Cet article est rédigé par des élèves. Il peut comporter des oublis et imperfections,
autant que possible signalés par nos relecteurs dans les notes d'édition.

CRYPTOGRAPHIE [1]

Année 2018 – 2019

Guillaume Fourche, Romain Pastorelli, Baptiste Pignier, Simon Robin, Amy Tontat élèves de la classe de seconde 3

Encadrés par CANAT Laurence

Établissements : Lycée Louis Armand // Collège de Côte-Rousse

Chercheur Chercheuse : Jimmy GARNIER (LAMA)

1. Présentation du sujet

Notre projet consiste à trouver des techniques de codage efficaces, discrètes et faciles d'utilisation, qui nous permettront de transmettre un message confidentiel qui sera à la vue de tous, sans que celui-ci ne puisse être lu. De plus, nous avons anticipé les erreurs qui pourraient subvenir lors de la transmission ou lors de l'interprétation.
Le but de ce projet à l'interface entre mathématiques et informatique est de comprendre et changer un langage courant en un langage plus mathématique voire informatique.

2. Annonce des conjectures et résultats obtenus

Nous avons commencé par travailler et s'entraîner avec les codes les plus connus en particulier avec le code César, ou la grille des franc-maçons, qui nous ont permis de nous familiariser avec la cryptographie. Par la suite nous avons commencé à rédiger nos propres codes. Nous avons trouvé des méthodes originales de codage comme :

- le code Zodiac
- le code Tricot,
- le code Phonétique

Chacun présentant des avantages et des inconvénients, mais ne satisfaisant pas toujours notre problématique et dépassant parfois aussi nos compétences informatiques et graphiques.

Du coup, nous nous sommes tournés vers le codage affine, simple de fonctionnement et de codage, plus difficile, pour nous de décryptage.

Nous avons réussi à coder en Python nos travaux; permettant ainsi à un utilisateur de coder un message ou de décrypter un message reçu.
Nous avons aussi travaillé sur un déchiffrage par force brute et étendu notre alphabet à 74 caractères (point, point d'interrogation, espace, majuscule ou minuscule…), mais lors de la mise en commun de nos programmes de décryptage et déchiffrage, nous n'avons pas réussi à étendre notre alphabet à 74, mais sommes restés avec

26 minuscules.

3. Texte de l'article

La cryptographie

1 . Introduction
- 1.1 La cryptographie à travers les âges :
  - le code César ✓
- 1.2 Enjeux et nouvelles techniques de la cryptographie ✓
  - anlayse fréquentielle
  - introduction de l'informatique dans la cryptographie
- 1.3 Créer un code simple et robuste ✓
  - présentation brève de notre projet
  - Une idée de nos travaux

2 . Première approche empirique :
- 2.1 Code du zodiac %s ✓
- 2.2 Tricode %s ✓
- 2.3 Code phonétique %s ✓

3 . Autour du code affine
- 3.1 Une approche plus mathématique et informatique :
- 3.2 Qu'est ce que le codage affine ?
- 3.3 Codage et décryptage ✓
- Qu'est-ce que l'inverse modulaire ? ✓
- 3.4 Implémentation de notre code ✓

L'homme a toujours essayé de communiquer, partager ses informations, mais aussi les protéger face à d'éventuelles menaces. Il s'est montré astucieux pour essayer de transmettre ses données en toute confidentialité à son seul destinataire et ses méthodes ont considérablement évolué au fil de l'histoire.
Le problème " transmettre de façon sûre un message" reste d'actualité dans un monde où Internet tient une place considérable et où les échanges sont de plus en plus nombreux.
Comment ne pas s'inspirer de ces recherches pour créer notre propre code ? Quelles en sont les limites ?

1 . Introduction

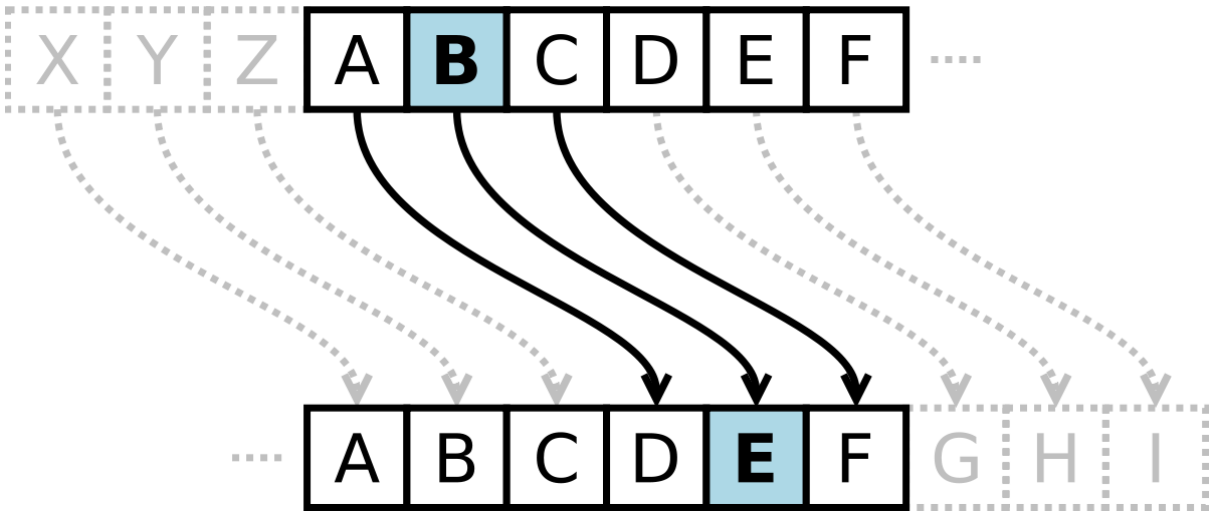
Littéralement, le mot **cryptographie** vient du grec “kruptos” (κρυπτός) “caché” et “graphein” (γράφειν) “écrire”.
Cette discipline a pour but de protéger et de cacher des messages afin de les transmettre en sécurité à leur destinataire et d’empêcher d’autres personnes d’avoir accès aux informations qu’ils comportent.

