Cryptanalyse Algébrique de Fonctions de Hachage

Luk Bettale
encadré par Jean-Charles Faugère et Ludovic Perret
équipe SALSA LIP6

24 août 2008

Le contexte général

Le stage effectué porte sur la cryptanalyse algébrique, et plus particulièrement l'application de celle-ci au domaine des fonctions de hachage cryptographiques.

De nombreuses personnes se sont intéressées à la cryptanalyse des fonctions de hachage, mais peu ont essayé une approche algébrique. En effet, à première vue, la façon dont les principales fonctions de hachages utilisées sont actuellement construites, ce n'est pas une approche naturelle. On pense plus à la cryptanalyse différentielle, et aux attaques proposés par Wang [25, 28]. Nous avons essayé cette approche algébrique sur une fonction de hachage utilisée en pratique (SHA-1), et sur une fonction qui se prête plus à ce genre d'analyse puisqu'il s'agit d'une nouvelle famille de fonction de hachage basée sur l'évaluation d'un système polynomial en plusieurs variables qu'on appelle fonctions de hachages multivariées.

Dans le domaine de l'analyse algébrique des fonctions de hachage "classiques", quelques travaux ont été accomplis, notamment par Sugita et al, que nous avons repris et étudiés dans le cadre du stage. Ils mélangent une attaque différentielle existante (Wang) avec une approche algébrique, ou du moins qui peut être modélisé de façon algébrique. En ce qui concerne les fonction de hachage multivariées, domaine assez récent (2007), il n'y a pas encore eu beaucoup de travaux, mais parmi ceux-ci, on trouve un article de Aumasson et Meier [2] qui mettent en évidence quelques problèmes dans la construction de certaines catégories de ces fonctions, et plus récemment une attaque en préimage qui est légèrement meilleure que la recherche exhaustive [18].

Le problème étudié

Nous avons essayé dans un premier temps de faire une analyse algébrique de SHA-1, fonction bien connue et largement utilisée, et dans un deuxième temps, nous avons étudié une famille de fonctions de hachage définie explicitement par un système polynomial multivarié.

Il est important d'essayer une approche algébrique sur des fonctions existantes, et il est aussi important d'essayer d'estimer la vulnérabilité des nouvelles fonctions proposés, notamment en donnant si possible des bornes sur la complexité d'une attaque algébrique avec les outils actuels.

Dans le cas de SHA-1, d'autres personnes, notamment Sugita et al ont évoqué l'idée d'une attaque algébrique, et laissaient une piste à explorer. Ils n'ont pas approfondi le côté algébrique

de leur attaque, et la contribution apportée était de tester en pratique si leur approche est efficace, c'est à dire si les outils actuels permettent un gain dans la recherche de collisions dans le cadre d'une attaque algébrique.

Dans le cas des fonctions de hachages multivariées, le problème est différent : On a déjà le problème sous une forme algébrique, et le but est de voir à quel point les schémas proposés sont résistants.

La contribution proposée

Pour SHA-1, nous avons proposé une modélisation de la fonction sous la forme d'un système de polynômes de degré inférieur ou égal à 3. Bien qu'on ne puisse pas l'utiliser telle quelle pour une attaque du fait du trop grand nombre de variables, elle peut être utilisable pour d'autres approches algébrique, c'est d'ailleurs ce que nous avons fait en utilisant cette modélisation pour implémenter l'attaque de Sugita [24], même s'il s'est avéré que l'attaque nécessitait le calcul de polynômes de trop haut degré impossible à réaliser en pratique.

Pour les fonctions de hachage multivariées, nous avons pu donner une borne de complexité sur une attaque en collision en se servant d'une technique que nous avions déjà présenté dans un article à la conférence AFRICACRYPT 2008 [8], et en remarquant que les systèmes étudiés se comportaient comme des systèmes ayant certaines propriétés (semi-régularité). De ce côté là, la démarche a été plutôt expérimentale, et le but était de voir comment se comportent les fonctions proposées face à des attaques existantes. Cette partie du stage a donné lieu à la rédaction d'un article que nous soumettrons à la conférence Inscrypt 2008.

Les arguments en faveur de sa validité

En ce qui concerne la modélisation de SHA-1, celle que nous proposons est un bon compromis entre le nombre de variables et le degré des polynômes. À notre connaissance, aucune modélisation de cette fonction n'a été publiée.

Pour les fonctions de hachage multivariées, notre travail a permis de mesurer la sécurité de différents types de construction proposés. Nous avons été en mesure de trouver effectivement des collisions sur certains paramètres.

Le bilan et les perspectives

Pour les deux parties du stage, l'approche est tout à fait générale, et l'analyse peut s'adapter à d'autres fonctions de hachage, mais aussi à d'autres primitives cryptographiques comme des schémas de chiffrement par blocs. Notre modélisation de SHA-1 peut très bien être utilisée comme un point de vue nouveau pour une analyse plus détaillée de la fonction, voire comme nouvelle base pour trouver des biais qui permettraient d'arriver à de nouvelles collisions, piste que nous n'avons pas abordé dans ce stage et qui pourrait être étudiée.

Les fonctions de hachage multivariées sont à notre avis une bonne piste pour l'avenir des fonctions de hachage quand il y a un besoin de sécurité prouvable, c'est pourquoi elles ont besoin d'être étudiées plus en profondeur.

Table des matières

1 Introduction

La cryptanalyse algébrique est une discipline assez récente qui consiste en l'étude de primitives cryptographiques avec des outils algébriques. On se ramène à l'étude de systèmes polynomiaux en plusieurs variables. Dans le cadre de ce stage, on a tenté de voir si une approche algébrique pourrait s'avérer efficace pour l'étude des fonctions de hachage. Dans un premier temps, nous présenterons donc les principes d'une fonction de hachage, ainsi que l'outil utilisé pour résoudre des systèmes polynomiaux multivarié, à savoir la base de Gröbner de l'idéal engendré par le système, ensuite, nous présenterons le travail qui a constitué la première partie de ce stage, à savoir l'étude de la fonction de hachage SHA-1 et les différentes approches étudiées pour son analyse. Enfin, nous parlerons de la deuxième partie du stage qui était l'étude d'une fonction de hachage basée directement sur l'évaluation d'un système de polynômes en plusieurs variables.

2 Fonctions de hachage cryptographiques

Dans cette section, nous rappelons brièvement le principe et l'utilité des fonctions de hachage cryptographiques, ainsi que les différents critères de sécurité et les types d'attaques existants. Par la suite, on écrira indifféremment *fonction de hachage* pour *fonction de hachage cryptographique* par abus de langage.

2.1 Présentation

Une fonction de hachage permet de générer à partir d'un message de longueur quelconque une somme de contrôle, ou empreinte, de longueur fixée. Si on note $\mathcal{A}$ l'alphabet du message, On définit plus formellement une fonction de hachage par

$$h : \mathcal{A}^* \to \mathcal{A}^m$$

Dans la pratique, on voudra hacher une suite de bits, et $\mathcal{A} = \{0, 1\}$.

On veut qu'il soit "difficile" de trouver deux messages qui donnent la même empreinte (collision) : on appelle une fonction ayant une telle propriété Collision Resistant Hash Function (CRHF). Pour une fonction de hachage cryptographique, on veut en plus qu'il soit "difficile" de retrouver le message à partir de son empreinte : on appelle une fonction ayant une telle propriété One Way Hash Function (OWHF). On emploie ici le terme "difficile" pour dire "impossible en pratique".

