

Rapport codage et crypto

MOURELON Yann et PINSON Daniel

Avril 2023



1 Introduction

De nos jour, le chiffrement de données est un sujet sensible car omniprésent dans notre quotidien. Le développement informatique et son utilisation constante fait que nous avons de plus en plus besoin de chiffrer des données personnels avec des systèmes de plus en plus robuste afin de les protéger au mieux. Aujourd’hui, il existe de nombreux systèmes de chiffrage différents tous aussi sécurisé les uns que les autres. Dans le cadre de notre formation, nous allons nous pencher sur un système de chiffrage inventé par Mr. Bruce Schneier : **Le solitaire**.

L’objectif de ce projet est de réaliser un système de chiffrement sécurisé en suivant la méthode du solitaire.

2 Explication du système

Comme son nom l’indique, le solitaire est un système de chiffrement utilisant un jeu de 54 cartes que nous allons utiliser pour générer une clé aléatoire de même longueur que le message que nous voulons chiffrer. Pour ce faire, on mélange le paquet de carte afin d’avoir un ordre unique. En effet, il existe 8.06×10^{67} mélanges possible dans un paquet de 52 cartes, on a donc $8.06 \cdot 10^{-67}\%$ de chance d’obtenir 2 fois le même mélange, ce qui rend ce système plutôt fiable. La création de la clé s’effectue en 5 étapes :

1. On récupère le joker noir, puis on le recule d’une position. Si celui-ci se trouve à la fin du paquet, il est inséré à la deuxième place.

2. On récupère le joker rouge que l'on va reculer de 2 positions. Si celui-ci se trouve dans les 2 dernières places du paquet, nous le passons respectivement dans les 2 premières place, toujours derrière la première carte.
3. On réalise une double coupe en fonction des jokers. on échange les cartes situés au dessus du premier joker rencontré avec celles situé en dessous du second.
4. On effectue une coupe simple suivant la dernière carte. On récupère la valeur de la dernière carte (suivant l'ordre du Bridge), puis on récupère les N premières cartes (suivant la valeur de la dernière carte) pour les placer à la fin du paquet devant la dernière carte.
5. On regarde le numéro de la première carte, puis on va chercher la $(n+1)$ -ième carte de notre paquet. Si celle-ci est un joker, on mélange à nouveau le paquet suivant les étapes citées plus tôt, sinon on récupère la valeur associée à notre carte modulo 26, enfin on récupère la lettre associée à cette valeur ($1=A$, $2=B$, $3=C\dots$).

On réitère ces opérations autant de fois que nécessaire puis après avoir eu autant de lettres que notre message d'origine, on peut chiffrer celui-ci en additionnant les valeurs de chaque lettres modulo 26. Cela nous donne une suite de lettre rendant incompréhensible notre message à qui le lira sans la 'clé' qui est le paquet.

Maintenant que nous avons chiffré notre message, il faut pouvoir le déchiffrer. Pour cela rien de plus simple, la seule condition est d'avoir le même paquet de base que votre interlocuteur. Ainsi, il pourra faire exactement le même mélange ce qui le mènera au même résultat et pourra par conséquent déchiffrer le message. Cela montre une faille dans ce système de chiffrement mais nous reviendrons dessus plus tard.

3 Implémentation du chiffrement

Pour réaliser le programme qui s'occupe de chiffrer du texte, nous avons utilisé le langage Python qui nous semblait être le plus adapté dû à sa facilité d'utilisation des tableaux. Le paquet est représenté par une liste de nombre compris entre 1 et 54, avec le joker noir valant 53 et le rouge 54. Les valeurs des cartes sont celles du Bridge, donc les couleurs sont dans l'ordre trefle-carreau-cœur-pique, et l'as valant 1. Nous avons découpé notre programme en plusieurs méthodes reprennent chacune une partie du mélange expliqué précédemment :