1.1 La cryptographie à travers les âges :

En effet, déjà *chez les égyptiens*, en 1900 av JC, des hiéroglyphes “inventés” auraient été dissimulées sur des tablettes.

Chez les grecs, en 500 av JC, on utilisait le procédé de la scytale; il s’agissait d’un bâton, de diamètre défini, autour duquel on enroulait un parchemin. Le message était noté sur toute la longueur de la scytale, puis déroulé et confié à un messager. Le destinataire n’avait plus qu’à entourer une scytale *de même diamètre* pour déchiffrer.

Jules César lui même se servait d’un stratagème judicieux (pour l’époque) pour envoyer des missives à ses campements ou coder ses correspondances personnelles.
Le Code de César : est le code le plus connu et le plus simple à utiliser. Il suffit de décaler les lettres de l’alphabet d’un certain nombre de places. Par exemple, si on réalisait un décalage de 3 lettres vers la droite, le A devient un D, le B un E...



Toutes les lettres subiraient un décalage dit **à clé** : trois (à choisir entre l’expéditeur et le destinataire).
Une lettre est toujours substituée par la même autre lettre de l’alphabet.

En ayant la clé (3), le message est facilement décryptable.

Une personne mal intentionnée interceptant ce type de message peut facilement “casser” ce code (même en ne connaissant pas la clé) car il suffit qu’elle teste les 25 décalages possibles ! A l’époque de

Jules César, vu le peu de personnes ayant accès à l'éducation, ce mode de codage paraissait intéressant.
Aujourd'hui, avec un simple tableur ou un petit algorithme, on peut facilement décoder ce type de décalage.

Par contre, si au lieu de choisir **une permutation circulaire des lettres de l'alphabet** (c'est-à-dire avec une clé de 3 : ABC devient DEF), on choisit **une permutation** de nos 26 lettres, pour la première lettre on a 26 choix possibles, pour la 2-ème : 25 choix possibles, pour la 3-ème : 24 choix possibles, soit $26 \times 25 \times 24 \times ... \times 2 \times 1$ = 26 ! (factorielle 26)

= 4×10^{26} possibilités !!!

Ce qui promet d'être nettement plus long à décoder...

1.2 Enjeux et nouvelles techniques de la cryptographie :

En étudiant le Coran et la fréquence d'apparition des lettres, il semble qu'au IX-ème siècle Abu Yusuf Ya'qub ibn Is-haq ibn as-Sabbah Oòmran ibn Ismaïl al-Kindi, plus connu sous le nom d'**Al-Kindi**, publie le premier ouvrage de cryptanalyse (Manuscrit sur le déchiffrement des messages codés). Ainsi, dans un texte assez long, il est facile en fonction de la langue utilisée de savoir la fréquence de 'e', de 'a' ou de 'w' !

A) Qu'est-ce que l'analyse fréquentielle ?

On sait que dans la langue française, les lettres les plus utilisées sont E, S, A, I, N, T, U, R, L et O.
On sait aussi que la lettre "q" est toujours suivie d'une "u" et que les articles "le, la, les, un" sont répandus.

On peut considérer que *les fréquences en pourcentage* de ces lettres sont :

E	**A**	**S**	**I**	**N**	**T**	**U**	**R**	**L**	**O**
17,7	**8,5**	**7,5**	**7,4**	**7,2**	**7,1**	**6,8**	**6,3**	**5,7**	**5,4**

On constate que dans la langue française, pour un texte suffisamment long, le "e" est la lettre qui apparaît le plus. Donc en se basant sur cette hypothèse, on peut facilement voir la lettre du code César qui se répète le plus et donc l'assimiler au "e"....et de proche en proche décoder FACILEMENT le reste du message.
Il est donc encore plus aisé de nos jours, de crypter, mais aussi de décrypter un message de type "César", où l'utilisation de l'ordinateur nous permet de gagner en rapidité.

