Analyse de la résistance aux attaques algébriques des fonctions de filtrage augmentées

Stage de recherche du MPRI

Stéphane JACOB, encadré par Anne CANTEAUT

Équipe-projet SECRET, Centre de recherche INRIA Paris-Rocquencourt

24 novembre 2008

Table des matières

Fiche de synthèse

Le contexte général

Le problème étudié ici est une attaque s'appliquant à certains chiffrements à flot présentée par Simon Fischer et Willi Meier en 2007 dans leur article *Algebraic Immunity of S-Boxes and Augmented Functions* [FM07]. Cette attaque fait partie de la famille des attaques dites des « attaques algébriques » qui consistent à effectuer une cryptanalyse grâce à la résolution d'un système d'équations polynomiales multivariées de petit degré. Elle se fonde sur les fonctions augmentées (*Déf. 2*) de ces algorithmes de chiffrements car celles-ci ont des propriétés algébriques qui ne sont pas prises en compte dans les précédents travaux sur l'immunité algébrique des fonctions booléennes de filtrage. Cela permet alors de monter des attaques avec une faible complexité en données sur certains chiffrements à flots, jusqu'alors immunisés contre les attaques algébriques.

Le problème étudié

L'article de Simon Fischer et Willi Meier [FM07] introduit donc un nouveau type d'attaques sur les fonctions augmentées (*Déf. 2*) des chiffrements à flot, ce qui ouvre de nouvelles perspectives dans ce domaine. En revanche, deux points sont laissés en suspens. Le premier est la faisabilité pratique de telles attaques ; en effet dans l'article, les exemples sont basés sur des LFSR de longueur assez faible (20-40 bits) par rapport aux cas courants (~ 200) et la faisabilité de l'attaque dans ces cas plus généraux n'est pas évoquée. Le second est de théoriser un peu plus l'attaque en déterminant mieux quels paramètres (fonction de mise à jour, fonction de filtrage...) sont prépondérants et dans quelle mesure. Enfin, un travail à plus long terme consiste à identifier l'origine de ces attaques et à comprendre pourquoi elles fonctionnent pour certaines fonctions de filtrage et/ou de mise à jour.

La contribution proposée

Pour répondre à la question des influences de tel ou tel paramètre dans l'attaque, nous avons commencé par reproduire les résultats présentés dans l'article de Fischer et Meier [FM07] en développant nos propres programmes et en suivant à la lettre la démarche proposée par les auteurs. Cela nous a ainsi permis de mieux comprendre les algorithmes mis en œuvre et nous a donné des pistes pour identifier les points qui empêchaient de les adapter à des cas pratiques où la valeur des paramètres est considérablement plus grande que dans ces exemples.

Dans ce contexte, nous avons tout d'abord montré que les deux versions de l'attaque proposée dans [FM07], celle utilisant des relations algébriques

satisfaites pour toute sortie et celle utilisant des relations algébriques conditionnées, avaient la même complexité et ne différaient que par leur phase de précalcul. Nous avons ensuite proposé une variante de l'algorithme qui réalise un meilleur compromis temps-mémoire que les algorithmes utilisés dans [FM07]. Enfin l'étude précise de la complexité du précalcul nous a montré, que, pour des paramètres réalistes, il était impossible d'examiner toutes les entrées de la fonction augmentée. Il est donc indispensable d'utiliser des relations algébriques satisfaites avec une probabilité $(1 - \varepsilon)$, pour ε petit. Nous proposons alors une nouvelle attaque dans ce cas, fondée sur l'utilisation d'un algorithme de décodage. Contrairement à l'attaque de Fischer et Meier, notre attaque n'exige pas la connaissance d'un nombre gigantesque de bits de suite chiffrante.

Les arguments en faveur de sa validité

Les améliorations apportées sont pertinentes car elles permettent théoriquement d'attaquer des LFSR de tailles couramment utilisées, c'est-à-dire des LFSR qui sont jusqu'à dix fois plus grands que ceux auxquels s'applique l'attaque proposée par Fischer et Meier [FM07].

Par ailleurs, l'algorithme de décodage que nous avons utilisé n'a pas été choisi au hasard, c'est en effet le plus adapté aux cas où les biais statistiques sont relativement élevés.

Le bilan et les perspectives

Le travail durant ce stage n'est qu'une petite partie du travail à effectuer sur cette attaque. Cette adaptation aux cas pratiques ne clôt pas la recherche sur ce sujet, car si elle permet de montrer que cette attaque est réellement intéressante car utilisable en pratique, elle ne donne aucune indication sur ce qui peut entraîner, dans un algorithme de chiffrement, une faiblesse vis-à-vis de celle-ci. En effet, les fonctions de filtrage et de mise à jour influent sur les paramètres de l'attaque et déterminent ainsi si l'attaque peut être mise en pratique ou non contre un algorithme de chiffrement donné. Il reste donc à définir précisément l'influence respective de ces fonctions sur les différents paramètres, et, si possible, à exhiber les caractéristiques précises de celles-ci qui rendent un chiffrement les utilisant vulnérable à cette attaque.

1 Introduction

L'article de Simon Fischer et Willi Meier [FM07] sur lequel nous avons travaillé introduit un nouveau type d'attaques exploitant certaines propriétés des fonctions augmentées des chiffrements à flot, ce qui ouvre de nouvelles perspectives dans ce domaine. Avant de se pencher sur cet article, commençons par présenter ce qu'est un chiffrement à flot et les attaques qui s'y appliquent, ainsi qu'un type de décodage qui nous sera utile par la suite, le décodage par ensemble d'information, qui est justement fréquemment utilisé pour des attaques sur des chiffrements à flot.

1.1 Les chiffrements à flot et leurs attaques [Can06]

Parmi les algorithmes de chiffrement à clef secrète, la distinction entre chiffrement à flot et chiffrement par blocs n'est pas toujours aisée. Habituellement, le chiffrement à flot désigne les algorithmes opérant sur des blocs de clair de taille relativement petite (généralement un bit, un octet ou un mot) au moyen d'une transformation qui varie au cours du temps. Par opposition, un algorithme de chiffrement par blocs applique la même fonction aux différents blocs de clair, qui sont ici de taille plus conséquente, typiquement 64, 128 ou 256 bits [MvOV97].

Cette définition permet de facilement différencier les chiffrements à flot synchrones additifs (qui opèrent au niveau du bit) des chiffrements par blocs en mode ECB, de même qu'elle permet de classer certains modes opératoires sur les algorithmes par blocs, par exemple les modes OFB et CTR, dans la catégorie des chiffrements à flots. Toutefois, elle ne permet pas de classer sans ambiguïté le mode CBC, car il peut également être considéré comme un chiffrement à flot auto-synchronisant opérant sur des blocs de grande taille. Dans la suite de ce document nous limiterons l'utilisation du terme chiffrement à flot aux seuls algorithmes synchrones car c'est à ceux-là que s'applique l'attaque étudiée, ceci étant aussi motivé par le fait que les chiffrements à flot auto-synchronisants sont plus ou moins tombés en désuétude.

Modèle et propriétés des chiffrements à flot synchrones

Modèle général. Un algorithme de chiffrement à flot synchrone consiste à combiner le clair avec une suite binaire de même longueur, la suite chiffrante. Notons $\boldsymbol{s} = (s_t)_{t\geq 0}$ cette suite qui est engendrée par un automate à états finis, appelé générateur pseudo-aléatoire, de façon indépendante à la fois du clair et du chiffré. Le rôle de ce générateur est de produire à chaque moment t un bloc de m bits, s_t, qui est fonction de son état interne $\boldsymbol{x}_t$. Dans toute la suite de ce document, nous ne considérerons que le cas $m = 1$. Le générateur peut se décomposer en trois fonctions (*Fig.* 1) :

- *une procédure d'initialisation* qui, à partir de la clef secrète et d'une valeur initiale publique qui correspond fréquemment à un numéro de

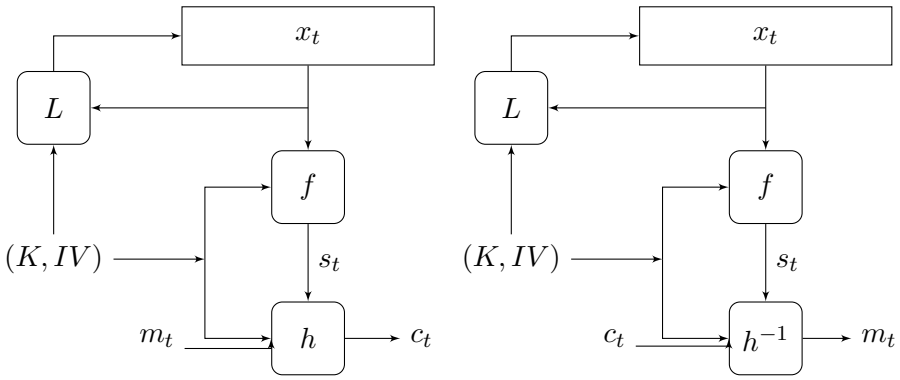

FIG. 1 – Chiffrement à flot synchrone et déchiffrement associé

trame, l'IV, calcule l'état initial du générateur, $\boldsymbol{x}_0$. Cette étape peut parfois être scindée en deux phases. La première, le chargement de clef, consiste en un calcul d'une valeur ne dépendant que de la clef. La seconde, appelée injection d'IV ou de re-synchronisation, détermine alors l'état initial du générateur à partir de la valeur obtenue précédemment et de l'IV. Ce découpage permet de gagner du temps lors d'un changement de l'IV sans changement de clef, ce qui est fréquent, surtout lors de l'utilisation d'un protocole de communication pour lequel la longueur des paquets échangés est relativement petite, comme par exemple pour les communications par GSM.

