Cryptanalyse algébrique : Optimisation des bases de Gröbner sur AES

Martin Grenouilloux le 7 juillet 2021

Table des matières

1 Introduction

Advanced Encryption Standard (AES) est la méthode de chiffrement symétrique la plus utilisée aujourd'hui et aucune attaque connue à ce jour n'est assez efficace pour mettre à mal ce cryptosystème. Une approche intéressante consiste à calculer la base de Gröbner du système de polynômes associé au processus de chiffrement. Une fois calculée, le résultat obtenu est correct et permet de retrouver le clair et la clef de chiffrement. Le problème avec cette méthode est qu'elle est beaucoup trop longue à calculer sur des cas pratiques, ce qui la rend inutilisable.

Le but de ce projet introspectif de la cryptanalyse algébrique sur AES est de chercher et trouver des pistes d'amélioration sur le calcul des bases de Gröbner pour en réduire la complexité.

Dans les trois premières parties sont présentés les fondamentaux algébriques autour des bases de Gröbner ainsi que les méthodes actuellement utilisées comme la génération par Buchberger et son algorithme. Dans la quatrième partie sont présentées les pistes et avancées sur le sujet d'optimisation qu'elles soient intéressantes ou non.

2 Bases de Gröbner

Une base de Gröbner regroupe un ensemble de polynômes g pour un idéal I si et seulement si $I = \langle g \rangle$ et que les termes principaux de g génèrent l'idéal des termes généraux de I, défini par la formule suivante :

$$\langle LT(g) \rangle = \langle LT(I) \rangle$$

Une base de Gröbner est dite réduite si le coefficient principal de chaque polynôme de g vaut 1. Les bases de Gröbner sont particulièrement efficaces lorsqu'il s'agit de résoudre (ou réduire) un système d'équations polynomiales.

2.1 Ordre monomial

Classer des polynômes à plusieurs inconnues ne peut pas exclusivement se faire en étudiant leurs termes principaux car plus de facteurs sont à considérer et c'est pourquoi plusieurs "classements" des monômes des polynômes sont proposés, chacun présentant des spécificités propres.

On dit d'un ordre de monômes que c'est une relation représentée par $>$ sur $\mathbb{Z}^n_{\geq 0}$ sur l'ensemble des monômes x^α ayant pour principe :
— $>$ est un ordre total ou linéaire sur $\mathbb{Z}^n_{\geq 0}$
— $\forall \gamma \in \mathbb{Z}^n_{\geq 0}$, si $\alpha > \beta$, alors $\alpha + \gamma > \beta + \gamma$
— Tout sous-ensemble de $\mathbb{Z}^n_{\geq 0}$ a un plus petit élément dans l'ordre $>$
Voici quelques exemples d'ordres sur les monômes.

L'ordre lexicographique Abrégé *lex*, pour tout exposants α et β dans $\mathbb{Z}^n_{\geq 0}$, si $\alpha >_{lex} \beta$ alors, nous allons ordonner les monômes de telle sorte : $x^\alpha >_{lex} x^\beta$. C'est la manière la plus "naturelle" de lire un polynôme et l'on pourra toujours réordonner les monômes d'une base de Gröbner après réduction d'un système pour une meilleure lecture.

L'ordre lexicographique des degrés inverses abrégé *degrevlex*, pour chaque α et β dans $\mathbb{Z}^n_{\geq 0}$ tels que α et β sont des exposants, $\alpha >_{degrevlex} \beta$ si $\deg(\alpha) \geq \deg(\beta)$ et l'ordre des monômes va suivre le rythme $x^\alpha >_{degrevlex} x^\beta$

Parmi les ordres ci-dessus, le plus naturel serait très certainement l'ordre lexicographique, mais un tel arrangement des monômes n'entraînerait pas nécessairement la base de Gröbner la plus optimale pour un idéal I donné. Pour cela, nous définissons les propriétés suivantes sur les polynômes :

Soit f un polynôme non-nul défini par un ordre des monômes $>_m$ tel que $f = \sum_\alpha a_\alpha x^\alpha$

— Le multidegré de f est noté $md(f)$ et défini par :

$$md(f) = \max(\alpha \in \mathbb{Z}^n_{>_m 0},\ a_\alpha \neq 0)$$

— Le coefficient principal de f est noté $LC(f)$ et est défini par :

$$LC(f) = a_{md(f)} \in \mathbb{K}$$

(aussi appelé coefficient du terme du plus haut degré)

— Le monôme principal de f noté $LM(f)$ et est défini par

$$LM(f) = x^{md(f)}$$

soit le terme du plus haut degré

— Le terme principal de f est noté $LT(f)$ et est défini en fonction du coefficient principal et du monôme principal :

$$Lt(f) = LC(f) * LM(f)$$

— Le plus petit multiple commun de deux monômes $m_1 = x^a$ et $m_2 = x^b$ est appelé $LCM(m_1,\ m_2)$

Nous pouvons maintenant définir une base de Gröbner.

2.2 Les S-polynômes

Les S-polynômes, aussi appelés *Substraction-polynoms*, permettent d'éliminer les termes principaux de deux polynômes f_1 et f_2 par la formule suivante :

$$S(f_1,\ f_2) = \frac{LCM(LM(f_1),\ LM(f_2))}{LT(f_1)} * f_1 - \frac{LCM(LM(f_1),\ LM(f_2))}{LT(f_2)} * f_2$$

Et si f_1 et f_2 appartiennent au même idéal I, alors on peut affirmer que $S(f_1,\ f_2) \in I$. L'élimination des termes principaux nous mènent à une simplification des polynômes dans un système et cette propriété des S-polynômes est aussi utile dans la génération de bases de Gröbner par l'algorithme de Buchberger.

