TD0 - Révisions Sage et Cryptanalyse Basique

Responsable : M. Bombar

Introduction

SageMath (ou encore Sage) [Sage24] est un logiciel libre et ouvert de calcul formel et symbolique fondé sur le langage de programmation Python. Il est développé depuis 2005 par une large communauté d'utilisateurs (enseignants, chercheurs, étudiants), et l'objectif est de proposer une alternative viable aux logiciels propriétaires comme Magma [BCP97], Maple [MAPLE24] ou encore Matlab [MATLAB24] [1]. Pour ce faire, Sage intègre de nombreux autres logiciels libres comme GAP [GAP24], PARI/GP [PARI/GP24] (développé ici par des chercheurs de l'Université de Bordeaux), ou encore des bibliothèques comme FLINT [FLINT24], M4RI [M4RI24]... [2]

Installation

Normalement, Sage est déjà installé sur les machines de cette salle informatique. Si vous souhaitez l'installer sur vos machines personnelles, il existe essentiellement deux façons de faire :

1. La plus simple : Installer la version déjà construite pour votre système.
2. La plus longue : Compiler Sage depuis les sources.

L'intérêt de la deuxième version est que vous pouvez contribuer à Sage en modifiant les fichiers sources, ou encore en en créant de nouveaux !

Vous trouverez plus d'information sur cette page `https://doc.sagemath.org/html/en/installation/index.html` (en anglais).

Utiliser Sage

Il existe de nombreuses façons d'utiliser Sage, qui ne sont pas mutuellement exclusives. Personnellement, je vous conseille le shell interactif pour tous vos tests, et de sauvegarder vos commandes dans un fichier texte.

1. À noter que les outils d'algèbre sont beaucoup plus présents dans Sage que ceux d'analyse, ce qui le rapproche plus de Magma.
2. Sage propose aussi une interface avec des logiciels propriétaires comme Magma s'il est présent sur votre machine

Le Shell Interactif Vous pouvez accéder à l'interpréteur Sage (au fonctionnement extrêmement proche de `iPython`) via la ligne de commande en ouvrant un terminal et en tappant simplement la commande

```
$ sage
```

Vous devriez obtenir quelque chose ressemblant à

```
SageMath version 10.4, Release Date: 2024-07-19
Using Python 3.12.5. Type "help()" for help.

sage:
```

Vous pouvez alors entrer vos commandes après le mot-clé `sage:` Pour quitter il suffit de tapper `exit` ou encore d'appuyer sur `Ctrl-D`.

Éditeurs de textes Comme n'importe quel langage de programmation, vous pouvez écrire vos programme Sagemath dans un fichier texte, qui peut ensuite être interprété par un shell via la commande `load`.

```
sage: load("myFile.sage")
```

Remarque 1. *Si vous modifiez le fichier "myFile.sage", vous devrez alors le recharger. Vous pouvez aussi choiser de l'attacher à votre session via la commande* `attach`*. Il sera alors rechargé automatiquement à chaque modification.*

```
sage: attach("myFile.sage")
```

Remarque 2. *Pensez à activer la coloration syntaxique dans vos éditeurs de textes. Une bonne solution est de faire en sorte que votre éditeur charge la même que si vous ouvriez un fichier Python. Par exemple, si vous utilisez Emacs, vous pouvez rajouter la commande suivante dans votre configuration*

```
(setq auto-mode-alist (cons '("\\.sage$" . python-mode) auto-mode-alist))
```

À noter qu'il existe aussi un `sage-mode` *pour Emacs, disponnible ici* `https://github.com/sagemath/sage-shell-mode`*, mais je ne l'utilise pas.*

Jupyter Notebook Si vous appréciez les environnements graphiques et les cellules offertes par les Notebooks Jupyter, il existe un noyau pour utiliser Sage. Pour exécuter un Notebook, vous pouvez lancer la commande suivante dans un terminal

```
$ sage -n jupyter MonNotebook.ipynb
```

Je n'aime pas particulièrement ce mode d'utilisation, et n'en dirai donc pas plus ici. Vous trouverez plus d'information dans la documentation officielle.