- *une fonction de transition*, notée L, qui fait passer l'état interne du générateur de l'instant t à l'instant $(t+1)$. En général, cette fonction est fixe, mais il se peut qu'elle varie avec la clef, l'IV, voire même avec le temps.
- *une fonction de filtrage*, notée f, qui retourne le bloc de suite chiffrante à partir de l'état interne courant, $\boldsymbol{x}_t$. Comme la fonction de transition, la fonction de filtrage est généralement fixe pour des raisons de simplicité et d'encombrement pour les implémentations matérielles.

Pour chiffrer, il suffit alors de combiner la suite chiffrante au clair à l'aide d'une fonction h inversible, qui associe un bloc de l bits de texte chiffré à un couple de l bits de suite chiffrante et de clair. Dans l'immense majorité des cas, la fonction h correspond au XOR bit à bit, c'est-à-dire à l'addition modulo 2, d'où le nom de chiffrement synchrone additif.

Dans la suite, on désignera par « la taille de l'état interne du générateur » la taille du plus petit automate à états finis permettant d'engendrer le même ensemble de suites que le générateur considéré. Par ailleurs, le nombre de variables de la fonction de filtrage correspond aussi à cette taille de l'état

interne car on considère que la fonction de filtrage ne dépend pas de certaines de ses variables, ce qui fait qu'elle possède ainsi des structures linéaires.

Contextes d'utilisation. Les avantages des algorithmes de chiffrements à flot découlent de la petite taille de bloc utilisée, ce qui est d'autant plus vrai dans le cas des chiffrements additifs où le bloc est réduit à un unique bit. Cette faible taille permet une réduction des délais et de la taille de la mémoire-tampon nécessaire pour stocker le message avant l'obtention d'un bloc complet. À cela s'ajoute le fait que les chiffrements à flot additifs n'ont évidemment pas besoin de padding, ce qui est fortement désirable lorsque la bande passante est faible ou que le protocole utilisé nécessite l'utilisation de paquets courts. L'emploi de blocs de petite taille limite aussi la propagation d'erreurs de transmission lors du déchiffrement.

Lorsque les ressources sont limitées, souvent parce qu'il faut restreindre la consommation électrique du circuit électrique dédié au chiffrement, ou qu'il est nécessaire de pouvoir chiffrer et déchiffrer très rapidement, on utilise généralement des algorithmes de chiffrement à flot, par exemple sur les téléphones portables.

Les grandes familles de générateurs pseudo-aléatoires. Il y a de nombreuses familles de générateurs pseudo-aléatoires, mais seules les constructions dédiées, au sens où elles sont spécialement conçues pour cet usage, permettent d'obtenir un très haut débit dans un environnement logiciel ou une mise en œuvre très peu coûteuse dans un contexte matériel ; ces deux objectifs étaient d'ailleurs ceux du projet eSTREAM [ECR05]. En l'occurrence, les deux classes de générateurs pseudo-aléatoires suivantes ne respectent pas ces contraintes malgré le grand intérêt qu'elles présentent du point de vue de la sécurité :

- *les générateurs sûrs d'un point de vue calculatoire,* dont la sécurité repose sur la difficulté de certains problèmes mathématiques. Il s'agit de générateurs pour lesquels l'existence d'un distingueur de complexité polynomiale est équivalente à un problème mathématique connu réputé difficile. Comme pour la plupart des cryptosystèmes à clef publique, ces générateurs pseudo-aléatoires sont fondés sur des problèmes de la théorie des nombres. Par exemple, le générateur RSA consiste à appliquer l'algorithme RSA à partir d'une graine $\boldsymbol{x}_0$. Sa sortie à l'instant t correspond au bit de poids faible de x_t [ACGS88]. Le problème de ce type de générateurs est leur lenteur qui ne leur permet d'avoir qu'un débit très faible, ainsi que la complexité de leur mise en œuvre. Ils ne sont donc pas utilisés en pratique pour les chiffrements à flot.
- *les générateurs inconditionnellement sûrs selon certains modèles,* dont l'objectif est d'apporter une sécurité démontrable identique à celle du masque jetable mais sous l'hypothèse que l'adversaire, même s'il dis-

pose d'une puissance de calcul infinie, a certaines contraintes, comme par exemple une capacité de stockage limitée [Mau92, AR99, Rab05]. La mise en œuvre de ces générateurs est beaucoup trop lourde pour qu'ils soient déployés autrement qu'à grande échelle, entre autres à cause de la source d'aléa complexe qu'ils utilisent.

Les contraintes imposées précédemment rendent la conception de tels générateurs très complexe, d'où leur rareté. Parmi les plus anciens figurent SNOW 2.0 [EJ02], SNOW 3G et MUGI [Hit01] qui ont été pris en compte dans la dernière version de travail de la norme internationale de chiffrement ISO/IEC 18033-4 [ISO05]. Plus récemment, le projet eSTREAM [ECR05] du réseau européen ECRYPT a recommandé des listes de générateurs pseudo-aléatoires : une pour une implémentation logicielle, l'autre pour une implémentation matérielle. Lors de la première révision de ces listes en septembre 2008, les générateurs recommandés pour être utilisés en logiciel sont HC-128, Rabbit, Salsa20/12 et Sosemanuk et ceux pour une implémentation matérielle sont Grain v1, MICKEY v2 et Trivium.

Sécurité

Pour évaluer la sécurité d'un algorithme de chiffrement à flot synchrone, on se place généralement dans le contexte d'une attaque à clair connu. Étant donné que la fonction h est publique et inversible, la connaissance d'un couple clair/chiffré implique celle de la suite chiffrante. Dans un tel contexte, l'attaquant dispose donc d'une certaine quantité de bits de la suite chiffrante.

Classification des attaques. Il y a quatre grandes catégories d'attaques, ici rangées de la plus forte à la plus faible :

- les attaques *par recouvrement de clef,* dont la finalité est de retrouver la clef secrète à partir d'un certain nombre de bits de la suite chiffrante ;
- les attaques *par recouvrement d'état,* qui visent à retrouver l'état interne initial complet, ce qui est d'ailleurs équivalent à retrouver n'importe quel état interne. C'est à ce type d'attaque que correspond l'algorithme proposé par Simon Fischer et Willi Meier [FM07] que nous avons étudié. Il faut remarquer que la portée d'une telle attaque, si elle permet à l'attaquant de calculer autant de bits qu'il le souhaite à partir de cette initialisation du générateur, ne lui permet a priori pas d'engendrer des suites dérivant de la même clef mais avec une IV différente. Ce type d'attaque est donc plus faible que le précédent ;
- les attaques *par prédiction du bit suivant,* dont l'objectif est, à partir d'un certain nombre de bits de suite chiffrante connus, de trouver le suivant ;
- les attaques *par distingueur,* dont le but est de déterminer si une suite d'un certain nombre de bits est un bout de suite chiffrante produit par

le générateur pseudo-aléatoire ou si c'est réellement une suite aléatoire.

Quelques types d'attaques Chacune de ces catégories contient de nombreuses attaques, dont certaines familles sont particulièrement utilisées. En voici quelques-unes parmi les plus connues :

- *Les attaques par corrélation*, qui entrent dans la catégorie des attaques de type « diviser pour régner », visent à retrouver une partie de l'état interne du générateur pseudo-aléatoire. Elles ont été introduites par Siegenthaler en 1985 [Sie85], puis améliorées sous le nom d'attaques par corrélation rapides par Meier et Staffelbach [MS88, MS89]. Le principe de telles attaques repose sur l'existence d'éventuelles corrélations entre la sortie de la fonction de filtrage et un sous-ensemble de ses entrées qui correspond à la partie incriminée de l'état interne [Can06].
- *Les attaques algébriques*, proposées par Courtois et Meier [CM03] en 2003, consistent à exprimer l'opération de chiffrement sous forme d'un système d'équations algébriques multivariées liant les bits du chiffré, les bits du clair et ceux de la clef, puis à instancier ce système à l'aide d'un ou plusieurs couples clairs / chiffrés et finalement de le résoudre afin de retrouver les bits de la clef. Pour appliquer ce principe général, il n'est pas nécessaire que la fonction de filtrage soit de degré faible. En effet, il suffit qu'il existe des relations de degré faible entre les entrées et la sortie de cette fonction, c'est-à-dire il suffit que la fonction de filtrage ait des multiples de petit degré.

Complexité en données. Traditionnellement, un algorithme est considéré comme sûr s'il n'est vulnérable à aucune attaque dont la complexité en temps, en mémoire et en données est inférieure à la taille de l'espace des clefs. Dans le cas des chiffrements à flot, la complexité en données est plus difficile à déterminer car les bits de la suite chiffrante utilisés lors de l'attaque ne proviennent pas obligatoirement de la même valeur initiale. En général, les trames étant relativement courtes, les bits de suite chiffrante sont issus de plusieurs valeurs initiales différentes, et décider si un algorithme est cassé quand il est vulnérable à une attaque nécessitant un nombre de bits de suite chiffrante inférieur mais proche de la taille de l'espace des clefs, est sujet à discussion.

Contraintes imposées par les attaques génériques. Des attaques classiques sur les chiffrements à flot imposent certaines conditions sur ses composantes :

- *La taille de l'état interne* doit être supérieure ou égale au double de la taille de la clef secrète afin de parer les attaques dites par compromis temps-mémoire-données [Bab98, Gol97].