La méthode la plus couramment utilisée pour obtenir une telle fonction, est la construction de Merkle-Damgård. Elle consiste à décomposer le message d'origine de longueur inférieure à 2^k en blocs de taille fixée l, quitte à ajouter des bits à la fin du message (padding) pour pouvoir encoder la taille du message sur les k derniers bits et avoir un nombre de bits multiple de l. Chaque bloc de message ainsi qu'une variable de chaînage sont envoyés dans une fonction de compression f, et le résultat devient la variable de chaînage suivante. Si le message se découpe en k blocs $(x_0, \ldots, x_{k-1})$, on note c_0 la première variable de chaînage appelée aussi Initial Value (IV), on aura donc

$$c_{i+1} = f(x_i, c_i)$$

où $f : \mathcal{A}^l \to \mathcal{A}^m$ est la fonction de compression. Le hash correspond à la dernière variable de chaînage

$$h(x) = c_k$$

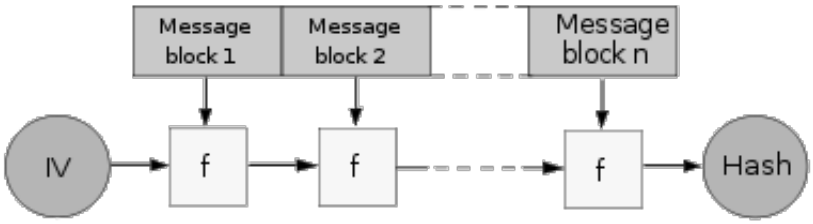

FIG. 1 – Construction de Merkle-Damgård

Les fonctions de hachages les plus utilisées à l'heure actuelle sont construites sur ce principe, il s'agit de MD5, et SHA-1. La sécurité d'une fonction de hachage repose principalement sur celle de la fonction de compression.

Théorème 1. *(Merkle-Damgård) Soit h la fonction de hachage construite par la méthode de Merkle-Damgård en itérant une fonction f. Pour une empreinte donnée, si on trouve une collision sur h, alors on peut en déduire une collision sur f.*

Démonstration. Soient x et x' tels que $x \neq x'$ et $h(x) = h(x')$. 2 cas se présentent :

- x et x' sont de longueur différentes :
  leur dernier bloc est différent : $x_k \neq x'_{k'}$, dans ce cas, c'est qu'à la dernière itération de l'algorithme, on a eu une collision sur f car $f(x_k, c_k) = f(x'_{k'}, c'_{k'})$
- x et x' sont de même longueur :
  Il faut autant d'itération pour les 2 messages, et à la dernière itération, on a $f(x_k, c_k) = f(x'_k, c'_k)$. Si $c_k \neq c'_k$ alors on a une collision sur f, sinon on peut remonter d'une itération et refaire le même raisonnement. Si on n'obtient pas de collision, alors $x = x'$ ce qui est en contradiction avec l'hypothèse.

□

Par contraposée, si une fonction f est résistante aux collisions, alors la fonction de hachage h construite à partir de f sera elle aussi résistante. Dans les 2 parties que nous allons traiter, on ne s'intéressera qu'à la fonction de compression.

On évalue la sécurité d'une fonction de hachage selon sa capacité à résister aux 3 différents types d'attaques que nous présentons dans les sous-sections suivantes.

2.2 Attaques en préimage et seconde préimage

Une attaque en préimage consiste à retrouver un message à partir d'une empreinte. On nomme h la fonction de hachage :

- Préimage : étant donné une empreinte y, trouver un message x tel que $h(x) = y$.
- Seconde Préimage : étant donné un message x, trouver un autre message x', $x' \neq x$ tel que $h(x) = h(x')$.

Pour n'importe quelle fonction de hachage, la complexité générique de ces 2 attaques est $\mathcal{O}((\log(\#\mathcal{A}))^n)$ en nombre de calcul d'empreintes, c'est à dire le coût d'une recherche exhaustive.

2.3 Attaque en collision

– Collision : trouver un couple de messages (x, x') tel que $h(x) = h(x')$.
La complexité générique de l'attaque en collision est $\mathcal{O}((\log(\#\mathcal{A}))^{n/2})$ en nombre de calcul d'empreintes (coût de la recherche exhaustive) en utilisant le paradoxe des anniversaires.

Après avoir présenté le contexte dans lequel nous avons travaillé, les fonctions de hachage, il convient à présent d'introduire les notions nécessaires pour décrire les attaque algébriques que nous allons tester sur ces fonctions de hachage.

3 Bases de Gröbner

Dans le cadre de la cryptanalyse algébrique, on se ramène toujours au même problème, a savoir celui de la résolution de systèmes polynomiaux en plusieurs variables. Soit $\mathbb{K}$ un corps, on définit $\mathbb{K}[x_1, \ldots, x_n]$ l'anneau des polynômes en n variables $x_1, \ldots, x_n$ à coefficients dans $\mathbb{K}$. On appelle un système polynomial un ensemble d'équations

$$\begin{cases} p_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ p_m(x_1, \ldots, x_n) = 0 \end{cases}$$

avec $p_i \in \mathbb{K}[x_1, \ldots, x_n]$ pour $i \in \{1, \ldots, m\}$.

Tout d'abord, il convient de définir les objets que l'on va manipuler, à savoir les *idéaux* et les *variétés*, ainsi que ce qu'est une *base de Gröbner* d'un *idéal*. Nous verrons ensuite l'utilité d'une telle base, et de quels moyens on dispose pour les calculer.

3.1 Définitions

Définition 1. *On appelle* ***idéal*** *engendré par $p_1, \ldots, p_m \in \mathbb{K}[x_1, \ldots, x_n]$ l'ensemble*

$$\mathcal{I} = \langle p_1, \ldots, p_m \rangle = \{\sum_{k=1}^{m} p_k \cdot h_k : h_1, \ldots, h_m \in \mathbb{K}[x_1, \ldots, x_n]\} \subseteq \mathbb{K}[x_1, \ldots, x_n]$$

Définition 2. *On appelle* ***variété*** *d'un idéal $\mathcal{I} \subseteq \mathbb{K}[x_1, \ldots, x_n]$ l'ensemble*

$$V_{\mathbb{L}}(\mathcal{I}) = \{\boldsymbol{z} \in \mathbb{L}^n : p_i = 0, i \in 1, \ldots, m\}$$

où $\mathbb{L}$ est une extension du corps $\mathbb{K}$. Il s'agit de l'ensemble des valeurs qui annulent tous les générateurs de l'idéal (et par conséquent tous les polynômes de l'idéal).

En cryptologie, on cherchera plutôt à calculer $V_{\mathbb{K}}$, c'est à dire l'ensemble des solutions dans le corps des coefficients. Comme on travaille toujours sur des corps finis, pour se restreindre à ces solutions, on peut chercher la variété de l'idéal engendré par le système de départ auquel on ajoute ce qu'on appelle les équations de corps.

Définition 3. *Soit $\mathbb{K}$ un corps de caractéristique non-nulle. Si on note k le nombre d'éléments du corps $\mathbb{K}$ ($k = \#\mathbb{K}$), on appelle* ***équations de corps*** *les équations $x_i^k + x_i = 0, i \in \{1, \ldots, n\}$*

Il est souvent difficile de retrouver la variété à partir d'un ensemble de générateurs (base) quelconque de l'idéal, et une base de Gröbner, dans certains cas, permet de décrire une variété. Informellement, c'est un ensemble de générateurs de l'idéal avec de bonnes propriétés. Les bases de Gröbner sont définies suivant un ordre monomial.

Définition 4. *On note $M(n)$ l'ensemble des monômes en n variables.*
On appelle un ***ordre monomial*** *une relation $\prec$ sur $M(n)$ tel que*
(i) $\prec$ est un ordre total sur $M(n)$
(ii) si $\boldsymbol{x} \prec \boldsymbol{y}$ et $\boldsymbol{z} \in M(n)$ alors $\boldsymbol{xz} \prec \boldsymbol{yz}$
(iii) $\prec$ est un ordre bien fondé, c'est à dire que tout sous-ensemble non vide de $M(n)$ possède un plus petit élément par rapport à $\prec$
On note $LM(p, \prec)$ le monôme de tête du polynôme p par rapport à l'ordre monomial $\prec$.

