans la plupart des cas un bon test d'arrêt avec en prime la permutation lorsque ce test est positif.

A - Codes de Goppa

Définition 16 *Soit $L = (\alpha_1, \ldots, \alpha_n)$ une suite de n éléments distincts de $\mathbf{F}_{2^m}$ et $g(z) \in \mathbf{F}_{2^m}[z]$ un polynôme unitaire de degré t, irréductible dans $\mathbf{F}_{2^m}[z]$. Le code de Goppa binaire irréductible de support L et de générateur $g(z)$, noté $\Gamma(L, g)$, est l'ensemble des mots $(a_1, \ldots, a_n) \in \mathbf{F}_2^n$ tels que*

$$R_a(z) = \sum_{i=1}^{n} \frac{a_i}{z - \alpha_i} = 0 \bmod g(z). \tag{6.3}$$

Il existe d'autres caractérisations équivalentes à (6.3) d'un code de Goppa binaire,

– par sa matrice de parité:

$$Ha^T = 0 \text{ avec } H = \begin{pmatrix} 1 & \cdots & 1 \\ \alpha_1 & \cdots & \alpha_n \\ \vdots & & \vdots \\ \alpha_1^{t-1} & \cdots & \alpha_n^{t-1} \end{pmatrix} \begin{pmatrix} g(\alpha_1)^{-1} & & \\ & \ddots & \\ & & g(\alpha_n)^{-1} \end{pmatrix}, \tag{6.4}$$

– par le polynôme localisateur, $\sigma_a(z) = \prod_{i=1}^{n}(z - \alpha_i)^{a_i}$:

$$a \in \Gamma(L, g) \Leftrightarrow g^2(z) \mid \frac{d\sigma_a(z)}{dz}. \tag{6.5}$$

B - L'attaque de Gibson

D'après (6.5), si une matrice G est une matrice génératrice de $\Gamma(L, g)$ où L est connu et g inconnu, il suffira de choisir deux éléments a et b de ce code (à l'aide de G) et le pgcd de $d\sigma_a(z)/dz$ et de $d\sigma_b(z)/dz$ sera probablement égal à $g^2(z)$. Le degré de $g(z)$ étant connu d'après les paramètres du code, on pourra au besoin effectuer un ou plusieurs autres calculs de pgcd.

L'attaque consiste ainsi à multiplier la clé publique à droite par une permutation jusqu'à obtenir un code de Goppa de support L. Lors de la description de cette attaque dans le cas où $n = 2^m$, Gibson [Gib91] remarque que l'ensemble des codes de Goppa de support L est globalement invariant sous l'action du groupe de permutations $\{\pi(z) = z^{2^i} + b \mid b \in \mathbf{F}_{2^m}, 0 \leq i < m\}$. En effet $\pi(\Gamma(L, g)) = \Gamma(\pi(L), g) = \Gamma(L, \pi(g))$ où $\pi(L) = (\pi(\alpha_1), \ldots, \pi(\alpha_n))$ et $\pi(g)(z) = g(z^{2^i} + b)^{2^{m-i}}$. Ceci permet de diviser le coût moyen de l'attaque par la taille d'une orbite sous l'action de ce groupe, c'est-à-dire en général mn, pour un coût total moyen de $(n-1)!/m$ calculs de pgcd.

En fait, comme nous le remarquons dans [LS01], l'attaque peut être améliorée encore en considérant l'action du groupe de permutations de la forme $\pi(z) = (az^{2^i} + b)/(cz^{2^i} + d)$, $ad - bc \neq 0$ sur les codes de Goppa étendus (cf. [MS77, p. 347]). La taille de l'orbite d'un code de Goppa étendu sous l'action de ce groupe sera en général de mn^3, ce qui divise encore le coût de l'attaque par n^2.

C - Attaques à l'aide de l'algorithme de séparation du support

L'algorithme de séparation du support permet de tester l'équivalence par permutation entre deux codes, chaque fois que ceux-ci auront un hull de faible dimension. De plus, lorsque deux codes sont équivalents, il permet d'obtenir une permutation qui les relie.

De cette manière il est possible de construire une attaque (décrite dans [LS01]) en parcourant la famille de codes secrets. Dans le cas des codes de Goppa binaires irréductibles t-correcteurs de longueur $n = 2^m$, le nombre de codes à considérer est environ égal à $2^{tm}/t$ (i.e. le nombre de polynômes irréductibles de degré t à coefficients dans $\mathbf{F}_{2^m}$ [LN83, p. 93]). Comme pour l'attaque de Gibson, si l'on considère des codes étendus, le coût de l'attaque peut être divisé par mn^3. Au total, sachant que dans le cas des codes de Goppa le coût de l'algorithme de séparation du support est de l'ordre de $O(n^3)$ opérations binaires, cette attaque aura un coût de l'ordre de $O(2^{tm}/tm)$. Notons enfin que dans la mise en œuvre effective, tester l'irréducibilité d'un polynôme n'est pas significativement moins cher que tester l'équivalence de deux codes, et donc le coût réel sera plutôt de l'ordre de $O(2^{tm}/m)$.

6.2.3 Clés faibles

Dans un travail récent [LS01] avec Pierre Loidreau nous nous sommes intéressés à un sous-ensemble particulier des codes de Goppa binaires : ceux dont le polynôme de Goppa est à coefficients dans $\mathbf{F}_2$ plutôt que $\mathbf{F}_{2^m}$. Nous avons pu montrer que le groupe de permutations de ces codes contient le Frobenius $z \mapsto z^2$; en fait, il s'agit en général exactement du groupe engendré par le Frobenius. Une telle propriété peut être mise en évidence à l'aide de l'algorithme de séparation du support. De plus nous montrons comment la connaissance des orbites des éléments du support sous l'action du Frobenius permet de diminuer les paramètres des codes sur lesquels une attaque exhaustive sera finalement effectuée. Nous montrons que l'at-

taque peut être facilement mise en œuvre pour des valeurs de t pas trop importantes (jusqu'à 30 ou 40).

A - Classification de l'espace des clés

Dans la lignée de ce travail sur les clés faibles du cryptosystème de McEliece, des questions plus fondamentales se posent. S'il est possible de détecter l'utilisation de certains codes à l'aide de certaines de leurs propriétés invariantes, pourquoi ne pas envisager la classification de l'ensemble des clés en un grand nombre de sous-ensembles, chacun d'eux étant identifiable par une propriété invariante et de taille suffisamment faible pour être ensuite parcouru de façon exhaustive. En d'autres termes peut-on trouver une classification « algorithmique » des codes de Goppa, et quel usage pourrait-on en faire ?

Soit $\mathcal{F}$ la famille dans laquelle le code secret est choisi. Pour tout invariant $\mathcal{V}$ nous noterons

$$E_{\mathcal{V}}(C) = \{\Gamma \in \mathcal{F}, \mathcal{V}(C) = \mathcal{V}(\Gamma)\}.$$

L'algorithme suivant prend en entrée une clé publique d'un cryptosystème basé sur les codes et qui retourne un élément de $\mathcal{F}$ qui lui est équivalent

`input` C
  `repeat`
    $\Gamma \in_R E_{\mathcal{V}}(C)$ *; i.e. tirer* Γ *uniformément dans* $E_{\mathcal{V}}(C)$
  `until` $\mathcal{SSA}(C, \Gamma) =$ `true`
`output` Γ

Nous noterons

- $K_{\mathcal{SSA}}(C)$ le coût d'un appel à l'algorithme de séparation du support pour les codes C et C' (ce coût peut être rendu indépendant de C'),
- $K_{\mathcal{V}}(C)$ le coût pour tirer uniformément un élément $\Gamma \in \mathcal{F}$ tel que $\mathcal{V}(\Gamma) = \mathcal{V}(C)$.

Pour un code C donné le coût moyen de l'algorithme ci-dessus est

$$(K_{\mathcal{V}}(C) + K_{\mathcal{SSA}}(C)) \frac{|E_{\mathcal{V}}(C)|}{N(C)}.$$

où $N(C)$ est le nombre d'éléments de $\mathcal{F}$ équivalents à C.

En pratique, c'est le cas pour les Goppa par exemple, $N(C)$ et $K_{\mathcal{SSA}}(C)$ sont polynomiaux en la longueur n de C pour presque tout code. Pour que l'algorithme fonctionne en temps polynomial il faudra donc trouver un invariant $\mathcal{V}$ qui soit à la fois

- *discriminant*, c'est-à-dire que l'ensemble $E_{\mathcal{V}}(C)$ doit être presque toujours de taille polynomiale en n,
- « *inversible* », c'est-à-dire qu'il est presque toujours possible en temps polynômial en n de tirer uniformément un élément dans $E_{\mathcal{V}}(C)$.

À notre connaissance un tel invariant n'existe pas pour les codes de Goppa.

Pour envisager une telle attaque sur les codes de Goppa, il faudrait d'une façon ou d'une autre relier une propriété invariante d'un code à une propriété explicite du polynôme de Goppa (c'est ce qui est fait, partiellement, dans [LS01]).

Chapitre 7

Remarques sur la mise en œuvre

La mise en œuvre des cryptosystèmes à clé publiques pose un certain nombre de questions. Les premières concernent évidemment l'implantation du chiffrement et du déchiffrement; nous verrons qu'il existe quelques différences de complexité entre la version de McEliece et celle de Niederreiter. Nous verrons ensuite comment et à quelles conditions il est possible de réduire la taille de la clé publique et de la clé secrète. Enfin dans la dernière section de ce chapitre nous examinerons en détail les moyens de coder de l'information à l'aide de mots de poids constant. Un tel codage est nécessaire pour mettre en œuvre la variante de Niederreiter.