2.3 Génération par Buchberger

L'algorithme de Buchberger permet de construire une base de Gröbner d'un idéal I depuis un système F de n générateurs. L'idée principale derrière cet algorithme est l'utilisation de S-polynômes pour réduire la taille des éléments générateurs en simplifiant leur terme principal. En construisant $S(f_1, f_2)$ dans I, on obtient des termes principaux plus petits que ceux de f_1 et f_2 mais qui en sont des multiples.

Critères de Buchberger - Une accélération de l'algorithme est possible en déterminant si $S(f_1, f_2)$ se réduit à zéro :

— f, g deux polynômes tels que leurs monômes principaux n'ont pas d'inconnues en commun, alors leur S-polynôme est nul.

— Soit F un système de polynômes f_1, f_2, f_3.... Si $LT(f_1)|LCM(LT(F_2), LT(f_3)$ et que les S-polynômes de f_1 et f_2 et de f_1, f_3 modulo F sont nuls, alors $S(f_2, f_3$ est nul.

Une implémentation de cet algorithme est disponible dans *Sage* et une vue d'ensemble ressemble à ça [1] :

Listing 1 – Algorithme de Buchberger

```
from sage.rings.polynomials.toy_buchberger import spol

def buchberger(F):
    G = set(F)
    G2 = set()
    while G2 != G:
        G2 = copy(G)
        for f1, f2 in cartesian_product-iterator([G2, G2]):
            if f < g:
                s_polynom = spol(f, g).reduce(G2)
                if spolynom != 0:
                    G.add(s)
    return G
```

L'algorithme de Buchberger termine et est correct, peu importe le système en entrée.

3 Utilisations pour la cryptanalyse

3.1 Procédé d'orthogonalisation de Gram-Schmidt

Le procédé d'orthogonalisation de Gram-Schmidt calcule une base orthogonale à la base donnée en paramètre en conservant la dimension et dans le même espace vectoriel. Ainsi, de toute base $v_1,\ v_2,\ ...,\ v_n$ à n dimensions, il est possible de calculer une base $w_1,\ w_2,\ ...,\ w_n$ orthogonale dans l'EV de la première.

$$w_1 = v_1$$

$$w_2 = v_2 - \frac{\langle v_2,\ w_1\rangle}{\langle w_1,\ w_1\rangle} w_1$$

$$w_3 = v_3 - \frac{\langle v_3,\ w_1\rangle}{\langle w_1,\ w_1\rangle} w_1 - \frac{\langle v_3,\ w_2\rangle}{\langle w_2,\ w_2\rangle} w_2$$

$$\dots$$

1. D'après la thèse de Martin Albrecht

$$w_n = v_n - \frac{\langle v_n,\ w_1\rangle}{\langle w_1,\ w_1\rangle} w_1 - \cdots - \frac{\langle v_n,\ w_{n-1}\rangle}{\langle w_{n-1},\ w_{n-1}\rangle} w_{n-1}$$

La base orthogonale obtenue après le processus :
— est continue
— préserve l'orientation de la base originale

L'algorithme de LLL est basé sur ce principe d'orthogonalisation des bases.

4 Optimisation des bases de Gröbner

4.1 Quel ordre choisir pour la meilleure base de Gröbner

Choisir un ordre monomial ne devrait théoriquement pas influencer la base de Gröbner obtenue en sortie de Buchberger, c'est pourtant parfois le cas et aucun document n'explique pourquoi. Dans sa thèse, Martin Albrecht défend l'idée qu'un ordre *degrevlex* est de manière générale, plus efficace.

Exemple - Création de la base de Gröbner avec le même système de polynômes avec Sage mais avec deux ordres différents, d'abord en ordre lexicographique des degrés inverses :

```
sage: P.<x, y, z> = PolynomialRing(GF(127), order='degrevlex')
sage: f = -46*z^2 + 51*x - 2*z + 38
sage: g = -51*x*y - 47*y^2 + 49*x*z + 62*y*z + 45*z^2
sage: h = -5*x^2 - 21*y*z - 49*z^2 + 48*x - 15*y
sage: I = Ideal(f, g, h)
sage: I.groebner_basis()
[y^3 - 61*y^2*z - 55*y^2 - 29*x*z - 63*y*z + 56*x + 16*y + 38*z + 53, x^2 + 55*y*z
- 28*x + 3*y + 57*z + 60, x*y - 19*y^2 + 9*x*z - 56*y*z + 57*x - 62*z + 35, z^2
- 37*x - 11*z - 45]
```

Puis en ordre lexicographique :

```
sage: I.groebner_basis()
[x + 24*z^2 - 10*z + 63, y + 13*z^7 - 55*z^6 - 10*z^5 - 57*z^4 - 50*z^3 + 39*z^2
- 10*z + 41, z^8 + 23*z^7 - z^6 + 16*z^5 - 58*z^4 + 23*z^3 - 10*z^2 + 40*z - 55]
```

On se rend compte immédiatement que le deuxième résultat de l'ordre lexicographique est plus intéressant car la base de Gröbner obtenue contient moins de monômes. Pourtant, avec d'autres systèmes polynomiaux, le résultat est totalement différent.

5 Étude de la connectivité des graphes dans AES

5.1 Composantes connexes

Chaque chiffrement par AES peut être représenté sous la forme d'un système de polynômes dont la solution amène au message en clair. Chaque système de polynôme peut être vu sous la forme d'un graphe comme présenté ci-dessous.