Deux ordres sont utilisés en pratique, l'ordre lexicographique (lex), et l'ordre du degré lexicographique inverse (DRL)

Définition 5. *Soit $\alpha = (\alpha_1, \ldots, \alpha_n)$ et $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{N}^n$. Alors :*
- $x_1^{\alpha_1} \cdots x_n^{\alpha_n} \succ_{\text{Lex}} x_1^{\beta_1} \cdots x_n^{\beta_n}$ *si l'élément non nul le plus à gauche de $\alpha - \beta$ est positif.*
- $x_1^{\alpha_1} \cdots x_n^{\alpha_n} \succ_{\text{DRL}} x_1^{\beta_1} \cdots x_n^{\beta_n}$ *si $\sum_{i=1}^n \alpha_i > \sum_{i=1}^n \beta_i$, ou $\sum_{i=1}^n \alpha_i = \sum_{i=1}^n \beta_i$ et l'élément non nul le plus à droite de $\alpha - \beta$ est négatif.*

On a maintenant tout le nécessaire pour définir les bases de Gröbner.

Définition 6. *On appelle* ***base de Gröbner*** *d'un idéal $\mathcal{I} \subseteq \mathbb{K}[x_1, \ldots, x_n]$ par rapport à un ordre monomial $\prec$, un ensemble de polynômes*
$G \subset \mathbb{K}[x_1, \ldots, x_n]$ tel que pour tout $p \in \mathcal{I}$, il existe $g \in G$ tel que $LM(g, \prec)$ divise $LM(p, \prec)$.

La base de Gröbner pour l'ordre lexicographique permet de calculer facilement la variété de l'idéal. En effet, on peut montrer que lorsque le système admet un nombre fini de solutions dans la clôture algébrique de $\mathbb{K}$ (système zéro-dimensionnel), alors une base de Gröbner pour l'ordre lexicographique de l'idéal engendré par le système sera toujours de la forme

$$(g_1(x_1), g_2(x_1, x_2), \ldots, g_{k_2}(x_1, x_2), \ldots, g_{k_n}(x_1, \ldots, x_n))$$

À partir d'une telle base, il ne reste plus qu'à résoudre les équations univariées et à substituer les variables dans les équations suivantes pour avoir les valeurs des $(x_1, \ldots, x_n)$. C'est cette stratégie que l'on utilise pour résoudre des systèmes polynomiaux.

Pour une introduction plus complète sur les bases de Gröbner, le lecteur pourra se référer à [1] ou [12].

L'algorithme historique de calcul de bases de Gröbner est celui introduit par Bruno Buchberger[10]. Dans cet algorithme, on passe beaucoup de temps à faire des calculs inutiles, et ce malgré des critères qui permettent d'éviter certains de ces calculs.

Les algorithmes F4 [15] et F5 [16] de Jean-Charles Faugère que nous ne présenterons pas ici améliorent la complexité moyenne de calcul de bases de Gröbner (la complexité au pire cas est toujours doublement exponentielle) en utilisant intensivement l'algèbre linéaire, et en

utilisant d'autres critères pour éviter le calcul inutile. L'algorithme F4 est celui que nous avons utilisé car c'est celui qui est implanté par défaut dans la plupart des logiciels de calcul formel, et en particulier Magma [19] que nous avons utilisé pour toutes les implémentations et les calculs.

Pour la suite de ce rapport, nous avons besoin d'introduire la notion de degré de régularité et de suite semi-régulière.

Définition 7. *On appelle* ***degré de régularité*** *d'un ensemble de polynômes homogène $p_1, \ldots, p_m \in \mathbb{K}[x_1, \ldots, x_n]$ et on note $d_{reg}(p_1, \ldots, p_m)$, le plus petit entier $d \geq 0$ tel que les polynômes de degré d dans l'idéal $\mathcal{I} = \langle p_1, \ldots, p_m \rangle$ génèrent l'ensemble de tous les monômes de degré d en n variables (il y en a $\binom{n+d-1}{d}$)*

$$d_{reg}(p_1, \ldots, p_m) = min\{d \geq 0 : dim_{\mathbb{K}}(\{f \in \mathcal{I} = \langle p_1, \ldots, p_m \rangle : deg(f) = d\}) = \#M_d(n)\}$$

On peut étendre cette définition pour des polynômes non-homogènes en prenant le degré de régularité de la partie homogène de plus haut degré du système.

En pratique, le degré de régularité est le degré maximal qu'atteindront les polynômes lors du calcul d'une base de Gröbner. Avec l'algorithme F5, on sait que pour un système $(f_1, \ldots, f_m)$ avec $f_i \in \mathbb{K}[x_1, \ldots, x_n], i \in \{1, \ldots, m\}$, la complexité du calcul de base de Gröbner est

$$\mathcal{O}\left(\left(m \cdot \mathrm{C}^{d_{\mathrm{reg}}}_{n+d_{\mathrm{reg}}-1}\right)^{\omega}\right)$$

où ω est la constante d'algèbre linéaire, $2 \leq \omega \leq 3$. [5]

On ne peut pas connaître à l'avance le degré de régularité d'un système en général, sauf pour le cas des *séquences semi-régulières*

Définition 8. *Soit $p_1, \ldots, p_m \in \mathbb{K}[x_1, \ldots, x_n]$ une séquence de polynômes homogènes de degré $d_1, \ldots, d_m$. On dit que c'est une* ***séquence semi-régulière*** *si*

- $\langle p_1, \ldots, p_m \rangle \neq \mathbb{K}[x_1, \ldots, x_n]$
- *quelque soit $f \in \mathbb{K}[x_1, \ldots, x_n]$ et pour tout $1 \leq i \leq m$:*
  $f \cdot p_i \in \langle p_1, \ldots, p_{i-1} \rangle$ *et* $deg(f \cdot p_i) \leq d_{reg}(p_1, \ldots, p_{i-1}) \Rightarrow f \in \langle p_1, \ldots, p_{i-1} \rangle$

Comme pour le cas du degré de régularité, on peut étendre la notion au cas de polynômes non-homogènes en prenant la partie homogène de plus haut degré de ces polynômes.

On peut montrer que pour des séquences semi-réguliers, on pouvait connaître à l'avance le degré de régularité du système [3, 5].

Proposition 1. *Soit $(f_1, \ldots, f_m)$ une séquence semi-régulière de degré respectifs $(d_1, \ldots, d_m)$. Le degré de la séquence est donné par le premier coefficients non-positif de la suite*

$$\sum_{k \geq 0} c_k \cdot z^k = \frac{\prod_{i=1}^{m}(1 - z^{d_i})}{(1 - z)^n}$$

3.2 Applications en Cryptanalyse

Pour faire des primitives cryptographiques, on se base sur des problèmes mathématiques difficile, et celui de la résolution de systèmes polynomiaux en plusieurs variables est connu comme étant NP-complet. C'est pourquoi une famille de cryptosystèmes (le plus souvent asymétrique) basés sur ce problème a émergé, notamment C^* de Matsumoto-Imai [20], ou HFE de Patarin [23]. Les algorithmes efficaces de calcul de base de Gröbner ont permis de casser ces cryptosystèmes pour certains paramètres [17].

Maintenant, nous avons voulu savoir si dans le domaine des fonctions de hachage une approche algébrique pourrait être utilisée, et nous présenterons nos travaux sur SHA-1, et sur les fonctions de hachages multivariées.