7.1 Implémentation

Patterson décrit dans [Pat75] un algorithme qui résout l'équation clé (7.1) à l'aide de l'algorithme de Berlekamp-Massey [Ber68, Mas69]. Toujours dans [Pat75], on trouve une variante spécifique pour le cas binaire avec un gain algorithmique très contestable, au moins pour une implémentation logicielle. Les solutions proposées par Patterson requièrent un nombre d'opérations dans $\mathbf{F}_{2^m}$ quadratique en t, le degré du polynôme de Goppa. En pratique, la résolution directe de (7.1) à l'aide de l'algorithme d'Euclide étendu permet d'obtenir également un coût quadratique avec une constante qui n'est pas plus importante en moyenne. Nous ne décrirons donc que cette dernière solution qui a en outre l'avantage d'être plus simple.

Une autre approche possible pour décoder les codes de Goppa consiste à les considérer comme des codes alternants (cf. par exemple [MS77, Ch. 12]). Dans ce cas l'équation clé a pour forme

$$\sigma(z)S(z) = \omega(z) \bmod z^{2t},$$

et peut être résolue directement par l'algorithme de Berlekamp-Massey. Cette solution apporte un gain dans la résolution de l'équation clé (un facteur 1/2 par rapport à Euclide), mais rend certains précalculs et le stockage d'une matrice $tm \times tm$ obligatoire. Comme la résolution de l'équation clé n'est pas l'opération la plus coûteuse, nous nous contenterons de décrire la première solution, que nous désignerons pour des raisons historiques par algorithme de Patterson.

7.1.1 Algorithme de Patterson

Soit $\Gamma(L, g)$ un code de Goppa binaire dont le polynôme générateur $g(z)$ est de degré t sans facteur multiple et sans racine dans le support $L = (\alpha_1, \ldots, \alpha_n) \subset \mathbf{F}_{2^m}$. Un mot

$a = (a_1, \ldots, a_n)$ sera dans Γ si et seulement si

$$R_a(z) = \sum_{i=1}^{n} \frac{a_i}{z - \alpha_i} = 0 \bmod g^2(z)$$

Ce polynôme permet donc de caractériser les mots de code de Γ. Notons que l'application $x \mapsto R_x(z)$ est linéaire, c'est-à-dire que $R_{a+e}(z) = R_a(z) + R_e(z)$. Ainsi pour tout y dans $\mathbf{F}_2^n$, tout mot e du coset $y + \Gamma$ vérifie $R_e(z) = R_y(z)$, en particulier si e est un mot de poids minimum de $y + \Gamma$, le mot $y - e$ est un mot de Γ le plus proche de y.

À tout mot $e = (e_1, \ldots, e_n) \in \mathbf{F}_2^n$ nous associons son polynôme localisateur $\sigma_e(z) = \prod_{i=1}^{n}(z - \alpha_i)^{e_i}$. Nous avons l'équation clé suivante, dont $\sigma_e(z)$ est l'inconnue :

$$\sigma_e(z)R_e(z) = \frac{d\sigma_e(z)}{dz} \bmod g^2(z) \tag{7.1}$$

Cette équation peut être résolue à l'aide de l'algorithme d'Euclide étendu chaque fois que e est de poids de Hamming inférieur à t. Donc pour tout mot y de $\mathbf{F}_2^n$ nous pouvons connaître son coset leader (*i.e.* le mot de plus petit poids dans $y+\Gamma$) si ce dernier est de poids inférieur à t. La table 7.1 décrit les étapes du décodage, le résultat sera la liste des indices des positions erronées.

Donnée : un mot y de $\mathbf{F}_{2^n}$ à distance au plus t de Γ
1. Calcul du syndrome : $y \mapsto R_y(z) = S(z)$
2. Résolution de l'équation clé : $S(z) \mapsto \sigma(z)$
3. Calcul des racines de $\sigma(z)$ dans $\mathbf{F}_{2^m}$: $\sigma(z) \mapsto (i_1, \ldots, i_t)$

TAB. 7.1: *Décodage d'un code de Goppa*

A - Complexité

1. Le calcul du syndrome $R_y(z)$ nécessite environ $3tn/2$ multiplications et additions dans le corps fini $\mathbf{F}_{2^m}$ (cf. ci-dessous les remarques sur le calcul de $R_e(z)$). En pré-calculant tous les inverses de $z - \beta$ modulo $g^2(z)$, il est possible de ramener cette complexité à tn additions dans $\mathbf{F}_{2^m}$.

2. La résolution de l'équation clé par l'algorithme d'Euclide étendu coûte de l'ordre de $4t^2$ multiplications et le même nombre d'additions dans $\mathbf{F}_{2^m}$.

3. Sans entrer dans les détails, le calcul des racines de $\sigma(z)$ peut se faire soit de façon directe en évaluant $\sigma(z)$ en tout point de L, ce qui coûte environ tn multiplications et additions dans $\mathbf{F}_{2^m}$, soit par l'algorithme des traces de Berlekamp [Ber68, Ch. 6] (voir plus bas), au prix d'environ mt^2 multiplications et autant d'additions dans $\mathbf{F}_{2^m}$.

B - Remarques sur le calcul de $R_e(z)$

Le mot reçu y aura le même syndrome que l'erreur e. Pour calculer

$$R_y(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i} = 0 \bmod g^2(z)$$

Il faudra calculer les inverses de $z - \beta$ modulo $g^2(z)$. En pratique il suffira de calculer les inverses modulo $g(z)$:

$$\frac{1}{z-\beta} \bmod g^2(z) = (z-\beta) f_\beta^2(z) \quad \text{où} \quad f_\beta(z) = \frac{1}{z-\beta} \bmod g(z).$$

Les coefficients de $f_\beta^2(z)$ se calculent aisément à partir de ceux de $g^2(z)$. Grâce à l'identité $z f_\beta(z) = 1 + \beta f_\beta(z) - g(z)/g(\beta)$ on obtient :

$$\begin{aligned} f^2_{\beta,t-1} &= \frac{1}{g^2(\beta)}, \\ f^2_{\beta,i} &= \beta^2 f^2_{\beta,i+1} + \frac{g^2_{i+1}}{g^2(\beta)}, \quad i = 0, \ldots, t-2. \end{aligned}$$

Que l'on peut écrire différemment en définissant la suite u_i :

$$u_0 = 1 \quad \text{et} \quad u_i = \beta^2 u_{i-1} + g^2_{t-i}, \quad i = 1, \ldots, t.$$

Nous aurons $u_t = g^2(\beta)$ et $f^2_{\beta,i} = u_{t-1-i}/u_t$, $i = 0, \ldots, t-1$. Chaque polynôme $1/(z - \beta) \bmod g^2(z)$ pourra ainsi être obtenu à l'aide d'environ $4t$ opérations dans le corps $\mathbf{F}_{2^m}$ (t additions, t divisions et $2t$ multiplications).

C - Algorithme des traces de Berlekamp

Cet algorithme consiste à calculer pour $i = 0, \ldots, m-1$

$$r_i(z) = \gcd\left(\sigma(z), T(\alpha^i z)\right), \quad \text{où} \quad T(z) = \sum_{i=0}^{m-1} z^{2^i}.$$

L'ensemble des polynômes $r_i(z)$ permet de « séparer » toutes les racines de $\sigma(z)$. Une description précise de l'algorithme est disponible par exemple dans [MvOV89]. En pratique il n'est pas toujours nécessaire de calculer tous les $r_i(z)$ si le degré de $\sigma(z)$ est petit. Les polynômes $r_i(z)$ peuvent être obtenus en précalculant tous les $z^{2^i} \bmod \sigma(z)$. Le coût de l'algorithme n'est pas significativement plus élevé que le coût de ce précalcul, c'est-à-dire mt^2 opérations dans le corps $\mathbf{F}_{2^m}$.

7.1.2 Déchiffrement pour le cryptosystème de McEliece

Le cryptogramme est un mot y de $\mathbf{F}_2^n$ donc l'algorithme de décodage peut s'appliquer directement. À l'issue nous pouvons corriger les erreurs et obtenir le mot de code $\tilde{y} = xG$ où x est le message clair recherché. Dans le cas où la matrice G n'est pas donnée sous forme systématique (cf. §7.2), il faudra encore multiplier (k bits de) $\tilde{y}$ par une matrice $k \times k$ qu'il est possible de pré-calculer. Si l'on compte qu'une opération dans $\mathbf{F}_{2^m}$ représente de l'ordre de m opérations binaires, le coût total sera borné par $ctmn$ opérations binaires, où c est une petite constante, plus éventuellement $k^2/2 = (n - tm)^2/2$ pour obtenir le texte clair.

7.1.3 Déchiffrement pour le cryptosystème de Niederreiter

Le cryptogramme est cette fois ci un mot y de $\mathbf{F}_2^{n-k}$ qui est le syndrome (par H) d'un mot de poids t de $\mathbf{F}_2^n$ représentant le texte clair. Dans le cas où la clé publique est sous forme systématique $H = (Id \mid R)$, le mot $\bar{y} = (y_1, \ldots, y_{n-k}, 0, \ldots, 0)$ de $\mathbf{F}_2^n$ (*i.e.* le syndrome y complété par des zéros) vérifie $H\bar{y}^T = y$. Nous pourrons donc lui appliquer l'algorithme de décodage. Le fait que seules les tm premières coordonnées du mot soient non nulles nous permet un gain appréciable pour le calcul de $S(z)$. En pratique, toutes les complexités (en temps, et également en mémoire s'il y a des pré-calculs) de la première étape peuvent être multipliées par tm/n. Le résultat de l'algorithme nous donnera directement le texte clair, c'est-à-dire un mot de poids t de $\mathbf{F}_2^n$. Au total, la complexité du déchiffrement sera donc de l'ordre de ct^2m opérations dans $\mathbf{F}_{2^m}$ ou ct^2m^2 opérations binaires où c est une petite constante (de l'ordre de 3, ou légèrement plus sans les pré-calculs).