Accéder à la Documentation

Si vous ne l'avez pas déjà, un des objectifs du cours est que vous gagniez en autonomie lors des séances de programmation. Cela passe en particulier par une recherche dans la documentation en ligne pour connaître les options qui s'offrent à vous, et/ou comprendre le fonctionnement des objets et algorithmes que vous manipulez. Dans le cas de Sage, la documentation officielle est disponible (en anglais) ici `https://doc.sagemath.org/html/en/reference/`. Elle est générée automatiquement à partir du code source de Sage. En particulier, ce dernier est **extrêmement** bien commenté (c'est même un des pré-requis pour pouvoir soumettre du code dans l'implémentation officielle), avec des exemples d'utilisation (qui servent pour les tests de débuggage). En particulier, et comme de manière générale dans un shell `iPython`, la documentation specifique à un objet peut s'obtenir en tappant `?` après un objet. Vous pouvez même accéder au code source de l'implémentation de cet objet en tappant `??`. Ce dernier peut-être très utile, en particulier pour savoir comment sont implémentés certains algorithmes, mais peut-être source de confusion compte-tenu de l'architecture de Sage.

Remarque 3. *Vous pouvez accéder à la liste des méthodes attachées à un objet grâce à la* ***completion automatique*** *en tappant* `TAB` *après* `monObjet.` *(le . est important).*

1 Manipulation de Structures Algébriques en Sage

La plupart des primitives cryptographiques reposent sur l'utilisation de structures discrètes comme les corps et anneaux finis ainsi que les espaces vectoriels, modules, matrices et polynômes (éventuellement à plusieurs variables) sur ces structures finies. Afin d'en faire la cryptanalyse, il est alors important de bien connaître les propriétés de ces objets. Cet exercice n'a pas vocation à faire un rappel exhaustif de toutes les propriétés dont vous auriez besoin, mais surtout vous aider à les manipuler en Sage. Si jamais vous avez des questions, que ce soit pendant ce TD ou les cours suivants, j'insiste, mais n'hésitez surtout pas.

Soit N un entier naturel et soit $R \overset{\text{def}}{=} \mathbb{Z}/N\mathbb{Z}$ l'anneau des entiers modulo N. En sage, l'anneau R est construit avec la commande suivante :

```
sage: N = 26
sage: R = IntegerModRing(N)
```

Ouvrez un shell Sage et vérifiez par vous même les propriétés de R que vous pouvez facilement manipuler avec Sage.

(Q1) Quelle est la caractéristique de R ?

(Q2) Rappelez sous quelle condition sur N l'anneau R est en réalité un corps.

(Q3) Dans le cas où N ne vérifie pas les conditions de la question précédente, quels sont les inversibles $R^\times$ de R ? Quel est le cardinal de $R^\times$?

On suppose à présent que N vérifie la propriété de (Q2), et qu'en particulier R est un corps. Dans ce cas, on peut aussi construire le corps fini à N éléments via

```
sage: F = GF(N) #Ici je suppose que N a la bonne propriete
```

Vérifiez que si N ne vérifie pas la propriété de (Q2), alors cette construction échoue.

Attention, vous pourriez être tentés de penser que F et R sont alors les mêmes objets. C'est vrai, **mathématiquement** (en réalité ils sont isomorphes), mais leur **représentation en machine** est différente.

```
sage: type(R)
<class 'sage.rings.finite_rings.integer_mod_ring.
   IntegerModRing_generic_with_category'>
sage: type(F)
<class 'sage.rings.finite_rings.finite_field_prime_modn.
   FiniteField_prime_modn_with_category'>
sage: R == F
False
```

En particulier, les algorithmes qui les manipulent sont différents, et en général ils sont nettement plus lents dans la représentation « anneau des entiers modulo N » que dans la représentation « Galois Field ». Attention à bien utiliser la bonne représentation dans les bons cas !

On rappelle que pour tout N vérifiant la propriété (Q2), et pour tout entier $d \geq 1$, il existe un corps fini de cardinal N^d. Par ailleurs, tous les corps finis de cardinal N^d sont isomorphes et cette classe d'isomorphisme est notée $\mathbb{F}_{N^d}$. Rappelons-en la construction.

Soit F un corps (non nécessairement fini). On rappelle que l'anneau $F[X]$ des polynômes en une indéterminée est un anneau euclidien (comme $\mathbb{Z}$), et donc en particulier principal, et factoriel. Soit $P \in F[X]$ un polynôme irréductible, et on note d son degré. On rappelle que dans le cas où $F = \mathbb{C}$, ce degré est forcément égal à 1, et dans le cas où $F = \mathbb{R}$, il est égal à 1 ou 2. En revanche, dans le cas général, et en particulier lorsque F est un corps fini, il existe des polynômes irréductibles de tout degré $d \geq 1$.