4 Étude de SHA-1

4.1 Présentation de SHA-1

SHA-1 est une fonction de hachage cryptographique conçue par la NSA et publié comme un standard de traitement de l'information (FIPS 180-1 [21]). Elle prend des messages de taille quelconque inférieure à 2^{64} bits et produit une empreinte de 160 bits. La fonction suit la construction de Merkle-Damgård :
Tout d'abord, on ajoute à la fin du message un 1 puis autant de 0 qu'il faut pour obtenir $k \times 512 - 64$ bits, puis on encode la longueur du message en nombre de bits sur les 64 dernier bits. Le message est ensuite découpé en blocs de 512 bits qui vont être utilisés dans une fonction de compression F avec la variable de chaînage. La fonction de compression prend en entrée un bloc de 512 bits et une variable de chaînage de 160 bits pour produire la variable de chaînage suivante (ou le hash si on traite le dernier bloc du message). Nous décrivons dans ce qui suit cette fonction de compression.

$$F : \mathbb{F}_2^{512} \times \mathbb{F}_2^{160} \to \mathbb{F}_2^{160}$$

Le bloc de 512 bits est découpé en 16 mots de 32 bits $(m_0, \ldots, m_{15})$, et on étend ce vecteur en calculant pour $i = 16, \ldots, 79$

$$m_i = (m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16}) \lll 1$$

où $\oplus$ représente le *ou exclusif*, et $\lll$ une rotation des bits vers la gauche.

La variable de chaînage est aussi découpée en blocs de 32 bits qu'on va noter $(a_0, b_0, c_0, d_0, e_0)$.

Ensuite, pour $i = 1, 2, \ldots, 80$, on calcule les valeurs suivantes :

$$\begin{aligned}
a_i &= (a_{i-1} \lll 5) + f_i(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + m_{i-1} + k_i \\
b_i &= a_{i-1} \\
c_i &= (b_{i-1} \lll 30) \\
d_i &= c_{i-1} \\
e_i &= d_{i-1}
\end{aligned}$$

Le + désigne l'addition de 2 mots machine, c'est à dire une addition modulo 2^{32}.

Enfin, on retourne $(a_0 \oplus a_{80}, b_0 \oplus b_{80}, c_0 \oplus c_{80}, d_0 \oplus d_{80}, e_0 \oplus e_{80})$.

On voit que chaque tour fait intervenir des fonctions f_i et des constantes k_i que nous définissons dans ce tableau :

i	fonction f_i		constante k_i
1,...,20	IF :	$(x \wedge y) \vee (\neg x \wedge z)$	$0x5a827999$
21,...,40	XOR :	$x \oplus y \oplus z$	$0x6ed6eba1$
41,...,60	MAJ :	$(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$	$0x8fabbcdc$
61,...,80	XOR :	$x \oplus y \oplus z$	$0xca62c1d6$

Tab. 1 – Fonctions booléennes et constantes dans SHA-1

Il ne reste plus qu'à définir la variable de chaînage initiale
$IV = (a_0, b_0, c_0, d_0, e_0)$:
$IV = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476, 0xc3d2e1f0)$

Maintenant que nous avons présenté le principe de la fonction de hachage SHA-1, on voit qu'elle fait appel à des opérations booléennes, des rotations de bits et des additions modulaires. On voudrait arriver à modéliser la fonction par un système d'équations en plusieurs variables à coefficients dans $\mathbb{F}_2$ pour pouvoir essayer des attaques algébriques.

4.2 Modélisation de la fonction par un système d'équations

Pour modéliser la fonction de compression de SHA-1, il faut au moins les 512 variables qui correspondent au bits du bloc du message, et les 160 qui vont constituer la sortie de la fonction. Si on cherche à exprimer directement la sortie en fonction de l'entrée, on obtient très vite au bout de quelques tours des équations de très haut degré, impossibles aussi bien à manipuler qu'à stocker. On va donc avoir besoin d'introduire comme variables celles qui interviennent dans chacun des 80 tours de l'algorithme de compression.

De la façon dont la variable de chaînage $(a_i, b_i, c_i, d_i, e_i)$ est mise à jour dans l'algorithme, on voit qu'on peut ne garder que 32 variables (sur 160) pour notre modélisation en remarquant que

$$\begin{aligned} b_i &= a_{i-1} \\ c_i &= (b_{i-1} \ll 30) = (a_{i-2} \ll 30) \\ d_i &= c_{i-1} = (a_{i-3} \ll 30) \\ e_i &= d_{i-1} = (a_{i-4} \ll 30 \end{aligned}$$

ces équations étant linéaires, on peut donc ne garder que les variables a_i dans notre modélisation sans monter de degré dans les équations. On utilisera donc pour les variables initiales (constantes) $(a_0, b_0, c_0, d_0, e_0)$ les notations $a_{-4}, \ldots, a_0$.

En ce qui concerne les fonctions booléennes, leur modélisation se fait simplement en donnant l'équivalence des opérations booléennes dans $\mathbb{F}_2$. Soit $x, y \in \mathbb{F}_2$ deux variables auxquelles sont associées les variables booléennes x' et y' alors on peut faire les associations suivantes :

$$x' \wedge y' \Leftrightarrow xy$$

$$x' \vee y' \Leftrightarrow x + y + xy$$
$$x' \oplus y' \Leftrightarrow x + y$$

Le principal problème lorsqu'on essaye de modéliser ce genre de fonction est l'addition modulaire (dans notre cas modulo 2^{32} vu qu'il s'agit d'additions sur des entiers machine de 32 bits). Ce problème a déjà fait l'objet de travaux dans [13] où l'on donne une description algébrique, du problème.

$$z = x + y \mod 2^n$$

$$\begin{cases} z_0 &= x_0 + y_0 \\ z_1 &= x_1 + y_1 + c_1 \\ z_2 &= x_2 + y_2 + c_2 \\ &\vdots \\ z_k &= x_k + y_k + c_k \\ &\vdots \\ z_{n-1} &= x_{n-1} + y_{n-1} + c_{n-1} \end{cases} \qquad \begin{cases} c_1 &= x_0 y_0 \\ c_2 &= x_1 y_1 + (x_1 + y_1)c_1 \\ &\vdots \\ c_k &= x_{k-1} y_{k-1} + (x_{k-1} + y_{k-1})c_{k-1} \\ &\vdots \\ c_{n-1} &= x_{n-2} y_{n-2} + (x_{n-1} + y_{n-1})c_{n-2} \end{cases}$$

Sur chaque ligne, c_i représente la retenue qui vient de l'addition des 2 bits x_{i-1} et y_{i-1}. On n'a pas besoin de garder les c_i dans notre modélisation car en remarquant que $c_i = z_i - x_i + y_i$ pour $i > 1$, on peut se ramener au système suivant en remplaçant les c_i :

$$\begin{cases} z_0 &= x_0 + y_0 \\ z_1 &= x_1 + y_1 + x_0 y_0 \\ z_2 &= x_2 + y_2 + x_1 y_1 + (x_1 + y_1)(z_1 + x_1 + y_1) \\ &\vdots \\ z_k &= x_k + y_k + x_{k-1} y_{k-1} + (x_{k-1} + y_{k-1})(z_{k-1} + x_{k-1} + z_{k-1}) \\ &\vdots \\ z_{n-1} &= x_{n-1} + y_{n-1} + x_{n-2} y_{n-2} + (x_{n-1} + y_{n-1})(z_{n-2} + x_{n-2} + z_{n-2}) \end{cases}$$

Dans l'algorithme de SHA-1, l'addition modulaire intervient dans

$$a_i = (a_{i-1} \lll 5) + f_i(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + m_{i-1} + k_i$$

On a 4 additions à effectuer sur des variables, et pour garder des polynômes de bas degré (≤ 3), on a du décomposer ces additions en ajoutant 3×32 nouvelles variables par tour. Ce sont les variables induites par les équations suivantes

$$\begin{aligned} v_{i,1} &= (a_{i-1} \lll 5) + f_i(b_{i-1}, c_{i-1}, d_{i-1}) \\ v_{i,2} &= v_{i,1} + e_{i-1} \\ v_{i,3} &= v_{i,2} + m_{i-1} \\ a_i &= v_{i,3} + k_i \end{aligned}$$

La modélisation du système en entier par des polynômes de degré inférieur ou égal à 3 nécessite 10752 variables. En effet, on a besoin des 512 bits du (bloc de) message $(x_{0,0}, \ldots, x_{0,31}, \ldots, x_{15,31})$, et de $4 \times 32 = 128$ bits par tour (pour le i-ème tour : la variable de

chaînage $(a_{i,0}, \ldots, a_{i,31})$ + les 3 variables intermédiaires $(v_{i,1,0}, \ldots, v_{i,1,31}, \ldots, v_{i,3,31})$) ce qui fait bien $512 + 80 \times 128 = 10752$. Nous pouvons bien sûr générer un système sur moins de tours pour étudier son comportement plus facilement.