Pour pallier cette difficulté, au XVI-ème siècle **Blaise Vigenère** proposa son idée : une clé alphabétique se reproduisant sur la longueur du message. Ainsi, une même lettre peut être codée de façons différentes.
Ce code résista et aida aux transmissions lors de l'apparition du télégraphe au milieu du XIX-ème siècle.

La cryptographie a souvent été utilisée pour transmettre des messages concernant les guerres et les batailles rendant des messages contenant des informations confidentielles sur les plans d'attaque (par exemple) impossibles à comprendre.

B) Introduction de l'informatique dans la cryptographie :

Le code de Vigenère sera cassé par **Charles Babbage**, le père de l'informatique, en 1854.
Des machines sont effectivement conçues pour crypter des données et les décrypter grâce à des sortes de cartes perforées. Ce qui permet de gagner un temps considérable. Mais ce n'est pas toujours facile de décrypter. Rappelons la machine "Enigma" dont se servait l'armée allemande pour encoder ses messages confidentiels et qui ne fut "brisée" par une équipe britannique qu'en 1944. Décodage, tenu secret, mais grâce auquel on gagna la guerre et deux ans de bataille !
A chaque fois qu'un code est décrypté, les cryptanalystes essaient d'en inventer un encore plus puissant ! Avec l'arrivée des ordinateurs et Internet, la multiplicité des échanges et des demandes (mail, compte bancaire, signature numérique…), on cherche des systèmes de plus en plus performants une seule clé publique ne suffit plus.
À partir du XXème siècle, la cryptographie a évolué, entre autres grâce aux mathématiques, en utilisant de nouveaux outils (comme l'inverse modulaire que l'on abordera plus loin), mais aussi, et surtout grâce au développement de l'informatique (comme avec le chiffrement RSA de 1983). Aujourd'hui, la cryptographie est très utilisée sur Internet pour chiffrer les pages Web, les transactions virtuelles, et toutes sortes d'autres choses.

A notre échelle que sommes nous capables de faire ?

1.3 Créer un code simple et robuste :

Nous sommes cinq élèves de seconde du Lycée Louis Armand. Nous avons choisi comme thème la cryptographie car pour nous, il était évident que l'informatique interviendrait dans notre problématique. Le projet consiste à envoyer un message crypté à une personne et cette personne doit le décrypter avant qu'une autre personne ne le décrypte à sa place.

Après avoir travaillé sur des codes connus, nous avons inventé nos propres codes, puis les avons testés entre nous pour voir leurs limites.

2 . Première approche empirique :

2.1 Code du zodiac %s ✓

Nous nous sommes inspirés d'un tueur en série, toujours non identifié à ce jour, qui opérait autour de San Francisco, dans les années 60 à 80 et qui aurait commis une quarantaine de meurtres revendiqués : le Zodiac.
Pour la petite histoire, il envoyait à la presse des lettres dans lesquelles il revendiquait la plupart de ses meurtres et donnait des détails pour le prouver. Sur chacune de ses lettres figurent des cryptogrammes dans lesquels il donnerait des informations sur son identité, par exemple.

Ce code était si compliqué que la police n'a pas réussi à le casser, seulement quelques parties dont elle ne n'est même pas sûre.
Nous avons immédiatement pensé à lui quand on avons commencé à chercher un code.
Nous avons analysé un des cryptogrammes et avons essayé d'imaginer un alphabet de substitution en associant un symbole à chaque lettre. Nous nous sommes vite rendus compte que ce code était

facilement cassable grâce à l'analyse fréquentielle : dans un texte en français, par exemple, le "e" est la lettre qui revient le plus souvent suivie du "a"...Nous avons donc modifié notre alphabet afin que les lettres les plus récurrentes aient 3 symboles, par exemple le "i" ou le "n", puis 2 pour celles qui apparaissent moins souvent comme le"b" ou le "p" et enfin 1 pour celles que nous rencontrons le moins le"x" ou le "y").
De plus, pour renforcer notre code, nous avons associé à chaque symbole 2 lettres différentes (comme pour le "a" et le "l").

***** Alphabet ZODIAC *****

En brouillant ainsi les pistes de l'analyse fréquentielle, il devient vraiment compliqué de décrypter un message.
Ceci est un sacré avantage, mais ne correspond plus exactement à notre problématique, si le destinataire ne parvient pas à retrouver notre message initial !!

Ce code présente néanmoins quelques imperfections : comme il y a plusieurs possibilités de chiffrage pour la plupart des lettres, et que les symboles codent plusieurs lettres, il est possible que, dans une phrase, deux mots soient codés identiquement et qu'ils aillent tous deux dans le contexte de la phrase. Ce cas là, ne nous est jamais arrivé mais c'est statistique.

Le deuxième inconvénient, qui peut aussi être un avantage, c'est que le déchiffrage est très complexe, même avec la clé. Aux vues de tous ces défauts et de la complexité à transcrire nos symboles nous avons décidé de ne pas le développer par la suite et l'avons abandonné. En effet, il nous apparaît que pour pouvoir décrypter "aisément", il ne faut pas qu'un même symbole ait plus d'un antécédent; sinon difficile de l'identifier.