- *La fonction de filtrage* f doit être équilibrée afin que la sortie du générateur soit uniformément distribuée pour éviter des attaques par distingueur triviales.
- *La fonction de transition* L doit avoir une période élevée pour la même raison.
- *Au minimum, l'une des fonctions précédentes,* f *ou* L*,* doit ne pas être linéaire, sinon avec n bits de suite chiffrante, on obtiendrait un système linéaire de n équations à n inconnues qui sont les bits de l'état initial. Ce système se résout aisément grâce à un pivot de Gauss avec une complexité en $\mathcal{O}\left(n^3\right)$, ce qui est très faible.

Les grandes familles de constructions dédiées

Classer ces algorithmes dédiés est difficile à cause à leur grande disparité mais on peut toutefois les séparer en trois grandes familles.

Les chiffrements à transition linéaire. En terme d'implémentation, c'est le choix naturel car le plus simple, mais il faut bien faire attention à ce que la fonction de filtrage f ne soit ni linéaire, ni proche d'une fonction linéaire, d'après ce qui précède. Les chiffrements utilisant des registres à décalage à rétroaction linéaire (LFSR) sont prépondérants dans cette catégorie à la fois parce que leur implémentation a un coût très faible et que l'on dispose de nombreux résultats théoriques sur les propriétés statistiques des suites chiffrantes qu'ils produisent. De plus ils peuvent être utilisés dans un environnement matériel (registres à décalage à rétroaction linéaire binaires) et dans un environnement logiciel, mais dans ce cas on opère sur un alphabet plus grand que le bit, en général sur un octet ou un mot de processeur. Les générateurs à base de LFSR les plus connus sont E0, déployé dans la norme Bluetooth, A5/1 et son cousin A5/2 utilisés dans la norme GSM pour les communications avec des téléphones portables, SNOW 2.0 qui est inclus dans la dernière norme ISO/IEC 18033 et son grand frère SNOW 3G qui est destiné aux téléphones 3G, ou encore Sosemanuk un des chiffrements retenu par le projet eSTREAM. Lors de la conception de tels chiffrements il faut faire très attention à l'émergence de nouvelles attaques apparues grâce aux récents progrès, notamment dus aux attaques algébriques.

Les chiffrements à transition non-linéaire. Leur raison d'être est d'éviter les faiblesses potentielles dues au caractère linéaire de la fonction de transition. Toutefois, la fonction de transition choisie doit garantir que les états internes du générateur ne forment pas une suite de faible période, et ce quelle que soit la valeur de l'état initial, ce qui est amplement plus difficile à prouver que pour des fonctions linéaires. Le seul moyen de contourner ce problème, dont l'exemple le plus connu est RC4, est de s'autoriser une

taille d'état interne très grande, ce qui entraîne généralement des problèmes d'implémentation. Mais même ce contournement n'est pas complètement satisfaisant car les chiffrements ainsi créés sont particulièrement vulnérables aux attaques par canaux cachés qui exploitent l'analyse du comportement du cache. Sans une grande taille d'état interne, il faut des résultats théoriques sur la période de la fonction de transition que peu de fonctions vérifient.

Les conceptions hybrides. Finalement, entre ces deux types de transition, il est possible de diviser l'état interne d'un générateur en deux parties, l'une étant mise à jour par une fonction linéaire, l'autre par une fonction non-linéaire. Si cette seconde partie est sensiblement plus petite que l'autre, elle est souvent assimilée à une mémoire interne et le générateur ainsi formé est alors classé comme un générateur à transition linéaire avec mémoire, comme par exemple pour SNOW 2.0, SNOW 3G et E0. Toutefois, il existe des générateurs dans lesquels les deux parties sont de tailles similaires, comme par exemple Grain v1 [HJM05] un des chiffrements recommandé par eSTREAM pour les implémentations matérielles.

1.2 Le décodage par ensemble d'information

Il existe de nombreux algorithmes de décodage, et les attaques par corrélation utilisent fréquemment le décodage par ensemble d'information ou ses dérivés, tels l'algorithme de Canteaut-Chabaud [CC98]. Ce type de décodage, initialement introduit par Prange [Pra62] pour les codes cycliques avant d'être généralisé à d'autres codes, se distingue des algorithmes à maximum de vraisemblance car il est probabiliste et non-exhaustif.

Avant de détailler cet algorithme, commençons par définir ce qui est son point central, l'ensemble d'information :

Définition 1. Soit $\mathcal{C}$ un code linéaire de longueur n et de dimension k, et G une matrice génératrice de $\mathcal{C}$. Un sous-ensemble I de $\{1, \ldots, n\}$ de taille k est un *ensemble d'information* pour le code $\mathcal{C}$ si et seulement si la restriction du code à ces k positions est un espace vectoriel de dimension k. Ceci équivaut à dire que la matrice carrée correspondant à la restriction de G aux positions de I est inversible.

Notons alors, si G est une matrice $n \times k$, $G = (U, V)_I$ où U est la restriction de G à l'ensemble I et V celle de G au complémentaire I.

Dans le contexte qui nous intéresse, nous disposons d'un mot x qui correspond au résultat de la transmission d'un mot du code $\mathcal{C}$ à travers un canal binaire symétrique de probabilité d'erreur ε, et l'on cherche à décoder ce mot, c'est-à-dire à retrouver le mot de code initial. Il est important de noter qu'on ne peut garantir que ce mot de code est unique seulement si le nombre de positions erronées ne dépasse pas la capacité de correction du code.

L'algorithme 1 consiste à sélectionner aléatoirement des ensembles d'information jusqu'à en trouver un qui ne contienne pas d'erreur. En effet, si l'on trouve un ensemble d'information I ne contenant aucune position erronée, en décomposant le mot bruité en $x = (x_I, x_J)_I$ et la matrice génératrice en $G = (U, V)_I$, le vecteur-erreur vaut $x + x_I U^{-1} G = (0, x_J U^{-1} V)_I$.

Algorithme 1 Décodage par ensemble d'information [Pra62, Can96]

Entrées :

- G, une matrice génératrice du code de longueur n et de dimension k.
- x, un mot de code bruité avec une probabilité d'erreur ε

Sorties : un mot de code à distance de l'ordre de εn de x

tant que un vecteur d'erreurs de poids environ égal à εn n'a pas été trouvé **faire**

1. Choisir aléatoirement un ensemble d'information I.
2. Mettre la matrice G sous la forme systématique $U^{-1}G = (Id, Z)_I$ et décomposer le vecteur x sous la forme $x = (x_I, x_J)_I$.
3. Calculer $w(x_J + x_I Z)$.

si ce poids est de l'ordre de εn **alors**

$(x_I, x_I Z)$ est un mot de code à distance $w(x_J + x_I Z)$ de x

fin si

fin tant que

Le nombre d'opérations binaires requises par cet algorithme est alors, d'après [Can96] :

$$\begin{aligned} N(n,k,\varepsilon n) &= \frac{\binom{n}{k}}{\binom{n(1-\varepsilon)}{k}} \left(\frac{k^2 n}{2} + \frac{n(n-k)}{2} + (n-k) \right) \\ &= \mathcal{O}\left(k^2 n \frac{\binom{n}{k}}{\binom{n(1-\varepsilon)}{k}} \right) \end{aligned} \tag{1}$$

2 Une attaque fondée sur les fonctions augmentées

2.1 Principe

Définition 2. Considérons un chiffrement à flot d'état interne x de longueur n bits, une fonction de mise à jour L et une fonction de filtrage f qui sort un bit de suite chiffrante par itération. La *fonction augmentée* S_m est définie comme suit :

$$\begin{array}{rccl} S_m : & \mathbb{F}_2^n & \to & \mathbb{F}_2^m \\ & x & \mapsto & y = \left(f(x), f(L(x)), \ldots, f(L^{m-1}(x)\right). \end{array}$$

Dans cette définition, on remarque que y est constitué des m premiers bits de la suite chiffrante générée par ce chiffrement à flot avec l'état initial x.

La notion de fonction augmentée a été introduite dans le contexte des attaques par corrélation par Anderson en 1995 [And95].

L'attaque décrite dans [FM07] exploite l'existence de relations algébriques de petit degré pour la fonction augmentée S_m. De telles relations sont définies de la manière suivante :

Définition 3. Soit F une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^m$. On appelle *relation algébrique pour* F toute relation du type

$$\sum_{\alpha \in \mathbb{F}_2^n} \sum_{\beta \in \mathbb{F}_2^m} c_{\alpha,\beta} x^\alpha y^\beta = 0$$

satisfaite pour tout couple $(x, y) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ où $y = F(x)$, avec $c_{\alpha,\beta} \in \mathbb{F}_2$ et la notation $x^\alpha := (x_1^{\alpha_1} \cdots x_n^{\alpha_n})$.

À partir de cela, le but est de trouver des relations algébriques de petit degré entre les bits de l'état initial, ce qui permet de le retrouver. En effet, si une telle relation algébrique pour S_m de degré d en les bits de l'état initial existe, elle fournit une relation de degré d entre ceux-ci dès que m bits consécutifs de suite chiffrante sont connus. L'attaque consiste alors à collecter suffisamment de relations de ce type pour pouvoir résoudre le système polynomial et retrouver l'état initial du générateur.

L'intérêt de cette attaque par rapport aux attaques algébriques classiques est qu'elle ne nécessite pas que la fonction de filtrage ait des multiples de petit degré et qu'elle s'applique donc dans des cas où les attaques algébriques classiques échouent.

2.2 Algorithme de Fischer et Meier

Dans cet algorithme [FM07], il n'y a, a priori, aucune restriction au cas linéaire. Nous présentons donc le cas général.

