Rapport Sécurité Informatique

Ruddy Fantelli et Gregory Hubert

17 avril 2020

Résumé

A l'intérieur de ce rapport, vous trouverez toutes les informations concernant notre projet de sécurité informatique lié aux différentes méthodes de cryptographie.

1 Introduction

Dans ce cours, nous avons étudier différentes méthodes de cryptographie afin de sécuriser des messages entre des intercoluteurs. Notre projet consiste à la réalisation d'une application mobile permettant de cyrpter ou de decrypter des messages avec une des méthodes suivantes : Cesar, Atbash, Vigenere, Hill et DES.

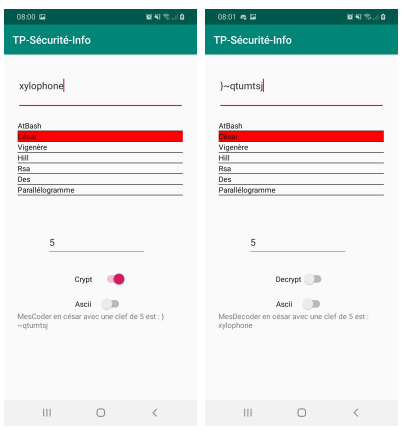
Vous trouverez l'intégralité du *Code Source* [3] sur Github. Des commentaires ont été laisser sur chacune des fonctions pour permettre une meilleur compréhension de celui-ci.

2 Detail du projet

Nous allons à présent vous présenter les différentes méthodes de cryptographies et de leur fonctionnement en terme de crytpage et de décryptage. Il faut savoir que chacune de ces méthodes ont été penser de manière à être facilement manipulée dans le code. C'est pourquoi chaque outil de cryptographie s'est vu être attribuer une classe dans laquelle est contenue 2 méthodes : crypt() et decrypt() servant chacune au chiffrement et déchiffrement de la méthode de cryptographie concernée. De plus, nous avons également la possibilité de choisir si le cryptage/décryptage se fera uniquement avec les lettres de l'alphabet ou avec l'ASCII étendu.

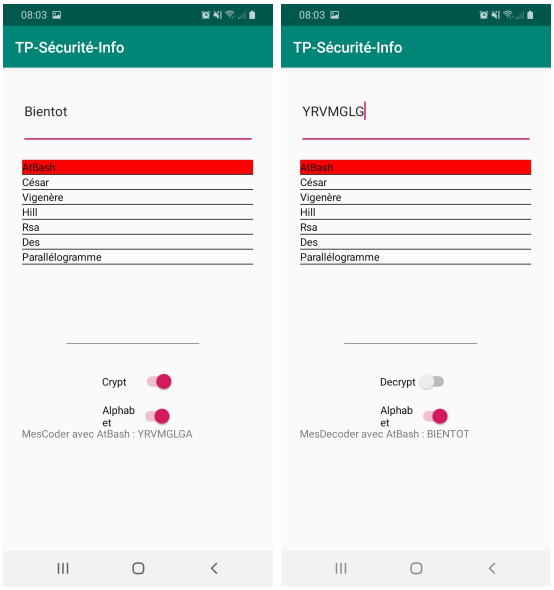
2.1 Cesar

Cesar est le premier chiffrement symétrique que nous avons implémenter dans l'application, c'est le plus simple à manipuler, il suffit simplement de décaler le caractère d'un certains nombre par rapport à sa position dans la table ASCII. Celui-ci affiche un résultat sous forme héxadécimal si le caractère pointé par ce résultat se trouve être impossible à écrire pour la machine.



2.2 Atbash

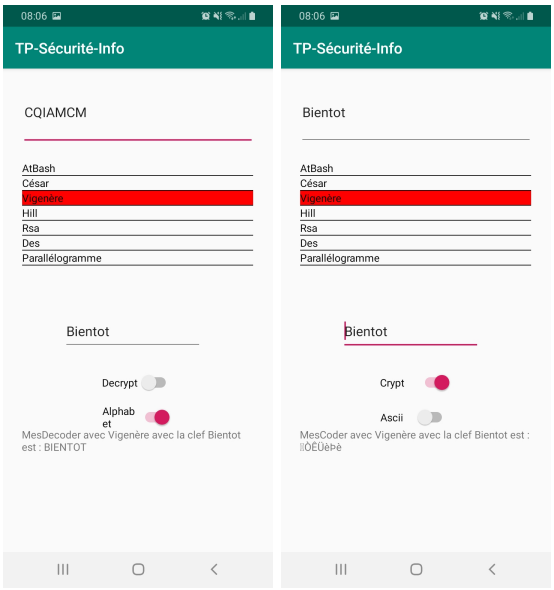
Atbash est également assez basique dans sa compréhension. Il suffit de renvoyer la position opposéé à chaque caractère qui est lu, si le chiffrement est appliqué à la table ASCII, on se retrouve simplement sur une plage de valeur allant jusqu'à 255 au lieu de 26 pour l'alphabet classique.



