Chapitre II

Le problème SD et sa cryptanalyse

"Il n'y a pas de problèmes ; il n'y a que des solutions. L'esprit de l'homme invente ensuite le problème."

A. Gide

Ce chapitre décrit un problème issue de la théorie du codage, celui du décodage du syndrome (problème SD). Ce dernier appartient à la classe des problèmes $\mathcal{NP}$-complets. Cependant, il existe des algorithmes probabilistes, capables d'en résoudre certaines instances. Une description de ces algorithmes est donnée, ainsi que les paramètres pour lesquels ils deviennent inefficaces. Ceci permettra de fixer le seuil de sécurité des schémas d'identification décrits au chapitre iii. En premier lieu, quelques notions sur la théorie de la complexité sont rappelées.

LA théorie de la complexité fournit une méthode pour analyser la complexité de calcul de différents algorithmes. Dans le domaine de la cryptographie, elle permet de déterminer le niveau de sécurité d'un protocole. La théorie de l'information nous apprend que tous les algorithmes cryptographiques peuvent être cassés (à une exception près). La théorie de la complexité nous indique si la seule solution pour assister au succès d'une attaque est la cryogénisation !

§1. Quelques notions de théorie de la complexité

L'efficacité d'un protocole cryptographique est déterminée par la puissance de calcul nécessaire pour qu'une éventuelle attaque aboutisse. La théorie de la complexité détermine les temps et espaces minimums nécessaires pour résoudre l'instance la plus difficile du problème sur un ordinateur théorique connu sous le nom de machine de Turing. Il s'agit d'une machine à états finis avec une mémoire en écriture et en lecture sous la forme d'un ruban infini. Il s'avère que la machine de Turing est un modèle réaliste de calcul.

Les différents problèmes à analyser sont classés d'après les algorithmes nécessaires pour les résoudre.

La classe $\mathcal{P}$ est définie comme étant l'ensemble des problèmes qui peuvent être résolus en temps polynomial par une machine de Turing déterministe. C'est la classe des problèmes solubles.

La classe $\mathcal{NP}$ est définie comme l'ensemble des problèmes pouvant être résolus en temps polynomial par une machine de Turing non déterministe. Lors de la recherche d'une solution d'un problème, face à deux alternatives différentes, une machine de Turing non déterministe peut créer une copie d'elle même et continuer sa recherche en parcourant les deux voies possibles (principe du *backtrack*). Ce procédé peut aboutir à un nombre exponentiel de copies. De plus la machine doit vérifier la validité de chaque alternative en un temps polynomial. Ainsi pour montrer qu'un problème D est dans la classe $\mathcal{NP}$, il suffit d'exhiber un algorithme contenant trois modules distincts :

- un module de génération de tous les candidats solutions de D (module non déterministe),
- un module de choix d'un candidat (module de divination),
- un module de vérification de la validité du candidat choisi (module déterministe),

Si le module non déterministe est de complexité en $\mathcal{O}(2^{p(n)})$, où n est la longueur d'une instance du problème D et p un polynôme, et si de plus la complexité du module de vérification est en $\mathcal{O}(n^d)$ ($d \in \mathbb{N}^\times$), alors D est dans la classe des problèmes $\mathcal{NP}$.

La classe $\mathcal{NP}$ contient la classe $\mathcal{P}$ parce que tout problème soluble en temps polynomial sur une machine de Turing déterministe, est aussi soluble en temps polynomial sur une machine de Turing non déterministe car l'étape de divination peut tout simplement être supprimée. On ne sait pas si $\mathcal{P} = \mathcal{NP}$, mais la plupart des chercheurs, en informatique théorique, pensent que $\mathcal{P}$ et $\mathcal{NP}$ ne représentent pas la même classe. Intuitivement la classe $\mathcal{P}$ est composée de problèmes qui peuvent être résolus rapidement. La classe $\mathcal{NP}$ est composée de problèmes pour lesquels une solution peut être rapidement validée.

En 1971, Cook montre que le problème SAT ("Etant donné une formule booléenne propositionnelle, y a-t-il un moyen d'assigner des valeurs de vérité aux variables de telle manière que la formule soit vraie ?") possède la propriété suivante : la résolution de tout problème D de la classe $\mathcal{NP}$ se ramène à la résolution du problème SAT. Ainsi, s'il existe un algorithme polynomial pour résoudre le problème SAT, alors cet algorithme peut être modifié afin de résoudre en temps polynomial le problème D. Intuitivement, un problème D peut être ramené à un autre problème D' si une instance quelconque de D peut être "facilement reformulée" comme une instance de D', dont la solution fournira une solution pour l'instance de D. Par exemple, le problème de la résolution d'équations linéaires à une inconnue x peut se réduire au problème de la résolution d'équations quadratiques. Etant donnée une instance $ax + b = 0$, on la transforme en $0x^2 + ax + b = 0$, dont la solution fournit une solution à $ax + b = 0$. Donc, si un problème D se réduit à un problème D', alors D n'est, dans un certain sens, "pas plus difficile à résoudre" que D'.