Une première approche est de chercher directement des relations algébriques pour la fonction S_m au sens de la défintion 3. Les y étant connus, le degré en y n'est pas limité car il n'a aucune influence sur la complexité de l'algorithme. Par contre, celui en x est le degré des équations à résoudre. Notons donc d le degré maximal que l'on s'autorise, soit $d := \max(w(\alpha) \text{ tq } c_{\alpha,\beta} = 1)$. Le nombre de monômes x^α possibles de degré inférieur à d est alors $D := \sum_{i=0}^{d} \binom{n}{i}$ et donc celui de monômes $x^\alpha y^\beta$ est $2^m D$. Pour chercher alors les combinaisons linéaires de monômes s'annulant, il suffit de créer la matrice M de taille $2^n \times 2^m D$ où chaque ligne correspond à un état initial x et chaque colonne à l'évaluation d'un monôme $x^\alpha y^\beta$ dans un ordre prédéterminé. Le rang de M détermine alors le nombre de solutions, qui correspondent au noyau de ${}^{\mathrm{t}}M$. Le problème, nous le verrons plus loin, est que la taille d'une telle matrice M est assez énorme au point qu'elle ne tient pas en RAM assez rapidement (dès $n \simeq 30$).

Pour éviter cet écueil, Fischer et Meier proposent de ne considérer que des sous-ensembles de cette matrice. Pour cela ils s'intéressent aux relations algébriques pour S_m conditionnées par la valeur de sortie, au sens de la définition suivante.

Définition 4. On appelle *relation algébrique conditionnée* par y_0 toute relation du type

$$\sum_{\alpha \in \mathbb{F}_2^n} c_\alpha x^\alpha = 0$$

satisfaite pour tout x tq $S_m(x) = y_0$, avec $c_\alpha \in \mathbb{F}_2$ et la notation $x^\alpha := (x_1^{\alpha_1} \cdots x_n^{\alpha_n})$.

Remarque. Les relations algébriques conditionnées par y_0 correspondent exactement aux relations algébriques pour la fonction booléenne $(\delta_{y_0} \circ F) : \mathbb{F}_2^n \to \mathbb{F}_2$ où δ_{y_0} est la fonction indicatrice :

$$\begin{aligned} \delta_{y_0}(y) &= 1 \text{ si } y = y_0 \\ &= 0 \text{ sinon.} \end{aligned}$$

Soit donc y un vecteur de m bits consécutifs de suite chiffrante. Notons $U_y := |S_m^{-1}(y)|$, le nombre d'antécédents de y par S_m, autrement dit le nombre d'états x donnant la suite chiffrante y. Pour chercher les équations entre les bits de x, il suffit de stocker les U_y vecteurs x dans une matrice M_y de taille $U_y \times D$ et de même on obtient le nombre de relations qui s'annulent en calculant le rang de M_y et on trouve les relations en cherchant le noyau de ${}^{\mathrm{t}}M_y$.

Dans ce cas, il est donc question de chercher des relations algébriques conditionnées par la sortie y_0 de S_m pour toutes les valeurs de y_0 possibles.

2.3 Lien entre les deux versions de l'attaque

Il est alors intéressant de faire le lien entre les relations trouvées dans ces deux cas.

Théorème 1. *Soit F une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^m$ et*

$$\sum_{\alpha \in \mathbb{F}_2^n} \sum_{\beta \in \mathbb{F}_2^m} c_{\alpha,\beta} x^\alpha y^\beta = 0$$

une relation algébrique de degré d non dégénérée pour F. Alors, il existe une relation algébrique de degré d conditionnée par y_0 pour au moins une valeur de y_0 dans $\mathbb{F}_2^m$.

Démonstration. Comme la relation algébrique est de degré d, il existe au moins un coefficient c_{α_0,β_0} non nul pour un α_0 de poids d. Il en découle que le polynôme $P(y) = \sum_{\beta \in \mathbb{F}_2^m} c_{\alpha_0,\beta} y^\beta$ n'est pas identiquement nul, ce qui implique l'existence d'au moins une valeur y_0 dans $\mathbb{F}_2^m$ pour laquelle la relation algébrique de départ conduit à la relation conditionnée pour y_0

$$\sum_{\alpha \in \mathbb{F}_2^n} x^\alpha \left(\sum_{\beta \in \mathbb{F}_2^m} c_{\alpha,\beta} y_0^\beta \right) = 0$$

de degré exactement d. □

Remarque. Il est important de remarquer qu'une relation de degré d peut également conduire à des relations conditionnées de degré strictement inférieur à d. Cette situation arrive quand il est possible de choisir une valeur de y_0 qui annule les coefficients de degré d :

$$\sum_{\beta \in \mathbb{F}_2^m} c_{\alpha,\beta} y_0^\beta = 0, \forall \alpha \text{ tq } w(\alpha) = d$$

et pour lequel il existe un α_0 de poids strictement inférieur à d tel que :

$$\sum_{\beta \in \mathbb{F}_2^m} c_{\alpha_0,\beta} y_0^\beta \neq 0.$$

Théorème 2. *Soit F une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^m$. Soit $y_0 \in \mathbb{F}_2^m$ tel qu'il existe une relation algébrique conditionnée par y_0 de degré d*

$$\sum_{\alpha \in \mathbb{F}_2^n} c_\alpha x^\alpha = 0$$

Alors

$$\sum_{\alpha \in \mathbb{F}_2^n} \sum_{\beta \in \mathbb{F}_2^m} c_\alpha \delta_\beta x^\alpha y^\beta = 0$$

est une relation algébrique de degré d pour F où $\sum_{\beta \in \mathbb{F}_2^m} \delta_\beta y^\beta$ est la forme algébrique normale de la fonction indicatrice δ_{y_0}.

Démonstration. Considérons une relation donnée pour un y_0 :

$$\forall x \in S_m^{-1}(y_0), \sum_{\alpha \in \mathbb{F}_2^n} c_\alpha x^\alpha = 0.$$

On en déduit les relations générales suivantes :

$$\sum_{\alpha \in \mathbb{F}_2^n} c_\alpha x^\alpha \delta_{y_0}(y), \forall (x, y), y = S_m(x)$$

où δ_{y_0} a déjà été définie à la page 14. □

Corollaire 1. *Soit F une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^m$. Alors le degré minimal d atteignable pour une relation algébrique pour F correspond à la plus petite valeur d pour laquelle il existe une relation conditionnée de degré d.*

Il est donc aisé de passer d'une variante à l'autre et on voit bien que seul le point de vue et la formulation des équations diffèrent et non le fond du problème et sa solution.

Malheureusement, ce changement d'échelle ne résout pas tout. En effet, si la taille mémoire occupée est ainsi fortement diminuée, on doit chercher les équations pour tous les y les uns après les autres ce qui impose deux boucles imbriquées de calculs de longueurs 2^n et 2^m, ce qui, pour des n courants pour les LFSR classiques limite considérablement la taille de m.

Il est donc intéressant d'étudier le problème consistant à déterminer quelle est la valeur de m minimale qui permet d'obtenir suffisamment de relations pour retrouver l'état initial.

2.4 Est-on sûr de bien obtenir des équations ?

On est certain d'obtenir des équations à partir d'une certaine valeur de m. En effet, dès que $U_y < D$, le noyau de ${}^{\mathrm{t}} M_y$ n'est plus réduit à l'ensemble vide et on trouve donc au moins une équation. On est sûr que cela arrive dès qu'il existe une valeur de y pour laquelle le nombre de lignes de M_y est plus petit que D, car alors, dans ce cas, au moins un des y a un nombre d'antécédents inférieur à 2^{n-m}. Du théorème 3, Fischer et Meier déduisent la valeur minimale de m, donnée dans la table 1 de [FM07], à partir de laquelle on est certain d'obtenir une équation indépendamment de la fonction de filtrage f et de la fonction de mise à jour L.

Théorème 3. *[FM07] Soit f une fonction booléenne de filtrage à n variables, et S_m sa fonction augmentée à m sorties. Alors, si $m \geq m_0$ avec $m_0 = n + 1 - \lfloor \log_2(D) \rfloor$ où $D = \sum_{i=0}^{d} \binom{n}{i}$, la fonction S_m possède des relations algébriques de degré inférieur ou égal à d quelle que soit la fonction de mise à jour linéaire L utilisée.*

2.5 Influence du nombre d'antécédents de y_0

En revanche, [FM07] ne donne aucun moyen de déterminer, a priori, quelle va être la plus petite valeur de m telle qu'il y ait une relation algébrique, ni la plus petite valeur telle qu'il y ait un nombre suffisant de relations pour retrouver les états initiaux dans tous les cas, ni même une meilleure approximation que m_0 qui ferait intervenir certaines propriétés de f ou de L. Une idée que nous avons explorée consiste à déterminer, en fonction de f et de L, à partir de quel m on arrive à un grand déséquilibre entre les tailles des matrices M_y pour les différents y ou plus simplement un m pour lequel on commence à obtenir de petites matrices, c'est-à-dire des matrices avec $U_y \sim D$. Mais comme le montrent nos résultats de simulation exposés au tableau 1, même si avec les m pour lesquels $U_y < D$ on a nécessairement des équations, la réciproque est loin d'être observée en pratique.

Fonction de filtrage	Premier m tel que	
	$\lvert S^{-1}(y)\rvert < D$	il y ait une relation
CanFil1	13	13
CanFil2	13	13
CanFil3	13	13
CanFil4	10-11	8-10
CanFil5	13-14	6-11
CanFil6	13	9-13
CanFil7	13	8-9

TAB. 1 – Comparaison entre la valeur de m pour laquelle des relations de degré 1 apparaissent, et celle pour laquelle la taille des matrices M_y est inférieure à n, pour les fonctions de filtrage et de mise à jour utilisées dans [FM07].

Autrement dit, l'existence pour certaines matrices M_y de noyau non trivial pour des valeurs de m très inférieures à m_0 ne semble pas provenir la plupart du temps d'un déséquilibre de la fonction S_m.