2.2 Tricode %s ✓

En cours, d'histoire, nous prenons des notes et avons souvent besoin d'écrire vite, aussi utilisons nous des abréviations telles que " tps" à la place de temps, les chiffres romains pour désigner les différents rois...et nos propres abréviations, connues et compréhensibles de nous seuls.

De telles abréviations apparaissent notamment dans les codifications du tricot.

En confectionnant une écharpe, l'une de nous, nous a montré son patron et son codage : "maille à l'endroit', "maille à l'envers" … et nous avons eu l'idée d'utiliser ses abréviations de tricot.
Cela nous a rappelé le morse, qui lui aussi possède 2 symboles récurrents et a été très utilisé dans les transmissions.
Ces acronymes sont K1, qui veut dire knit 1 en anglais ou une maille à l'endroit, et P1, qui veut dire purl 1 ou une maille à l'envers.
On a donc remplacé les points par K1 et les tirets par P1.

La seule règle c'est que le numéro qui se trouve après la lettre, change en fonction du nombre de points ou de tirets qui se suivent.

Ex : H → morse : **....** tricode : K4.

Les avantages de ce code sont que c'est une façon originale de cacher un mot, c'est assez simple et on peut facilement le maîtriser, à condition de connaître le morse.

Les inconvénients sont que ça fait de très longues phrases et nous n'avons malheureusement trouvé aucune façon de raccourcir les abréviations de façon cohérente.

***** Alphabet Tricode *****

A = .- K1 P1	**N = -. P1 K1**
B = -... P1 K3	**O = --- P3**
C = -.-. P1 K1 P1 K1	**P = .--. K1 P2 K1**
D = -.. P1 K2	**Q = --.- P2 K1 P1**
E = . K1	**R = .-. K1 P1 K1**
F = ..-. K2 P1 K1	**S = … K3**
G = --. P2 K1	**T = - P1**
H = …. K4	**U = .. K2**
I = .. K2	**V = …- K3 P1**
J = .--- K1 P3	**W = .-- K1 P2**
K = -.- P1 K1 P1	**X = -..- K1 P2 K1**
L =.-.. K1 P1 K2	**Y = -.-- P1 K1 P2**
M = -- P2	**Z = --.. P2 K2**

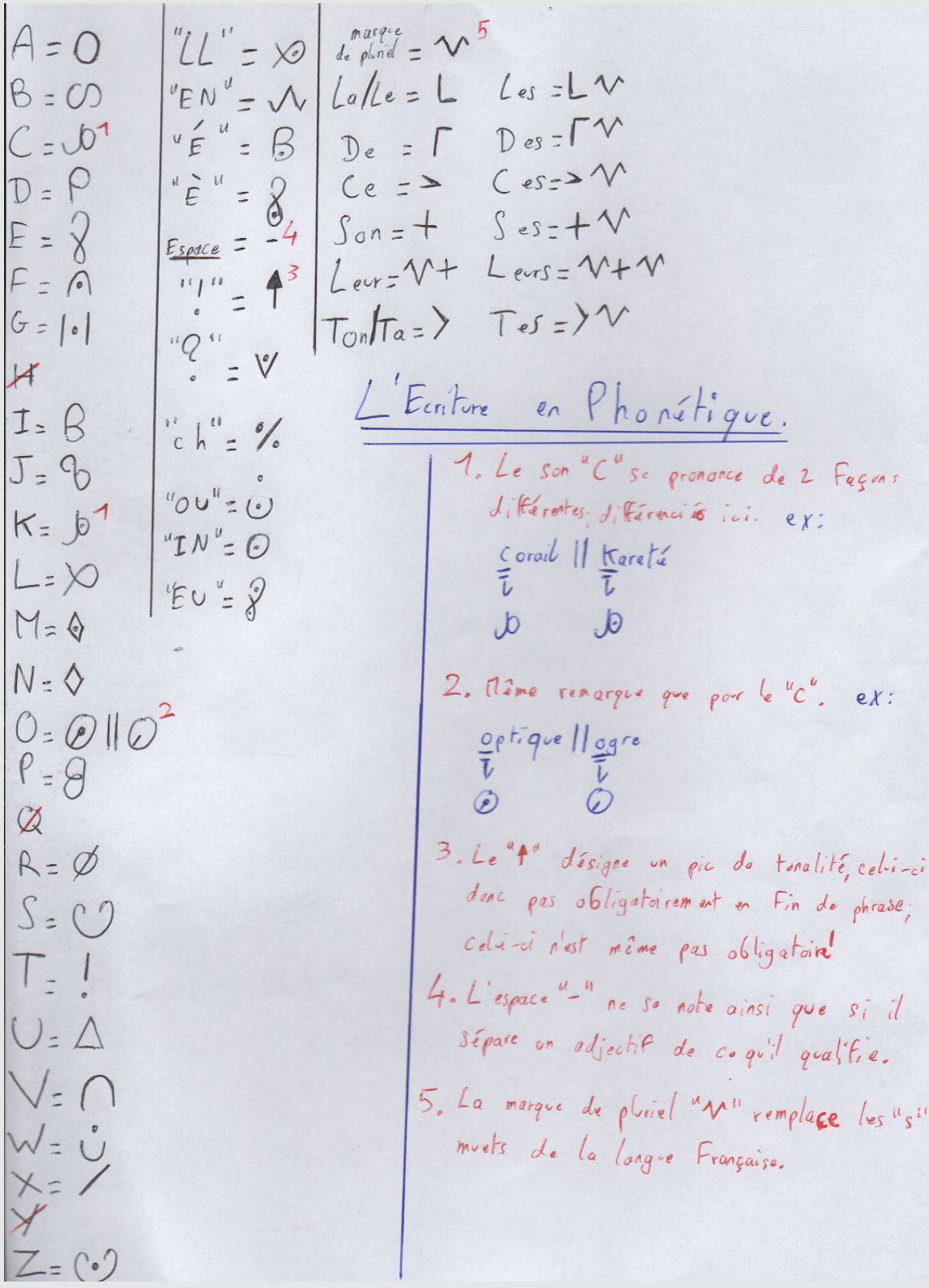
Par contre, telle la scytale des hébreux, nous pouvons porter notre écharpe codée sans que quiconque s'en aperçoive...à moins de connaître les mailles et les correspondances...ou d'avoir un bon algorithme avec un procédé graphique pour identifier nos symboles. Ce que nous ne nous sentons pas capables, pour l'heure, de réaliser.