Dans [GaJo], Garey et Johnson dressent la liste de plus de trois cents problèmes dont la résolution est équivalente à la résolution du problème SAT. C'est la classe des problèmes $\mathcal{NP}$-complets. Pour montrer qu'un problème D est $\mathcal{NP}$ complet, il faut :

- Démontrer que D est dans la classe $\mathcal{NP}$,
- Choisir un problème connu D' connu pour être $\mathcal{NP}$-complet,
- Réduire D en D' en un temps polynomial.

La raison qui pousse le plus les théoriciens de l'informatique à croire que $\mathcal{NP} \neq \mathcal{P}$ est sans doute l'existence de la classe des problèmes $\mathcal{NP}$-complets. Il n'est pas exclus que quelqu'un fournisse un jour un algorithme polynomial pour un problème $\mathcal{NP}$-complet, démontrant du même coup que $\mathcal{P} = \mathcal{NP}$. Néanmoins, comme aucun algorithme de ce type n'a encore été découvert pour aucun problème $\mathcal{NP}$-complet, démontrer qu'un problème est dans cette classe donne un excellent indice, théorique, de son intraitabilité. Cependant, dans un cadre pratique, les instances d'un même problème peuvent donner lieu à des problèmes de recherche plus ou moins difficile.

1.1. Les problèmes de décision

Pour des raisons de simplicité, la théorie de la $\mathcal{NP}$-complétude est restreinte aux problèmes de décision : ceux qui ont pour solution *oui/non*. Dans ce cas, on peut voir un problème de décision, comme une fonction qui fait correspondre l'ensemble des instances I, d'un problème, à l'ensemble des solutions $\{0, 1\}$. Un problème de décision se présente toujours de la même manière. Voici un exemple pour le problème SAT :

Nom : SAT
Entrée : C est une collection de clauses sur un ensemble fini de variables.
Question : C est-elle satisfaisable ?

De nombreux problèmes ne sont pas des problèmes de décision, mais des problèmes d'optimisation, dans lesquels une certaine valeur doit être minimisée ou maximisée. Pour appliquer la théorie de la $\mathcal{NP}$-complétude aux problèmes d'optimisation, il faut les reformuler pour en faire des problèmes de décision. Le plus souvent, il suffit d'imposer une borne sur la valeur à optimiser (ce qui est le cas pour le problème SD). Bien que la théorie de la $\mathcal{NP}$-complétude nous impose de transformer les problèmes d'optimisation en problèmes de décision, cette contrainte ne diminue pas l'impact de la théorie. De manière générale, si l'on peut résoudre rapidement un problème d'optimisation, on peut aussi résoudre rapidement son problème de décision associé. On compare simplement la valeur obtenue avec la borne fournie en entrée du problème de décision. Ainsi si l'on peut convaincre qu'un problème de décision est difficile, on convainc du même coup de la difficulté du problème d'optimisation associé.

§2. Le problème SD

Soit $G(k,n)$ *(resp. $H(n-k,n)$)* la matrice génératrice *(resp. la matrice de contrôle)* d'un code linéaire binaire C, t-correcteur. Soient c un mot du code C et y un mot de la forme $c+e$, où $\omega(e) \leqslant t$. Soit z le syndrome du mot y, alors $z = Hy^{\mathrm{t}} = He^{\mathrm{t}}$. De plus, $\{x \in \mathbb{F}_{2^n}, Hx^{\mathrm{t}} = He^{\mathrm{t}}\} = \{c+e, c \in C\}$. Or

$$\forall c \in C, (c \neq 0), \forall e \text{ telque } \omega(e) \leqslant t, \qquad \omega(c+e) > \omega(e).$$

Ainsi, connaissant H et le mot y, la recherche de la quantité c revient à trouver la solution de poids minimum de l'équation $z = Hy^{\mathrm{t}}$. C'est donc un problème d'optimisation que l'on transforme en problème de décision en imposant une borne sur le poids de la solution recherchée. On obtient alors le problème de décision suivant :

Nom : SD
Entrée : $H(k,n)$ une matrice binaire de rang k, i un vecteur binaire de longueur k et p un entier.
Question : Existe-t-il un vecteur e de longueur n tel que $He^{\mathrm{t}} = i$ et $\omega(e) \leqslant p$?

THÉORÈME 2.1. *SD est $\mathcal{NP}$-complet*

Preuve. Ceci a été démontré par E. R. Berlekamp, H. C. Van Tilborg et R. J. McEliece [BMV]. Les auteurs réduisent polynomialement le problème SD au problème $\mathcal{NP}$-complet dit des 3-Mariages, dont le problème de décision est :

Nom : 3-Mariages
Entrée : Un sous-ensemble $U \subseteq T \times T \times T$, où T est un ensemble fini.
Question : Existe-t-il un ensemble $W \subseteq U$ tel que $\mathrm{card}(W) = \mathrm{card}(T)$, et tel qu'aucun couple d'éléments de W ne possède de coordonnées communes ?