3 De la théorie à la pratique

3.1 Description de l'algorithme de recherche des équations

3.1.1 Chercher les équations

Comme nous l'avons vu, Fischer et Meier proposent deux façons de procéder, soit avec une unique matrice M (*algorithme 2*), soit avec 2^m matrices M_y (*algorithme 3*), auxquelles nous pouvons ajouter un troisième algorithme (*algorithme 4*) moins coûteux en mémoire que l'algorithme 2 et ne

nécessitant pas la boucle de calculs sur les y qu'il y a dans l'algorithme 3. Ce dernier algorithme est donc en fait une variante du premier où l'on ne stocke que les $\{x^\alpha S_m(x) \text{ tq } w(\alpha) \leq d\}$ au lieu des $\left\{x^\alpha S_m(x)^\beta \text{ tq } w(\alpha) \leq d, \beta \in \mathbb{F}_2^m\right\}$ mais où l'on doit procéder à un tri sur les $S_m(x)$ pour récupérer les matrices M_y utilisées dans le deuxième algorithme avant de terminer par un pivot de Gauss sur ces M_y.

Algorithme 2 Algorithme de recherche des relations algébriques pour S_m, cas avec une unique matrice M

Entrées : une fonction de mise à jour L et une fonction de filtrage f.

Sorties : les relations de la forme $\sum c_{\alpha,\beta} x^\alpha S_m(x)^\beta = 0$ de degré au plus d en x.

{**Calcul de la matrice génératrice G du LFSR**}

À partir de L, calculer la matrice génératrice G telle que :

$$(x_0 \dots x_{n-1}) \begin{pmatrix} 1 & & 0 & \cdot & \dots & \cdot \\ & \ddots & & \cdot & \dots & \cdot \\ 0 & & 1 & \cdot & \dots & \cdot \end{pmatrix} = (x_0 \dots x_{n-1+m})$$

{**Calcul de M**}
pour tout x dans $\mathbb{F}_2^n$ **faire**
  calculer $S_m(x) \in \mathbb{F}_2^m$
  pour tout $\alpha \in \mathbb{F}_2^n$ tq $w(\alpha) \leq d$ **faire**
    pour tout $\beta \in \mathbb{F}_2^m$ **faire**
      stocker $x^\alpha S_m(x)^\beta$
    fin pour
  fin pour
fin pour
{**Recherche des relations**}
Faire un pivot de Gauss sur M pour trouver son rang et les relations algébriques pour S_m

---

Algorithme 3 Algorithme de recherche des relations algébriques conditionnées pour y, cas avec une matrice M_y pour chaque y

Entrées : une fonction de mise à jour L et une fonction de filtrage f.

Sorties : pour chaque y, les relations conditionnées par y de la forme $\sum_\alpha c_\alpha x^\alpha = 0$ pour tout $x \in S_m^{-1}(y)$ de degré au plus d

{**Calcul de la matrice génératrice G du LFSR**}

À partir de L, calculer la matrice génératrice G telle que :

$$(x_0 \dots x_{n-1}) \begin{pmatrix} 1 & & 0 & \cdot & \dots & \cdot \\ & \ddots & & \cdot & \dots & \cdot \\ 0 & & 1 & \cdot & \dots & \cdot \end{pmatrix} = (x_0 \dots x_{n-1+m})$$

pour tout y dans $\mathbb{F}_2^m$ **faire**
  {**Calcul de** M_y}
  pour tout x dans $\mathbb{F}_2^n$ **faire**
    calculer $S_m(x) \in \mathbb{F}_2^m$
    si $S_m(x) = y$ **alors**
      pour tout $\alpha \in \mathbb{F}_2^n$ tq $w(\alpha) \leq d$ **faire**
        stocker x^α dans M_y
      fin pour
    fin si
  fin pour
  {**Recherche des relations**}
  Faire un pivot de Gauss sur M_y pour trouver son rang et les relations de dépendance linéaires
fin pour

Algorithme 4 Algorithme de recherche des relations algébriques conditionnées, cas hybride

Entrées : une fonction de mise à jour L et une fonction de filtrage f.
Sorties : pour chaque y, les relations conditionnées par y, de la forme $\sum_\alpha c_\alpha x^\alpha = 0$ pour tout $x \in S_m^{-1}(y)$, de degré au plus d.

{**Calcul de la matrice génératrice G du LFSR**}
À partir de L, calculer la matrice génératrice G telle que :

$$(x_0 \dots x_{n-1}) \begin{pmatrix} 1 & & 0 & \cdot & \dots & \cdot \\ & \ddots & & \cdot & \dots & \cdot \\ 0 & & 1 & \cdot & \dots & \cdot \end{pmatrix} = (x_0 \dots x_{n-1+m})$$

{**Calcul de M'**}
pour tout x dans $\mathbb{F}_2^n$ **faire**
  calculer $S_m(x) \in \mathbb{F}_2^m$
  pour tout $\alpha \in \mathbb{F}_2^n$ tq $w(\alpha) \leq d$ **faire**
    stocker x^α
    Ajouter $S_m(x)$ à la fin de la ligne courante dans M'
  fin pour
  incrémenter le nombre d'antécédents de $S_m(x)$
fin pour
{**Préparation de M'**}
trier M' en fonction de la valeur des m derniers bits de chaque ligne
{**Recherche des relations**}
pour tout y dans $\mathbb{F}_2^m$ **faire**
  faire un pivot de Gauss sur la sous-matrice de M' dont les m derniers bits forment y et qu'on a enlevés.
fin pour

3.1.2 Complexité du précalcul

Ici, on ne s'intéresse qu'au cas linéaire, c'est-à-dire $d = 1$ et $D = n + 1$, qui est également le principal cas traité dans [FM07].

- Cas de l'algorithme avec une unique matrice M (*algorithme* 2) :
  1. le calcul de la matrice génératrice G du LFSR est en $\mathcal{O}(m)$;
  2. l'obtention d'une unique matrice M coûte 2^n pas de boucle lors desquels on fait un calcul de $S_m(x)$ puis des $x^\alpha S_m(x)^\beta$ pour tout couple $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ avec $w(\alpha) \leq 1$ soit une complexité en $\mathcal{O}\left(2^n\left(m + 2^m(n+1)\right)\right)$;
  3. le pivot de Gauss sur M coûte $\mathcal{O}\left(n^2 2^{2m+n}\right)$ car M est de taille $2^n \times (n+1)2^m$.

  Le coût total du précalcul est donc d'environ $2^n(m + 2^m n) + n^2 2^{2m+n}$.

Le facteur dominant est donc le coût du pivot de Gauss, ce qui correspond à une complexité en $\mathcal{O}\left(2^{n+2m}n^2\right)$.

- Cas de l'algorithme avec une matrice M_y pour chaque y (*algorithme* 3) :
  1. le calcul de la matrice génératrice G du LFSR est en $\mathcal{O}(m)$.
  2. l'obtention des M_y coûte 2^m étapes pour les $y \in \mathbb{F}_2^m$, chacune examinant les 2^n valeurs de x pour lesquelles le calcul de $S_m(x)$ vaut $\mathcal{O}(m)$, auquel s'ajoute le calcul des x^α pour tout $\alpha \in \mathbb{F}_2^n$ tel que $w(\alpha) \leq 1$ quand $S_m(x) = y$, soit au total environ $2^{m+n}m + 2^n(n+1)$ opérations ;
  3. finalement, le pivot de Gauss sur M_y coûte en moyenne $n^2 2^{n-m}$, car M_y est en moyenne de taille $2^{n-m} \times (n+1)$, soit un coût total de $\mathcal{O}\left(n^2 2^n\right)$.

  Le coût total du précalcul est donc, ici, d'environ $m2^{m+n} + 2^n n + n^2 2^n$, soit en $\mathcal{O}\left(2^n(2^m m + n^2)\right)$.
- Cas de l'algorithme hybride avec une matrice M' (*algorithme* 4) :
  1. le calcul de la matrice génératrice G du LFSR est en $\mathcal{O}(m)$.
  2. l'obtention d'une unique matrice M' coûte 2^n pas de boucle lors desquels on fait un calcul de $S_m(x)$ puis des x^α pour tout $\alpha \in \mathbb{F}_2^n$ tel que $w(\alpha) \leq 1$ soit une complexité en $\mathcal{O}\left(2^n\,(m+n)\right)$;
  3. le tri de la matrice M' sur les m derniers bits de ses vecteurs coûte au pire environ $\mathcal{O}\left(2^n n\right)$;
  4. finalement, le pivot de Gauss sur les sous-matrices de M' coûte en moyenne $n^2 2^{n-m}$, car elles sont en moyenne de taille $2^{n-m} \times n$, soit un coût total de $\mathcal{O}\left(n^2 2^n\right)$.

  Le coût total du précalcul est donc, ici, d'environ $2^n\left(m + 2n + n^2\right)$.. Le facteur dominant est maintenant le coût des pivots de Gauss, ce qui conduit à une complexité en $\mathcal{O}\left(n^2 2^n\right)$ qui ne dépend pas de m pour des valeurs de m raisonnables.

Complexité en mémoire. Lors du précalcul il faut soit stocker M, soit M_y, soit M'. La première matrice fait exactement $2^n \times n2^m$, la seconde a en moyenne une taille de $2^{n-m} \times n$, et la dernière fait $2^n \times (n+m)$.

	Temps de précalcul	Mémoire
Algorithme 2	$2^{n+2m}n^2$	$2^{n+m}n$
Algorithme 3	$2^n(2^m m + n^2)$	$2^{n-m}n$
Algorithme 4	$n^2 2^n$	$2^n(n+m)$

TAB. 2 – Comparatif des complexités des trois algorithmes proposés.