Soit m le nombre de tours de SHA-1 sur lequel on travail.
Soit l'anneau de polynômes

$$\mathbb{F}_2[\mathbf{x}_0, \ldots, \mathbf{x}_{15}, \mathbf{a}_1, \ldots, \mathbf{a}_m, \mathbf{v}_{1,1}, \ldots, \mathbf{v}_{1,3}, \ldots, \mathbf{v}_{m,3}]$$

où $\mathbf{x}_i = (x_{i,0}, \ldots, x_{i,31}), i \in \{0, \ldots, 15\}$,
$\mathbf{a}_i = (a_{i,0}, \ldots, a_{i,31}), i \in \{1, \ldots, m\}$,
$\mathbf{v}_{i,j} = (v_{i,j,0}, \ldots, v_{i,j,0}), i \in \{1, \ldots, m\}, j \in \{1, \ldots, 3\}$.

Soit $\succ$ l'ordre lexicographique sur $M(n)$ tel que $x_{15,31} \succ \cdots \succ x_{15,0} \succ \ldots x_{0,0}$ et que quelque soit le tour i, on a

- $a_{i,31} \succ a_{i,30} \succ \cdots \succ a_{i,0}$
- $v_{i,k,31} \succ v_{i,k,30} \succ \cdots \succ v_{i,k,0}$ pour tout $k \in \{1, \ldots, 3\}$
- pour tout k et $k' \in \{1, \ldots, 3\}$, si $k > k'$ alors $v_{i,k,0} \succ v_{i,k',31}$
- $a_{i,0} \succ v_{i,3,31}$
- $v_{i,1,0} \succ x_{15,31}$

et, pour tout i et $i' \in \{1, \ldots, m\}$, si $i > i'$ alors $v_{i,1,0} \succ a_{i',31}$

On construit le système en mettant une par une les équations de l'algorithme

$$\begin{aligned} v_{i,1} + (a_{i-1} \ll 5) + f_i(b_{i-1}, c_{i-1}, d_{i-1}) \\ v_{i,2} + v_{i,1} + e_{i-1} \\ v_{i,3} + v_{i,2} + m_{i-1} \\ a_i + v_{i,3} + k_i \end{aligned}$$

en remplaçant $b_{i-1}, c_{i-1}, d_{i-1}, e_{i-1}$ par les bonnes équations qui dépendent des a_i définies plus haut, et avec à chaque fois $m_i = (m_{i-3} + m_{i-8} + m_{i-14} + m_{i-16}) \ll 1$ pour $i \geq 16$ où on a substitué au fur et à mesure. Comme cette équation est linéaire, là non plus, on ne fait pas monter le degré. Pour n'avoir que des solutions dans le corps des coefficients $\mathbb{F}_2$, il ne reste plus qu'à ajouter les équations de corps.

On remarque que par construction du système, chaque variable ne dépend que de ses prédécesseurs pour l'ordre lexicographique choisi. Le système obtenu est donc directement une base de Gröbner pour cet ordre, et on pourrait facilement en obtenir la variété si elle ne contenait pas 2^{512} éléments !

4.3 Attaques sur SHA-1

4.3.1 Attaque directe

Nous avons tout d'abord essayé d'attaquer directement le problème en essayant de trouver la variété de l'idéal engendré par le système dans lequel nous avons fixé les variables qui correspondent au hash pour obtenir une attaque en préimage. Le nombre de variable est beaucoup trop important pour avoir une attaque en pratique, même si les polynômes ne dépassent pas le degré 3. Nous avons donc abandonné l'idée et nous avons étudié les attaques en collisions basées sur la cryptanalyse différentielle.

4.3.2 Attaque différentielle

Les attaques différentielles sur les fonctions de hachage cryptographiques sont sans aucun doute celles qui ont donné les meilleurs résultats, avec en tête les travaux de Wang et al [26, 29, 25] qui ont permis de trouver des collisions sur MD5 et SHA-0 ainsi que d'autres fonctions basées sur le même principe. Ils ont aussi une attaque en collision sur SHA-1 présentée à la conférence CRYPTO 2005 [28, 27] qui a une complexité estimée à 2^{63} calculs d'empreintes.

Le principe de l'attaque différentielle a été introduit par Biham et Shamir [9] pour analyser la sécurité du DES. Ils définissent la cryptanalyse différentielle comme étant l'analyse des effets que peuvent produire une certaine différence dans un couple de messages clairs sur les différences entres les chiffrés. On voit très bien qu'on peut adapter cette analyse sur une fonction de hachage telle que SHA-1. L'attaque que propose Wang peut se décomposer de la manière suivante.

1. choisir un "bon" vecteur Γ de 80×32 bits qui correspondra à la différence entre les 2 messages étendus de la collision.
2. choisir un chemin différentiel qui correspondra aux différences qu'auront les variables de chaînage au fil des 80 tours.
3. trouver un ensemble de conditions suffisantes sur m qui assurent avec une bonne probabilité que le le couple de message $(m, m + \Gamma)$ suive le chemin différentiel (donc qui mène à une collision)
4. choisir aléatoirement un message m et le "modifier" jusqu'à ce qu'il satisfasse les conditions suffisantes.

Nous ne nous sommes pas intéressé à la recherche de bonnes différentielles (nous renvoyons le lecteur curieux à [11]), mais plutôt à la manière dont on pourrait combiner une attaque différentielle avec une modélisation algébrique du problème, notamment pour l'étape 4. C'est ce que nous présentons dans la sous-section suivante.

4.3.3 Attaque algébrique

Nous avons étudié l'article de Sugita [24] où les auteurs proposent une méthode pour trouver des collisions sur SHA-1. Le principe est basé sur l'attaque différentielle telle que proposée par Wang légèrement modifiée appliquée à une version de SHA-1 réduite à 58 tours. La principale différence est que la modification de message ne s'effectue pas sur les bits du message mais celles des variables de chaînage. (on peut facilement voir que dans SHA-1 les 512 bits du message d'origine sont en bijection avec les 512 bits des 16 premiers a_i). Le principe est de voir les "conditions suffisantes" de l'attaque de Wang d'un point de vue algébrique, et de transformer ces conditions en polynômes dans $\mathbb{F}_2[\mathbf{x}]$. Malheureusement, les conditions suffisantes sont exprimées comme fonctions booléennes sur les variables du message (dans notre modélisation les x_i), et pour la modification de message, on a besoin de connaître la fonction f telle que $\mathbf{x}_i = f(\mathbf{a}_0, \ldots, \mathbf{a}_{15})$ qui n'est pas triviale, et correspond à des polynômes de très haut degré.

Nous avons donc ajouté à ces conditions suffisantes les équations de notre modélisation de SHA-1 qui modélisent à fortiori les relations qui existent entre les $\mathbf{x}_i$ et les $\mathbf{a}_i$. Le problème reste le même que pour l'attaque en préimage et si on arrive à appliquer cette méthode sur des systèmes jouets (messages de 4×4 bits sur 6 tours), avec les paramètres de SHA-1, même

pour un nombre de tour restreint, le calcul de base de Gröbner nécessite un trop grand espace en mémoire pour pouvoir être mené à bout.