2.3 Code phonétique %s ✓

Le principe du codage phonétique est d'écrire des phrases avec une base grammaticale très différente de celle de la langue française. Inspiré des hiéroglyphes égyptiens ainsi que des syllabaires japonais, le code phonétique permet de construire des mots en se basant sur les sons qui constituent celui-ci, c'est à dire que chaque son sera représenté par un symbole.

Les lettres muettes (tel que les “H”; “S” muets;...) ne sont pas représentées alors que certains sons ne possédant pas de signe propre (tel que le “ou”; “in”; “ch”;...) en possède un dans ce code. De plus, les sons qui se ressemblent, tel que le m // n; é // è; on // en, on aussi des signes qui se ressemblent. Pour finir, on peut ajouter à tout cela plusieurs règles de grammaire qui diffèrent du français comme décrit plus haut. On trouve par exemple un symbole permettant d’indiquer les pics de tonalité de la phrase comme il existe des symboles spéciaux pour les pluriels ainsi que pour les déterminants.

Tout cela a pour but de perdre les éventuels personnes qui voudrait lire les messages écrits de cette façon sans connaître l’écriture tout en permettant aux personnes un peu expérimentées de lire ces même messages avec une grande facilité.
Car en effet, ce code peut être décodé très facilement si l’on connaît bien les symboles, permettant même de lire et décoder des messages directement à l’oral.



***** Alphabet Phonétique *****

Ce code est très anthropocentré, mais difficilement réalisable sur une programmation simple (même avec la force brute en essayant toutes les possibilités et en les comparant aux mots existants dans un dictionnaire, cela nous paraît long…). Mais avec l'avènement du Machine Learning et du Deep Learning, ce problème devrait pouvoir être résolu. Si on parvient effectivement à connaître nos habitudes de langage, nos préférences, on devrait être capable de reconstituer un de nos messages.

Nous avons aussi décidé d'abandonner cette idée de codage; certes intéressante pour des message personnel, mais moins réalisable pour des messages longs et qui ont besoin d'une plus grande sécurité.

Nous avons fini par choisir le code affine, car c'est une méthode de chiffrement simple, connue, et qui nous permettait d'appliquer des connaissances mathématiques acquises au cours de l'année de seconde (fonctions affines). De plus, l'une de nous l'avait travaillée en option MPS (Méthodes et Pratiques Scientifiques) et a pu nous l'expliquer.

3 . Autour du code affine

3.1 Une approche plus mathématique et informatique :

Après nos essais de codage, nous avons compris qu'il nous fallait une fonction "simple" permettant de coder notre alphabet. La fonction "affine" nous est apparue assez naturellement, puisque nous l'avions étudiée et qu'il nous était facile d'obtenir les images de chacune des lettres. Et cette fois, nous nous sentions au point pour automatiser le processus avec un algorithme.

3.2 Qu'est ce que le codage affine ? ✓

Le codage affine est une méthode simple de cryptographie utilisant la substitution d'une lettre de l'alphabet par une autre lettre en utilisant une fonction affine.
On rappelle qu'une fonction $f(x) = ax + b$ est une fonction affine de paramètres a et b où a et b sont des réels quelconques.

3.3 Codage et décryptage ✓

Pour convertir un mot que tout le monde peut lire en message chiffré, il faut réaliser une opération qu'on appelle **le codage.** Cette opération consiste à substituer une lettre d'un mot par une autre. Ce procédé se rapproche du code de César évoqué plus tôt, à la seule différence que le décalage de position de lettre dans l'alphabet n'est plus un nombre, mais est une fonction affine. Cette fonction affine sous la forme $ax + b$ est définie par deux valeurs, a et b.