3.1.3 Complexité de l'attaque

Données nécessaires. Notons m_1 le m minimal tel qu'il existe au moins un $y_0 \in \mathbb{F}_2^m$ pour lequel une relation conditionnée de degré d existe. Remarquons que l'on a toujours $m_1 \leq m_0$.

Supposons que l'on dispose d'une relation algébrique de degré 1 pour S_{m_1} de la forme

$$\left(\sum_{\beta\in\mathbb{F}_2^m} c_{0,\beta}y^\beta\right) + \sum_{i=1}^{n}\left(\sum_{\beta\in\mathbb{F}_2^m} c_{i,\beta}y^\beta\right)x_i = 0$$

obtenue avec l'algorithme 2. Alors l'observation de m_1 bits consécutifs de suite chiffrante, y, fournit une relation linéaire sur les bits de l'état initial si et seulement s'il existe i, $1 \leq i \leq n$, tel que

$$g_i(y) = \sum_{\beta\in\mathbb{F}_2^m} c_{i,\beta}y^\beta \neq 0.$$

Notons $\mathcal{S} = \bigcup_{i=1}^{n}\{y \in \mathbb{F}_2^{m_1} \text{ tq } g_i(y) = 1\}$. Comme l'observation de N bits de suite chiffrante correspond à $(N - m_1 + 1)$ valeurs de y, on en déduit que pour obtenir n relations linéaires sur les bits de l'état initial, il faut observer de l'ordre de

$$N = \frac{n2^{m_1}}{|\mathcal{S}|} + m_1 - 1$$

bits de suite chiffrante.

Si l'on utilise des relations algébriques de degré 1 conditionnées, trouvées par exemple avec l'algorithme 3 ou 4, la complexité en données de l'attaque est identique. En effet, la relation algébrique de degré 1 obtenue avec l'algorithme 2 correspond exactement à une relation de degré 1 conditionnée par y, pour tout $y \in \mathcal{S}$.

Complexité en temps de l'attaque. Il suffit de faire tourner le LFSR suffisamment pour récupérer assez d'équations indépendantes, puis de résoudre un système à n équations et n inconnues, ce qui se fait aisément en $\mathcal{O}\left(n^3\right)$.

3.1.4 Problème : taille et temps nécessaires

La taille requise par M est de 8×2^n octets pour stocker les pointeurs avec un processeur 64 bits et $\frac{n}{8}2^n$ octets pour son contenu. En effet, M est stockée de façon à ce que chaque x corresponde à une ligne, ce qui entraîne l'utilisation de plus de pointeurs que si l'on stockait la transposée de M, mais rend beaucoup plus aisé son remplissage car l'accès à la bonne ligne lorsque l'on connait x est alors trivial. En gigaoctets, cela donne $\left(\frac{n}{8} + 8\right)2^{n-30}$ Go. Donc, même avec 8 Go, on ne peut aller qu'à $n \approx 30$. Or la longueur du

LFSR dans un registre filtré est généralement de l'ordre de 200 ce qui rend cette attaque inefficiente en pratique.

À l'opposé, si l'on se contente de stocker M_y, l'espace mémoire requis est de $\left(\frac{n}{8}+8\right)2^{n-m}$ octets. Donc, quand m est de l'ordre de m_0, avec $m_0 = n+1-\lfloor \log_2(D) \rfloor = n+1-\lfloor \log_2(n+1) \rfloor$, on a besoin de $\left(\frac{n}{8}+8\right)2^{\log_2(n+1)-1}$, ce qui ne pose plus problème. En revanche, la complexité en temps est alors en $\mathcal{O}\left(2^n\left(2^m m + n^2\right)\right)$, ce qui est bloquant assez rapidement, d'autant plus que dès que m_1 est suffisamment grand, le chercher nécessite de procéder par tâtonnements car on ne le connaît pas a priori.

3.2 Maintenir m petit

Une solution pour remédier à ces problèmes, en particulier à celui du temps de calcul, est de se limiter à des m petits. Alors, au lieu d'être toujours vraies, les équations ne le sont plus qu'avec une forte probabilité $p = 1-\varepsilon$, avec $\varepsilon \ll 1$. On a alors les modifications suivantes par rapport à l'algorithme hybride (*algorithme* 4) :

- on fixe m a priori, il n'y a donc plus à le chercher ;
- vu que les équations ne sont plus vraies qu'avec une certaine probabilité, il en faut un nombre $\nu > n$ pour nous ramener à un problème de décodage (*algorithme* 1) d'un code linéaire de longueur ν et de dimension n pour le canal binaire symétrique de probabilité d'erreur ε ; si l'on note alors $\mathrm{C}(\varepsilon) = 1 + \varepsilon \log_2(\varepsilon) + (1-\varepsilon)\log_2(1-\varepsilon)$ la capacité d'un tel canal, il nous faut au moins $\nu \geq \frac{n}{\mathrm{C}(\varepsilon)}$ et donc pour les calculs de complexité suivants nous utiliserons l'égalité $\nu = \lceil \frac{n}{\mathrm{C}(\varepsilon)} \rceil$;
- dans l'étape de précalcul, le pivot de Gauss est remplacé par une recherche de mots de petit poids, et par exemple par le calcul exhaustif des poids de toutes les combinaisons linéaires possibles, ce qui coûte $2^n 2^{n-m} = 2^{2n-m}$ en moyenne pour chaque sous-matrice de M', soit 2^{2n} au total. Des algorithmes plus perfectionnés peuvent être utilisés : la recherche exhaustive utilisant la transformée de Fourier rapide, l'algorithme de Canteaut-Chabaud [CC98]. À chaque fois les relations entre les colonnes donnant les mots de plus petits poids sont celles qui seront utilisées.

L'attaque est également modifiée : à la place d'une attaque algébrique, on se situe dans le cadre des attaques par corrélation. En effet, on a maintenant affaire à un mot transmis à travers un canal binaire symétrique et on essaye de le décoder grâce à l'algorithme de décodage par ensemble d'information [Can96] (*algorithme* 1) dont la complexité est en $\mathcal{O}\left(n^2 \nu \frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}}\right)$ (*cf.* 1). On utilise cet algorithme car sa complexité est plus simple à calculer, mais lors d'une implémentation pratique, nous utiliserions plutôt une variante plus efficace, comme l'algorithme de Canteaut-Chabaud [CC98].

Pour pouvoir comparer avec les complexités précédentes, il faut évaluer ν et les binomiaux ci-dessus pour n grand et ε petit. Cela donne le résultat asymptotique suivant d'après les calculs détaillés à l'annexe A.

$$\mathcal{O}\left(n^2\nu\frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}}\right) = \mathcal{O}\left(n^3\left(1-(\ln 2)^{-1}\varepsilon(1-\ln\varepsilon)+o\left(\varepsilon\right)\right)e^{2n(\varepsilon+o(\varepsilon))}\right)$$

Le surcoût est donc asymptotiquement raisonnable avec des paramètres bien choisis, typiquement quand $n\varepsilon$ reste constant.

3.3 Maintenir très petit ($m \leq 3$)

Avec un m très petit, on peut aussi limiter le nombre de calculs à faire sur les états du LFSR car la fonction de filtrage f ne voit pas passer des éléments dérivant de chacun des $x_{i\in[1\ldots n]}$ avant quelques tours, bien que cela arrive assez rapidement si L joue correctement son rôle de rétroaction. Si P est le polynôme de rétroaction associé à L et $w(P)$ son poids de Hamming, et si v est le nombre de variables de f, alors le nombre de x_i passant dans S_m varie en moyenne comme indiqué dans le tableau 3.

m	x_i passant dans S_m
1	v
2	$< 2v + wt(P) - 1$
3	$\sim \frac{3}{4}n$
$\vdots$	$\vdots$
6	n

TAB. 3 – Nombre de x_i passant dans S_m

On voit donc qu'on ne peut y gagner qu'avec un m très petit de l'ordre de 2 ou 3. L'attaque va donc ressembler fortement à une attaque par corrélation rapide [MS88]. Par analogie avec ce cas, il faut alors regarder ce qui se passe avec $p = \frac{1}{2}+\varepsilon$, toujours avec $\varepsilon \ll 1$. Les résultats classiques [CT00] montrent alors que la complexité en données devient exponentielle en n.

3.4 N'explorer qu'un nombre limité d'x

Néanmoins, maintenir m petit n'est toujours pas suffisant car le coût de tous les algorithmes de précalcul est exponentiel en n (*cf.* tableau 2), même si les algorithmes autres que l'algorithme hybride ont aussi un facteur exponentiel en m.

Une autre solution consiste alors à ne pas parcourir tous les x possibles mais seulement un sous-ensemble de $\mathbb{F}_2^n$. Cela est d'ailleurs suggéré par Fischer et Meier, mais ils essaient d'obtenir des équations vraies avec une

probabilité de l'ordre de $99,999\%$ et ensuite de résoudre le système obtenu classiquement. En effet, l'attaque de Fischer et Meier ne fonctionne que quand on a trouvé n relations de degré 1 vérifiées par les bits de l'état initial. Si la relation trouvée dans l'étape de précalcul n'est satisfaite qu'avec une probabilité $(1-\varepsilon)$, le coût de l'attaque et la complexité en données sont multipliés par un facteur $(1-\varepsilon)^{-n}$. Ceci impose une valeur de ε très faible, par exemple, pour $\varepsilon = 10^{-1}$, la complexité en données de l'attaque est multipliée par 10^9.

Nous pouvons pousser l'idée que nous avons développée précédemment plus loin en diminuant de façon conséquente la taille du sous-ensemble de $\mathbb{F}_2^n$ à explorer. Les relations obtenues ne sont alors vérifiées qu'avec une probabilité $(1-\varepsilon)$, où ε est petit. Une attaque beaucoup plus efficace que celle suggérée dans [FM07] consiste alors à utiliser le décodage par ensemble d'information, ce qui nous conduit à la complexité précédemment évoquée.