(Q4) Rappelez pourquoi $K \stackrel{\text{def}}{=} F[X]/(P)$ est un corps.

(Q5) On note d le degré de P. Montrez que K est un F-espace vectoriel de dimension d.

(Q6) En particulier, lorsque F est le corps $\mathbb{Z}/N\mathbb{Z}$ lorsque N vérifie (Q2), quel est le cardinal de K.

Remarque 4. *Ces questions rappellent l'**existence** des corps finis, mais n'en démontrent pas l'unicité.*

De la même manière que précédemment, l'efficacité des algorithmes dans les corps finis dépendent fortement de leur représentation : choix du polynôme P, choix de la base de K vu comme F-espace vectoriel.

En Sage, on peut construire l'anneau des polynômes en l'indéterminée X de la façon suivante

```
sage: A.<X> = F[]
```

Dans la suite du cours, on aura aussi l'occasion de manipuler des anneaux de polynômes à plusieurs variables. On peut les construire en nommant les variables explicitement

```
sage: A.<X, Y> = F[]
sage: A.gens()
(X, Y)
```

Lorsqu'il y a beaucoup de variables, on peut aussi utiliser des variables indicées.

```
sage: A = PolynomialRing(F, 'x', 20)
sage: A.gens()
(x0, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14,
   x15, x16, x17, x18, x19)
```

(Q8) Construire le corps $\mathbb{F}_8$ avec Sage, de façon native (donc pas à la main). Déterminez quel polynôme est utilisé.

(Q9) Vérifiez que $\mathbb{Z}/8\mathbb{Z}$ n'est pas un corps. Attention à ne pas faire l'erreur !

(Q10) Vérifiez (avec Sage par exemple) que le polynôme X^3+X^2+1 est irréductible sur $\mathbb{F}_2$, mais n'est pas le polynôme précédent. Construire alors une autre représentation de $\mathbb{F}_8$.

Lorsque l'on manipulera des chiffrements par flots, on aura besoin de certains polynômes qu'on appelle **primitifs**.

On rappelle que lorsque K est un **corps fini**, le groupe $K^\times$ des éléments inversibles est **cyclique**, c'est à dire qu'il existe un élément $\alpha \in K$ tel que $K^\times = \langle\alpha\rangle = \{1, \alpha, \alpha^2, \ldots, \alpha^{r-1}\}$ pour un certain r. Soit $\mathbb{F}_N = \mathbb{Z}/N\mathbb{Z}$ le corps fini à N éléments, où N vérifie (Q2). Soit $P \in \mathbb{F}_N[X]$ un polynôme irréducible de degré d, et soit $K \overset{\text{def}}{=} \mathbb{F}_q[X]/(P) = \mathbb{F}_{N^d}$ le corps fini à N^d éléments. P est dit **primitif** si toutes ses racines engendrent le groupe cyclique $K^\times$. En particulier, $K = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{N^d-2}\}$.

(Q11) À l'aide d'un élément de $\mathbb{F}_{2^4}$, trouver (avec Sage) un polynôme de degré 4, irréductible sur $\mathbb{F}_2$, mais **non primitif**.

2 Le chiffrement de Hill

Cet exercice s'intéresse à un chiffrement polyalphabétique par substitution inventé par Lester S. Hill en 1929 [Hil29]. Comme nous allons le voir, son caractère linéaire le rend complètement vulnérable à une attaque à clair connu, dans laquelle un adversaire a accès à une paire `(clair, chiffré)` pour un texte clair suffisamment long. En particulier, il n'a donc jamais vraiment été utilisé en pratique. Cependant, il a pendant très longtemps été considéré comme resistant aux attaques exploitant uniquement des messages chiffrés (**ciphertext-only attacks**) ; la première attaque complète dans ce modèle datant de 2015 [KA15].