□

Etant donné qu'il existe un algorithme polynomial permettant d'obtenir, à partir d'une matrice de contrôle d'un code C, une matrice génératrice, le problème SD énoncé en termes de matrice génératrice est aussi $\mathcal{NP}$-complet.

L'un des schémas d'identification proposé au prochain chapitre repose sur cette vision du problème SD.

Nom : G-SD
Entrée : $G(k, n)$ une matrice binaire de rang k, y' un vecteur binaire de longueur n et p' un entier.
Question : Existe-t-il un vecteur e' de longueur n et de poids inférieur ou égal à p' tel que $y' + e'$ soit une combinaison linéaire des lignes de G ?

Théorème 2.2. *G-SD est $\mathcal{NP}$-complet*

Preuve. G-SD se réduit polynomialement à SD de la façon suivante. Construisons la matrice $H(n-k, n)$ vérifiant $GH^{\mathrm{t}} = 0$. Alors

$$\exists m \text{ et } e' \text{ telque } \begin{cases} \omega(e') \leqslant p' \\ y' + e' = mG \end{cases} \iff \exists e' \text{ telque } \begin{cases} \omega(e') \leqslant p' \\ He'^{\mathrm{t}} = Hy'^{\mathrm{t}} \end{cases}.$$

□

Remarques :

- Du point de vue de la théorie de la complexité, ces deux problèmes de décision sont équivalents. Cependant, dans un cadre pratique, nous verrons, qu'à paramètres égaux et pour un seuil de sécurité identique, les performances d'un schéma d'identification diffèrent selon le problème sur lequel repose sa sécurité,
- Les résultats donnés, dans le prochain paragraphe, montrent qu'il est facile de construire des instances difficiles du problème SD *(resp. de G-SD)*. Ceci est un avantage, lors de l'élaboration d'un protocole dont la sécurité

dépend de la difficulté de résolution d'un tel problème. En effet, à titre d'exemple, les instances difficiles du problème du sac à dos se situent en densité 1 [CJL..], mais on ne connaît pas, à ce jour, de méthodes permettant de construire des suites, de longueur inférieure à 120, et de densité 1, qui soient résistantes à l'attaque de l'algorithme LLL [Bou].

§3. Cryptanalyse du problème SD

La sécurité des schémas d'identification, décrits dans le prochain chapitre, dépend de la difficulté de résolution des problèmes SD et G-SD.

Proposition 3.1. *S'il existe, pour une instance donnée, une solution au problème SD (resp. au problème G-SD), alors la recherche de celle-ci revient à trouver un mot d'un certain poids dans un code linéaire binaire.*

Preuve.

• Le problème G-SD : S'il existe un vecteur e de poids au plus p tel que $x + e$ appartienne au code engendré par G, ceci signifie que $x = mG + e$. Considérons alors le code C' admettant pour matrice génératrice, la matrice

$$G' = \begin{pmatrix} G \\ x \end{pmatrix}.$$

La recherche du vecteur e est équivalente à la recherche d'un mot de poids au plus p dans le code C'.

• Le problème SD : Supposons qu'il existe un vecteur e de poids au plus p tel que $He^{\mathrm{t}} = i$. Considérons le code C' dont une matrice de contrôle est la matrice

$$H' = (H \mid i).$$

La recherche de e revient à trouver un mot de poids au plus $p + 1$, dont la dernière composante vaut 1, dans le code C'. □

Il existe, actuellement, quatre algorithmes probabilistes permettant d'exhiber une solution (si elle existe) du problème SD *(resp. du problème G-SD)*. Bien que leur facteur de travail croisse de façon exponentielle, il est nettement inférieur à celui de l'attaque systématique qui consiste à tester les 2^{n-k} solutions de l'équation $z = Hx^{\mathrm{t}}$ (si H est une matrice $k \times n$). Voici une description de ces algorithmes, donnée dans l'ordre chronologique de leur apparition. Le lecteur trouvera de plus amples informations en consultant les travaux de Florent Chabaud ([Cha$_1$], [Cha$_2$]), et plus récemment dans le travail de synthèse effectué par A. Canteaut et F. Chabaud [CaCh$_2$].

Remarque : dorénavant on appellera “facteur de travail” d’une attaque la valeur de la quantite T/P où P correspond à la probabilité de succès dans l’espace de tous les cas possibles (espace muni de la probabilité uniforme) et T est la complexité de calcul pour un cas favorable (pour plus de simplicité on se place dans l’hypothèse où chaque cas d’attaque a la même complexité de calcul).