7.2 Taille de la clé publique

Il est possible de réduire la taille de la clé publique en choisissant une matrice publique sous forme systématique. Un tel choix ne pose pas de problème dans le système de Niederreiter, par contre pour McEliece, il nécessite une précaution : les messages doivent être choisis selon une loi uniforme. Il est aussi possible d'ordonner la matrice publique d'une façon ou d'une autre, toutefois cela ne permet pas un gain appréciable.

7.2.1 Clé publique sous forme systématique

Soit C un code (n, k) sur le corps fini $\mathbf{F}_q$ correcteur de t erreurs utilisé dans une instance des cryptosystèmes de McEliece et de Niederreiter.

Nous nous interrogeons ici sur l'impact que peut avoir un choix particulier des matrices publiques G et H sur la sécurité du système. En particulier, elles ont pu être choisies de telle façon que G ou H soit sous forme systématique. Un tel choix permettrait de réduire de moitié environ la taille de la clé publique.

A - Cryptosystème de McEliece

Résoudre l'instance (G, y) du cryptosystème de McEliece consiste à trouver l'unique couple (x, e) de $\mathbf{F}_q^k \times CW_q(n, t)$, s'il existe, solution de l'équation $xG + e = y$. Soit $\chi(G, y)$ le coût de cette résolution.

Proposition 17 *Il existe une constante positive c telle que pour toute matrice inversible U*

$$\forall y \in \mathbf{F}_q^n, \chi(UG, y) \leq \chi(G, y) + cn^3. \tag{7.2}$$

Preuve : Si (x, e) est la solution de l'instance (G, y), alors la solution de (UG, y) est (xU^{-1}, e) et peut être déduite de (x, e) pour un coût n'excédant pas cn^3 pour une certaine constante c.

Soit l'espace probabilisé $\Omega = \mathbf{F}_q^k \times CW_q(n, t)$ dont chaque occurrence (x, e) est composée d'un message clair x et d'une erreur e (le message chiffré sera donc $y = xG+e$). Nous noterons $P(x, e)$ la probabilité de l'événement (x, e) et

$$P_x : x \mapsto \sum_{e \in CW_q(n,t)} P(x, e) \quad \text{et} \quad P_e : e \mapsto \sum_{x \in \mathbf{F}_q^k} P(x, e).$$

Nous supposerons que la loi de probabilité P est indépendante de G. Les deux hypothèses supplémentaires suivantes seront utiles pour énoncer la proposition 18.

(H1) La loi d'émission P_x est uniforme.

(H2) Les lois P_x et P_e sont indépendantes.

Le coût moyen du décryptage pour une clé publique donnée est égal à

$$\chi(G) = \sum_{(x,e)\in\Omega} P(x,e)\chi(G, xG+e). \tag{7.3}$$

Proposition 18 [1] *Si les hypothèses (H1) et (H2) sont vérifiées, alors il existe une constante positive c telle que pour toute matrice inversible U*

$$\chi(UG) \leq \chi(G) + cn^3. \tag{7.4}$$

Preuve : D'après les hypothèses de l'énoncé, nous avons $P(x,e) = q^{-k}P_e(e)$ et, en utilisant la proposition 17

$$\begin{aligned}
\chi(UG) - \chi(G) &= \sum_{(x,e)\in\Omega} P(xU^{-1},e)\chi(UG, xG+e) - P(x,e)\chi(G,xG+e) \\
&= \sum_{(x,e)\in\Omega} q^{-k}P_e(e)\left(\chi(UG,xG+e) - \chi(G,xG+e)\right) \\
&\leq \sum_{(x,e)\in\Omega} q^{-k}P_e(e)cn^3 = cn^3.
\end{aligned}$$

La signification de ce résultat est que, moyennant le fait que les messages sont choisis uniformément dans $\mathbf{F}_q^k$, toute matrice génératrice du code public offre le même niveau de sécurité, en particulier on peut la choisir systématique, ce qui réduit sa taille de $k \times n$ à $k \times (n-k)$ éléments de $\mathbf{F}_q$.

Notons que la loi de probabilité de e est indifférente ; cela ne signifie pas que le système est sûr quel que soit le système de génération d'erreur, mais simplement que son impact sur la sécurité, bon ou mauvais, est le même quelle que soit la matrice génératrice.

Si la loi d'émission n'est pas uniforme ? Si la matrice génératrice est sous forme systématique, le début du cryptogramme est égal au texte clair modifié en quelques positions. S'il existe une forme de redondance dans le texte clair il paraît alors possible dans certains cas de retrouver l'information. On peut s'en convaincre par un petit exemple. Nous avons modifié un message de 65 caractères (soient 520 bits en ASCII) en 25 bits tirés aléatoirement (code $(1024, 524)$ avec 50 erreurs). Nous obtenons le cryptogramme suivant :

```
Le{ cryptosystèmås0basés suv les code{‘corveãteurs soît-ils sýòs?
```

Il est clair qu'avec un peu d'astuce et quelques informations contextuelles il est possible de retrouver le message clair.

1. D'après un résultat de G. Kabatianski, communication privée.

Absence de sécurité symbole par symbole. Nous avons vu que si la loi d'émission est uniforme, la mise sous forme systématique de la clé publique ne permet pas de déterminer plus facilement quel message a été émis. Toutefois, comme cela apparaît dans l'exemple du paragraphe précédent, chaque bit du message est connu, individuellement, avec une probabilité élevée (de l'ordre de $1 - t/n = 0.95$). La connaissance d'une telle information *locale* pourrait s'avérer problématique selon le contexte (notons que la version de Niederreiter n'est pas sensible à ce problème).

Si l'on souhaite malgré tout mettre en œuvre le cryptosystème de McEliece avec une clé publique de taille réduite, il faudra appliquer une fonction inversible publique au texte clair avant le chiffrement. Cette fonction devra être simple, afin de n'augmenter significativement ni la taille de la clé ni la complexité des procédures de chiffrement et de déchiffrement. Elle devra également posséder de bonnes propriétés de diffusion afin de rendre inexploitables les propriétés locales évoquées plus haut. Une telle fonction pourra, par exemple, être mise en œuvre à l'aide d'un registre à décalage à rétroaction linéaire.

Ceci ne protégera le système d'attaques par manipulation de messages. Par exemple, une évidente faiblesse du système peut être mise en évidence en additionnant deux textes chiffrés provenant du même clair [Ber97]. D'autres mises en œuvre plus complexes du cryptosystème [BDPR98, KI01] permettent d'obtenir des versions « sémantiquement sûres ».

B - Cryptosystème de Niederreiter

L'espace probabilisé est cette fois $CW_q(n,t)$, une occurrence x est un message clair. Nous supposerons que la loi de probabilité notée $P(x)$ est indépendante de la clé publique H. Résoudre l'instance (H,y) du cryptosystème de Niederreiter consiste à trouver l'unique mot x de $CW_q(n,t)$, s'il existe, solution de l'équation $Hx^T = y$. Soit $\chi(H,y)$ le coût de cette résolution.

Proposition 19 *Pour toute matrice inversible U et tout élément y de $\mathbf{F}_q^{n-k}$*

$$\chi(UH,Uy) = \chi(H,y).$$

Preuve : Une solution à l'instance (H,y) existe si et seulement une solution à l'instance (UH,Uy) existe. De plus elles sont égales.

Le coût moyen du décryptage pour une clé publique donnée est égal à

$$\chi(H) = \sum_{m \in CW_q(n,t)} P(m)\chi(H, Hm^T).$$

Proposition 20 *Pour toute matrice inversible U, nous avons*

$$\chi(UH) = \chi(H).$$

Preuve : Nous avons (d'après la proposition 19) :

$$\chi(UH) - \chi(H) \quad = \quad \sum_{m \in CW_q(n,t)} P(m)\left(\chi(UH, UHm^T) - \chi(H, Hm^T)\right) = 0.$$

Le cryptosystème de Niederreiter peut donc inconditionnellement être utilisé avec une clé publique sous forme systématique.

7.2.2 Tri de la clé publique

En faisant l'hypothèse que toutes les lois de probabilités sont uniformes (y compris pour la variante de Niederreiter), nous pouvons prouver des résultats similaires lorsque la matrice génératrice ou de parité est multipliée à droite par une matrice de permutation. Il devient ainsi possible d'utiliser une clé publique dont les colonnes ont été préalablement triées. Si la clé publique est sous forme systématique avec $k > (n-k)$, le gain peut en théorie aller jusqu'à environ $\log_2(k!)$ bits. Toutefois le gain sera relativement faible et une mise en œuvre efficace est probablement très difficile ...

7.3 Taille de la clé secrète

Pour effectuer le déchiffrement pour les systèmes de McEliece et de Niederreiter, il n'est pas nécessaire de conserver la clé publique. Tout d'abord, si celle-ci est sous forme systématique, elle peut être reconstituée à l'aide du polynôme de Goppa et de la permutation. Si l'on examine les procédures de déchiffrement, cette reconstitution n'est même pas nécessaire. En pratique, si l'on excepte le cas du système de McEliece avec une matrice génératrice quelconque, il faudra conserver le polynôme de Goppa, soient tm bits et la permutation, soit $\log_2(n!)$ bits. Au besoin, si l'on accepte une dégradation, très théorique, de la sécurité, on peut engendrer la permutation à l'aide d'un générateur pseudo-aléatoire utilisant le polynôme de Goppa comme graine[2].

7.4 Codage des mots de poids constant

Nous noterons $CW_q(n,t)$ l'ensemble des mots de poids t de l'espace $\mathbf{F}_q^n$. Nous avons vu que le système de Niederreiter utilisait cet espace comme ensemble de textes clairs. Il nous faudra donc établir une correspondance entre ces mots et l'information que nous souhaitons transmettre. Ceci signifiera en pratique que nous allons devoir coder de façon non équivoque les mots de $CW_q(n,t)$ par des séquences binaires.