2.1 Chiffrement et Déchiffrement

Pour chiffrer un texte écrit avec un alphabet de taille N, on va encoder chaque lettre par un identifiant unique dans $\mathbb{Z}/N\mathbb{Z}$. Pour simplifier, dans cet exercice on va considérer l'alphabet Latin classique sans diacritiques et avec toutes les lettres en minuscules. Ainsi, $N = 26$ et chaque lettre est remplacée par son rang dans l'alphabet (`a` est encodé par 0 et `z` par 25). Le chiffrement de Hill va opérer sur des blocs de taille fixée d qui est un paramètre du cryptosystème. La clé de chiffrement est une matrice M dans $M_d(\mathbb{Z}/26\mathbb{Z})$ et

le chiffré correspondant à un bloc clair $(m_1, \ldots, m_d) \in (\mathbb{Z}/26\mathbb{Z})^d$ est le vecteur $(c_1, \ldots, c_d) \in (\mathbb{Z}/26\mathbb{Z})^d$ obtenu par le produit matrice-vecteur :

$$M \cdot \begin{pmatrix} m_1 \\ \vdots \\ m_d \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_d \end{pmatrix}.$$

Si la longueur du message clair n'est pas un multiple de d, on réalise un padding par des 0.

(Q12) Quelles matrices $M \in M_d(\mathbb{Z}/26\mathbb{Z})$ correspondent à un chiffrement valide ? C'est-à-dire qui produisent un chiffré pouvant être déchiffré de manière unique.

(Q13) Quelle est la taille de l'espace des clés valides en fonction de d ?

(Q14) En déduire la plus petite valeur de d offrant une sécurité d'au-moins 128 bits vis-à-vis de l'attaque par recherche exhaustive sur les clés.

(Q15) Est-ce que plusieurs tours de chiffrement (avec plusieurs clés différentes) augmente la sécurité ?

(Q16) Implémentez en Sage les fonctions `hillEnc(d, M, m)` et `hillDec(d, M, c)` qui prennent en argument le paramètre `d`, une clé `M` et soit un message clair `m` soit un chiffré `c` et implémentent respectivement le chiffrement et le déchiffrement.

Commencez par implémenter une fonction simple qui réalise l'encodage d'un texte écrit dans l'alphabet latin vers $\mathbb{Z}/26\mathbb{Z}$.

2.2 Attaque à Clair Connu

Vous venez d'intercepter un message chiffré dont le contenu est

```
gptbwbyhiejlsymeimtnyggxgfaawdzuyir
```

et vous soupçonnez qu'il a été obtenu par la méthode de Hill avec une certaine clé secrète $M \in (\mathbb{Z}/26\mathbb{Z})^d$ pour un paramètre `d` inconnu. Cependant, vous savez aussi que le texte suivant

```
apmppgoarpnhnjpemimplojfnfysdfgzofwjbhpbhpqfqukbcezrgloxgoeulqhce
evlismqmhsmcdeflcbmvbaqepzsyatnoyrkiwthdgtimrxt
```

se déchiffre en

```
attheannualarcticgaladancingpenguinsalwaysstealtheshowateveryparty
leavingtheotheranimalsclappinginaweandlaughter
```

avec la même clé M.

(Q17) Proposez une méthode pour retrouver la clé secrète M.

(Q18) Implémentez l'attaque en Sage pour déchiffrer le message initial.

3 Le cryptosystème de Vigenère

L'un des plus célèbres cryptosystèmes par substitution polyalphabétique est un chiffrement datant de la seconde moitié du XVIème siècle, et souvent attribué à Blaise de Vigenère [Vig86]. Rappelons-en le principe : Comme dans l'exercice précédent, on représente chaque lettre par son rang dans l'alphabet, vu comme un élément de $\mathbb{Z}/26\mathbb{Z}$. La clé est une chaîne de caractères de longueur ℓ secrète vue comme un élément de $(\mathbb{Z}/26\mathbb{Z})^\ell$ et un texte clair est chiffré en ajoutant à chaque lettre le décalage donné par la clé, ce décalage étant appliqué successivement et de façon cyclique. Par exemple, si la clé est `alexandrin`, représentée par $(0, 11, 4, 23, 0, 13, 3, 17, 8, 13) \in (\mathbb{Z}/\mathbb{Z})^{10}$, alors le texte clair

```
portezcevieuxwhiskyaujugeblondquifume
```

est chiffré en

```
pzvqemfvdvefbthvvbgnuuydeoofvqqfmcuzh
```

(Q19) Implémentez les fonctions `vigenereEnc(clear, key)` et `vigenereDec(cipher, key)` qui prennent en argument une clé et un texte clair (respectivement un chiffré) et implémentent le chiffrement et déchiffrement de Vigenère.

(Q20) On suppose que la longueur ℓ de la clé est connue.