3.1. L’algorithme de P.J. Lee et E.F. Brickell

Le cryptosystème, à clef publique, de Mc Eliece [McEl] est le premier protocole cryptographique utilisant les codes correcteurs d’erreurs. A tout message m, à chiffrer, de longueur k, est associé un mot d’un code de Goppa binaire, de dimension k, auquel on ajoute un mot erreur de poids inférieur à la capacité de correction du code utilisé. Ainsi, le cryptogramme associé au message m est de la forme $x = mG + e$, où G est la matrice génératrice du code.

```
entrée : - une matrice G(k,n) sur F_2, p ∈ N
         - x = mG + e un vecteur de longueur n, ω(e) = p
début
    succès ← 0 ;
    Tant que succès<>1 faire
        début
            sélectionner k positions aléatoires i_1,...,i_k ;
            x̄ ← (x_{i_1},...,x_{i_k}) ;
            Ḡ ← (G_{i_1},...,G_{i_k}), (G_j = j^ième colonne de G) ;
            m̄ ← x̄Ḡ^{-1} ;
            Si ω(m̄G + x) = p alors
                début
                    succès ← 1 ;
                    e ← m̄G + x ;
                fin
        fin
fin.
sortie : - le mot e de poids p
```

Fig. 1. Algorithme de McEliece

L’algorithme de Lee et Brickell a été développé afin de cryptanalyser le système Mc Eliece . Contrairement aux autres algorithmes, celui-ci ne se propose pas de retrouver un mot de poids p dans un code linéaire binaire quelconque. Il permet en fait, à partir d’une matrice G et d’un mot x de la forme $mG+e$, de retrouver le mot erreur e de poids p. Cette méthode permet de façon évidente de résoudre le problème G-SD. On peut l'appliquer aussi afin de résoudre le problème SD (cf. preuve du théo. 2.2).

L'algorithme proposé par P.J. Lee et E.F. Brickell est une optimisation de la cryptanalyse, proposé par R.J. McEliece lui-même, pour son cryptosystème. Nous présenterons donc cette version initiale, le lecteur intéressé pourra consulter [CACH2], [CHA2], [LEBR] en ce qui concerne la version optimisée. L'idée principale de R.J. McEliece est basée sur le fait que, dans son cryptosystème, le mot erreur e étant de petit poids par rapport à la longueur du code n, il ne recouvre que très partiellement le mot mG. Le principe général est alors d'essayer de trouver parmi les composantes de x, celles qui ne sont pas perturbées par l'erreur e. Cette heuristique s'appliquera aussi dans le cadre de nos schémas d'identification, car, pour des raisons de sécurité (cf.§4), le poids p du mot à rechercher sera proche de la distance minimale du code.

• Facteur de travail

La probabilité pour qu'une sélection de k colonnes aboutisse à un cas favorable (i.e les k positions correspondent à des positions pour lesquelles les composantes de e sont nulles) est :

$$\pi(n,k,p) = \frac{\binom{n-p}{k}}{\binom{n}{k}}.$$

Pour chaque itération, le nombre d'opérations binaires à effectuer est :

- environ k^3 pour l'inversion de $\bar{G}$,
- $k(2k-1)$ pour le produit $m\bar{G}^{-1}$,
- une addition entre deux vecteurs de longueur n.

Une estimation du facteur de travail de cet algorithme est donc :

$$W(n,k,p) = \frac{k^3 + k(2k-1) + n}{\pi(n,k,p)}.$$

L'optimisation de P.J. Lee et E.F. Brickell consiste à ne pas rejeter systématiquement, lors de la sélection de k composantes du mot x, celles contenant des bits erronés, mais à conserver celles pour lesquelles le poids de l'erreur est inférieur à un certain paramètre ℓ.

3.2. L'algorithme de J.S. Leon

Cet algorithme recherche un mot de poids p dans un code linéaire binaire C, à partir de sa matrice génératrice ([LEO]). Il dépend de deux paramètres s ($>k$) et ℓ. Nous décrivons ici, une optimisation de cet algorithme, proposée par

F. Chabaud [CHA2]. Soit e un mot du code C tel que $\omega(e) = p$. L'idée générale de l'algorithme est d'essayer de trouver une permutation des composantes de e de telle façon que :

- le poids des s premières composantes soit égal à ℓ (ℓ petit, i.e. les s premières composantes de e sont presque toutes nulles),
- le poids des $n - s$ dernières composantes soit égal à $p - \ell$.

La première condition montre que, si après permutation des colonnes de G, cette dernière est mise sous la forme $(I_k \mid M)$, alors les s premières colonnes de G contiennent une combinaison linéaire d'au plus ℓ lignes, donnant un mot de poids inférieur ou égal à ℓ. Une telle combinaison linéaire engendre un mot qui a de fortes chances d'être le mot recherché. Une telle heuristique est bien adaptée à la recherche des mots de faible poids.

```
entrée : - une matrice G(k,n) sur F2,
         - p,ℓ,s ∈ ℕ
début
    Permuter aléatoirement les colonnes de G ;
    Appliquer l'algorithme de Gauss afin de mettre G sous
    la forme G = (Ik | Z | B), où Z est une matrice (k,s − k)
    et B est une matrice (k,n − s) ;
    T ← (Ik | Z)
    Pour tous les mots λ, de longueur k, vérifiants ω(λ) ⩽ ℓ faire
        début
            z ← ∑_{i=1}^{k} λi Ti  (Ti = i^ième ligne de T) ;
            Si ω(z) ⩽ ℓ alors
                début
                    y ← ∑_{i=1}^{k} λi Gi  (Gi = i^ième ligne de G) ;
                    Si ω(y) = p alors
                        début
                            e ← y ;
                            exit ;
                        fin ;
                fin ;
        fin ;
fin.
sortie : - un mot e de poids p
```