Nous pouvons ramener cette étude à celle de l'ensemble $CW_2(n,t)$ (que nous notons $W_{n,t}$) car il existe une bijection évidente entre $CW_q(n,t)$ et $\mathbf{F}_q^* \times W_{n,t}$.

7.4.1 Une solution exacte

Nous représenterons les éléments de $W_{n,t}$ par des t-uplets $(i_1, \ldots, i_t)$ d'entiers tels que $1 \leq i_1 < \ldots < i_t \leq n$. Pour plus de commodité, nous noterons l le plus petit entier tel que $\binom{n}{t} \leq 2^l$. Nous établissons une bijection entre $W_{n,t}$ et l'intervalle $[0, \binom{n}{t}[$ de la façon suivante :

$$\begin{array}{rccl} \varphi & : & W_{n,t} & \to & [0, \binom{n}{t}[\\ & & (i_1, \ldots, i_t) & \mapsto & \binom{i_t-1}{t} + \ldots + \binom{i_1-1}{1} \end{array}$$

Le codage est relativement simple, il consiste à additionner t coefficients binomiaux, ce qui requiert environ tl opérations binaires si ces coefficients ont été pré-calculés (mémoire ntl bits). Si ce n'est pas le cas le coût sera de l'ordre de $tB(l)$ opérations binaires pour calculer

2. Engendrer une permutation pseudo-aléatoire à partir d'un séquence pseudo-aléatoire peut se réaliser à un coût raisonnable en utilisant, par exemple, un schéma de Feistel à 3 ou 4 tours [Pie90, Pat91].

une image par φ, où $B(l)$ est le coût du calcul d'un coefficient binomial de taille l bits (voir plus bas).

Pour le décodage, la situation se complique. Le t-ème élément de $\varphi^{-1}(x)$ sera l'unique entier i_t tel que

$$\binom{i_t - 1}{t} \leq x < \binom{i_t}{t}.$$

Là encore si les coefficients binomiaux ont été pré-calculés, il faudra environ tl opérations binaires pour retrouver l'élément de $W_{n,t}$ auxquelles il faudra ajouter un temps de recherche, qui ne sera pas plus élevé si les coefficients sont correctement rangés (à l'aide d'une table de hachage par exemple). Sans les pré-calculs il faut inverser la fonction $i \mapsto \binom{i}{t}$. Si nous cherchons à exprimer i en fonction de $x = \binom{i}{t}$, nous obtenons pour t fixé[3]

$$i = X + \frac{t-1}{2} + \frac{t^2-1}{24}\frac{1}{X} + O\left(\frac{1}{X^3}\right) \tag{7.5}$$

où $X = (t!x)^{1/t}$. Ainsi, lorsque t est petit par rapport à n, l'indice i peut être obtenu à l'aide de (7.5) et pour un coût indépendant de la taille du problème car nous pouvons nous contenter d'une valeur approchée par un nombre flottant. Si nous négligeons ce dernier coût, l'inversion de φ en un point va coûter de l'ordre de $tB(l)$ opérations binaires.

Calcul des coefficients binomiaux. Calculer un coefficient binomial à l'aide de la formule de récurrence

$$\binom{n}{t} = \frac{n-t+1}{t}\binom{n}{t-1}$$

va coûter $O(l^2)$ opérations binaires où l est la taille en bit du coefficient cherché. Il est possible baisser ce coût en utilisant des techniques dichotomiques [Bre76] et des opérations arithmétiques accélérées sur les grand entiers. Le coût du calcul d'un coefficient binomial peut alors être abaissé à $B(l) = O(l \log^3 l)$. Toutefois cette technique ne sera avantageuse que pour des valeurs de l supérieures à plusieurs centaines, voire plusieurs milliers.

7.4.2 Codage des mots de poids constant : une solution approchée

Nous présentons ici une méthode pour coder des mots de $W_{n,t}$ en des séquences binaires. Soit l'ensemble d'entiers $S = \{0, 1, 2, \ldots, n-t-1\}$, nous considérons l'injection

$$\begin{array}{rcl} W_{n,t} & \to & S^t \\ (i_1, \ldots, i_t) & \mapsto & (i_1 - 1, i_2 - i_1 - 1, \ldots, i_t - i_{t-1} - 1) \end{array}$$

Dans un mot de $W_{n,t}$, le nombre de '0' entre deux '1' consécutifs sera égal à i, $0 \leq i \leq n-t-1$, avec la probabilité $P_i = \binom{n-i-1}{t-1}/\binom{n}{t}$. Nous munissons S de cette loi de probabilité et nous allons construire un codage de source pour S.

L'ensemble S étant trop grand, nous allons à nouveau le transformer. Soit d un entier positif, pour tout $i \in S$ nous notons q_i et r_i respectivement le quotient et le reste de la

3. Cette formule a été obtenue en Maple grâce aux indications de Bruno Salvy.

division euclidienne de i par d. Enfin nous notons S_d la source $\{0, 1, \ldots, d-1, d\}$ munie de la loi de probabilité :

$$Pr_d(i) = P_i, 0 \leq i < d \quad \text{et} \quad Pr_d(d) = 1 - \sum_{i=0}^{d-1} P_i = \frac{\binom{n-d}{t}}{\binom{n}{t}}.$$

Nous pouvons à présent définir $f_d : S_d \to \{0,1\}^*$ un code de Huffman (ou autre) de S_d. La lettre i de S sera représentée par le (q_i+1)-uplet $(d, \ldots, d, r_i)$ de $S_d^{q_i+1}$ dans lequel d est répété q_i fois. De cette manière, nous pourrons représenter tout élément S^t et donc de $W_{n,t}$ comme une séquence de lettre de S_d. Finalement en utilisant f_d nous obtenons un codage de source de $W_{n,t}$.

Bien entendu ce codage ne sera pas optimal car en codant les éléments de $W_{n,t}$ comme des éléments de S^t par concaténation nous supposons implicitement que les distributions des écarts entre positions consécutives sont indépendantes, ce qui n'est évidemment pas le cas. Ceci a pour effet de réduire l'efficacité du codage (au sens de la théorie de l'information).

A - Codage

En pratique, le code de Huffman de S_d sera le plus efficace lorsque $Pr_d(d)$ sera proche d'une puissance de 1/2. On va donc choisir d tel que $Pr_d(d)$ soit proche de 1/2. Les valeurs des probabilités P_i, pour $i < d$, sont lentement décroissantes et le rapport P_0/P_{d-1} ne dépasse pas 2 pour les valeurs de d qui nous intéressent. Cela signifie que la longueur de $f_d(i)$ sera croissante et que $f_d(d-1)$ sera plus long que $f_d(0)$ d'au plus une unité. Nous noterons u le plus grand entier vérifiant $2^u \leq d$, et $D_2(i,l)$ les l bits les moins significatifs de la décomposition en base 2 de l'entier i (bit de poids fort à gauche). Le code suivant de S_d est un code binaire préfixe, généralement optimal, de S_d :

$$f_d(i) = \begin{cases} D_2(2^u + i, u+1) & i < d_1 = 2^{u+1} - d \\ D_2(2^{u+1} + i + d_1, u+2) & d_1 \leq i < d \\ 0 & i = d \end{cases}$$

Si i vaut d il sera codé par une séquence de longueur 1, s'il est inférieur à d_1, il sera codé par une séquence de longueur $u+1$ et sinon par une séquence de longueur $u+2$. On en déduit une procédure de codage pour S :

```
procedure codage
input: un entier i
output: une séquence binaire
    (q, r) ← euclide(i, d)
    if r < d_1
        return D_2(2^u + r, u + 1 + q)
    else
        return D_2(2^{u+1} + r + d_1, u + 2 + q)
```

où `euclide`(i,d) retourne le quotient et le reste de la division euclidienne de i par d et `D_2`(x,l) retourne les l bits les moins significatifs de l'écriture de l'entier x en base 2. Pour coder les éléments de $W_{n,t}$ on procédera comme suit :

```
procedure CWtoB
```

```
input: élément (i_1, i_2, ..., i_t) de W_{n,t}
output: une séquence binaire
    s ← codage(i_1 − 1)
    for j from 2 to t
        s ← s ⊙ codage(i_j − i_{j−1} − 1)
    return s
```

Nous notons $s_1 \odot s_2$ la concaténation de deux séquences binaires s_1 et s_2. Remarquons que cette procédure de codage est simple et ne nécessite pas de parcourir un arbre.

B - Décodage

Le décodage est également relativement simple. On peut le décrire par la procédure suivante :

```
procedure décodage
input: un flot binaire B
output: un entier naturel
    q ← 0
    while (read(B, 1) = 0)
        q ← q + 1
    r ← read(B, u)
    if r < d_1
        return r + q * d
    else
        return 2 * r + read(B, 1) − d_1 + q * d
```

où $\texttt{read}(B, l)$ avance de l bits dans le flot binaire et retourne l'entier correspondant aux l bits lus. Comme pour le codage, on constate que le cas $d = 2^u$ est particulièrement simple. Le décodage d'une séquence binaire produite par `CWtoB` en un mot de $W_{n,t}$ s'écrit :

```
procedure CWtoBinv
input: un flot binaire B
output: un t-uplet d'entiers naturels
    i_1 ← 1 + décodage(B)
    for j from 2 to t
        i_j ← i_{j−1} + 1 + décodage(B)
    return (i_1, ..., i_t)
```

C - Efficacité

L'efficacité du codage binaire d'une source est le rapport entre l'entropie de cette source et la longueur moyenne de la séquence binaire codant un élément. L'efficacité est toujours inférieure ou égale à 1.