- **reculJokerNoir** : on identifie la position du joker noir dans le paquet, puis, si celui-ci est en dernière position dans le paquet, on le met à la 2ème position. Sinon on interchange le joker avec la carte qui suit.
- **reculJokerRouge** : on identifie la position du joker rouge dans le paquet, puis, si celui-ci est dans les 2 dernières positions, on le met à la 2^{eme} ou à la 3^{eme} position respectivement. Sinon, on déplace le joker rouge derrière les 2 cartes qui suivent.
- **coupeDouble** : on déclare 3 tableaux p1, p2, p3 :
 - **p1** : contient les 1^{ères} cartes jusqu'au 1er joker
 - **p2** : contient les cartes du 1^{er} joker au 2nd (joker compris)
 - **p3** : contient le reste des cartes.

Pour ce faire, on parcours notre tableau pour trouver l'emplacement du 1^{er} joker, puis on insère dans p1, ensuite on continue de parcourir notre tableau jusqu'à ce qu'on trouve le 2nd joker, une fois trouvé, on insère dans p2 et p3, et on quitte la boucle. Pour finir, on concatène les tableaux dans cet ordre : **p3, p2, p1** (ce qui revient à échanger p1 et p3).

- **coupeSimple** : on récupère la 54^{eme} carte du tableau (à l'indice 53), puis on crée 2 nouveaux tableaux dont le premier va contenir les cartes d'incise 0 à X non-compris (X, la valeur Bridge de la carte récupérée) et le 2nd les cartes de X à 52 compris, puis on retourne la concaténation de ces 2 tableaux dans l'ordre **p2, p1** ; suivie de la carte récupérée.

Ces méthodes représentent les différentes étapes de mélange du chiffrement du solitaire. Nous avons aussi créé une méthode **melange** les appelant dans cet ordre, ce qui nous permet de faciliter la lecture de notre programme.

Les instructions pour lancer le code sont dans le fichier README.md dans le dossier du projet.

4 Valeur ajoutée

Nous avons créé une interface graphique avec la bibliothèque **tkinter** qui nous simplifie la création de celle-ci. Cette interface nous permet une utilisation plus intuitive de notre programme avec quatre zones de texte. La 1^{ere} nous permet d'écrire ou d'éditer le message que l'on veut chiffrer, la 2^{eme} affiche le message chiffré ou bien nous permet d'inscrire nous même un message chiffré, la 3^{eme} nous permet de voir le message décodé grâce au paquet visible dans la 4^{eme} zone. Il y a 2 boutons servant à chiffrer le message que l'on a entré ou bien à déchiffrer le message codé ainsi que 2 autres boutons permettant d'ouvrir un fichier .txt, ce qui importe le texte de ce dernier dans la zone correspondante pour éviter d'avoir à recopier l'entièreté de ce dernier.

Au niveau du paquet, nous avons 2 boutons nous permettant d'insérer un paquet de carte contenus dans un fichier .txt, un autre nous permettant de générer un nouveau paquet dont le programme va se servir pour coder et décoder les différents messages.

Nous avons aussi générés différents tests unitaires pour tester nos méthodes afin de vérifier que celles-ci fassent bien ce que l'on veut qu'elles fassent en fonction de certaines prédispositions. Nous avons mis les résultats de test dans les annexes et vous pourrez trouver ceux-ci dans le code fourni avec ce pdf.

5 Conclusion

Ce système de chiffrement est simple, rapide, efficace et ne demande pas une grande technologie à son utilisation car un simple jeu de carte nous est demandé pour chiffrer / déchiffrer un message. Malgré sa simplicité d'implémentation et de réflexion, le nombre de possibilité aléatoire font que ce système simple est très efficace jusqu'à un certain point. Ce système est initialement conçu pour des messages courts et ne convient pas à autre chose qu'une phrase composée uniquement de lettres. Cependant, il est possible de mettre des accents et les caractères spéciaux compris dans la langue française (ç et é par exemple). Mais ceux-ci sont convertis en leurs homonymes dans la langue anglaise pour correspondre à la table ASCII, faisant que l'on perd de l'information si il y a des accents.