Fig. 2. Algorithme de J.S. Leon

• Facteur de travail

Supposons qu'il existe dans le code C un mot e de poids p. Une permutation des colonnes de G aboutit à un cas favorable si, son action sur e, a pour effet de positionner, parmi les s premières composantes de ce mot, au plus ℓ coordonnées non nulles. La probabilité d'une sélection correcte est donc donnée par

$$\pi(s,\ell,n,k,p) = \frac{\sum_{i=1}^{\ell}\binom{p}{i}\binom{n-p}{s-i}}{\binom{n}{s}}.$$

A chaque itération les calculs effectués nécessitent :

- $k \times \frac{k}{2} \times n$ opérations binaires pour l'élimination gaussienne,
- $\sum_{i=1}^{\ell}\binom{k}{i}(i-1)$ additions binaires sur des vecteurs de $s-k$ bits.

De plus dans le cas où $\omega(z) = i(\leqslant \ell)$, on calcule la somme de $i-1$ vecteurs de longueur $n-s$. Le nombre moyen de cas où $\omega(y) \leqslant \ell$ est donnée par

$$\sum_{i=1}^{\ell}\binom{k}{i}\frac{\sum_{j=0}^{\ell-i}\binom{s-k}{j}}{2^{s-k}}.$$

Le facteur de travail de cet algorithme est donc de

$$W(s,\ell,n,k,p) = \frac{k \times \frac{k}{2} \times n + \sum_{i=1}^{\ell}\binom{k}{i}(i-1)\left(\frac{(s-k)+n\sum_{j=0}^{\ell-i}\binom{s-k}{j}}{2^{s-k}}\right)}{\pi(s,\ell,n,k,p)}.$$

3.3. L'algorithme de J. Stern

Cet algorithme se propose de chercher un mot de poids p dans un code linéaire binaire ([STE₁]). Contrairement à l'algorithme de Leon, celui de Stern utilise la matrice de contrôle du code. C'est en quelque sorte une version duale de l'algorithme précédent. Il dépend de deux paramètres s et $\ell(< \frac{k}{2})$.

Soit $H(n-k,n)$ une matrice de contrôle du code C, alors un mot x du code est caractérisé par $Hx^{\mathrm{t}} = 0$. Le principe de départ est le même. Il consiste, dans un premier temps, à sélectionner aléatoirement $n-k$ colonnes de cette matrice afin de les amener, par permutation, d'un côté de la matrice. On réalise alors un pivot de Gauss afin de se ramener à la forme $H = (Q \mid I_{n-k})$. L'idée est alors de décomposer Q de la façon suivante :

$$Q = \left(\begin{array}{c|c} X_\ell & Y_\ell \\ \hline X_{n-k-\ell} & Y_{n-k-\ell} \end{array}\right),$$

où X_j est une matrice à j lignes et $\frac{k}{2}$ colonnes. On a donc

$$H = \left(\begin{array}{c|c|c} X_\ell & Y_\ell & \\ \cline{1-2} X_{n-k-\ell} & Y_{n-k-\ell} & I_{n-k} \end{array}\right). \tag{3.3.1}$$

Pour chaque groupe P_X *(resp.* P_Y*)* de s colonnes de X_ℓ *(resp. de* Y_ℓ*)*, on calcule leur somme $\Sigma_s(P_X)$ *(resp.* $\Sigma_s(P_Y)$*)*.

```
entrée : - une matrice H(n − k, n) sur F2,
         - p, ℓ, s ∈ ℕ
début
  succès ← 0 ;
  Tant que succès<>1 faire
    début
      Choisir aléatoirement n − k colonnes de H
      et les placer en fin de matrice ;
      Appliquer Gauss afin de mettre H sous la forme
      donnée par la relation (3.3.1) ;
      Pour chaque groupe P_X et P_Y de s colonnes faire
        début
          Si succès<>1 alors
            début
              calculer Σ_s(P_X) et Σ_s(P_Y) ;
              Si Σ_s(P_X)= Σ_s(P_Y) alors
                début
                  y ← (y_1,...,y_n), y_i = 1 si i ∈ P_X ∪ P_Y, 0 sinon ;
                  v ← (v_1,...,v_{n−k−ℓ}), v_j = Σ_{i∈P_X∪P_Y} h_{j+ℓ,i} ;
                  Si ω(v) = p − 2s alors
                    début
                      succès ← 1 ;
                      z ← (z_1,...,z_n),
                      z_i = 0, 1 ⩽ i ⩽ k + ℓ; z_i = v_{i−k−ℓ}, k + ℓ + 1 ⩽ i ⩽ n − k ;
                      x ← y + z ;
                    fin ;
                fin ;
            fin ;
        fin ;
    fin ;
fin.
sortie : - un mot x de poids p
```