Pour tout $i = qd+r \in S$ la procédure `codage` retournera une séquence de longueur $u+1+q$ si $r < d_1$ et de longueur $u + 2 + q$ sinon. Nous avons pour la longueur moyenne:

$$\bar{L}_d = \sum_{q \geq 0} \left(\sum_{r=0}^{d_1-1} (u + 1 + q) P_{qd+r} + \sum_{r=d_1}^{d-1} (u + 2 + q) P_{qd+r} \right).$$

En exprimant les probabilités en fonction des π_j

$$\pi_j = 1 - \sum_{i=0}^{j-1} P_i = \frac{\binom{n-j}{t}}{\binom{n}{t}},$$

nous obtenons après quelques simplifications

$$\bar{L}_d = u + 1 + \sum_{q \geq 0} \pi_{qd+d_1},$$

Notons que cette somme n'est pas infinie puisque $\pi_i = 0$ pour $i > n - t$. La longueur moyenne du codage d'un élément de $W_{n,t}$ par `CWtoB` sera $\bar{L} = t\bar{L}_d$.

En pratique, nous pourrons déduire de l'expression de $\bar{L}_d$ la valeur optimale pour d, c'est-à-dire la valeur de d qui permet d'exprimer les éléments de $W_{n,t}$ en utilisant le moins possible de bits en moyenne. On peut obtenir une valeur approximative de d à l'aide d'un développement de $\bar{L}_d$ lorsque π_d est voisin de 1/2:

$$\bar{L}_d \approx \log_2(d) + \frac{1}{1 - \pi_d},$$

dont le minimum est atteint pour $n/(2t\log(2)) = 0.72\, n/t$. La valeur de d optimale observée en pratique est légèrement inférieure comme on le constate dans la table 7.2. On constate

(n,t)	d	$t\bar{L}_d$	$H = \log_2 \binom{n}{t}$	$H/t\bar{L}_d$
$(2048, 26)$	32	205.14	197.39	96.22%
$(2048, 26)$	52	200.51	197.39	**98.44**%
$(2048, 26)$	64	201.38	197.39	98.01%
$(4096, 9)$	256	91.31	89.51	98.04%
$(4096, 9)$	275	91.28	89.51	**98.06**%
$(4096, 9)$	512	93.53	89.51	95.72%
$(2^{16}, 9)$	4096	127.32	125.53	98.59%
$(2^{16}, 9)$	4398	127.30	125.53	**98.61**%
$(2^{16}, 9)$	8192	129.53	125.53	96.91%

TAB. 7.2: *Efficacité de* `CWtoB`

également que l'efficacité varie peu avec d. En particulier si d est une puissance de 2, le codage et le décodage sont simplifiés à l'extrême pour une perte très faible.

D - Codage adaptatif

Le fait de coder les éléments de $W_{n,t}$ comme des éléments de S^t n'est pas optimal car les distributions des écarts entre les positions des '1' dans les éléments de $W_{n,t}$ ne sont pas indépendants. En faisant varier la valeur de d au cours du calcul, il est possible d'améliorer l'efficacité.

`procedure CWtoBAdaptatif`
`input:` deux entiers n et t, un t-uplet $(i_1, i_2, \ldots, i_t)$
`output:` une séquence binaire

```
    if t = 0 or n ≤ t
        return
    d ← [λn/t]
    u ← [log_2(d)]
    d_1 ← 2^{u+1} − d
    if i_1 >= d
        return 0 ⊙ CWtoBAdaptatif(n − d, t, (i_1 − d, i_2, ... , i_t))
    else if i < d_1
        return 1 ⊙ D_2(i, u) ⊙ CWtoBAdaptatif(n − i − 1, t − 1, (i_2, ... , i_t))
    else
        return 1 ⊙ D_2(i + d_1, u + 1) ⊙ CWtoBAdaptatif(n − i − 1, t − 1, (i_2, ... , i_t))
```

Cette procédure dépend d'un paramètre $\lambda \leq 1$ dont la valeur optimale est entre 0.6 et 0.7 selon les valeurs de n et t. Nous notons $[x]$ la partie entière de x. Enfin, la valeur de d doit au moins être égale à 1 pour que la procédure s'arrête.

Il semble difficile d'évaluer théoriquement la longueur moyenne obtenue de cette manière, mais les simulations montrent des efficacités très proche de 1 (cf. table 7.3).

(n,t)	λ	$\bar{L}$	$H = \log_2 \binom{n}{t}$	$H/\bar{L}$
$(2048, 26)$	0.67	198.24	197.39	99.57%
$(4096, 9)$	0.64	89.82	89.51	99.66%
$(2^{16}, 9)$	0.64	125.83	125.53	99.76%

TAB. 7.3: *Efficacité de* `CWtoBAdaptatif` *(simulations)*

La procédure de décodage se déduit aisément.

```
procedure BtoCW
input: deux entiers n ≤ t, un flot binaire B
output: un élément de W_{n,t}
    i_1, i_2, ... , i_t ← CWtoBAinv(n, t, 0, B)
    i_1 ← i_1 + 1
    for j from 2 to t
        i_j ← i_{j−1} + i_j + 1
    return (i_1, ... , i_t)

procedure CWtoBAinv
input: trois entiers n, t et i, un flot binaire B
output: un t-uplet d'entiers naturels
    if t = 0
        return
    else if n ≤ t      ; en fait n = t
        return i, 0, ... , 0
    else
        d ← [λn/t]
        u ← [log_2(d)]
        d_1 ← 2^{u+1} − d
        if read(B, 1) = 0
```

```
        return CWtoBAinv(n − d, t, i + d, B)
    else
        r ← read(B, u)
        if r ≥ d_1
            r ← 2 ∗ r + read(B, 1) − d_1
        return i + r, CWtoBAinv(n − r − 1, t − 1, 0, B)
```

On vérifie facilement par récurrence que, si `CWtoBAinv` est initialement appelé depuis `BtoCW`, la somme des éléments du t-uplet retourné par `CWtoBAinv`(n, t, i, B) est inférieure ou égale à $n - t + i$. En conséquence `BtoCW` retourne bien un élément de $W_{n,t}$.

7.4.3 Codage d'une séquence binaire en mots de poids constant

Le problème n'est pas exactement le même que celui que nous venons de résoudre, toutefois les outils que nous venons de mettre en place vont pouvoir servir. La procédure `CWtoBinv` peut être utilisée pour tout flot binaire, et elle retournera un t-uplet d'entiers. Malheureusement ce t-uplet n'est pas un élément de $W_{n,t}$ (ni même un élément de S^t) mais de $\mathbf{N}^t$. En fait la procédure `CWtoBinv` fournit un mot de $CW(\infty, t)$, l'ensemble des séquences binaires finies de poids t. En revanche, la procédure adaptative `BtoCW` retournera toujours un élément de $W_{n,t}$.

A - Efficacité

On pourra définir une efficacité qui sera le rapport entre le nombre moyen $\bar{L}_{inv}$ de bits lus par `BtoCW` avant de produire un élément de $W_{n,t}$ et le logarithme en base 2 du cardinal de $W_{n,t}$. L'efficacité obtenue est très proche de 1 comme nous pouvons le constater dans la table 7.4. Dans cette table nous donnons également l'entropie H_{CW} de l'ensemble $W_{n,t}$ engendré par la procédure `BtoCW`.

(n, t)	λ	$\bar{L}_{inv}$	$\log_2 \binom{n}{t}$	$\bar{L}_{inv} / \log_2 \binom{n}{t}$	H_{CW}
$(2048, 26)$	0.67	196.53	197.39	99.56%	194.85
$(4096, 9)$	0.64	89.21	89.51	99.66%	88.63
$(2^{16}, 9)$	0.64	125.22	125.53	99.76%	124.64

TAB. 7.4: *Efficacité de* `BtoCW` *(simulations)*

7.4.4 Complexité

Toutes les procédures présentées requièrent en moyenne un nombre d'opérations binaires proportionnel à la taille de la séquence binaire générée ou lue. En pratique et dans le cadre de ce travail, ce coût sera de toute façon faible par rapport au coût du chiffrement ou du déchiffrement.

Chapitre 8

Signature digitale

L'absence de système de signature digitale efficace a certainement été un frein à l'utilisation des cryptosystèmes basés sur les codes correcteurs d'erreurs. Dans un travail récent avec Matthieu Finiasz et Nicolas Courtois, nous proposons une solution à ce problème [CFS01].

8.1 Introduction

Une signature digitale est une information de petite taille, dépendant d'un message et d'un utilisateur (le signataire). Il est possible de construire un schéma de signature à partir d'un système de chiffrement à clé publique.

- Soient X l'ensemble des textes clairs, Y l'ensemble des textes chiffrés, $E : X \to Y$ la fonction de chiffrement et $D : Y \to X$, telle que $D \circ E = id$, la fonction de déchiffrement.
- Soient $\mathcal{M}$ l'ensemble des messages et $h : \mathcal{M} \to Y$ une fonction de hachage publique à sens unique et sans collision (*i.e.* trouver un couple (M, M') tel que $h(M) = h(M')$ est calculatoirement difficile).
- La signature d'un message M sera $s = D(h(M))$ que seul le signataire doit être à même de produire mais qui peut être universellement vérifiée à l'aide de l'identité $h(M) = E(s)$.

Notons que pour calculer $D(h(M))$ il est suffisant mais non nécessaire de connaître la fonction de déchiffrement $D()$. Si, en plus de son caractère « sans collisions », l'on fait l'hypothèse que la fonction $h()$ est indépendante des fonctions $E()$ et $D()$ (c'est-à-dire de la clé publique) et ne dépend que de la famille dans laquelle la paire (E, D) est choisie, alors le calcul de $D(h(M))$ devient aussi difficile que celui de $D(y)$ pour un y arbitraire (modèle dit de « l'oracle aléatoire » [BR96]).