Les figures 2 et 3 comparent alors la complexité en temps des deux attaques quand ε varie, dans le cas où $n = 256$: celle de [FM07] avec des relations algébriques satisfaites avec probabilité $(1-\varepsilon)$ (courbes rouges), et celle que nous avons proposée (courbes bleues). On constate que l'attaque de [FM07] devient inaccessible dès que ε dépasse 5.10^{-2} car elle nécessite la connaissance de plus de 2^{m+19} bits de suite chiffrante. Pour notre attaque, le surcoût en termes de données est très faible : le nombre de bits de suite chiffrante requis ne dépasse pas $2^{m+1,84}$ quand $\varepsilon \leq 0,2$. Notre attaque a en revanche un surcoût en termes de temps de calcul, mais celui-ci n'excède pas 2^{20} même quand ε augmente.

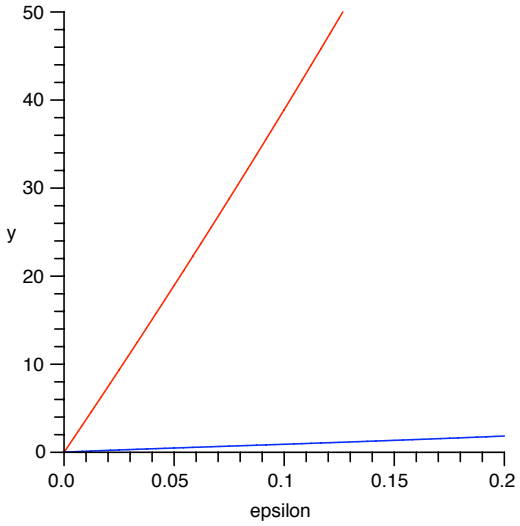

FIG. 2 – Complexité en données : ($\log_2$ (nb de bits de suite chiffrante) - m)

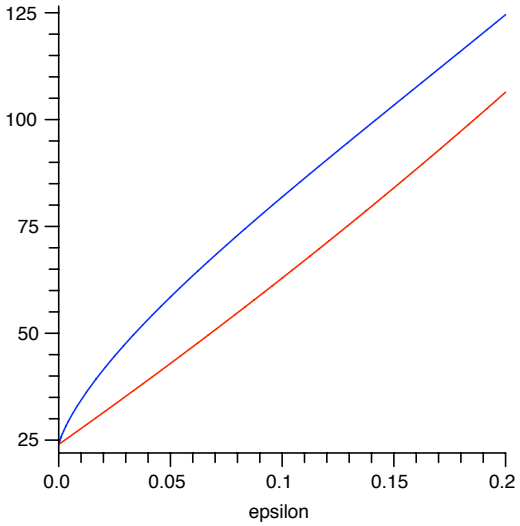

FIG. 3 – Complexité en temps : ($\log_2$ (nombre d'opérations) - m)

4 Conclusion

Fischer et Meier ont proposé en 2007 dans [FM07] un nouveau type d'attaques algébriques, particulièrement intéressant car il s'applique à des algorithmes de chiffrement immunisés jusqu'à présent contre les attaques algébriques. Nous avons alors cherché à l'adapter pour le rendre utilisable en pratique, d'où les trois améliorations que nous proposons :

- *un algorithme hybride*, se situant entre les deux proposés par Fischer et Meier et exploitant au mieux les contraintes de temps et d'espace mémoire disponibles ;
- *un moyen de garder m petit* pour limiter la complexité ;
- *une restriction sur le nombre de valeurs initiales parcourues* permettant aussi un gain important en complexité.

Ces améliorations permettent de rendre l'attaque réaliste pour des tailles de paramètres de l'ordre de celles des systèmes de chiffrement existants, ce qui n'était pas le cas de l'attaque initialement présentée par Fischer et Meier. Toutefois, il reste à étudier les influences des fonctions de filtrage et de mise à jour sur la vulnérabilité des algorithmes de chiffrement à ce nouveau type d'attaque.

Références

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C.P. Shnorr. RSA and Rabin functions : certain parts are as hard as the whole. *SIAM Journal on Computing*, 17 :194–209, 1988.

[And95] R. J. Anderson. Searching for the optimum correlation attack. In *Fast Software Encryption 1994*, volume 1008 of *Lecture Notes in Computer Science*, pages 137–143. Springer-Verlag, 1995.

[AR99] Y. Aumann and M.O. Rabin. Information theoretically secure communication in the limited storage space model. In *Advances in Cryptology - CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 65–79. Springer-Verlag, 1999.

[Bab98] S. Babbage. A space/time trade-off in exhaustive search attacks on stream ciphers. (408) :367–378, janvier 1998.

[Can96] Anne Canteaut. *Attaques de cryptosystèmes à mots de poids faible et construction de fonctions t-résilientes*. PhD thesis, Université Paris 6, octobre 1996.

[Can06] A. Canteaut. *Analyse et conception de chiffrements à clef secrète*. Mémoire d'habilitation à diriger des recherches, Université Paris 6, septembre 2006. `http://www-rocq.inria.fr/codes/Anne.Canteaut/canteaut-hdr.pdf`.

[CC98] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code : application to primitive narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1) :367–378, janvier 1998.

[CM03] N. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.

[CT00] A. Canteaut and M. Trabbia. Improved fast correlation attack using paritycheck equations of weight 4 and 5. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT'2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.

[ECR05] ECRYPT - European Network of Excellence in Cryptology. The eSTREAM Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`, 2005.

[EJ02] P. Ekdahl and T. Johansson. *A new version of the stream cipher SNOW*, volume 2295 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2002.

[FM07] S. Fischer and W. Meier. Algebraic immunity of S-boxes and augmented functions. In *Fast Software Encryption - FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 366–381. Springer, 2007.

[Gol97] J. Golic. *Cryptanalysis of alleged A5 stream cipher*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997.

[Hit01] Hitachi, Ld. Mugi specification, 2001. `http://www.sdl.hitachi.co.jp/crypto/mugi/mugi_spe.pdf`.

[HJM05] M. Hell, T. Johansson, and W. Meier. Grain : A stream cipher for constrained environments. Soumission au projet eSTREAM [ECR05], 2005. `http://www.ecrypt.eu.org/stream/`.

[ISO05] ISO/IEC 18033-4. Information technology - security techniques - encryption algorithms - part 4 : Stream ciphers, juin 2005.

[Mau92] U. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1) :53–66, 1992. Disponible sur `ftp://ftp.inf.ethz.ch/pub/crypto/publications/Maurer92b.pdf`.

[MS88] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In *Advances in Cryptology - EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer-Verlag, 1988.

[MS89] W. Meier and O. Staffelbach. Fast correlation attack on certain stream ciphers. *Journal of Cryptology*, pages 159–176, 1989.

[MvOV97] A.J. Menezes, P.C. van Oorshot, and S.A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997. Disponible sur `http://www.cacr.math.uwaterloo.a/hac/`.

[Pra62] E. Prange. *The use of information sets in decoding cyclic codes*, volume IT-8, pages S5–S9. 1962.

[Rab05] M.O. Rabin. Provably unbreakable hyper-encryption in the limited access model. In *Proceedings of the 2005 IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security*, pages 34–37. IEEE, 2005.

[Sie85] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. C-34(1) :81–84, 1985.

A Complexité de l'attaque pour ε petit

Nous avons besoin d'évaluer $\mathcal{O}\left(n^2\nu\frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}}\right)$ pour comparer la complexité obtenue à la section 3.2 avec les complexités des algorithmes donnés antérieurement à celle-ci. Pour cela, nous allons utiliser la formule suivante :

$$\frac{2^{n\mathrm{H}_2\left(\frac{t}{n}\right)}}{\sqrt{8t\left(1-\frac{t}{n}\right)}} \leq \binom{n}{t} \leq \frac{2^{n\mathrm{H}_2\left(\frac{t}{n}\right)}}{\sqrt{2\pi t\left(1-\frac{t}{n}\right)}}$$

avec $\mathrm{H}_2(t) = -t\log_2(t) - (1-t)\log_2(1-x)$.
Il en découle :

$$\begin{aligned}
\frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}} &\geq \frac{2^{\nu\mathrm{H}_2\left(\frac{n}{\nu}\right)}\sqrt{2\pi n\left(1-\frac{n}{\nu(1-\varepsilon)}\right)}}{\sqrt{8n\left(1-\frac{n}{\nu}\right)}2^{\nu(1-\varepsilon)\mathrm{H}_2\left(\frac{n}{\nu(1-\varepsilon)}\right)}} \\
&\geq 2^{\nu\left(\mathrm{H}_2\left(\frac{n}{\nu}\right)-(1-\varepsilon)\mathrm{H}_2\left(\frac{n}{\nu(1-\varepsilon)}\right)\right)}\sqrt{\frac{\pi}{4}\left(1-\frac{\varepsilon}{(1-\varepsilon)\left(\frac{\nu}{n}-1\right)}\right)}
\end{aligned}$$

$$\begin{aligned}
\frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}} &\leq \frac{2^{\nu\mathrm{H}_2\left(\frac{n}{\nu}\right)}\sqrt{8n\left(1-\frac{n}{\nu(1-\varepsilon)}\right)}}{\sqrt{2\pi n\left(1-\frac{n}{\nu}\right)}2^{\nu(1-\varepsilon)\mathrm{H}_2\left(\frac{n)}{\nu(1-\varepsilon)}\right)}} \\
&\leq 2^{\nu\left(\mathrm{H}_2\left(\frac{n}{\nu}\right)-(1-\varepsilon)\mathrm{H}_2\left(\frac{n}{\nu(1-\varepsilon)}\right)\right)}\sqrt{\frac{4}{\pi}\left(1-\frac{\varepsilon}{(1-\varepsilon)\left(\frac{\nu}{n}-1\right)}\right)}
\end{aligned}$$