- (a) Décrivez une procédure simple pour retrouver le texte clair à partir d'un texte chiffré suffisamment long.
- (b) Implémentez cette procédure.

3.1 Cryptanalyse du chiffrement de Vigenère

On appelle **indice de coïncidence** d'un texte T, qu'on note $IC(T)$, la probabilité que deux lettres tirées uniformément dans T soient égales.

(Q21) Donnez une expression pour $IC(T)$, en fonction de la longueur de celui-ci.

(Q22) Quelle doit être sa valeur typique lorsque T est un texte aléatoire ?

On s'attend à ce qu'un bon chiffrement ait un indice de coïncidence qui soit très proche de l'indice de coïncidence typique d'un texte aléatoire. Cependant, pour un texte écrit dans un langage structuré comme le Français ou l'Anglais, l'indice est en réalité très loin de cet indice typique. Par exemple, pour le Français, une analyse statistique indique qu'il est de l'ordre de 0.078.

(Q23) Montrez que l'indice de coïncidence n'est pas modifié par l'application d'un chiffrement monoalphabétique.

En particulier, c'est cet indice statistique qui rend un texte obtenu par un chiffrement monoalphabétique très **distinguable** d'un texte aléatoire. L'idée de cette cryptanalyse proposée par William Friedman en 1920 est de se servir de ce distingueur pour retrouver la taille de la clé utilisée dans le chiffrement de Vigenère, avant d'appliquer l'attaque décrite à la question (Q20).

(Q24) Implémentez une attaque permettant d'estimer la longueur de la clé.

(Q25) Déchiffrez le texte suivant :

```
ufzbdemltfnlfgmoneefrttkophfeiuplbfxbtrmltfczfygipzjygblyyjivigpf
pffveptmfmfavjyxbymfcisptpyhjeuvppmpemwejeiopjgpxbweqpgipwpygilre
lmmciqcmtpioeinzmhyejeiuditpwqlhstpmpwslgpcrjpwqlvmpwfw
```

3.2 Challenge

Parfois, nos implantations ne sont pas forcément assez puissantes pour retrouver automatiquent le chiffré. Une intervention humaine est alors nécessaire. Essayez de déchiffrer le texte suivant :

```
appiofmkvatxewovjynppbkumdisfvewsaysbmmtlsbcilpgtxizukczyvkwzryay
wsxgxrkedynxpwmreaermrmciucgakokklrpitnbxvydzpenfpzwtzmrx
```

Références

[BCP97] Wieb Bosma, John Cannon et Catherine Playoust. « The Magma algebra system. I. The user language ». In : **J. Symbolic Comput.** 24.3-4 (1997). Computational algebra and number theory (London, 1993), p. 235-265. ISSN : 0747-7171.

[FLINT24] William Hart et al. **FLINT : Fast Library for Number Theory (Version 3.1.2)**. http://flintlib.org. FLINT Development Team. 2024.

[GAP24] The GAP Group. **GAP – Groups, Algorithms, and Programming (Version 4.13.1)**. https://www.gap-system.org. The GAP Group. 2024.

[Hil29] Hill, Lester S. ≪ Cryptography in an Algebraic Alphabet ≫. In : **The American Mathematical Monthly** 36.6 (1929), p. 306-312.

[KA15] Shahram Khazaei et Siavash Ahmadi. **Ciphertext-Only Attack on $d \times d$ Hill in $O(d13^d)$**. Cryptology ePrint Archive, Paper 2015/802. http://eprint.iacr.org/2015/802. 2015.

[M4RI24] Martin Albrecht et Gregory Bard. **The M4RI Library – Version 20240729**. https://github.com/malb/m4ri. The M4RI Team. 2024.

[MAPLE24] Maplesoft, a division of Waterloo Maple Inc. **Maple (Version 2024)**. https://www.maplesoft.com. Maplesoft. 2024.

[MATLAB24] The MathWorks, Inc. **MATLAB (Version R2024a)**. https://www.mathworks.com. The MathWorks, Inc. 2024.

[PARI/GP24] The PARI Group. **PARI/GP (Version 2.15.5)**. http://pari.math.u-bordeaux.fr/. Université de Bordeaux. 2024.

[Sage24] W. A. Stein et al. **Sage Mathematics Software (Version 10.4.0)**. http://www.sagemath.org. The Sage Development Team. 2024.

[Vig86] Vigenère, Blaise De. **Traicté des chiffres, ou Secrètes manières d'escrire**. 1586.