Cette fonction va prendre comme antécédent la position d'une lettre, et lui attribuer une nouvelle valeur, soit l'image de la fonction.

- Si cette valeur est supérieure à 26, on lui applique alors l'opération modulo 26 pour la ramener à une valeur inférieure à 26, c'est-à-dire qu'on recherche le reste de la division euclidienne de cette valeur et 26.

Cette même lettre va alors être changée par la lettre correspondant à la nouvelle valeur.

Exemple : En choisissant la fonction affine $f(x) = 3x + 2$

! Pour l'exemple, nous prendrons l'alphabet commençant à 0 tel que **A = 0 et B = 1** … pour être cohérent avec l'exemple de la programmation !

*Pour coder la lettre B par la fonction $3x + 2$,on récupère la place de B dans l'alphabet, 1.
On calcule **l'image** de 1 par la fonction f, on obtient : 3 * 1 + 2 = 5.

Le chiffre 5 correspond à la lettre F. La lettre B se code donc par F avec la fonction $3x+2$.

*Pour coder la lettre O par la fonction $3x+2$,on récupère la place de O dans l'alphabet, 14.
On calcule **l'image** de 14 par la fonction f, on obtient : $3\times 14+2 = 44 > 26$
or $44 = 26\times 1 + 18$ c'est-à-dire le reste de la division euclidienne de 44 par 26 est 18 $44 \equiv 18[26]$
Le chiffre 44 correspond à la 18-ème lettre soit S. La lettre O se code donc par S avec la fonction $3x+2$

Le codage du mot BONJOUR par la fonction $3x+2$

Lettre :	B	O	N	J	O	U	R
Place de la lettre :	1	14	13	9	14	20	17
Application de la fonction $3x+2$:	5	44	41	29	44	62	53
Modulo 26	5	18	15	3	18	10	1
Substitution :	F	S	P	D	S	K	B

Le mot BONJOUR est donc codé par FSPDSKB

Avec cette méthode de cryptage affine, on a constaté que "a" ne pouvait pas prendre toutes les valeurs.
En effet, **a** doit être **premier avec le nombre de lettres de notre alphabet.**
C'est-à-dire que notre paramètre a choisi et le nombre de lettres de notre alphabet (26, ici) ne doivent avoir que 1 comme diviseur commun.

À cette étape de nos recherches, nous avons eu besoin d'une nouvelle notion : **l'inverse modulaire.**

Qu'est-ce que l'inverse modulaire ?

Nous avons eu besoin de cette notion lors du déchiffrage, quand il fallait inverser toutes les opérations pour retrouver la lettre originelle. Or, on sait que toutes les opérations ont un inverse (l'addition et la soustraction, ou la multiplication et la division), il nous fallait donc en trouver une pour "défaire" le modulo. Nous avons alors fait quelques recherches internet pour trouver cette opération, et voilà comment l'inverse modulaire est naturellement apparu au cours nos recherches.

L'inverse modulaire est une notion que l'on apprend en terminale spécialité math, autant dire que ce n'est pas une notion mathématique très simple.
Mais petit à petit, nous avons à peu près réussi à comprendre son fonctionnement et, toujours avec l'aide de notre professeure, nous l'avons codée. Nous allons ici, essayer d'expliquer au mieux ce qu'il y a derrière ce nom. Tout d'abord, il faut savoir que le modulo (noté **mod (z) ou [z]**) est **le reste de la division euclidienne**, il n'y a donc pas vraiment d'opération directe qui l'annule

y l''inverse modulaire **[z]** d'un entier x, existe si pgcd(x, z) = 1 (c'est-à-dire si x et z sont premiers entre eux) et est défini comme suit : $x\times y \equiv 1[z]$
Cet inverse, s'il existe, est **unique.**

[2]

Pour trouver notre inverse modulaire, il nous a alors fallu trouver le nombre qui, multiplié par le "a" de notre fonction affine, modulo "le nombre de lettres dans l'alphabet (26)", donnait pile 1.

Pour exemple, en prenant $f(x) = 3x+2$ comme fonction de départ, on a :

$3x+2 \equiv w[26]$ x étant le numéro de la lettre de départ et w le numéro de la lettre finale, après le chiffrage

Soit encore

- $3x \equiv (w-2)\,[26]$
- On cherche **l'inverse modulo 26 de 3,** c'est-à-dire le nombre qui, multiplié par 3 et modulo 26 (nombre de lettres dans l'alphabet), donne pile 1 :

$3 \times 1[26] = 3[26]$
$3 \times 2[26] = 6[26]$
$3 \times 3[26] = 9[26]$
$3 \times 4[26] = 12[26]$
$3 \times 5[26] = 15[26]$
$3 \times 6[26] = 18[26]$
$3 \times 7[26] = 21[26]$
$3 \times 8[26] = 24[26]$
$3 \times 9[26] = 1[26]$ (trouvé !!)

Maintenant qu'on a trouvé un 1, on peut s'arrêter car on peut être sûr qu'il n'y en a qu'un seul et que c'est celui qu'on a trouvé : 9.