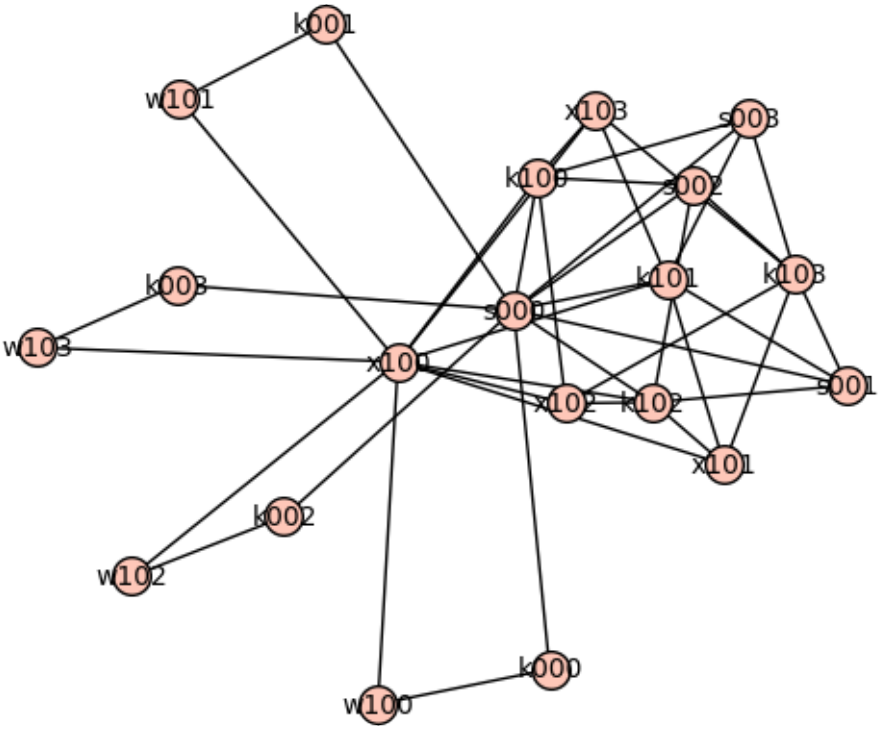

FIGURE 1 – Exemple de graphe pour AES-4 bits composé de 36 polynômes et 20 inconnues

Le graphe de la figure ci-dessus ne présente qu'une seule composante connexe mais il est théoriquement possible de le séparer en deux graphes indépendants ce qui engendrerait deux systèmes de polynômes à résoudre et une résolution de ces systèmes plus simples (car plus petits et avec potentiellement moins d'inconnues).

Quelque soit la taille de la clef lors de la génération, aucun graphe associé ne présente plusieurs composantes connexes. Après variation du nombre de tours sur AES, les graphes obtenus sont largement plus complexes et atteignent 7500 polynômes pour 4300 inconnues (AES-128). AES est donc assez solide pour ne pas présenter de faiblesses dans les graphes associés à ses systèmes de polynômes.

5.2 Création & étude de communautés dans les graphes

En étudiant moins naïvement les graphes et en se penchant sur les communautés présentes, on peut espérer détecter des communautés pouvant se détacher les unes des autres, toujours dans le but de séparer le graphe en deux.

5.2.1 Notions de communautés

La détection de communautés est une méthode très utilisée dans l'analyse de graphes. Soit un graphe $G = (V,\ E)$ avec V l'ensemble des sommets et E l'ensemble des arêtes de G, le principe de détection de communautés est d'identifier des ensembles, groupes de sommets densément connectés. On appelle les communautés identifiées d'un graphe une partition, et la qualité de celle-ci peut être mesurée avec sa modularité. La modularité d'une partition est une valeur comprise entre -1 et 1 qui mesure la densité des connexions entre les sommets d'une communauté. Une modularité optimale conduit donc au meilleur partitionnement du graphe possible et calculer une telle optimisation est long et coûteux, mais un algorithme y arrive très bien : la méthode de Louvain.

5.2.2 Algorithme de Louvain

La méthode de Louvain, proposé en 2008 par Vincent Blondel de l'Université de Louvain est basé sur un algorithme multi-itératif qui part d'un état initial de V pour un graphe $G = (V,\ E)$ et qui va itérativement améliorer la pertinence du choix des sommets pour former les communautés [2] au travers de leur modularité jusqu'à ce que le gain potentiel soit négligeable.

Cet algorithme possède principalement deux parties toutes deux itératives. Soit un graphe $G = (V,\ E)$ dont le nombre de sommets est $|V| = n$, chacun de ces sommets est d'abord assigné à une communauté : il y a donc autant de communautés de que sommets dans le graphe. Ensuite pour chaque voisin n_j de chaque sommet n_i, on évalue le gain de modularité lorsque n_i est retiré de sa communauté pour être placé dans celle de n_j. Ainsi, le sommet n_i est placé dans la communauté où le gain de modularité est le plus important seulement si le gain estimé maximum est positif non nul. Dans l'autre cas, n_i ne change pas de communauté. Ce processus est appliqué et répété jusqu'à qu'il n'y ait plus d'optimisation possible ou que celle-ci soit négligeable. Le seuil de négligeabilité, appelé τ, est arbitraire et défini à l'avance. La majeure partie de l'efficacité de cette méthode repose sur la facilité de calcul du gain de modularité ΔQ lors du mouvement d'un sommet d'une communauté dans une autre.

$$\Delta Q = \left[\frac{\sum_{in} + 2k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

avec $\sum_{in}$ la somme des poids dans la communauté, $\sum_{tot}$ la somme des poids des arêtes incidentes à la communauté, k_i la somme des poids des voisins du sommet i et m la somme des poids de toutes les arêtes du graphe. Comme les graphes que nous utilisons ne sont pas pondérés, chaque arête est annotéeRetirer tous les sommets de plus haut degré du graphe (jusqu'à une certaine limite) permet de réduire drastiquement la densité de celui-ci en peu de calculs. Même si les graphes

2. il est à noter que le nombre optimal de communautés dans un graphe n'est pas prédéfini et ne dépend pas du nombre de sommets

ont tendance à être extrêmement creux (une densité < 0.05 et qui diminue encore plus lorsque l'on monte dans les tours), ça permet de diminuer aussi le nombre de liens entre les sommets et par conséquent les relations entr

comme ayant un poids de 1, ce qui accélère encore plus les calculs.