Il est possible de restituer les accents, espaces, Majuscules et ponctuations en gardant dans une Hashtable les indices des éléments remplacé/supprimé en tant que clé et le caractère remplace/supprimé en tant que valeur pour ensuite remplacer dans le message final les caractères aux indices dans la Hashtable par leur valeur associée.

6 Annex

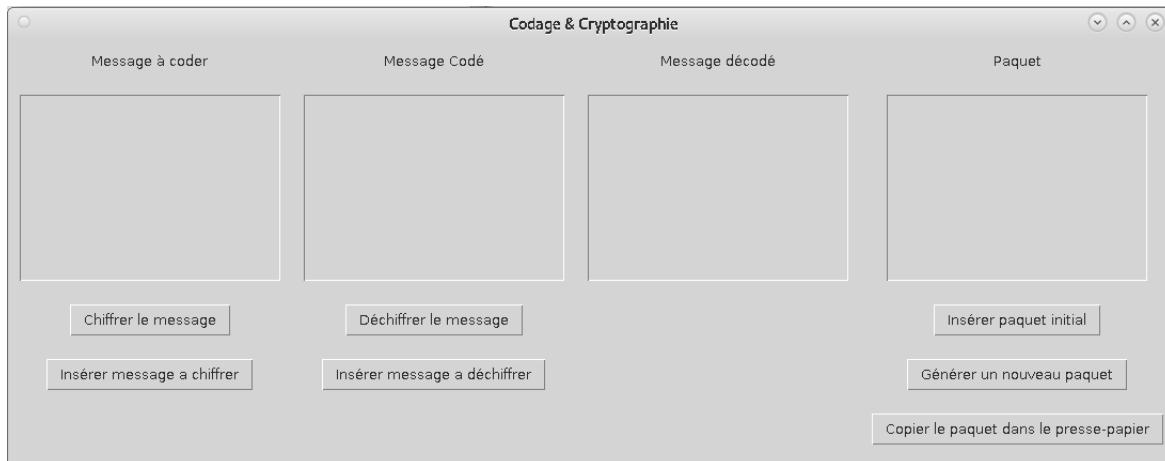


Figure 1: Initialisation de l'interface graphique

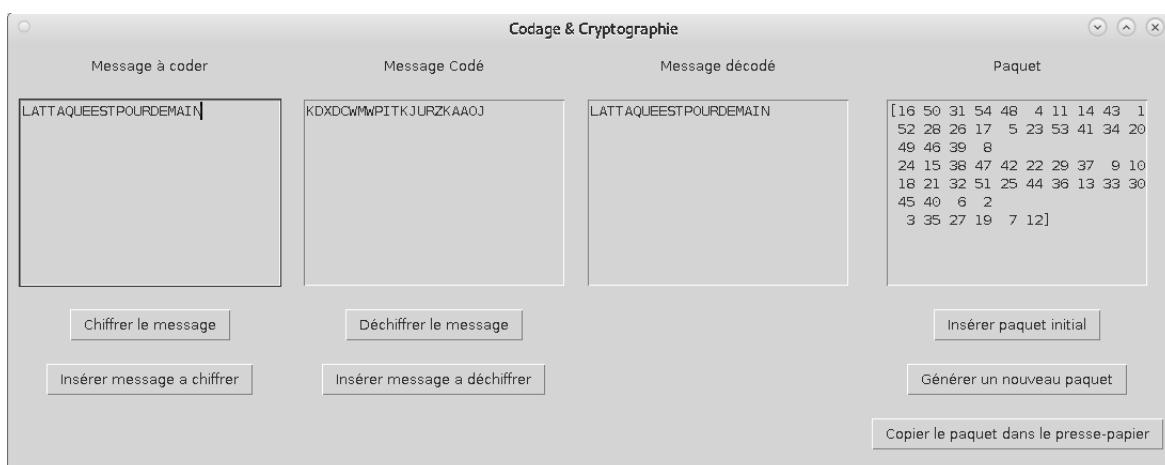


Figure 2: Jeu de test avec un paquet A

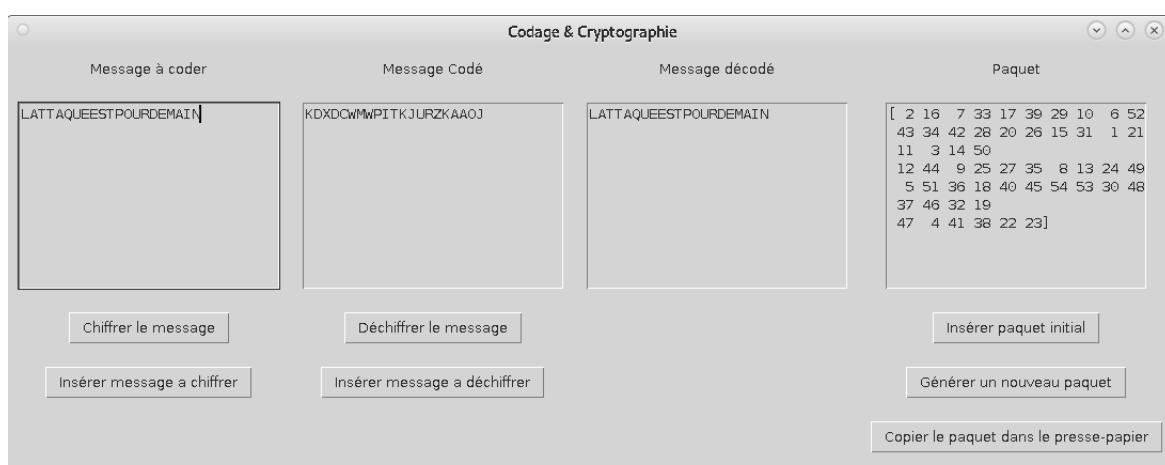


Figure 3: Jeu de test avec un paquet B



Figure 4: Jeu de test message avec caractères spéciaux

```
ym186975@M1I04-06:~/Documents/Master1/M1/s8/codageCrypto/projet$ make test
python3 -B src/test.py
=====
=====Test recul du joker noir de une position=====
Valide : testReculJokerNoirPaquetNeuf
Valide : testJokerNoirDernierePosition
=====
=====Test recul du joker rouge de deux position=====
Valide : testReculJokerRougeDernierePosition
Valide : testReculJokerRougeAvantDernierePosition
Valide : testReculJokerRougePositionNormale
=====
=====Test coupe double par rapport aux joker=====
Valide : testCoupeDouble
Valide : testCoupeDoubleNeuf