4.4 Conclusion sur l'analyse algébrique de SHA-1

On a vu qu'il est très difficile de faire une bonne approche algébrique sur une fonction comme SHA-1 de par le fait d'opérations non linéaires telles que l'addition modulaire qui impliqueraient des équations de trop haut degré. Pour éviter cela, le parti a été pris d'ajouter des variables pour avoir des polynômes au plus cubique. Cette modélisation a servi à implémenter une attaque algébrique proposée par Sugita et al dans [24], mais elle s'est avéré infructueuse. Cependant, on ne peut pas dire qu'une approche algébrique est impossible dans ce cadre. D'autres attaques pourront peut-être tirer parti d'une modélisation algébrique du problème.

5 Étude d'une fonction de hachage basée sur des polynômes multivariés

Une partie de ce chapitre a fait l'objet d'un article (co-écrit avec Jean-Charles Faugère et Ludovic Perret pour la partie "bases de Gröbner") soumis à la conférence internationale Inscrypt 2008, et nous renvoyons le lecteur à [7].

5.1 Principe de la fonction de hachage

Nous reprendrons ici les constructions données par Ding et Yang dans [14].

On appelle "fonction de hachage multivariée" une fonction explicitement décrite par un ensemble de polynômes en plusieurs variables. En réalité, c'est la fonction de compression qui sera ainsi décrite, et cette fonction sera placée dans une construction de Merkle-Damgård.

$$F : \mathbb{K}^{m+n} \to \mathbb{K}^m$$

$$f_1(y_1, \ldots, y_m, x_1, \ldots, x_n)$$

$$\vdots$$

$$f_m(y_1, \ldots, y_m, x_1, \ldots, x_n)$$

On utilise les variables de chaînage $a_i \in \mathbb{K}^m, i \in \{0, \ldots, k\}$ et on calcule

$$a_{i+1} = F(a_i, v_i)$$

où $v_i \in \mathbb{K}^n, i \in \{0 \ldots k-1\}$ sont les k blocs d'un message. a_0 est la Valeur Initiale (IV), et l'empreinte sera a_k.

Le problème de résoudre un système polynomial est un problème NP-complet, et on se base sur la difficulté de se problème pour justifier la sécurité en préimage de la construction. Cela dit, on ne peut rien dire sur la résistance de la fonction aux collisions. La sécurité de la fonction de hachage dépend de celle de la fonction de compression F, c'est donc le système polynomial qui décrit cette fonction qui sera étudié par la suite.

5.2 Attaques Algébriques sur les fonctions de hachage multivariées

Sur des fonctions de hachage multivariées, il est naturel d'utiliser des attaques algébrique. Les attaques en préimage et seconde préimage consistent naturellement en la résolution du système polynomial multivarié qui décrit la fonction en les valeurs du hash.

Soit $z = (z_1, \ldots, z_n) \in \mathbb{K}^n$ le hash dont on cherche une préimage. Pour le cas de la seconde préimage, on peut poser $h(x) = z$, et on obtient le même problème, à savoir résoudre le système suivant :

$$\begin{aligned} f_1(y_1, \ldots, y_m, x_1, \ldots, x_n) &= z_1 \\ &\vdots \\ f_m(y_1, \ldots, y_m, x_1, \ldots, x_n) &= z_m \end{aligned}$$

Le système obtenu est sous-déterminé (plus d'équations que de
variables). Pour ne pas avoir un nombre de solutions trop important, on doit fixer des variables, on fixera donc les valeurs de $y_i, i \in \{1, \ldots, m\}$ à celles de la variable d'initialisation.

Dans une attaque en collision, le but est de trouver une paire de messages (x, x') tel que $h(x) = h(x')$. Étant donné une différence δ entre deux messages x et x' (et éventuellement les variables de chaînage), on peut trouver des collisions en résolvant le système suivant :

$$\begin{aligned} f_1(y_1, \ldots, y_m, x_1 - \delta_1, \ldots, x_n - \delta_n) - f_1(y_1, \ldots, y_m, x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(y_1, \ldots, y_m, x_1 - \delta_1, \ldots, x_n - \delta_n) - f_m(y_1, \ldots, y_m, x_1, \ldots, x_n) &= 0 \end{aligned}$$

Il s'agit en fait de calculer la dérivée discrète des f_i en un certain point
(ici $(0, \ldots, 0, \delta_1, \ldots, \delta_n)$), puis de résoudre le système obtenu qui sera de degré $d - 1$ où $d = max(degre(f_i), i \in \{1, \ldots, m\})$ (les monômes de plus haut degrés vont disparaître). Ici aussi on aura besoin de fixer les valeurs des y_i. On peut signaler que si on choisit une différence aussi sur les valeurs des y_i, on peut essayer de chercher une "freestart" collision.

5.2.1 Construction cubique dense

Les auteurs de [14] proposent de construire la fonction en tirant aléatoirement n polynômes *cubiques* en $2n$ variables ($m = n$) à coefficients dans $\mathbb{F}_q$ avec, au choix, un des jeux de paramètres suivants :
Pour une empreinte de 160 bits (SHA-1)
$q = 2^8, n = 20$
$q = 2^4, n = 40$
Pour une empreinte de 256 bits (SHA-256)
$q = 2^{16}, n = 16$
$q = 2^8, n = 32$
$q = 2^4, n = 64$

Les paramètres donnés sont suffisants pour résister à une attaque algébrique directe en préimage, nous nous sommes donc concentrés sur l'attaque en collision. Le but est de trouver

$n-k$	d_{reg} (théorique)	d_{reg} (observé)
15	9	9
14	7	7
13	6	6
12	5	5
11	5	5

TAB. 2 – Comparaison avec les résultats théoriques pour $n = 16$

$n-k$	d_{reg} (théorique)	d_{reg} (observé)
17	8	8
16	7	7
15	6	6
14	6	6
13	5	5

TAB. 3 – Comparaison avec les résultats théoriques pour $n = 20$

un message $\mathbf{m} = (m_1, \ldots, m_n)$ tel que $\mathbf{m}$ et $\mathbf{m}+(\delta_1, \ldots, \delta_n)$ soient en collision. Nous calculons donc le système suivant

$$\begin{aligned} f_1(y_1, \ldots, y_n, x_1 + \delta_1, \ldots, x_n + \delta_n) - f_1(y_1, \ldots, y_n, x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(y_1, \ldots, y_n, x_1 + \delta_1, \ldots, x_n + \delta_n) - f_n(y_1, \ldots, y_n, x_1, \ldots, x_n) &= 0 \end{aligned}$$

où $(\delta_1, \ldots, \delta_n)$ est une différence choisie aléatoirement. En fixant les valeurs des y_i aux valeurs initiales, les système obtenu devient un système de degré 2 avec n équations en n variables.

$$f'_1(x_1, \ldots, x_n), \ldots, f'_n(x_1, \ldots, x_n)$$

qui peut être traité comme un système quadratique aléatoire dans la mesure où les polynômes cubiques d'origine, de même que la valeur des δ_i ont été choisis aléatoirement.

Sur les paramètre proposés, les machines actuelles ne permettent pas de pouvoir calculer la variété de l'idéal engendré car les polynômes manipulés durant le calcul de base de Gröbner par l'algorithme F4 atteignent un degré trop élevé (degré de régularité). Néanmoins, comme nous l'avons vu dans la Sect. 3, des travaux effectués dans le domaine [3, 5, 6] donnent une estimation précise du degré de régularité dans le cas de systèmes semi-réguliers. En se basant sur une approche hybride entre le calcul de bases de Gröbner et la recherche exhaustive que nous avions déjà testée sur un schéma de signature dans [8], nous avons fixé k variables supplémentaires de manière à obtenir des systèmes surdéterminés pour faire chuter le degré de régularité. Lorsqu'on spécialise k variables, on a besoin de faire en plus une recherche exhaustive sur ces variables, c'est à dire, calculer non plus une, mais $(\#\mathbb{K})^k$ bases de Gröbner. La différence avec [8] où on était sûr de trouver une solution, ici on n'obtiendra pas forcément une collision si on choisit une mauvaise différence δ.