2.3 Vigenere

De même pour Vigenere, nous avons conservé le même schéma de logique que pour les précédentes classe, une méthode crypt() permettant d'aller du message en clair vers un message non lisible, et une méthode decrypt() permettant de faire le chemin inverse. La différence se notera au niveau du calcul qui, au lieu d'être positif (donc d'effectuer un décalage vers la droite), sera négatif (décalage vers la gauche).

Nous avons créer une fonction intermédiaire permettant de générer un String qui se compose tout seul du bon nombre de répétition de la clé, permettant ainsi de pouvoir calculer bien plus efficacement par la suite le message à chiffrer car ils auront la même longueur.



2.4 Hill

A partir de celui-ci, les choses ont commencer à être plus délicate. En effet, nous avons dû séparer le travail en plusieurs partie afin de rendre le cryptage et le décryptage de Hill plus digeste étant donné qu'il y a plusieurs étape dans son fonctionnement. Ainsi, nous obtenons une classe Hill() composé de plusieurs fonctions permettant de :

- Séparer le message en groupe de 2. (Ou plus ! Car la méthode permet de séparer un message par petit paquet de X lettres que l'on passe en paramètre)
- Savoir les positions ASCII (ou Alphabet) d'un groupe donné. C'est pratique pour avoir le position du groupe sans avoir à le calculer en plein milieu de l'algorithme (Par exemple le groupe "AB" donnera en retour [0,1] si il est effectué avec un Alphabet simple)
- Obtenir le déterminant
- Réaliser une combinaison linéaire
- Calculer la matrice inverse

Tout cela mis ensemble permet d'arriver au cryptage et decryptage de Hill avec plus d'aisance et de facilité, permettant à ces méthodes de ne tenir que sur quelques lignes seulement.

```
public String crypt(String message, int groupe, int a, int b, int c, int d) {
String final_message = "";

if (determinant(a,b,c,d)%2 == 0 && determinant(a,b,c,d)%13 == 0 )
return "Erreur: [Impossible de decrypter!]";

ArrayList<String> liste_groupe = separeMessage(message,groupe);
ArrayList<int[]> liste_rang = rangGroupe(liste_groupe);
ArrayList<int[]> liste_combinaison = new ArrayList<int[]>();

for(int i = 0; i < liste_rang.size(); i++) {
int[] rang = liste_rang.get(i);
int[] combinaison = combinaisonLineaire(rang[0],rang[1],a,b,c,d);
liste_combinaison.add(new int[] {combinaison[0]%256, combinaison[1]%256});
}
for (int i = 0; i < liste_combinaison.size(); i++) {
int[] combi = liste_combinaison.get(i);
final_message += ExtendedAscii.getChar(combi[0], 0) + "" + ExtendedAscii.getChar(combi[1], 0);
}
return final_message;
}
```

2.5 DES

C'est probablement celui qui aura monopoliser le plus de temps pour sa réalisation. Il y a énormément d'étapes à faire pour parvenir à un chiffrement à l'aide du DES. En suivant la même logique que les précédents, en terme d'utilisation, il suffit de lancer la méthode crypt() se trouvant dans sa classe. De plus, pleins de petites fonctions intermédiaire ont été créées pour facilité certaines tâches à effectué. Nous avons réaliser le DES en suivant l'explication détaillée de *bibmath* [1]. Cependant nous avons eu quelques souci de compréhension de certaines étapes qui n'était pas forcément très claires, et parfois se sont même les tableaux qui se sont avérés ne pas être identique à ceux présentés ailleurs. C'est pourquoi les tableaux de permutations et de substitutions ont été repris de *Wikipedia* [2] Une partie des fonctions intermédiaire consistent à aider à tracer le code, comme une méthode permettant d'afficher le contenu d'une liste, ou de transformer une liste en String. Une autre partie des fonctions intermédiaire traite de tout type de conversion nécessaire dans l'algorithme du DES : notement des conversions d'entier en binaire, d'héxadécimal en binaire, de calcul d'opération comme le XOR à également été refait à la main.