- On a donc trouvé l'inverse modulo 26 de 3 : 9.
  On a ensuite : $3 \times 9 \times x \equiv 9 \times (w-2)\,[26]$
  Soit $x \equiv 9w - 18\,[26]$

- La fonction inverse modulo 26 de $3x+2$ est donc $9w-18$, c'est la fonction de décryptage. (On rappelle que x correspond à la place de la lettre <u>avant chiffrage</u> dans l'alphabet et que w, lui, correspond à la place de la lettre <u>après chiffrage</u>, dans l'alphabet. C'est pour cela que l'on n'utilise pas la même inconnue.)

3.4 Implémentation de notre code ✓

<u>Programmation :</u>

Contrairement à nos précédents codes, notre nouveau code se prête bien à une implémentation sur ordinateur. Comme nous étudions, cette année, le Python, nous avons décidé d'utiliser ce langage pour coder, déchiffrer et décrypter nos messages.

Python est un langage simple et globalement dérivé de l'anglais, il était donc facile de s'approprier les quelques règles de syntaxe. Deux d'entre nous ont déjà des connaissances et de l'expérience Python et de la programmation en général, c'est alors à eux qu'a été remis la tâche de transformer nos compétences mathématiques en programmation.
Pour ce faire, nous nous sommes servis uniquement du module de mathématiques inclus nativement dans Python.
L'expérience, l'interface utilisateur et l'esthétique du programme n'a pas été notre priorité sur ce projet. Nous avons mis l'accent sur l'optimisation des procédés mathématiques afin d'obtenir un temps de calcul assez court en adéquation à la puissance des ordinateurs fournis, notamment sur la dernière partie de notre programme qui est le décryptage par méthode dite '' brute-force'' nécessitant beaucoup de calculs.

En effet, notre programme se divise en trois sous-programmes : le chiffrage, le déchiffrage et le décryptage.

L'un de nous deux s'est d'abord penché sur le chiffrage et le déchiffrage, pendant que l'autre abordait le décryptage. Quand nous avons tous deux terminé, nous avons mis nos codes en commun et apporté quelques modifications afin que les deux programmes se "comprennent".

*Le chiffrage fut le plus simple à programmer.
Il suffisait au programme :

- de demander à l'utilisateur le "a", puis le "b" de la fonction affine $ax+b$ de son choix,
- de vérifier que le "a" choisi était bien premier avec le nombre de lettres de l'alphabet,
- puis un mot à chiffrer.

Ceci étant fait, il devait aller chercher :

- la place dans l'alphabet de la première lettre du mot,
- puis appliquer, à ce nombre, la fonction affine et il obtenait un résultat.
- Dans certains cas, ce résultat pouvait être plus grand que le nombre de lettres, il fallait alors lui faire subir un "modulo 26", noté "mod [26]". On était alors sûr que le résultat serait inférieur à 26,
- il pouvait donc prendre la lettre qui se trouvait à cette place et la garder en mémoire.

Il répétait cette opération autant de fois qu'il y avait de lettres dans le mot et terminait par écrire le résultat final, à savoir **le mot chiffré** grâce à la fonction choisie précédemment.
Voici la "traduction" de ce paragraphe, en lignes de code : [3]

```
17  if programme == 1:
18      while 1:
19          try:
20              a = int(input("Pour la fonction, veuillez saisir un a :"))
21              if pgcd(a,len(dico))==1:
22                  break
23              else:
24                  print("A doit être premier avec",str(len(dico)))
25          except:
26              print("Veuillez saisir un nombre")
27      while 1 :
28          try:
29              b = int(input('Puis un b (nombre):'))
30              break
31          except:
32              print("Veuillez saisir un nombre")
33      text_clair = list(input('Veuillez saisir un mot à chiffrer :'))
34      print('Cela donne :',''.join([list(dico.keys())[list(dico.values()).index(((a * dico[i]) + b) % len(dico))] for i in text_clair]))
```

*Le déchiffrage, lui, nous a donné beaucoup plus de fil à retordre car il ne suffisait pas seulement d'effectuer toute la liste de calculs à l'envers en les remplaçant par leur inverses (addition <-> soustraction et multiplication <-> division).
En effet, comme expliqué dans "le chiffrage", la dernière opération effectuée est un modulo, or, nous avons déjà parlé de l'inverse modulaire dans l'article du même nom et nous y avons mis en évidence les problèmes qu'il nous a causé, nous n'allons donc pas revenir dessus en détail.
Le programme demande donc :

- encore une fois le "a" et le "b" de la fonction affine choisie,
- et enfin le mot à déchiffrer.
- Ensuite, il cherche donc l'inverse modulaire du "a" comme expliqué plus haut.

- Une fois qu'il l'a trouvé, il effectue le produit de l'inverse modulaire par la différence de la place de la première lettre du mot et de "b", le tout, mod[26].
- Il obtient encore une fois un résultat qui correspond à une lettre, il garde cette dernière en mémoire et
- il refait de même avec chaque lettre du mot.