8.2 Signature basée sur les codes

Fonder un schéma de signature sur le système de chiffrement de McEliece (ou de Niederreiter) pose un problème difficile ; la fonction de hachage $h()$ dont il est question plus haut doit fournir un texte chiffré et ne peut donc pas être indépendante de la clé publique retenue

(l'espace de texte chiffré dépend de cette clé). En pratique, lorsque $E()$ et $D()$ sont respectivement les fonctions de chiffrement et de déchiffrement d'un système à clé publique basé sur les codes, nous n'avons pas connaissance d'un procédé qui permette de produire pour tout message M un cryptogramme $h(M)$ autrement qu'en écrivant explicitement $h(M) = E(g(M))$ (*i.e.* $g = D \circ h$), où $g()$ est une fonction impossible à cacher.

8.2.1 Un procédé de signature

Nous allons considérer $E()$ et $D()$ des fonctions de chiffrement et de déchiffrement d'une instance du cryptosystème de Niederreiter (n et k seront la longueur et la dimension du code utilisé). Nous noterons $h()$ une fonction de hachage sans collision prenant en argument une séquence binaire de longueur quelconque et à valeur dans $\{0,1\}^{n-k}$. Nous produisons une signature de la façon suivante :

1. pour tout $i \geq 0$, soit $y_i = h(M \oplus i) \in \{0,1\}^{n-k}$ (où $\oplus$ est l'addition bit-à-bit modulo 2),
2. soit i_0 le plus petit entier tel que y_{i_0} puisse être déchiffré, nous notons $x_{i_0} = D(y_{i_0})$,
3. la signature de M est égale à $s = (x_{i_0}, i_0)$.

La fonction de déchiffrement $D()$ est nécessaire pour produire s, par contre vérifier la validité de la signature (x, i) d'un message M ne requiert que l'identité $h(M \oplus i) = E(x)$. Dans le cas des paramètres originaux de McEliece ($m = 10$, $n = 1024$, $k = 524$ et $t = 50$), la probabilité pour qu'un syndrome choisi au hasard dans $\mathbf{F}_2^{n-k}$ soit celui d'un mot de poids t est $\binom{n}{t}/2^{n-k} \approx 2^{-216}$. Clairement, le procédé de signature est loin d'être satisfaisant dans ce cas.

8.2.2 Choix des paramètres

Comme nous le montrons dans [CFS01], il est possible de choisir des paramètres pour lesquels le coût du procédé de signature est raisonnable tout en conservant un niveau de sécurité suffisant. Pour un code de Goppa binaire de longueur $n = 2^m$ et correcteur de t erreurs avec $t \ll n$, la proportion de syndromes « décodables » est d'environ $1/t!$. Le coût de signature devient ainsi raisonnable lorsque t ne dépasse pas 9. La Table 8.1 donne le coût d'une

$n = 2^m$	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
$t = 9$	$2^{54.6}$	$2^{59.9}$	$2^{69.3}$	$2^{74.0}$	$2^{78.8}$	$2^{83.7}$	$2^{88.2}$
$t = 10$	$2^{60.9}$	$2^{66.8}$	$2^{72.3}$	$2^{77.4}$	$2^{87.4}$	$2^{90.9}$	$2^{94.6}$

TAB. 8.1: *Facteur de travail binaire pour le décodage complet d'un code binaire* $(n, n - tm)$

cryptanalyse par l'algorithme de Canteaut-Chabaud pour diverses valeurs des paramètres t et n. La meilleure attaque structurelle a un coût de l'ordre de $tm2^{(t-2)m}$ qui est toujours supérieure.

8.2.3 Mise en œuvre

En choisissant les paramètres $t = 9$ et $m = 16$, nous arrivons à un niveau de sécurité suffisant. Il subsiste toutefois deux défauts, une clé publique de grande taille (environ 1 Mo)

et un temps de signature d'une à deux minutes[1]. Par contre la vérification est rapide et les signatures obtenues sont très courtes (de l'ordre d'une centaine de bits). Dans la Table 8.2 sont indiquées les caractéristiques principales du système. Divers compromis sont possibles

coût de la signature	$t!t^2m^2$
longueur de la signature	$(t-1)m + \log_2 t$
coût de la vérification	t^2m
taille de la clé publique	$tm2^m$
coût de l'attaque par décodage [(a)]	$2^{tm(1/2+o(1))}$
coût de l'attaque structurelle	$tm2^{m(t-2)}$

[(a)] À l'aide du « split syndrome decoding » (*cf.* [Bar98]).

TAB. 8.2: *Schéma de signature basé sur des codes de Goppa binaires* ($n = 2^m, k = n - tm, d \geq 2t + 1$)

entre la longueur de la signature et le coût de la vérification ; dans la table nous indiquons la version offrant la vérification la plus rapide. Les paramètres que nous proposons sont $t = 9$ et $m = 16$. Pour ces valeurs, la longueur de la signature est de 132 bits en moyenne et il est possible de la réduire à 80 bits environ, mais pour un temps de vérification de l'ordre de la minute[1]. Les temps de calculs donnés pour la vérification et la signature sont susceptibles d'être améliorés.

1. Les temps sont donnés pour une implémentation logicielle en C sur une machine moyenne : processeur Intel PIII 1 Ghz

Chapitre 9

Preuve de sécurité

Dans ce chapitre, nous nous proposons de montrer comment la sécurité des système de chiffrement utilisant les codes de Goppa peut être formellement réduite à la difficulté de deux problèmes clairement identifiés. Il s'agit du caractère pseudo-aléatoire des codes de Goppa et de la difficulté en moyenne du décodage d'un code linéaire.

9.1 Distingabilité des codes de Goppa

Nous noterons $\mathcal{G}_{n,t}$, avec $n = 2^m$, l'ensemble des matrices de parité des codes de Goppa ayant $\mathbf{F}_{2^m}$ comme support et comme générateur un polynôme de degré t. Nous noterons $\mathcal{M}_{r,n}$ l'ensemble des matrices binaires $r \times n$. Notons que $\mathcal{G}_{n,t}$ est inclus dans $\mathcal{M}_{tm,n}$.

Définition 21 *Un distingueur de codes de Goppa $\mathcal{D}$ est une machine de Turing probabiliste dont l'entrée est une matrice binaire et dont la sortie vaut* 0 *ou* 1*. Pour tous entiers positifs t, m, $n = 2^m$ et $r = tm$, nous notons $T_{\mathcal{D}}(n,t)$ le temps d'exécution maximal pour une entrée dans $\mathcal{M}_{tm,n}$ et nous appelons* avantage *de $\mathcal{D}$ le nombre*

$$Adv_{\mathcal{D}}(n,t) = \left| Prob\left[\mathcal{D}(H) = 1 \mid H \in \mathcal{M}_{r,n}\right] - Prob\left[\mathcal{D}(H) = 1 \mid H \in \mathcal{G}_{n,t}\right]\right|.$$

L'avantage d'un distingueur de codes de Goppa est ainsi une mesure de sa capacité à faire la différence entre un code linéaire quelconque et un code Goppa. Bien sûr l'avantage doit être mis en rapport avec le temps d'exécution. Il peut être significatif et le temps d'exécution grand (exponentiel), ou bien le temps d'exécution peut être faible (polynomial) et l'avantage négligeable. Dans les deux cas il s'agira de distingueurs peu discriminants.

Le meilleur distingueur de codes de Goppa connu ayant un avantage proche de 1 est basé sur l'algorithme de séparation du support et possède un temps d'exécution exponentiel (proportionnel à n^t/tm). De même, nous n'avons pas connaissance d'un distingueur dont le temps d'exécution serait polynomial en n et qui distinguerait plus d'une proportion exponentiellement faible de codes de Goppa.

Nous conjecturons qu'il n'existe pas de « bons » distingueurs de codes de Goppa.

Conjecture 22 *Pour tout distingueur de code de Goppa $\mathcal{D}$, le rapport $T_{\mathcal{D}}(n,t)/Adv_{\mathcal{D}}(n,t)$ ne peut pas être borné supérieurement par un polynôme en n et t.*

Une autre façon d'exprimer cette conjecture est de dire que le problème suivant est difficile en moyenne

Problème 23 (Goppa Code Distinguishing - GD)

Instance : *Une matrice $r \times n$ binaire H.*

Question : *H appartient-t'elle à $\mathcal{G}_{n,t}$?*

Le problème GD est clairement NP, car vérifier qu'une matrice H est bien une matrice de parité d'un code de Goppa $\Gamma(L, g)$ avec L et g connus peut être effectué en temps polynomial. En revanche aucun élément ne nous permet de trancher sur la question de savoir s'il est complet.

9.2 Difficulté du décodage des codes linéaires

Pour tous entiers $r = tm$ et $n = 2^m$, une paire matrice/syndrome (H, s) de paramètres (r, n) est constituée d'une matrice $r \times n$ binaire H et d'un mot s de $\mathbf{F}_2^r$. Nous notons $W_{n,t}$ l'ensemble des mots de $\mathbf{F}_2^n$ de poids t.

Définition 24 *Un décodeur borné par les paramètres de Goppa (GB-décodeur) φ est une machine de Turing probabiliste dont l'entrée est une paire matrice/syndrome et dont la sortie vaut 0 ou 1. Pour tous entiers positifs t, m, $n = 2^m$ et $r = tm$, nous notons $T_\varphi(n, t)$ le temps d'exécution maximal pour une entrée de paramètres (tm, n) et nous appelons* avantage *de φ le nombre*

$$Adv_\varphi(n,t) = \big|Prob\left[\varphi(H, He^T) = 1 \mid H \in \mathcal{M}_{tm,n}, e \in W_{n,t}\right] - Prob\left[\varphi(H, s) = 1 \mid H \in \mathcal{M}_{tm,n}, s \in \mathbf{F}_2^r\right]\big| .$$

Les meilleurs GB-décodeurs connus pour un taux fixé $R = tm/n$ ont un temps d'exécution proportionnel à $\exp(cn/m(1 + o(1)))$, avec $c = -(1 - R)\ln(1 - R)$. Pour des codes dont le taux de transmission est proche de 1 (codes utiles pour le schéma de signature), le temps d'exécution est proportionnel à $\exp(tm/2\ln(2)(1 + o(1)))$.