Les observations expérimentales reportées dans les tableaux sur les degrés atteints durant les calculs de bases de Gröbner suggèrent que nous avons affaire à des systèmes semi-réguliers, ce qui va dans le sens de la conjecture qu'on peut trouver dans [4] :

Conjecture 1. *Presque tout système polynomial surdéterminé est une suite semi-régulière.*

Grâce à cette approche hybride, nous pouvons monter une attaque pratique avec une complexité bornée par (voir Sect. 3) :

$$\mathcal{O}\left((\#\mathbb{K})^k\left((n-k)\cdot \mathrm{C}_{n+d_{\mathrm{reg}}-1}^{d_{\mathrm{reg}}}\right)^{\omega}\right)$$

où ω est la constante d'algèbre linéaire, $2 \leq \omega \leq 3$.

L'avantage de cette approche est qu'elle est totalement parallélisable, et on peut stopper la recherche dès qu'une collision est trouvée. Pour les paramètres $n = 16$,$\mathbb{K} = \mathbb{F}_{2^{16}}$, on peut trouver des collisions en moins de 38 heures, en supposant l'accès à $2^{16} = 65536$ processeurs, ce qui reste raisonnable.

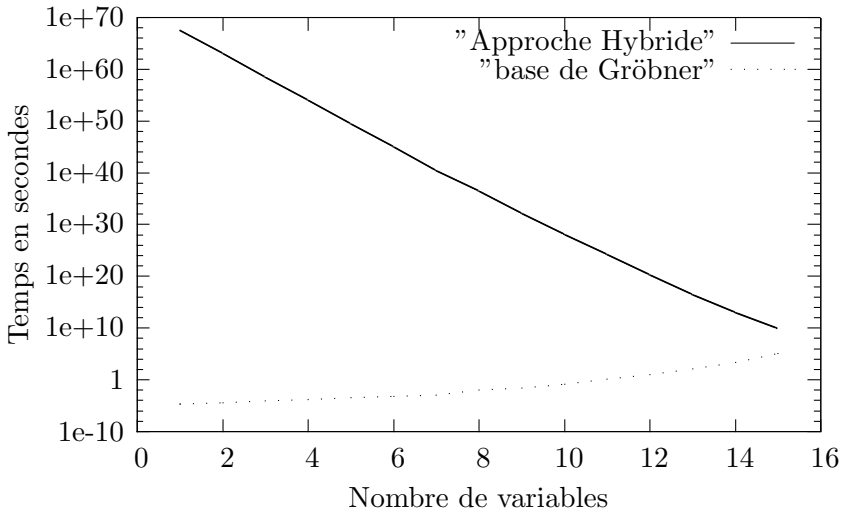

FIG. 2 – Approche Hybride : $n = 16$, $\mathbb{K} = \mathbb{F}_{2^{16}}$

La figure montre qu'on ne peut pas se permettre de fixer trop de variables sous peine de faire exploser le nombre de calculs à effectuer, mais si le corps des coefficients n'est pas trop grand, on peut fixer plus de variables. La principale barrière lors du calcul de bases de Gröbner c'est la mémoire nécessaire durant le calcul. Lorsqu'on a affaire à des systèmes semi-réguliers, on voit qu'on peut rendre l'attaque possible en calculant plus de bases de Gröbner qui montent moins en degré.

5.2.2 Construction cubique creuse

Pour accélérer le temps de calcul d'une empreinte, les auteurs de [14] ont proposé une deuxième construction basée sur des polynômes **creux** tirés aléatoirement. Pour cela, on spécifie un paramètre supplémentaire, il s'agit de ϵ, le pourcentage de monômes dans chaque polynômes du système. Ils proposent donc les jeux de paramètres suivants :
Pour une empreinte de 160 bits (SHA-1)

$q = 2^8, n = 20, \epsilon = 0.2\%$
$q = 2^4, n = 40, \epsilon = 0.1\%$

Pour une empreinte de 256 bits (SHA-256)

$q = 2^{16}, n = 16, \epsilon = 0.2\%$
$q = 2^8, n = 32\epsilon = 0.1\%$
$q = 2^4, n = 64\epsilon = 0.1\%$

Si en préimage, le problème est toujours difficile, en collision, c'est différent. Les auteurs de [2] on d'ailleurs mis en évidence une collision triviale qui arrive quand une variable x_k n'apparaît dans aucun polynôme (ce qui peut arriver au vu de les valeur des ϵ). Alors n'importe quel couple de message ayant cette différence sera en collision.

Nous avons essayé notre attaque en collision pour la construction creuse en calculant le système

$$\begin{aligned} f_1(y_1, \ldots, y_n, x_1 + \delta_1, \ldots, x_n + \delta_n) - f_1(y_1, \ldots, y_n, x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(y_1, \ldots, y_n, x_1 + \delta_1, \ldots, x_n + \delta_n) - f_n(y_1, \ldots, y_n, x_1, \ldots, x_n) &= 0 \end{aligned}$$

comme pour la construction dense, $(\delta_1, \ldots, \delta_n)$ est une différence choisie aléatoirement, et on fixe les valeurs des y_i aux valeurs initiales pour obtenir un système de degré 2 avec n équations en n variables.

Ici, comme les polynômes sont très creux, plus le vecteur de différence δ est de poids faible, et plus le système à résoudre sera creux, donc plus le calcul de base de Gröbner est rapide. En revanche, en contraignant δ a avoir un poids faible, on diminue l'espace dans lequel on recherche des collisions, et on a moins de chance d'en trouver. Il faut donc choisir un δ avec le plus grand poids possible mais qui nous permette de calculer la base de Gröbner.
Expérimentalement, voici les meilleur compromis que l'on obtient :

paramètres	poids de δ
$q = 2^8, n = 20, \epsilon = 0.2\%$	4
$q = 2^{16}, n = 16, \epsilon = 0.2\%$	5
$q = 2^8, n = 32, \epsilon = 0.1\%$	2

TAB. 4 – Poids de δ en fonction des paramètres

Pour toutes les fonctions de hachage construites avec ces paramètres, nous avons été en mesure de trouver au moins une collision grâce à cette technique :

1. choisir une différence δ de poids k aléatoirement
2. construire le système $F(\mathbf{y}, \mathbf{x} + \delta) - F(\mathbf{y}, \mathbf{x}) = 0$, et fixer les valeurs de $\mathbf{y}$ aux valeurs initiales.
3. calculer la variété de l'idéal engendré par F
4. si la variété est vide, retourner à l'étape 1

5.2.3 Composition de systèmes quadratiques

La construction d'une fonction de compression à base de polynômes tirés aléatoirement n'était pas entièrement satisfaisante. En effet, choisir des polynômes denses conduit à avoir des temps de calcul de hash plus longs et nécessite un espace de stockage plus important. D'un autre côté, choisir des polynômes creux réduisent considérablement la résistance aux collisions comme nous l'avons vu précédemment.