La deuxième partie est simplement la construction d'un nouveau graphe dont les sommets sont les communautés trouvées dans la première partie. il suffit de réitérer la première partie sur le graphe obtenu et de recommencer jusqu'à ce que le seuil de modularité soit atteint (Le graphe va ainsi perdre des sommets à chaque fin de la deuxième partie et résultera *ad fine* en un graphe de quelques sommets seulement).

L'algorithme de Louvain se termine, quelque soit le graphe de départ.

5.2.3 Observations

En terme de communautés et de dépendances entre les sommets, les graphes générés sont très intéressants.

AES-4		
round	sommets	communautés
1	20	6
2	36	7
5	84	7
10	164	9

AES-32		
round	sommets	communautés
1	144	6
2	256	6
5	592	12
10	1152	24

AES-64		
round	sommets	communautés
1	272	10
2	480	7
5	1104	20
10	2144	28

AES-128		
round	sommets	communautés
1	544	28
2	960	12
5	2208	21
10	4288	42

La taille des graphes obtenus évolue linéairement avec le nombre de tours sur AES. Le nombre de communautés semble adopter le même comportement dans le sens ou l'évolution sur le nombre de tour suit la même progression pour les différentes variantes d'AES. On peut identifier des points communs forts entre les variantes, ainsi les graphes tendent à réagir sensiblement de la même manière et l'on peut espérer que si l'on peut trouver un graphe séparable pour AES-4, cela est aussi possible pour AES-64, AES-128...

Cepandant, on peut constater une aspérité dans les données concernant le nombre de communautés pour *round* qui vaut 1 ou 2. On peut expliquer cela par la densité du graphe qui est élevée dès les premiers tours et qui va par conséquent en déduire qu'il existe un nombre important de communautés.

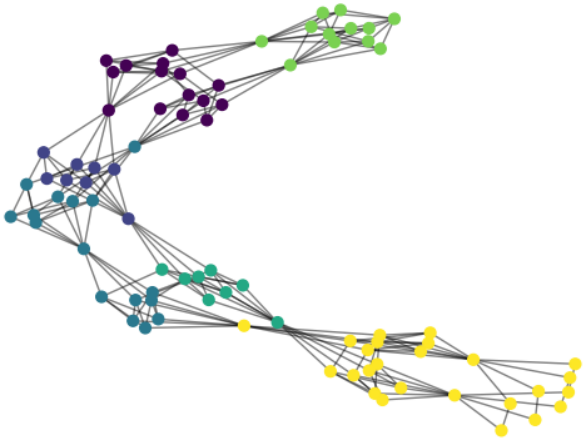

FIGURE 2 – Graphe pour AES-4, 5 tours

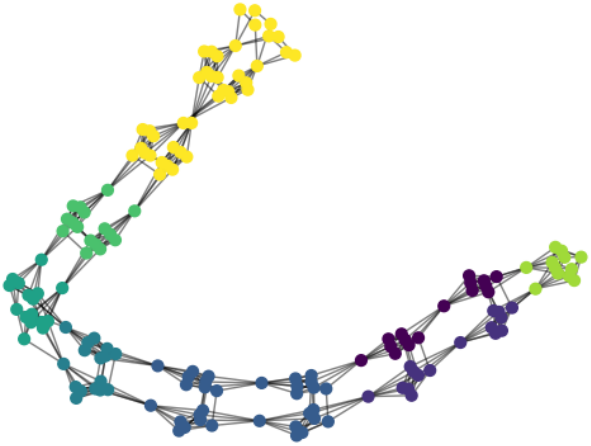

FIGURE 3 – Graphe pour AES-4, 10 tours

On peut constater ici qu'augmenter le nombre de tours ne change pas le comportement intrinsèque des communautés ni même du graphe mais va simplement "rajouter" des sommets tout en préservant sa structure. le graphe grossit énormément avec les tours mais ne rentre pas dans une spirale chaotique ce qui aurait pu rendre encore plus compliqué la tache de le séparer. On estime ceci être une imperfection (le mot faille serait trop puissant ici) de AES puisque monter dans les tours ne complexifie pas les graphes correspondants. Il est montré plus tard d'autres exemples suivant la même logique mais pour des longueurs de clef plus importantes.

Un point bien plus important dans les figures ci-dessus est la présence de *cut points* ou points de coupures dans le graphe. Ces *cut points* sont des sommets du graphe ayant la particularité de le scinder en deux si on les supprime. Pour un tel cas que celui de AES-4, seuls deux *cut points* suffisent pour couper les graphes ci-dessus (à noter que ce nombre n'augmente pas avec les tours comme vu juste avant) On notera aussi la forme "tressée" du graphe assez perturbante qui transparaît une certaine faiblesse et que l'on retrouve pour AES-64, 10 tours.