A - Problèmes NP liés au décodage

Les instances des systèmes de chiffrement de McEliece ou de Niederreiter peuvent être réduites à des instances du problème SD (Syndrome Decoding) ci-dessous, qui est NP-complet [BMvT78].

Problème 25 (Syndrome Decoding - SD)

Instance : *Une matrice $r \times n$ binaire H, un mot s de $\mathbf{F}_2^r$ et un entier $w > 0$.*

Question : *Existe-t'il un mot x dans $\mathbf{F}_2^n$ de poids $\leq w$ tel que $Hx^T = s$?*

Pour garantir l'inversibilité du chiffrement il est nécessaire de se restreindre à des instances particulières de SD :

Problème 26 (Bounded-Distance Decoding - BDD)

Instance : *Une matrice $r \times n$ binaire H et un mot s de $\mathbf{F}_2^r$.*

Promesse : *Tout ensemble de $d-1$ colonnes de H est linéairement indépendant.*

Question : *Existe-t'il un mot x dans $\mathbf{F}_2^n$ de poids $< d/2$ tel que $Hx^T = s$?*

Parce que le prédicat de la promesse est NP-complet, le problème BDD n'est probablement pas NP [Var97], cependant il est conjecturé comme étant NP-dur [Var97, Bar98]. De nouveau ce problème est trop général dans notre contexte. En pratique les instances de chiffrement sont construites à l'aide de famille particulières de codes, et le nombre d'erreurs à corriger lors du déchiffrement dépend généralement d'une distance construite plutôt que de la distance minimale. Pour les codes de Goppa nous utilisons le problème suivant :

Problème 27 (Goppa Parameterized Bounded Decoding - GPBD)

Instance : *Une matrice $r \times n$ binaire H et un mot s de $\mathbf{F}_2^r$.*

Question : *Existe-t'il un mot x dans $\mathbf{F}_2^n$ de poids $\leq r/\log_2 n$ tel que $Hx^T = s$?*

Ce problème est NP, mais malheureusement il est difficile d'en dire plus dans l'état actuel des connaissances. Construire un GB-décodeur efficace est un problème très voisin de celui de résoudre une instance moyenne (pour une distribution uniforme) de GPBD. Nous sommes aussi intéressé à savoir si les problèmes basés sur les codes, et en particulier GPBD, sont ou non difficiles *en moyenne* (cf. [Lev86, Gur91]) plutôt que seulement dans le pire cas. Ce problème est ouvert [Bar98, p. 747].

9.2.1 Réduction de la sécurité

Nous établissons ici que si un programme est capable de casser efficacement le système de chiffrement de McEliece, alors ce programme peut être utilisé pour construire soit un distingueur de codes de Goppa soit un GB-décodeur efficace.

Définition 28 *Un attaquant $\mathcal{A}$ est une machine de Turing probabiliste prenant en entrée une paire matrice/syndrome (H, s). Pour tous entiers m, t et $n = 2^m$, nous noterons $T(n,t)$ le temps d'exécution pour toute entrée de paramètres (tm, n). La* probabilité de succès *de A est définie par*

$$P_{\mathcal{A}}(n,t) = Prob\left[\mathcal{A}(H, He^T) = e \mid H \in \mathcal{G}_{n,t}, e \in W_{n,t}\right].$$

Proposition 29 *Soient $t \geq 2$, m et $n = 2^m$ des entiers positifs. S'il existe un attaquant $\mathcal{A}$ fonctionnant en temps T pour toute entrée dans $\mathcal{M}_{tm,n} \times \mathbf{F}_2^{tm}$ avec une probabilité de succès ε, alors il existe soit un distingueur de codes de Goppa soit un GB-décodeur fonctionnant en temps $\leq T + \delta$ – où δ est borné supérieurement par un polynôme en n – pour toute entrée dans $\mathcal{M}_{tm,n} \times \mathbf{F}_2^{tm}$ et $\mathcal{M}_{tm,n}$ respectivement, avec un avantage au moins égal à $\varepsilon/3$.*

Preuve : À partir de $\mathcal{A}$ nous construisons le GB-décodeur

$$\varphi(H,s) = \begin{cases} 1 & \text{si } He^T = s \text{ et } w_H(e) = t, \text{ où } e = \mathcal{A}(H,s), \\ 0 & \text{sinon,} \end{cases}$$

et le distingueur de codes de Goppa

$\mathcal{D}$: `input` H
$e \leftarrow random(W_{n,t})$
`return` $\varphi(H, He^T)$

où $random(W_{n,t})$ retourne un élément de $W_{n,t}$ choisi selon la loi de distribution uniforme. Clairement ces deux programmes fonctionnent en un temps qui n'excède pas T de plus d'une petite quantité, bornée par un polynôme en n. Nous noterons

$$p = Prob\left[\mathcal{A}(H, He^T) = e \mid H \in \mathcal{M}_{tm,n}, e \in W_{n,t}\right].$$

1. Avantage de $\mathcal{D}$. Il est égal à la différence des deux nombres suivants

$$\begin{aligned} Prob\left[\mathcal{D}(H) = 1 \mid H \in \mathcal{G}_{n,t}\right] &= Prob\left[\mathcal{A}(H, He^T) = e \mid H \in \mathcal{G}_{n,t}, e \in W_{n,t}\right], \\ Prob\left[\mathcal{D}(H) = 1 \mid H \in \mathcal{M}_{tm,n}\right] &= Prob\left[\mathcal{A}(H, He^T) = e \mid H \in \mathcal{M}_{tm,n}, e \in W_{n,t}\right], \end{aligned}$$

et donc $Adv_{\mathcal{D}} = |p - \varepsilon|$.

2. Avantage de φ. Il est égal à la différence des deux nombres suivants

$$\begin{aligned} Prob\left[\varphi(H, He^T) = 1 \mid H \in \mathcal{M}_{tm,n}, e \in W_{n,t}\right] &= p, \\ Prob\left[\varphi(H, s) = 1 \mid H \in \mathcal{M}_{tm,n}, s \in \mathbf{F}_2^{tm}\right] &= \frac{\binom{n}{t}}{n^t} p, \end{aligned}$$

La première égalité est la définition de p. La seconde est due au fait que les seules paires (H, s) vérifiant $\varphi(H, s) = 1$ sont telles que $s = He^T$ pour un certain e dans $W_{n,t}$.

3. Finalement, puisque $t \geq 2$ nous avons $1 - \binom{n}{t}/n^t > 1/2$, si $p \geq 2\varepsilon/3$ alors $Adv_{\varphi} > \varepsilon/3$. Si au contraire $p < 2\varepsilon/3$ alors $Adv_{\mathcal{D}} = |\varepsilon - p| > \varepsilon/3$.

La conséquence principale de ce résultat est que si les problèmes 23 et 27 sont difficiles *en moyenne*, alors les cryptosystèmes à clé publiques basés sur les codes correcteurs d'erreurs possèdent de forts arguments de sécurité.

Bibliographie

[AK90] E. F. Assmus, Jr et J. D. Key. « Affine and projective planes ». *Discrete Mathematics*, 83:161–187, 1990.

[AK92] E. F. Assmus, Jr et J. D. Key. *Designs and their Codes*. Cambridge University Press, 1992.

[AM87] C. Adams et H. Meijer. « Security-related comments regarding McEliece's public-key cryptosystem ». Dans C. Pomerance, éditeur, *Advances in Cryptology - CRYPTO'87*, numéro 293 dans LNCS, pages 224–228. Springer-Verlag, 1987.

[Bar98] A. Barg. Complexity issues in coding theory. Dans V. S. Pless et W. C. Huffman, éditeurs, *Handbook of Coding theory*, volume I, Chapitre 7, pages 649–754. North-Holland, 1998.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, et P. Rogaway. « Relations Among Notions of Security for Public-Key Encryption Schemes ». Dans H. Krawczyk, éditeur, *CRYPTO'98*, numéro 1462 dans LNCS, pages 26–46. Springer-Verlag, 1998.

[Ber68] E. R. Berlekamp. *Algebraic Coding Theory*. Aegen Park Press, 1968.

[Ber96] T. Berger. « *Groupes d'automorphismes et de permutations des codes affine-invariants* ». Habilitation à diriger des recherches, Université de Limoges, janvier 1996.

[Ber97] T. Berson. « Failure of the McEliece Public-Key Cryptosystem Under Message-Resend and Related-Message Attack ». Dans B. Kalisky, éditeur, *CRYPTO'97*, numéro 1294 dans LNCS, pages 213–220. Springer-Verlag, 1997.

[BMvT78] E. R. Berlekamp, R. J. McEliece, et H. C. van Tilborg. « On the inherent intractability of certain coding problems ». *IEEE Transactions on Information Theory*, 24(3), mai 1978.

[BN90] J. Bruck et M. Naor. « The hardness of decoding linear codes with preprocessing ». *IEEE Transactions on Information Theory*, 36(2):381–385, mars 1990.

[BR96] M. Bellare et P. Rogaway. « The exact security of digital signatures how to sign with RSA and Rabin ». Dans U. Maurer, éditeur, *EUROCRYPT'96*, numéro 1070 dans LNCS, pages 399–416. Springer-Verlag, 1996.

[Bre76] R. P. Brent. The Complexity of Multiple-Precision Arithmetic. Dans R. S. Anderssen et R. P. Brent, éditeurs, *Complexity of Computational Problem Solving*. Univ. of Queensland Press, Brisbane, 1976.