Pour contourner ces problèmes, les auteurs de [14] proposent de composer deux systèmes polynomiaux de degré 2 pour obtenir des polynômes quartiques (degré 4). Cette idée avait aussi été développée dans [22] dans un schéma que les auteurs nomment "MQ-HASH". La construction consiste d'abord à effectuer une expansion avec le premier système F, puis une compression avec le deuxième G.

$$F : \mathbb{K}^{n+m} \to \mathbb{K}^{n+m+k}, G : \mathbb{K}^{n+m+k} \to \mathbb{K}^{n}$$

La fonction de compression sera $H = G \circ F$
Les auteurs proposent d'utiliser $n = m = k$. si on note $F = (f_1, \ldots, f_{3n})$ et $G = (g_1, \ldots, g_n)$
On obtient donc les systèmes suivants :

$$\begin{cases} f_1(x_1, \ldots, x_{2n}) \\ \vdots \\ f_{3n}(x_1, \ldots, x_{2n}) \end{cases} \quad \begin{cases} g_1(x_1, \ldots, x_{3n}) \\ \vdots \\ g_n(x_1, \ldots, x_{3n}) \end{cases}$$

Pour une attaque en préimage, soit $z = (z_1, \ldots, z_n)$ l'empreinte dont on cherche une préimage. La composition de ces 2 systèmes nous donnerait à résoudre

$$\begin{aligned} z_1 + g_1(f_1(x_1, \ldots, x_{2n}), \ldots, f_{3n}(x_1, \ldots, x_{2n})) = 0 \\ \vdots \\ z_n + g_n(f_1(x_1, \ldots, x_{2n}), \ldots, f_{3n}(x_1, \ldots, x_{2n})) = 0 \end{aligned}$$

Pour éviter d'avoir un système trop lourd avec des polynômes de degré 4, on introduit $3n$ nouvelles variables $v_1, \ldots, v_{3n}$, et on récrit le système de la manière suivante :

$$\begin{aligned} v1 + f_1(x_1, \ldots, x_{2n}) = 0 \\ \vdots \\ v_{3n} + f_{3n}(x_1, \ldots, x_{2n}) = 0 \\ z_1 + g_1(v_1, \ldots, v_{3n}) = 0 \\ \vdots \\ z_n + g_n(v_1, \ldots, v_{3n}) = 0 \end{aligned}$$

Avec l'approche utilisée précédemment, nous n'avons pas été en mesure de trouver de collision pour les fonctions issus de cette construction qui semble être résistante aux collisions.

5.3 Conclusion sur les fonctions de hachage multivariées

Les fonctions de hachage multivariées ont été introduite avec la volonté d'avoir une sécurité prouvée, que ce soit en collision ou en préimage, c'est pour cela qu'on se base sur la résolution

de système polynomiaux qui est un problème NP-complet. Une fois le principe de ces fonctions de hachages défini, on a vu qu'en prenant des polynômes aléatoires, même si on ne remet pas en cause la résistance à l'attaque en préimage, il s'avère que la fonction est sensible aux collisions, et nous avons pu exhiber une attaque avec une complexité inférieure à la recherche exhaustive. Cette attaque ne s'appliquait que dans le cas de polynômes aléatoire à cause de la faiblesse de la construction qui laissait au hasard la résistance aux collisions.

La construction plus élaborée qui consiste à expanser d'abord le message dans un espace plus grand avant d'utiliser une fonction de compression semble plus résistante et à notre connaissance, ne souffre d'aucune attaque en collision.

Nous tenons à signaler qu'une attaque générique sur les fonctions de hachage multivariées a été présentée tout récemment dans [18]. Cet article montre qu'il est possible de monter une attaque en préimage avec une complexité un petit peu meilleure que la recherche exhaustive en montrant que pour un systèmes de polynômes F de degré d, le calcul de $2^d - 1$ empreintes ainsi que la connaissance de la dérivée d-ième de F (qui est donc une constante) permet de calculer une autre empreinte. De ce fait, si $\mathbb{K}$ est le corps des coefficients des polynômes, la probabilité d'avoir une collision en faisant $2^d - 1$ appels à la fonction est de $\frac{2^d}{\#\mathbb{K}}$ au lieu des $\frac{2^d-1}{\#\mathbb{K}}$ qu'on devrait avoir.

6 Conclusion

Dans le cadre de ce stage, nous avons pu appliquer les principes de la cryptanalyse algébrique sur des représentants de 2 familles de fonction de hachage. En premier lieu, les fonctions de la famille MD utilisées depuis longtemps en pratique et qui ne sont pas basés sur des concepts algébriques. Nous avons proposé une modélisation algébrique de la fonction SHA-1 [21], modélisation que nous avons utilisé pour adapter l'attaque proposée par Sugita [24] et basée sur les travaux de Wang [28] pour chercher des collisions sur SHA-1 réduite à 58 tours.

L'autre famille de fonctions de hachage étudiée est beaucoup plus récente, et est basée sur l'évaluation de systèmes polynomiaux [14, 2]. Là, l'attaque algébrique s'imposait, et nous avons pu donner une borne de complexité pour l'attaque en collision et nous avons été en mesure de trouver effectivement des collisions pour certains paramètres proposés par Ding et Yang [14]. Nous avons aussi étudié une construction qui nous a semblé bien plus robuste et qui pourrait être prometteuse pour l'avenir des fonctions de hachage.

La sécurité informatique est un domaine sensible, et il est normal qu'on attende des preuves fortes sur la sûreté d'un schéma de chiffrement. C'est la raison pour laquelle l'étude et l'analyse de ces schémas sont importants.

Références

[1] W.W. Adams and P. Loustaunau. *An Introduction to Gröbner Bases*, volume 3 of *Graduate Studies in Mahematics*. AMS, 1994.

[2] Jean-Philippe Aumasson and Willi Meier. Analysis of multivariate hash functions. In *Information Security and Cryptology - ICISC 2007*, volume 4817 of *LNCS*, pages 309–323. Springer, 2007. isbn : 978-3-540-76787-9.

[3] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Université de Paris VI, 2004.

[4] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. Complexity study of Gröbner basis computation. Technical report, INRIA, 2002. http://www.inria.fr/rrrt/rr-5049.html.

[5] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proc. International Conference on Polynomial System Solving (ICPSS)*, pages 71–75, 2004.

[6] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Bo-Yin Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *Proc. of MEGA 2005, Eighth International Symposium on Effective Methods in Algebraic Geometry*, 2005.

[7] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Cryptanalysis of multivariate polynomials for hashing. submitted to INSCRYPT 2008, 2008.

[8] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Cryptanalysis of the TRMS signature scheme of PKC'05. In *Progress in Cryptology – AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 143–155. Springer, 2008.

[9] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1) :3–72, January 1991.

[10] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[11] Christophe De Cannière and Christian Rechberger. Finding SHA-1 characteristics : General results and applications. In *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.

[12] David A. Cox, John B. Little, and Don O'Shea. *Ideals, Varieties and Algorithms*. Springer, 2005.

[13] Blandine Debraize and Nicolas Courtois. Algebraic description and simultaneous linear approximations of addition modulo 2^n.

[14] Jintai Ding and Bo-Yin Yang. Multivariates polynomials for hashing. Cryptology ePrint Archive, Report 2007/137, 2007.

[15] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139 :61–88, June 1999.

[16] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In T. Mora, editor, *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation ISSAC*, pages 75–83. ACM Press, July 2002. isbn : 1-58113-484-3.

[17] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 44–60. Springer, 2003.

[18] Yiyuan Luo and Xuejia Lai. Higher order differential cryptanalysis of multivariate hash functions. Cryptology ePrint Archive, Report 2008/350, 2008. http://eprint.iacr.org/.

[19] MAGMA. High performance software for algebra, nuber theory, and geometry — a large commercial software package. http ://magma.maths.usyd.edu.au.

[20] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In *Advances in Cryptology – EUROCRYPT 1988*, volume 330 of *LNCS*, pages 419–453. Springer–Verlag, 1988.

[21] National Institute of Standards and Technology (NIST). Federal Information Processing Standards Publication (FIPS) 180-1, 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[22] Matthew J. B. Robshaw Olivier Billet and Thomas Peyrin. On building hash functions from multivariate quadratic equations. In *ACISP*, volume 4586 of *LNCS*, pages 82–95. Springer, 2007.

[23] Jacques Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP) : two new families of asymmetric algorithms. In *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *LNCS*, pages 33–48. Springer–Verlag, 1996.

[24] Makoto Sugita, Mitsuru Kawazoe, Ludovic Perret, and Hideki Imai. Algebraic cryptanalysis of 58-round SHA-1. In Alex Biryukov, editor, *FSE*, volume 4593 of *LNCS*, pages 349–365. Springer, 2007.

[25] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004.

[26] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.

[27] Xiaoyun Wang, Andrew Yao, and Frances Yao. New collision search for SHA-1. CRPTYO 2005 Rump Session, 2005.

[28] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

[29] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 1–16. Springer, 2005.