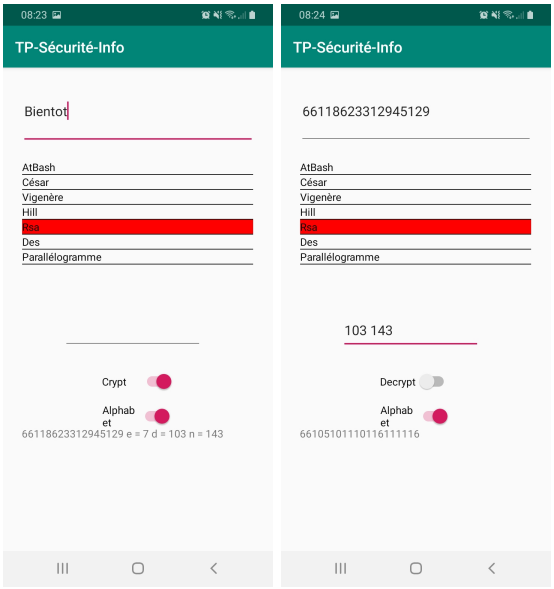
De plus, l'opération de décalage de bits à été fait de manière souple, permettant ainsi de réaliser un décalage de bit (avec retour de l'autre côté), de pouvoir choisir combien de bits à renvoyer au total et de choisir si le décalage se fait par la gauche ou par la droite.

```
    private long cyclicShift(long number, int offset, boolean leftSide, int nbTotalDigit) {
if (leftSide)
return (number << offset) | (number >> (nbTotalDigit - offset));
return (number >>> offset) | (number << (nbTotalDigit - offset));
}
```

2.6 RSA

Afin de faire le cryptage RSA, nous suivrons chaque étape vu dans le cours pour le faire. Dans un premier temps nous choisissons au hasard une valeur de p et q qui sont premiers. Puis nous calculons n en fonction des deux valeurs obtenues et de même pour phi(n). A partir des valeurs obtenus, nous cherchons e qui sera premier avec phi(n) qui lui sois strictement inférieur. Pour finir nous calculons d qui sera l'inverse de e modulo phi(n). Après toutes ces étapes, nous pouvons crypter le message M, grâce à la méthode crypt(), avec la formule vu en cours en fonction de e et n (Avec M mot traduit en ASCII de taille inférieur à n. S'il ne l'est pas on le découpe pour qu'il le sois.)

Le decryptage se fera alors de la même manière avec la formule decrypt() dans laquelle en paramètre nous donnerons les valeurs de d et n. La valeur final obtenu sera en ASCII et devra être convertie par la suite pour obtenir le message en clair.



2.7 Parallélocode

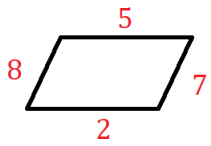
Enfin, terminons par celui que nous avons inventé : Le parallélocode !

Comme son nom l'indique, c'est une technique de cryptographie prenant en entrée le message en clair que l'on souhaite chiffrer, ainsi que 4 chiffres minimum (les chiffres supplémentaires ne seront pas pris en compte) que l'on fait passer dans un "paralélogramme fictif". En effet, chacun de ces chiffres représente une longueur de ce parallélogramme et chacune des lettres subit un tour complet du périmètre de ce parallèlogramme en subissant un chiffrement de Cesar positif ou négatif en atteignant chacune des arêtes.

Après avoir chiffré un caractère, afin d'éviter les décryptage via probabilité des lettres, nous rajoutons 4 caractères "poubelle" en utilisant la clef pour faire des calcul d'addition et multiplication sur le caractère originalement encodé. Pour le décodage on omet donc les caractère poubelles en ne prenant que les caractères en position modulo 5 et en faisant le chemin inverse.

Ce que nous avons trouvé intéréssant à travers cette méthode, c'est que contrairement aux autres, elle fait intervenir de la géométrie, chose qui n'est pas très commun si on le compare aux autres méthodes de chiffrements.

Bien entendu c'est un algorithme qui pourrait être étendu à n'importe quel autre polygone.



3 Conclusion

Pour conclure, nous pouvons dire que ce projet nous aura été extrêmement formateur sur les différentes notions de Sécurité Informatique.

Concernant le projet en lui-même, la partie la plus délicate, en dehors de l'algorithme du DES, aura été de traiter la transformation de l'ASCII étendu de Java dans notre ASCII étendu à nous, ce n'était vraiment pas une chose évidente, c'est pourquoi nous avons ajouté la feature de convertir uniquement avec l'alphabet également (cela nous permettait de vérifier en amont que nos algos sont corrects), On aurait certainement gagner plus de temps si on avait pas à se soucier de convertir une norme ASCII en une autre et de garder tout simplement l'ASCII que propose Java.

En tout cas, ce fut un réel plaisir de pouvoir plonger dans le coeur de toutes ces méthodes de chiffrements et déchiffrements, nous en avons acquis une certaine expérience et surtout une meilleure compréhension de ce domaine dont on n'en connaissait que la surface, et ça aura été un véritable plaisir de pouvoir manipuler tout cela directement pour de vrai.

4 Bibliographie

Références

[1] La saga du des. http://www.bibmath.net/crypto/index.php?action=affiche&quoi=moderne/des.

[2] Constantes du des. https://fr.wikipedia.org/wiki/Constantes_du_DES.

[3] Projet github. https://github.com/Gorgory24/TP-Securite-Info.