Fig. 3. Algorithme de J. Stern

L'idée générale de l'algorithme est alors d'étudier les cas où $\Sigma_s(P_X) = \Sigma_s(P_Y)$ (puisque $l < \frac{k}{2}$). Dans ce cas, considérons alors le vecteur y, de poids $2s$, et de longueur n tel que

$$y_i = \begin{cases} 1 & \text{si } i \in P_X \cup P_Y, \\ 0 & \text{sinon.} \end{cases}$$

Alors

$$Hy^{t} = \begin{pmatrix} 0 \\ \nu \end{pmatrix} \begin{matrix} \}\ell \\ \}n-k-\ell \end{matrix}$$

où ν est un vecteur de longueur $n-k-\ell$, qui correspond à la somme des colonnes, de la sous-matrice $(X_{n-k-\ell} \mid Y_{n-k-\ell})$, appartenant à $P_X \cup P_Y$. Si $\omega(\nu) = p - 2s$, alors on forme le vecteur z, de longueur n, vérifiant

$$z_i = \begin{cases} 0 & \text{pour } i = 1, \ldots, k+\ell, \\ \nu_{i-k-l} & \text{pour } i = k+\ell+1, \ldots, n. \end{cases}$$

alors,

$$Hz^{t} = H \begin{pmatrix} 0 \\ \nu \end{pmatrix} \begin{matrix} \}k+\ell \\ \}n-k-\ell \end{matrix} = I_{n-k} \begin{pmatrix} 0 \\ \nu \end{pmatrix} \begin{matrix} \}\ell \\ \}n-k-\ell \end{matrix} = \begin{pmatrix} 0 \\ \nu \end{pmatrix} \begin{matrix} \}\ell \\ \}n-k-\ell \end{matrix} = Hy^{t}.$$

Ainsi, le vecteur $x = y + z$ vérifie $Hx^{t} = 0$ et $\omega(x) = p$.

• Facteur de travail

Supposons qu'il existe un mot x de poids p appartenant au code dont une matrice de contrôle est H. Une permutation des colonnes de H aboutit à un cas favorable si, l'action de cette permutation sur le vecteur x produit le résultat suivant :

- les $\frac{k}{2}$ premières composantes de x ne comportent que s éléments non nuls. Le nombre de configurations possibles est $\binom{p}{s}\binom{n-p}{k/2-s}$,
- les $\frac{k}{2}$ composantes suivantes vérifient la même contrainte. Ce qui donne $\binom{p-s}{s}\binom{n-p-(k/2-s)}{k/2-s}$ choix possibles,
- les $n-k-\ell$ composantes suivantes contiennent le reste du support de x (i.e $p-2s$ composantes non nulles). Ce qui donne $\binom{n-p-(k-2s)}{n-k-\ell-(p-2s)}$ alternatives.

La probabilité d'obtention d'un cas favorable est donc :

$$\pi(s, \ell, n, k, p) = \frac{\binom{p}{s}\binom{n-p}{k/2-s}\binom{p-s}{s}\binom{n-p-(k/2-s)}{k/2-s}\binom{n-p-(k-2s)}{n-k-\ell-(p-2s)}}{\binom{n}{k/2}\binom{n-k/2}{k/2}\binom{n-k}{n-k-\ell}}.$$

Le nombre moyen de fois où l'on obtient le même vecteur de ℓ lignes en effectuant des combinaisons linéaires de s colonnes de la matrice $X_\ell(\ell \times k/2)$ *(resp. de la matrice Y_ℓ)* est de l'ordre de $\frac{\binom{k/2}{s}}{2^\ell}$. Par suite le nombre total de collisions $(\Sigma_s(X_\ell) = \Sigma_s(Y_\ell))$ est de l'ordre de $\frac{\binom{k/2}{s}^2}{2^\ell}$ Ainsi à chaque itération les calculs effectués sont :

- $(n-k)\frac{(n-k)}{2}n$ additions binaires pour la réduction gaussienne,

- $2\binom{k/2}{s}(p-1)$ additions de vecteurs de ℓ bits,
- $2s-1$ additions entre des vecteurs de $n-k-\ell$ bits pour chaque collision rencontrée.

Le facteur de travail de cet algorithme est donc :
$W(s,\ell,n,k,p) =$

$$\frac{n(n-k)(n-k)/2 + 2\ell\binom{k/2}{s}(s-1) + (n-k-\ell)(2s-1)\frac{\binom{k/2}{s}^2}{2^\ell}}{\pi(s,\ell,n,k,p)}.$$

3.4. L'algorithme de A. Canteaut et H. Chabanne

Initialement, cet algorithme a été élaboré afin d'optimiser le facteur de travail de l'algorithme de Mc Eliece. Cependant, les travaux récents de A. Canteaut et de F. Chabaud montrent que la méthode proposée peut s'appliquer aussi aux algorithmes de J.S. Leon et de J. Stern.