[Can95] A. Canteaut. « A new algorithm for finding minimum-weight words in large linear codes ». Dans Colin Boyd, éditeur, *Cryptography and Coding ; proceedings of the 5th IMA conference*, numéro 1025 dans LNCS, pages 205–212. Springer-Verlag, décembre 1995.

[Can96] A. Canteaut. « *Attaques de cryptosystèmes à mots de poids faible et construction de fonction t-résilientes* ». Thèse de doctorat, Université Paris 6, octobre 1996.

[CC93] H. Chabanne et B. Courteau. « Application de la méthode de décodage itérative d' Omura à la cryptanalyse du système de McEliece ». Rapport de recherche 122, Université de Sherbrooke, Canada, octobre 1993.

[CC94] A. Canteaut et H. Chabanne. « A Further Improvement of the Work Factor in an Attempt at Breaking McEliece Cryptosystem ». Dans P. Charpin, éditeur, *Livre des résumé – EUROCODE 94*, pages 163–167, Abbaye de la Bussière sur Ouche, France, octobre 1994. INRIA.

[CC98] A. Canteaut et F. Chabaud. « A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511 ». *IEEE Transactions on Information Theory*, 44(1):367–378, janvier 1998.

[CFS01] N. Courtois, M. Finiasz, et N. Sendrier. « How to achieve a McEliece-based Digital Signature Scheme ». Dans *Asiacrypt 2001*, numéro 2248 dans LNCS, pages 157–174. Springer-Verlag, 2001.

[Com74] L. Comtet. *Advanced Combinatorics*. Reidel, Dordrecht, 1974.

[CS98] A. Canteaut et N. Sendrier. « Cryptanalysis of the original McEliece cryptosystem ». Dans *Advances in Cryptology - ASIACRYPT'98*, numéro 1514 dans LNCS, pages 187–199. Springer-Verlag, 1998.

[Gib91] J. K. Gibson. « Equivalent Goppa codes and trapdoors to McEliece's public key cryptosystem ». Dans D. W. Davies, éditeur, *Advances in Cryptology - EUROCRYPT'91*, numéro 547 dans LNCS, pages 517–521. Springer-Verlag, 1991.

[Gir90] M. Girault. « A (non-practical) three-pass identification protocol using coding theory ». Dans J. Seberry et J. Pieprzyk, éditeurs, *Advances in Cryptology - AUSCRYPT'90*, numéro 453 dans LNCS, pages 265–272. Springer-Verlag, 1990.

[GPT91] E. Gabidulin, A. Paramonov, et O. Tretjakov. « Ideals over a non-commutative ring and their application to cryptology ». Dans D. W. Davies, éditeur, *Advances in Cryptology - EUROCRYPT'91*, numéro 547 dans LNCS, pages 482–489. Springer-Verlag, 1991.

[Gur91] Y. Gurevich. « Average case completeness ». *Journal of Computer and System Sciences*, 42(3):346–398, 1991.

[KI01] K. Kobara et H. Imai. « Semantically Secure McEliece Public-Key Cryptosystems -Conversions for McEliece PKC- ». Dans K. Kim, éditeur, *PKC'2001*, numéro 1992 dans LNCS, pages 19–35. Springer-Verlag, 2001.

[LB88] P. J. Lee et E. F. Brickell. « An observation on the security of McEliece's public-key cryptosystem ». Dans C. G. Günther, éditeur, *Advances in Cryptology – EUROCRYPT'88*, numéro 330 dans LNCS, pages 275–280. Springer-Verlag, 1988.

[LDW94] Y. X. Li, R. H. Deng, et X. M. Wang. « On the Equivalence of McEliece's and Niederreiter's Public-Key Cryptosystems ». *IEEE Transactions on Information Theory*, 40(1):271–273, janvier 1994.

[Leo82] J. S. Leon. « Computing Automorphism Groups of Error-Correcting Codes ». *IEEE Transactions on Information Theory*, 28(3):496–511, mai 1982.

[Leo88] J. S. Leon. « A Probabilistic Algorithm for Computing Minimum Weights of Large Error-Correcting Codes ». *IEEE Transactions on Information Theory*, 34(5):1354–1359, septembre 1988.

[Lev86] L. Levin. « Average case complete problems ». *SIAM Journal on Computing*, 15(1):285–286, 1986.

[LN83] R. Lidl et H. Niederreiter. *Finite Fields*. Cambridge University Press, 1983.

[LS01] P. Loidreau et N. Sendrier. « Weak keys in McEliece public-key cryptosystem ». *IEEE Transactions on Information Theory*, 47(3):1207–1212, avril 2001.

[Mas69] J. Massey. « Shift-Register Synthesis and BCH Decoding ». *IEEE Transactions on Information Theory*, 15(1):122–127, janvier 1969.

[McE78] R. J. McEliece. « A public-key cryptosystem based on algebraic coding theory ». *DSN Prog. Rep.,* Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pages 114–116, janvier 1978.

[Mon87] A. Montpetit. « Note sur la notion d'équivalence entre deux codes linéaires ». *Discrete Mathematics*, 65:177–185, 1987.

[MS77] F. J. MacWilliams et N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[MvOV89] A. Menezes, P. van Oorschot, et S. Vanstone. « Some computational aspects of root finding in $GF(q^m)$ ». Dans P. Gianni, éditeur, *ISSAC'88*, numéro 358 dans LNCS, pages 259–270. Springer, 1989.

[Nie86] H. Niederreiter. « Knapsack-type crytosystems and algebraic coding theory ». *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.

[Pat75] N. J. Patterson. « The algebraic decoding of Goppa codes ». *IEEE Transactions on Information Theory*, 21(2):203–207, mars 1975.

[Pat91] J. Patarin. « *Étude des Générateurs de Permutations pseudo-aléatoires basés sur le schéma du D.E.S.* ». Thèse de doctorat, Université Paris 6, novembre 1991. (in French).

[Pie90] J. Pieprzyk. « How to Construct Pseudorandom Permutations from Single Pseudorandom Functions ». Dans *Advances in Cryptology - EUROCRYPT'90*, numéro 437 dans LNCS, pages 140–150. Springer-Verlag, 1990.

[Ple65] V. Pless. « The number of isotropic subspaces in a finite geometry ». *Rend. Sc. Fis. Mat. e Nat., Accad. Naz. Lincie, Ser. VIII*, 39:418–421, décembre 1965.

[PR97] E. Petrank et R. M. Roth. « Is Code Equivalence Easy to Decide? ». *IEEE Transactions on Information Theory*, 43(5):1602–1604, septembre 1997.

[RS85] R. M. Roth et G. Seroussi. « On generator matrices of MDS Codes ». *IEEE Transactions on Information Theory*, 31(6):826–830, mai 1985.

[RSA78] R. L. Rivest, A. Shamir, et L. M. Adleman. « A Method for Obtaining Digital Signatures and Public-Key Cryptosystems ». *Communications of the ACM*, 21(2):120–126, février 1978.

[Sen91] N. Sendrier. « *Codes Correcteurs d'Erreurs à Haut Pouvoir de Correction* ». Thèse de doctorat, Université Paris 6, décembre 1991.

[Sen94] N. Sendrier. « On the structure of a randomly permuted concatenated code ». Dans P. Charpin, éditeur, *Livre des résumé – EUROCODE 94*, pages 169–173, Abbaye de la Bussière sur Ouche, France, octobre 1994. INRIA.

[Sen97a] N. Sendrier. « Finding the Permutation between Equivalent Binary Codes ». Dans *IEEE Conference, ISIT'97*, Ulm, Germany, juin 1997.

[Sen97b] N. Sendrier. « On the dimension of the hull ». *SIAM Journal on Discrete Mathematics*, 10(2):282–293, mai 1997.

[Sen98] N. Sendrier. « On the Concatenated Structure of a Linear Code ». *AAECC*, 9(3):221–242, 1998.

[Sen00] N. Sendrier. « Finding the permutation between equivalent codes: the support splitting algorithm ». *IEEE Transactions on Information Theory*, 46(4):1193–1203, juillet 2000.

[Sid94] V. M. Sidel'nikov. « A public-key cryptosystem based on Reed-Muller codes ». *Discrete Mathematics and Applications*, 4(3):191–207, 1994.

[Ske99] G. Skersys. « *Calcul du groupe d'automorphisme des codes. Détermination de l'équivalence des codes* ». Thèse de doctorat, Université de Limoges, octobre 1999.

[SP92] V. M. Sidelnikov et A. S. Pershakov. « Decoding of Reed-Muller Codes with a Large Number of Errors ». *Problems of Information Transmission*, 28(3):269–281, 1992.

[SS92] V. M. Sidel'nikov et S. O. Shestakov. « On cryptosystem based on generalized Reed-Solomon codes ». *Discrete mathematics (in russian)*, 4(3):57–63, 1992.

[SS99] N. Sendrier et G. Skersys. « Permutation groups of error-correcting codes ». Dans *WCC'99, Book of abstracts*, pages 33–42, Paris, France, janvier 1999.

[SS01] N. Sendrier et G. Skersys. « On the Computation of the Automorphism Group of a Linear Code ». Dans *IEEE Conference, ISIT'2001*, Washington D.C., USA, juin 2001.

[Ste89] J. Stern. « A method for finding codewords of small weight ». Dans G. Cohen et J. Wolfmann, éditeurs, *Coding theory and applications*, numéro 388 dans LNCS, pages 106–113. Springer-Verlag, 1989.

[Var97] A. Vardy. « The Intractability of Computing the Minimum Distance of a Code ». *IEEE Transactions on Information Theory*, 43(6):1757–1766, novembre 1997.

[vT90] J. van Tilburg. « On the McEliece cryptosystem ». Dans S. Goldwasser, éditeur, *Advances in Cryptology - CRYPTO'88*, numéro 403 dans LNCS, pages 119–131. Springer-Verlag, 1990.