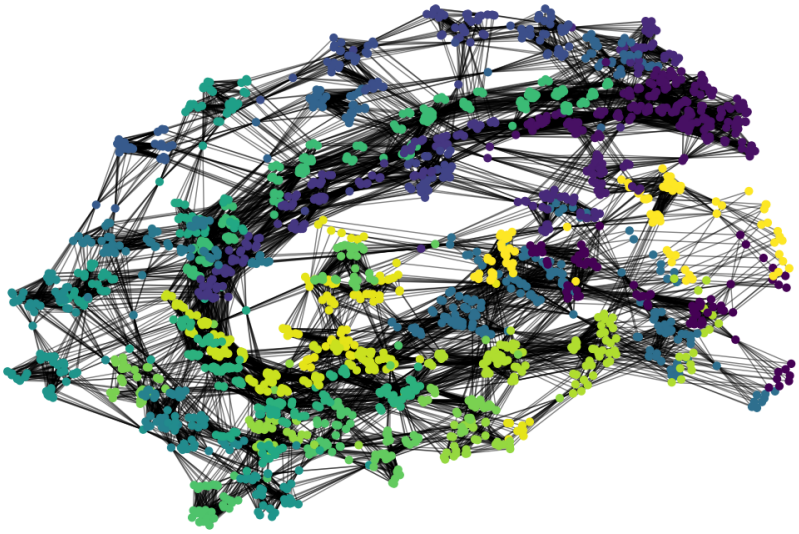

FIGURE 4 – AES-64, 10 tours avec cet aspect de tresse

La forme absolue "tressée" du graphe prend sa limite pour AES-64 puisque certaines connexions se font entre les communautés mais on perçoit quand même cette forme et toujours beaucoup de *cut points* qui peuvent être le principal angle d'attaque pour de tels graphes.

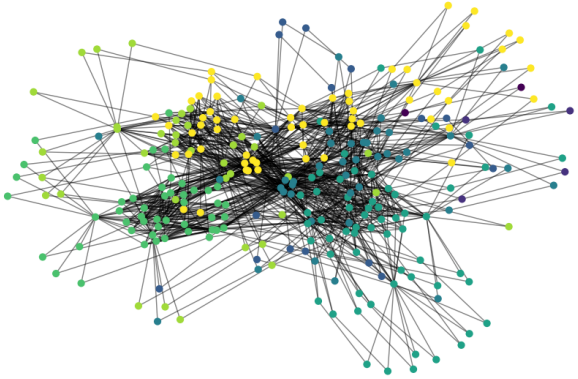

FIGURE 5 – Graphe pour AES-64, 5 tours

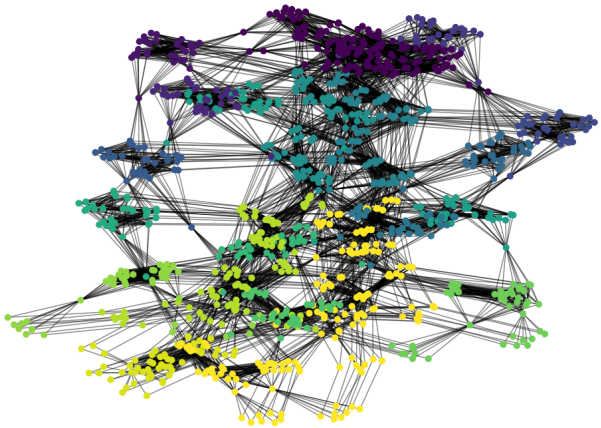

FIGURE 6 – Graphe pour AES-64, 5 tours

Une autre observation est la présence d'une "couronne" extérieure dans des graphes dont l'évolution suit celle des tours. Un ensemble de sommets de degré 2 uniquement, principal obstacle dans la quête de séparation du graphe en deux puisqu'elle a tendance à relier un sommet à deux communautés différentes. Les figures ci-dessus démontrent ce principe de "couronne".

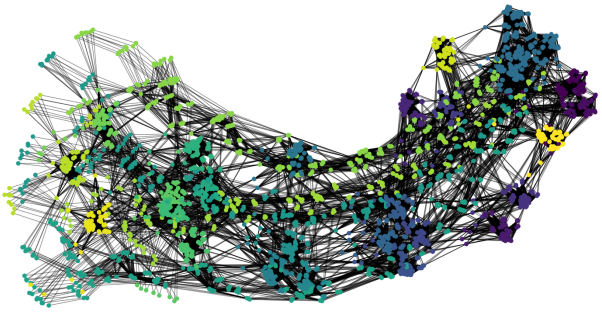

FIGURE 7 – Graphe pour AES-128, 7 tours

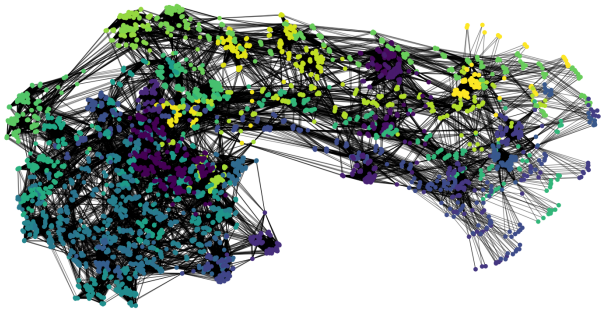

FIGURE 8 – Graphe pour AES-128, 10 tours

Une extension à AES-128, mais les résultats ici sont intraitables. Il est plus judicieux de traiter d'aussi gros graphes en termes d'ensembles de *cut points*, et de voir s'il est raisonnable de scinder les graphes selon ces ensembles [3].

Lorsque cela est possible, réduire le nombre de colonnes et/ou lignes du corps d'extension fini pour avoir un graphe plus simple pour une même taille de clef.