Nous ne donnerons que le principe général de cet algorithme. Pour plus d'informations le lecteur pourra consulter [CaCh1], [CaCh2].

Tous les algorithmes précédents ont un point commun. A chaque étape, un groupe de colonnes est sélectionné et transformé à l'aide d'une élimination gaussienne. La complexité de cette opération est de l'ordre de $\frac{nk^2}{2}$, si la matrice considérée est de rang k et de longueur n.

L'idée est la suivante : à une étape donnée, on dispose d'une matrice M de la forme $M = (I_k \mid N)$. Si l'algorithme utilisé n'a pas réussi à trouver un mot de poids p, alors au lieu de choisir, à nouveau, k colonnes au hasard, les auteurs proposent de permuter une colonne de la matrice I_k, avec une colonne de la matrice N. La nouvelle matrice, ainsi obtenue est de la forme

$$M' = \left(\begin{array}{ccc|c} 1 & \times & & \\ & \ddots \;\times & & \\ & 1 & & N' \\ & \times & \ddots & \\ & \times & 1 & \end{array}\right)$$

Les auteurs montrent comment choisir une colonne de la matrice N afin que les k premières colonnes de M' soient toujours inversibles.

Pour cette matrice, le coût de l'élimination gaussienne, afin d'obtenir, à nouveau, la forme $(I_k \mid N)$, est de l'ordre de $\frac{k(n-k)}{2}$ opérations.

Les résultats obtenus dans [CaCh2] montrent que l'algorithme le plus efficace, pour la recherche d'un mot de poids p dans un code linéaire binaire, est obtenu en combinant cette méthode et l'algorithme de J. Stern.

§4. Efficacité de ces algorithmes

L'étude expérimentale menée dans [CHA2], et confirmée dans [CACH2], montre que si l'on se fixe une limite de complexité de l'ordre de 2^{45} pour le facteur de travail, alors il est possible de trouver, à l'aide des différents algorithmes, des mots de poids assez faibles, même dans des codes de grande dimension. On constate aussi que plus p est proche de la distance minimale théorique du code et plus la recherche devient complexe. Une estimation de la distance minimale théorique, d, d'un code linéaire binaire quelconque, est obtenue à l'aide de la borne de Gilbert-Varshamov [MASL] :

$$H_2(\frac{d}{n}) \simeq 1 - \frac{k}{n},$$

où $H_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ est la fonction d'entropie.

Bien que le facteur de travail croisse en fonction de p, pour p grand, il est plus judicieux d'effectuer un tirage aléatoire parmi les mots du code pour en trouver un de poids p. En effet, la distribution de poids d'un code binaire aléatoire suit une loi binomiale.

Code		$(256,128)$	$(512,256)$	$(1024,512)$
Distance min. estimée		30	56	115
Stern	Borne sup.	30	37	35
	Borne inf.	30	72	171
Leon	Borne sup.	30	29	26
	Borne inf.	30	78	186

Tab. 4. Bornes sur les poids accessibles

Ainsi, au vu de ces différents résultats, si la sécurité d'un système cryptographique dépend essentiellement de la recherche d'un mot de poids p, dans un code linéaire, alors :

- p ne doit pas être trop petit, vu l'efficacité des algorithmes probabilistes,

- p ne doit pas être trop grand, en raison de la distribution de poids du code.

A titre d'information, la Table 4. donne la zone des poids accessibles pour les algorithmes de J.S Leon et J. Stern (optimisé). Une évaluation, de ces différents algorithmes, montre que pour obtenir un facteur de travail de l'ordre de 2^{70}, les paramètres à utiliser sont

$$\boxed{n = 512, k = 256, p = 56}$$

ou

$$\boxed{n = 510, k = 255, p = 56}$$

Pour un facteur de travail de l'ordre de 2^{60}, les paramètres sont

$$\boxed{n = 512, k = 120, p = 114}$$

p étant un estimation de la distance minimale théorique d'un code linéaire binaire de longueur n, et de dimension k.

§5. Le cas q-aire

Dans le cas où la matrice étudiée est à coefficients dans $\mathbb{F}_q$ ($q = p^r$, p premier), le facteur de travail de ces algorithmes est modifié. Tout d'abord, ceci ne change en rien le nombre d'itérations des algorithmes décrits précédemment, puisque ce dernier dépend essentiellement de la recherche d'une bonne permutation. Cependant, pour une étape fixée, le nombre de calculs va augmenter puisque les opérations ne s'effectuent plus sur des bits, mais sur des éléments de $\mathbb{F}_q$.