On sait aussi que :

$$\frac{\varepsilon}{(1-\varepsilon)\left(\frac{\nu}{n}-1\right)} = \frac{\varepsilon}{1-\frac{n}{\nu}}\left(1+\mathcal{O}(\varepsilon)\right)$$

et que :

$$\begin{aligned}
\frac{\varepsilon}{1-\frac{n}{\nu}} &= \frac{\varepsilon}{1-\mathrm{C}(\varepsilon)} \\
&= \frac{\varepsilon}{(\ln 2)^{-1}\varepsilon(1-\ln\varepsilon)+\mathcal{O}(\varepsilon^2)} \\
&= \frac{1}{(\ln 2)^{-1}(1-\ln\varepsilon)+\mathcal{O}(\varepsilon)} \\
&= \mathcal{O}(1)
\end{aligned}$$

D'où :

$$\begin{aligned}\frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}} &= \mathcal{O}(1)\, 2^{\frac{n}{\mathrm{C}(\varepsilon)}\left(\mathrm{H}_2(\mathrm{C}(\varepsilon))-(1-\varepsilon)\mathrm{H}_2\left(\frac{\mathrm{C}(\varepsilon)}{(1-\varepsilon)}\right)\right)} \\ &= \mathcal{O}(1)\, 2^{n\left(\frac{\mathrm{H}_2(\mathrm{C}(\varepsilon))}{\mathrm{C}(\varepsilon)}-\frac{(1-\varepsilon)}{\mathrm{C}(\varepsilon)}\mathrm{H}_2\left(\frac{\mathrm{C}(\varepsilon)}{(1-\varepsilon)}\right)\right)}\end{aligned}$$

Il ne reste plus qu'à calculer

$$\lambda = \frac{\mathrm{H}_2(\mathrm{C}(\varepsilon))}{\mathrm{C}(\varepsilon)} - \frac{(1-\varepsilon)}{\mathrm{C}(\varepsilon)}\mathrm{H}_2\left(\frac{\mathrm{C}(\varepsilon)}{(1-\varepsilon)}\right)$$

à l'aide de quelques formules supplémentaires :

$$\mathrm{H}_2(1-\varepsilon) = \frac{\mathrm{H}_2(1-\varepsilon)}{1-\varepsilon} = \varepsilon\left(\frac{1-\ln\varepsilon}{\ln 2}\right) + \mathcal{O}\left(\varepsilon^2\log_2(\varepsilon)\right)$$

$$\mathrm{C}(\varepsilon) = 1 - (\ln 2)^{-1}\varepsilon(1-\ln\varepsilon) + \mathcal{O}\left(\varepsilon^2\right)$$

$$\frac{\mathrm{C}(\varepsilon)}{1-\varepsilon} = 1 - \varepsilon\left((\ln 2)^{-1} - 1 - \log_2(\varepsilon)\right) + \mathcal{O}\left(\varepsilon^2\log_2(\varepsilon)\right)$$

$$\begin{aligned}\lambda &= \left(1-(\ln 2)^{-1}\varepsilon(1-\ln\varepsilon)\right)(\ln 2)^{-1}\left(1-\ln\left(1-(\ln 2)^{-1}\varepsilon(1-\ln\varepsilon)\right)\right) \\ &\quad -\left(1-\varepsilon\left((\ln 2)^{-1}-1-\log_2(\varepsilon)\right)\right)(\ln 2)^{-1}\left(1-\ln\left(1-\varepsilon\left((\ln 2)^{-1}-1-\log_2(\varepsilon)\right)\right)\right) \\ &\quad +\mathcal{O}\left(\varepsilon^2\log_2(\varepsilon)\right) \\ &= (\ln 2)^{-1}\left(\varepsilon+\ln\left(1-\varepsilon\left((\ln 2)^{-1}-1-\log_2(\varepsilon)\right)\right)-\ln\left(1-(\ln 2)^{-1}\varepsilon(1-\ln\varepsilon)\right)\right) \\ &\quad -(\ln 2)^{-1}\varepsilon\left((\ln 2)^{-1}-1-\log_2(\varepsilon)\right)\ln\left(1-\varepsilon\left((\ln 2)^{-1}-1-\log_2(\varepsilon)\right)\right) \\ &\quad +(\ln 2)^{-2}\varepsilon(1-\ln\varepsilon)\ln\left(1-(\ln 2)^{-1}\varepsilon(1-\ln\varepsilon)\right)+\mathcal{O}\left(\varepsilon^2\log_2(\varepsilon)\right)^2 \\ &= 2(\ln 2)^{-1}\varepsilon-(\ln 2)^{-3}(\varepsilon(1-\ln\varepsilon))^2+(\ln 2)^{-1}\left(\varepsilon\left((\ln 2)^{-1}-1-\log_2(\varepsilon)\right)\right)^2 \\ &\quad +\mathcal{O}\left(\varepsilon^2\log_2(\varepsilon)\right)^2 \\ &= 2(\ln 2)^{-1}\varepsilon+(\ln 2)^{-1}\varepsilon^2-2(\ln 2)^{-2}\varepsilon^2+2(\ln 2)^{-2}\varepsilon\ln\varepsilon+\mathcal{O}\left(\varepsilon^2\log_2(\varepsilon)\right)^2 \\ &= 2(\ln 2)^{-1}\varepsilon+o(\varepsilon)\end{aligned}$$

D'où :

$$\frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}} = \mathcal{O}(1)\, e^{2n(\varepsilon+o(\varepsilon))}$$

Et finalement :

$$\mathcal{O}\left(n^2\nu\frac{\binom{\nu}{n}}{\binom{\nu(1-\varepsilon)}{n}}\right) = \mathcal{O}\left(n^3\left(1-(\ln 2)^{-1}\varepsilon(1-\ln\varepsilon)+o(\varepsilon)\right)e^{2n(\varepsilon+o(\varepsilon))}\right)$$

B Comparaison des attaques pour différents n

Les figures 4, 5, 6 et 7 comparent la complexité en temps des deux attaques quand ε varie, dans le cas où $n = 512$ pour les deux premières et où $n = 1024$ pour les deux dernières.

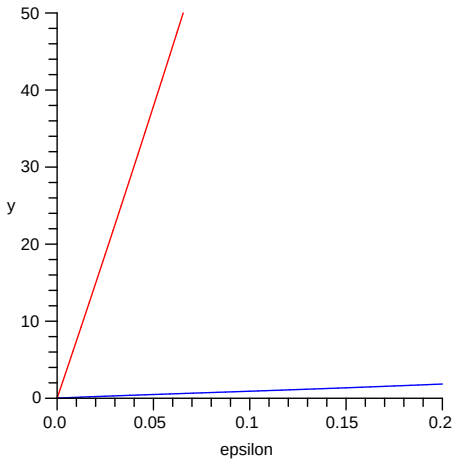

FIG. 4 – Complexité en données : ($\log_2$ (nb de bits de suite chiffrante) - m)

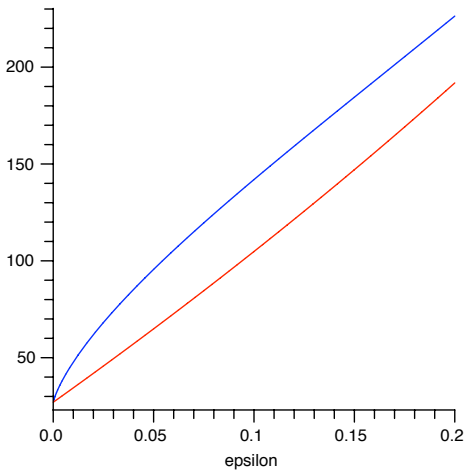

FIG. 5 – Complexité en temps : ($\log_2$ (nombre d'opérations) - m)

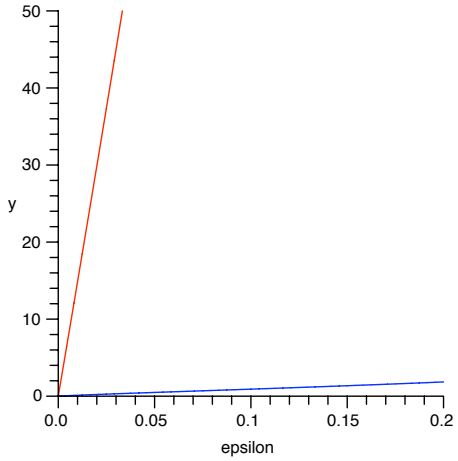

FIG. 6 – Complexité en données : ($\log_2$ (nb de bits de suite chiffrante) - m)

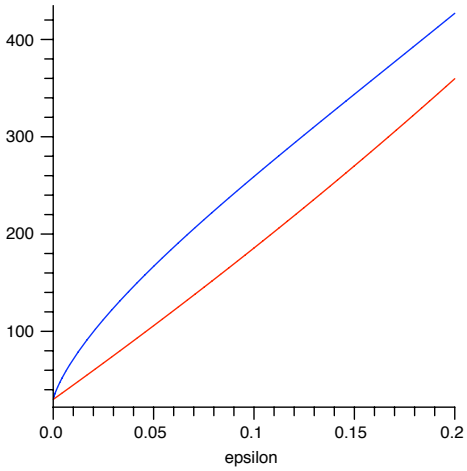

FIG. 7 – Complexité en temps : ($\log_2$ (nombre d'opérations) - m)