5.3 Évolution de la connectivité avec *bit-guessing*

Le *bit-guessing* est le fait de proposer une valeur pour une inconnue en lui attribuant successivement 0 puis 1, et de voir avec laquelle le système possède une solution. Cette méthode peut grandement aider dans la simplification des systèmes de polynômes mais a pour inconvénient de devoir tester exhaustivement toutes les valeurs possibles de toutes les inconnues que l'on soumet au *bit-guessing*, ce qui résulte en un accroissement exponentiel des calculs (2^n avec n le nombre d'inconnues *bit-guessed*).

5.3.1 *bit-guessing* des *cut points* de plus haut degré

Retirer tous les sommets de plus haut degré du graphe (jusqu'à une certaine limite) permet de réduire drastiquement la densité de celui-ci en peu de calculs. Même si les graphes ont tendance à être extrêmement creux (une densité < 0.05 et qui diminue encore plus lorsque l'on monte dans les tours), ça permet de diminuer aussi le nombre de liens entre les sommets et par conséquent les relations entre les inconnues, ce qui aide au calcul d'une base de Gröbner.

3. On considère environ 115 sommets comme étant raisonnable pour AES128.

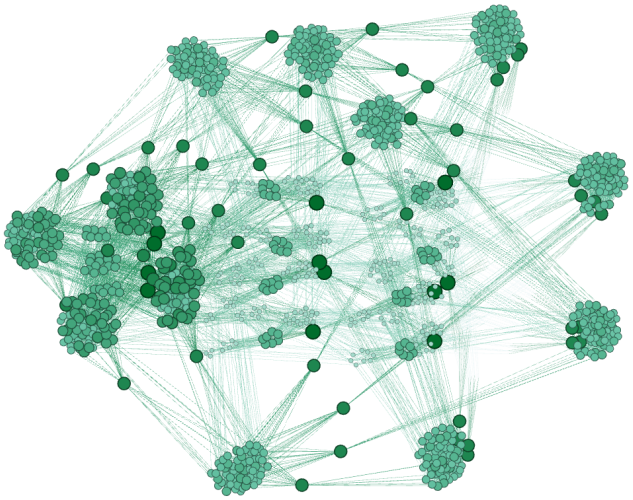

FIGURE 9 – AES128, 3 tours

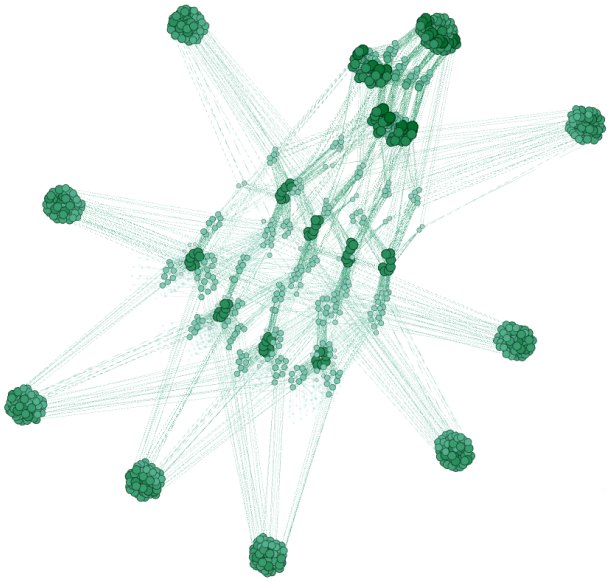

FIGURE 10 – Même graphe mais sans les noeuds de plus haut degré

Le graphe prend une forme d'étoile, avec des regroupements de sommets excentrés mais reliés à au moins deux groupes centraux, ce qui les rend très difficilement séparables du reste. Cette formation après retrait des plus gros noeuds du graphe est un comportement que l'on retrouve aussi dans AES128. De manière générale et comme déjà observée avant dans ce rapport, quelle que soit la taille de la clef utilisée, le graphe aura toujours le même comportement et les mêmes formes dans des situations similaires. Et d'ailleurs, on peut observer la même répartition des bits dans le graphe que ce soit pour AES64 ou AES128.