Il termine, là aussi par écrire le résultat, c'est-à-dire, **le mot déchiffré.**

Encore une fois, voici, pour ceux que ça intéresse, le code :

```
35  if programme == 2:
36      while 1:
37          try:
38              a = int(input('Pour la fonction, veuillez saisir un a (nombre):'))
39              break
40          except:
41              print("Veuillez saisir un nombre")
42      while 1:
43          try:
44              b = int(input('Puis un b:'))
45              break
46          except:
47              print("Veuillez saisir un nombre")
48      text_à_déchiffré = list(input('Veuillez saisir votre mot à déchiffrer :'))
49      for j in range(1,len(dico)+1):
50          if (j*a)%len(dico)==1 :
51              inv=j
52      try:
53          print ('Cela donne :',''.join([list(dico.keys())[list(dico.values()).index((inv*(dico[i] - b))%len(dico))] for i in text_à_déchiffré]))
54      except NameError as error:
55          print('Erreur : Verifiez les entrée de A et de B ... Détails -->', error)
```

Et voici le code du décryptage : [4]

```
56  if programme == 3:
57      try:
58          texte = list(input('Texte à decrypter : '))
59          nbs = 0
60          for a in range(1,len(dico),2):
61              if a % int(len(dico)/2) != 0:
62                  for b in range(0,len(dico)):
63                      dico1 = {}
64                      for c in range(0,100):
65                          for i in range(0,len(texte)):
66                              if [(dico[i] + c * len(dico) - b) / a for i in texte][i] < len(dico) and [(dico[i] + c * len(dico) - b) % a for i in texte][i] == 0:
67                                  dico1[i + 1] = int([(dico[i] + c * len(dico) - b) / a for i in texte][i])
68                      dico1 = {i[0]:i[1] for i in sorted(list(dico1.items()))}
69                      h = [[[[list(dico.keys())[list(dico.values()).index(i)] for i in list(dico1.values())][i],[list(dico.keys())[list(dico.values()).index(i)] for i in list
70                      if len([list(dico.keys())[list(dico.values()).index(i)] for i in list(dico1.values())]) % 2 == 1:
71                          h.append([[list(dico.keys())[list(dico.values()).index(i)] for i in list(dico1.values())][-2],[list(dico.keys())[list(dico.values()).index(i)] for i
72                      if ('1' in ['1' if (i in h)==True else '0' for i in [['a','a'],['b','p'],['b','q'],['b','w'],['b','x'],['b','z'],['c','j'],['c','p'],['c','v'],['c','w']
73                          nbs += 1
74                      else:
75                          if len([list(dico.keys())[list(dico.values()).index(i)] for i in list(dico1.values())]) == len(texte):
76                              print(''.join([list(dico.keys())[list(dico.values()).index(i)] for i in list(dico1.values())]),'Codé par : a =',a,', b =',b)
77                          else:
78                              nbs +=1
79          print('Nombre de mots ne respectant pas la syntaxe française :',nbs)
80      except:
81          print('Impossible à décrypter ou non achevé')
```

Les lignes 69, 71 et 72 s'étendant trop loin, on ne les a pas mises dans la capture d'écran.

4. Conclusion

En conclusion, nous pouvons dire qu'avec le codage affine, le chiffrage de ce code est très simple, même à la main, il est très pédagogique et permet d'aborder les bases de la cryptographie. Il a aussi une très bonne application en programmation.
Par contre, à la main, il est fastidieux, mais plus simple à l'ordinateur.
De même, les programmes de déchiffrage et de décryptage sont compliqués à réaliser, mais pas forcément difficiles à comprendre.
Ce code est facilement "cassable", c'est à dire que l'on peut facilement contrer sa sécurité, ce qui n'en fait pas un code suffisamment fiable pour des messages à coder ultra-sensibles. Nous avons découvert qu'il existe un chiffrage appelé "chiffrage de Hill", qui consiste à chiffrer le message en substituant les lettres du message, non plus lettre à lettre, mais par groupe de lettres, toujours à partir d'une fonction affine et qui rend, du coup, plus difficile le cassage du code. Mais nous ne nous sommes pas encore engagés sur cette nouvelle piste.

Notes d'édition

[1] Cet article ne correspond pas vraiment aux standards de Math.en.Jeans, car il s'agit plus d'un travail de recherche bibliographique que d'une recherche mathématique. Cependant, la qualité de sa rédaction justifie sa publication.

[2] Pour être plus précis : on dit que y est un inverse de x modulo z si $xy \equiv 1\ [z]$. Si $pgcd(x,z) = 1$, alors tout entier x tel que $1 \leq x < z$ possède un unique inverse modulaire y , modulo z, vérifiant $1 \leq y < z$.

[3] Il aurait été souhaitable d'expliquer le code python, dont certaines parties ne sont pas du tout évidentes.

[4] Les auteurs distinguent "déchiffrage" et "décryptage". Le premier utilise une méthode intelligente (le calcul de l'inverse modulaire de a). Le second fait une recherche "bête" en parcourant toutes les lettres, jusqu'à trouver la bonne.