Dans [CaCh2], les auteurs donnent une évaluation du coût des calculs à effectuer, lors d'un étape de l'algorithme de J.S. Leon. Nous avons évalué, pour notre part, les modifications du facteur de travail de l'algorithme de J. Stern. Pour cela, les notations suivantes, introduites dans [CaCh2], sont utilisées :

Notations : $\mathcal{M}$ représentera le coût d'une multiplication, $\mathcal{A}$ celui d'une addition et $\mathcal{I}$ celui d'une inversion, toutes ces opérations étant exécutées sur $\mathbb{F}_q$.

Tout d'abord, voici les résultats obtenus dans [CaCh2] :

- le nombre moyen d'opérations pour une élimination gaussienne est

$$\mathcal{G}(n,k) = k(\mathcal{I} + \frac{nk(q-1)}{q})(\mathcal{M} + \mathcal{A}),$$

- pour la méthode de Canteaut-Chabanne, ce nombre se réduit à

$$g(n,k) = \mathcal{I} + \frac{nk(q-1)}{q}(\mathcal{M} + \mathcal{A}),$$

  ceci si la matrice considérée est de longueur n et de dimension k,
- le facteur de travail de l'algorithme de J.S. Leon est
  $W(s,\ell,n,k,p,q) =$

$$\frac{\mathcal{G}(n,k) + \sum_{i=1}^{\ell}(i(\mathcal{A}+\mathcal{M}) + \mathcal{M})\binom{k}{i}(q-1)^i\left((s-k) + (n-s)\frac{\sum_{j=0}^{s-i}(q-1)^j\binom{s-k}{j}}{q^{s-k}}\right)}{\pi(s,\ell,n,k,p)}.$$

La quantité $\pi(s,\ell,n,k,p)$ étant la même que celle de l'algorithme initial.

Voici les résultats que nous avons obtenus pour l'algorithme de J. Stern :

- Le calcul de $\Sigma_s(P_X)$ revient à calculer toutes les combinaisons linéaires, sur $\mathbb{F}_q$, de s colonnes, choisies parmi $\frac{k}{2}$. Il y a $\binom{k/2}{s}(q-1)^s$ vecteurs de poids s et de longueur $\frac{k}{2}$. Une combinaison linéaire, effectué avec ce vecteur et les colonnes de la matrice X_ℓ, nécessitent $\ell(s\mathcal{M} + (s-1)\mathcal{A})$ opérations. Ainsi le calcul de $\Sigma_s(P_X)$ et $\Sigma_s(P_Y)$ nécessite

$$2\binom{k/2}{s}(q-1)^s(s\mathcal{M} + (s-1)\mathcal{A})\ell$$

  opérations.
- Le nombre moyen de fois où le même vecteur est obtenu, en effectuant une de ces combinaisons linéaires, est

$$\frac{\binom{k/2}{s}(q-1)^s}{q^\ell}.$$

  Ainsi le nombre moyen de collisions ($\Sigma_s(P_X) = \Sigma_s(P_Y)$) est

$$\frac{\binom{k/2}{s}^2(q-1)^{2s}}{q^\ell}.$$

- Pour chaque collision, on calcule une combinaison linéaire de $2s$ colonnes de la matrice $(X_{n-k-\ell} \mid Y_{n-k-\ell})$, ce qui donne

$$(n-k-\ell)(2s\mathcal{M} + (2s-1)\mathcal{A})$$

opérations.

Le facteur de travail de l'algorithme de J. Stern est donc

$$W(s,\ell,n,k,p,q) = \Big[\mathcal{G}(n,n-k) + 2\binom{k/2}{s}(q-1)^s\ell(\mathcal{M}+(s-1)(\mathcal{A}+\mathcal{M})) + (n-k-\ell)(\mathcal{M}+(2s-1)(\mathcal{A}+\mathcal{M}))\frac{\binom{k/2}{s}^2(q-1)^{2s}}{q^\ell}\Big]\Big/\pi(s,\ell,n,k,p).$$

La quantité $\pi(s,\ell,n,k,p)$ étant la même que celle de l'algorithme initial.

Dans le cas où $q = 2^m$, on peut évaluer, en termes d'opérations **binaires** :

- une addition entre deux éléments à m additions binaires,
- une multiplication à environ m^2 additions binaires,
- l'inversion d'un élément à environ $m^2\log_2(m)$ opérations binaires.

Une évaluation, de cet algorithme, montre que, sur le corps $\mathbb{F}_{2^8}$, pour obtenir un facteur de travail de l'ordre de 2^{70}, les paramètres à utiliser sont

$$\boxed{n = 96, k = 48, p = 35}$$

Remarque : contrairement aux paramètres suggérés dans le cadre binaire, la validité de ceux-ci n'a pas encore été confirmée expérimentalement.

En considérant les performances de ces divers algorithmes, une cryptanalyse du schéma d'identification de S. Harari est proposée au chapitre v. De plus, en considérant les résultats obtenus dans [CaCh2], nous montrons qu'il est impossible de diminuer les paramètres de ce schéma afin d'en améliorer les performances.