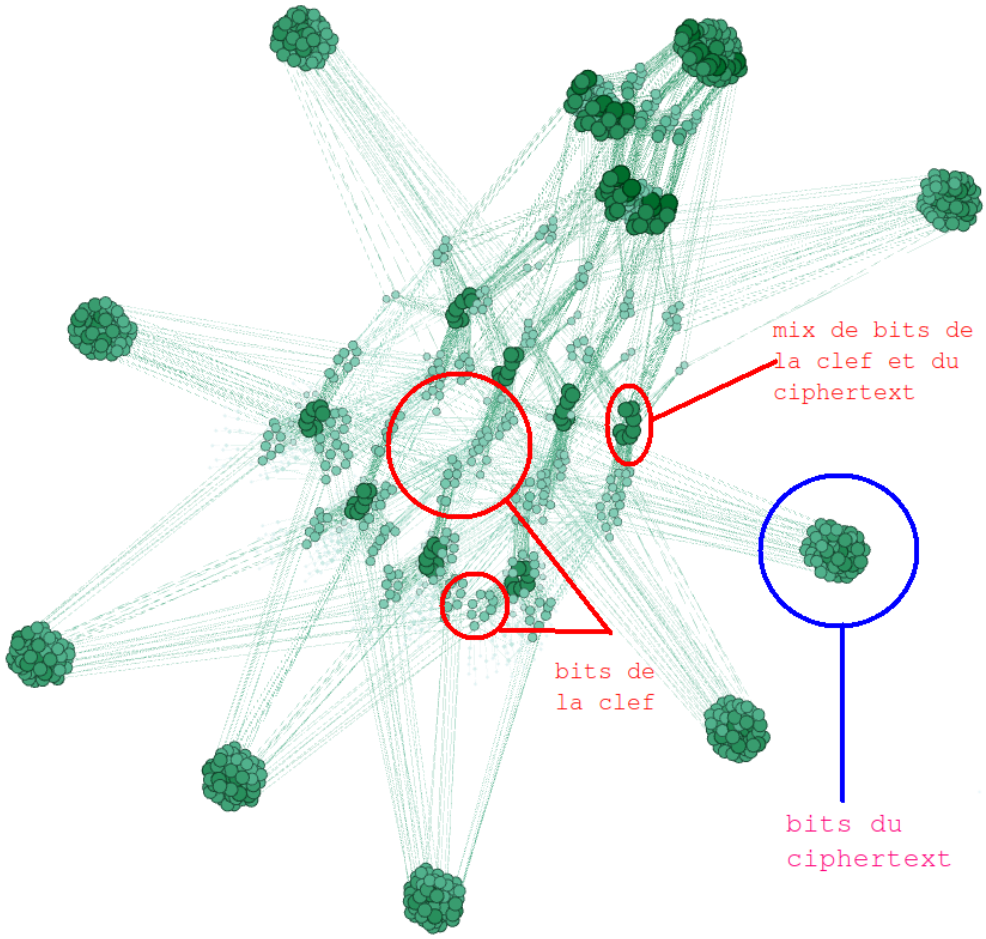

FIGURE 11 – Répartiton des bits dans le graphe

Les bits sont toujours ordonnés de la même manière. Les regroupements excentrés ne sont constitués que de bits du texte chiffré tandis que les noeuds du graphe de plus petit degré mais ayant des connections dans plusieurs communautés sont les bits de la clef de chiffrement. D'autres motifs apparaissent et sont un mélange des deux.

Pousser le *bit-guessing* exclusivement pour les noeuds de plus haut degré dans le graphe ne donnera pas de composante connexe, ou du moins le nombre de bits à deviner serait bien trop élevé. Pour cela, il faut s'attaquer au problème de séparation du graphe et du choix des noeuds car il est possible de séparer un graphe en deux parties pour des résultats assez raisonnables.

5.3.2 Obtention de deux composantes connexes sur AES-128, 10 tours

Pour information, le graphe étudié est non-orienté, possède 4288 sommets ainsi que 36130 arêtes pour une densité $d \approx 0.004$ (un graphe très creux, dont la densité est 3 fois inférieure à son équivalent sur 64 bits). La modularité du graphe est $m_G \approx 0.877$ qui est une valeur très haute en terme de distinction de communautés.

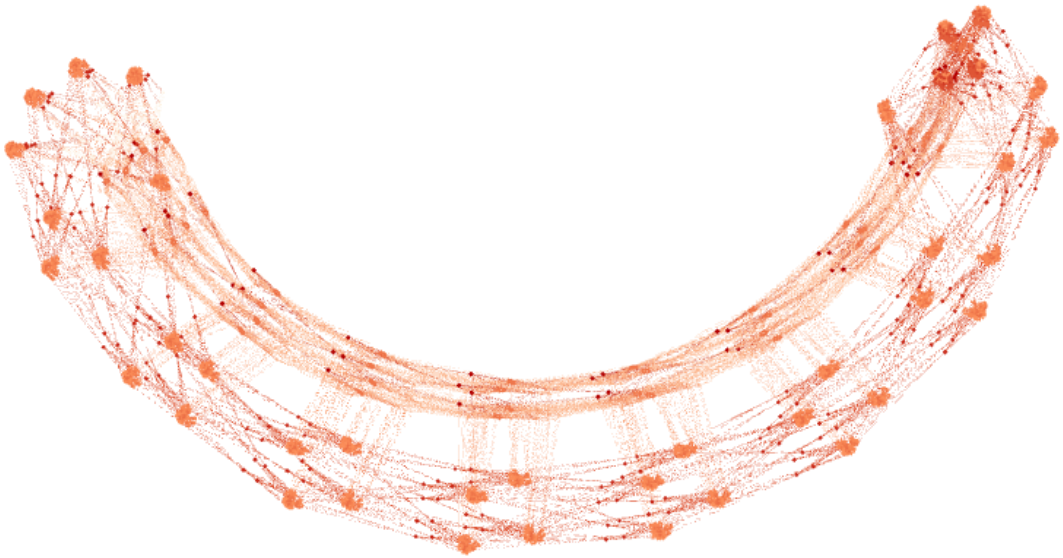

FIGURE 12 – Représentation graphique de AES-128, 10 tours

Un premier scindement pertinent se fait au milieu du graphe, simplement en retirant les sommets reliant deux communautés apparentes car un avantage du graphe est que la plupart de ses communautés n'ont que peu de liaisons aux autres. Les premiers résultats basés sur des combinaisons de *ciphertext* et clef aléatoires donnent une séparation en environ 145 sommets, ce qui n'est pas efficace mais peut s'optimiser.

Considérant la première approche, le nombre de sommets à retirer peut sûrement être réduit sur certains graphes, mais cela reste beaucoup de calculs. Une autre méthode est d'isoler un regroupement de sommets, et de manière récursive, recommencer sur un autre groupe excentré dans le but d'affaiblir le graphe en y allant pas à pas.

Les premiers résultats sont prometteurs, en seulement 40 suppressions nous obtenons une seconde composante connexe d'environ 65 sommets comme vus ci-dessous.