Valide : testCoupeDoubleNormal
Valide : testCoupeDoubleNormalJokerInverse
=====
=====Test coupe simple déterminée par la dernière carte===
Valide : testCoupeSimpleNeuf
Valide : testCoupeSimpleNormal
=====
=====Test lecture pseudo aléatoire=====
Valide : testLectureAleatoireNeuf
Valide : testLectureAleatoireJokerPremier
=====
=====Test validité paquet=====
Valide : testVerificationPaquetNormal
Valide : testVerificationPaquetCarteTrop
Valide : testVerificationPaquetCarteDouble
Valide : testVerificationPaquetCarteInvalide
=====
=====Test generation cle=====
Valide : testGenerationCleLettreUnique
Valide : testGenerationCleLettre
=====
=====Test formattage message=====
Valide : testFormattageMessageNormal
Valide : testFormattageMessageAccent
Valide : testFormattageMessageEspace
Valide : testFormattageMessageEspaceAccent
=====
=====Test chiffrage message=====
Valide : testChifffrageMessageValide
Valide : testChifffrageMessageAccent
Testsuite terminée
```

Figure 5: Jeu de test des méthodes

7 Sources

- <https://www.schneier.com/academic/solitaire/fr/>
- https://fr.wikipedia.org/wiki/Bruce_Schneier
- [https://fr.wikipedia.org/wiki/Solitaire_\(chiffrement\)](https://fr.wikipedia.org/wiki/Solitaire_(chiffrement))