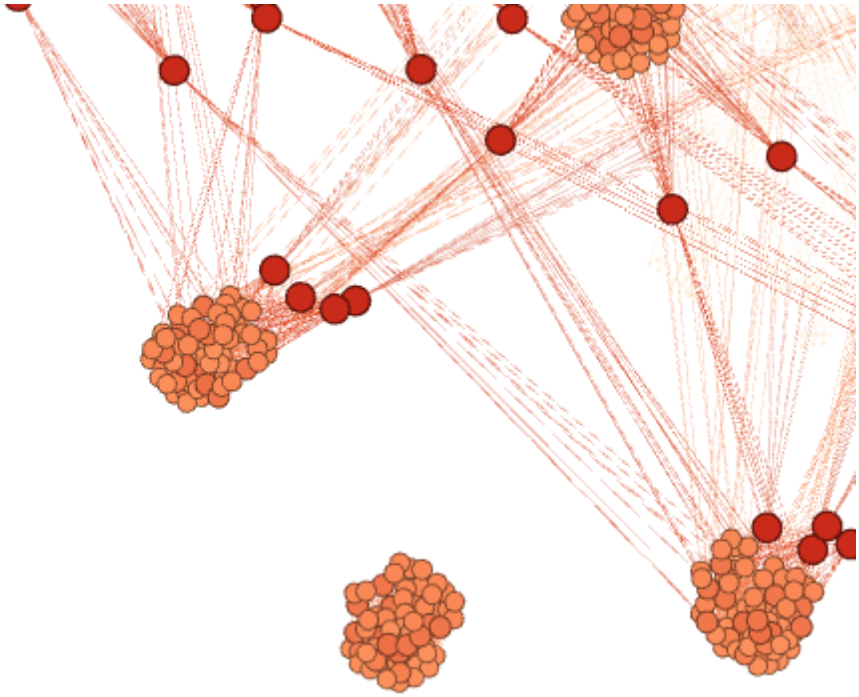

FIGURE 13 – Regroupement de sommets excentré dont les liens avec le graphe ont été supprimés

Une étude plus approfondie sur tous les tours de AES-128, fournit une courbe aux aspects logarithmiques, les graphes présentent ce même minima pour $n \geq 4$ car ils sont morphologiquement similaires et ne se distinguent que par leur taille. Ainsi, la configuration idéale pour les scinder en deux se retrouve partout.

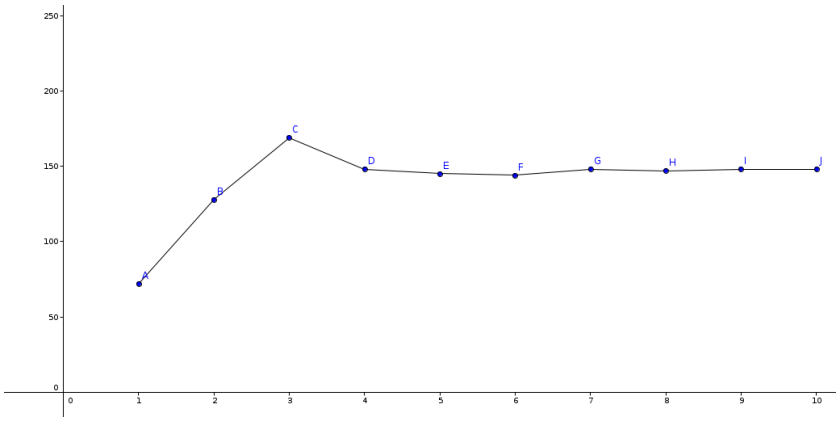

FIGURE 14 – Minima de noeuds à supprimer pour chaque tour dans AES-128

Références

[1] Thomas Debris-Alazard, Léo Ducas and Wessel P.J. van Woerden. *An Algorithmic Reduction Theory for Binary Codes : LLL and more.*
LIX, CNRS UMR 7161, École Polytechnique, 91128 Palaiseau, FRANCE

[2] Gregory V. Bard. *Algebraic Cryptanalysis.*
Department of Mathematics, Fordhma University, Bronx, NY 10458 USA

[3] Matthew Musson. *Attacking the Elliptic Curve Discrete Logarithm Problem.*
Acadia University, Spring Convocation, 2006.

[4] Vincent D. Blondel, Jean-Loup Guillaume and Renaud Lambiotte. *Fast Unfolding of Communities in Large Networks.*
J. Stat. Mech. (2008) P10008 (`http://iopscience.iop.org/1742-5468/2008/10/P10008`)

[5] Sayan Ghosh, Mahantesh Halappanavar, Antonio Tumeo, Ananth Kalyanaraman, Hao Lu, Daniel Chavarrià-Mianda, Arif Khan and Assefaw H. Gebremedhin. *Distributed Louvain Algorithm for Graph Community Detection.*
Washington State University, Pullman, WA, USA.

[6] *Lecture 30 : The Gram-Schmidt process.*

[7] Jan Verschelde. *Analytic Symbolic Computation.*
Passage on Gröbner bases, UIC, Dept of Math, Stat & CS

[8] Martin Albrecht. *Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis.*
Royal Holloway, University of London for the degree of Doctor of Philosophy 2010.

[9] Alin Bostan, Frédéric Chyzak, Marc Giusti, Romain Lebreton, Grégoire Lecerf, Bruno Salvy and Éric Schost. *Algorithmes Efficaces en Calcul Formel.*
CrateSpace Edition, september 2017.

[10] David A. Cox, John Little and Donal O'Shea. *Ideals, Varieties, and Algorithms.*
An Introduction to Computational Algebraic Geometry and Commutative Algebra, 2015.

[11] Thomas Becker, H. kredel and Volker Weispfenning. *Gröbner Bases.*
A Computational Approach to Commutative Algebra, 2012.

[12] Martin Kreuzer and Lorenzo Robbiano. *Computational Commutative Algebra 1.* Volume 1, 2008.