**Enigma Machine**

**Bac Informatique - UQO**

**Introduction à la cybersécurité**

An old typewriter with buttons

Description automatically generated

Loux14

**Introduction**

Enigma était une machine utilisée par les Allemands depuis 1930, largement rependue pour les communications militaires Nazis durant la Seconde Guerre Mondiale.

Sa cryptanalyse par les Alliés aurait permis d'avancer de 2 ans leur victoire et la fin de la Guerre, ce qui aurait épargné plusieurs dizaines de millions de vies.

Son fonctionnement repose sur un encryptage lettre par lettre, en pressant sur les touches du clavier. Le courant électrique généré passe par 3 rotors qui substituent chacun leur tour la lettre par une autre, puis atteint le réflecteur qui refait passer le courant par ses mêmes rotors pour enfin faire illuminer une lettre du clavier, qui est donc la lettre correspondante. A chaque touche du clavier pressée, le premier rotor tourne d'un cran. Le résultat est qu'une même lettre pressée 2 fois d'affilé donne 2 lettres différentes. Après 26 rotations, un rotor fait tourner d'un cran le rotor suivant, et se retrouve lui même sur sa position de départ.

Ce système, grâce au réflecteur, peut être utilisé pour encrypter un message ou le décrypter en utilisant la même position de départ des rotors.

La complexité de sa cryptanalyse repose sur le fait que les rotors sont amovibles, permettant des centaines de millions de combinaisons de départ. Les Nazis avaient une liste de combinaisons pré-établis pour chaque jour. Un même message encrypté est donc différent d'un jour à l'autre.

Dans notre cas, les réglages possibles seront de choisir l'ordre des 3 rotors, leur sens de rotation, ainsi que leur position de départ (décalage).

Le but du projet est de produire un logiciel simulant la machine Enigma, en étant capable de choisir la position (ordre) des 3 rotors, le sens de leur rotation, ainsi que leur décalage de départ (position de départ par rapport à une 'position zéro' du rotor). La machine devra être capable de crypter un message et de le décrypter en utilisant la même combinaison de réglages (Key), et devra afficher la position des rotors avant et après le message.

**Utilisation**

L'application est une application Java, à utiliser dans le terminal.

Il est nécessaire de compiler l'application afin de pouvoir l'utiliser avec la commande :

**" javac Enigma.java "**

Le programme devra être appelé avec 9 arguments comprenant les 3 rotors (**'R1'**, **'R2'**, **'R3'**), leur sens (**'D'** ou **'G'** pour droite et gauche) et leur décalage (**'1'**, **'2'**, **'-4'**, etc...) comme selon l'image fig.1 suivante.

fig.1

Dans cet exemple, le programme utilisera le Rotor 1 qui tourne à droite et dont la position de départ n'est pas décalée, puis le rotor 2 et le 3 dans les mêmes conditions.

Dans l'étape suivante, le programme fait une présentation des rotors selon leurs positions(fig.2).

On retrouve tout en bas l'alphabet puis en montant, le premier rotor utilisé, le second, le troisième, puis le réflecteur. Au-dessous, on trouve un rappel de la 'Key' utilisée, puis le programme nous invite à écrire le message à encrypter ou décrypter (aucune différenciation ne sera faite car le principe de la machine est que l'encryption et la décryption utilisent la même clé).

A screenshot of a computer

Description automatically generated

fig.2

A l'entrée de notre message, on verra le message encrypté, puis la nouvelle position des rotors après les rotations (fig.3).

A screenshot of a computer

Description automatically generated

fig.3

Dans cet exemple, on voit qu'à l'écriture de 'Lucas' (5 lettres), le rotor 1 de 5 cases vers la droite.

A ce point le programme s'arrête, et il est nécessaire de refournir une 'Key' pour ressaisir un nouveau message.

**Implémentation**

Chronologiquement, le code comprend une section de donnée statiques, puis la classe des rotors, la fonction compute() pour le calcul, les fonctions d'impression des tableaux puis le main().

***Les données statiques***

- R1, R2, R3 sont des tableaux à 2 niveaux (int[][]). Ils sont destinés à être copiés dans de nouveaux tableaux de la classe Rotors lors de la prise en compte de la Key en début de main().

La première ligne est utilisée pour le calcul lors du trajet allant vers le réflecteur, et la deuxième ligne sert au trajet retour. Les deux lignes sont donc complémentaires car elles représentent une connexion physique (fil électrique) dans la machine réelle.

- Reflector est un tableau à une ligne, qui est symétrique en son milieux, permettant la réversibilité de l'encryption.

- 2 HashMap permettant d'accéder a une lettre selon son indice et inversement.

***La classe Rotor***

La classe Rotor comprend principalement un constructeur permettant de dire quel Rotor statique on utilise, dans quel sens il tournera, ainsi que le décalage de départ.

Les Rotors possèdent un tableau a deux lignes (int[][]) qui sera utile pour le calcul dans la fonction Compute().

Ils possèdent également un 'count' permettant de suivre les rotations effectuées lors de l’écriture d'un message, un boolean précisant le sens de rotation, et un Rotor 'next' permettant la rotation en chaine dans la fonction de rotation.

Elle comprends enfin une fonction rotate() qui va recopier le tableau du rotor en le décalant d'une case, selon le sens spécifié. Plus important, elle vérifie si le 'count' du rotor est a 25 (tour complet), et si c'est le cas elle remet le compteur à zéro et demande une rotation du rotor suivant 'next'. Enfin, si le 'count n'est pas à 25, elle l'incrémente de 1.

***Fonction Compute()***

La fonction prend en entrée le message, ainsi que les 3 rotors (ordre des rotors).

Pour chaque lettre du message, elle récupère la valeur associée (0 a 25) via le HashMap, qui est l'index d'entrée du premier rotor.

On utilisera la première ligne de chaque tableau jusqu’à atteindre le réflecteur, puis le second niveau du tableaux (ce qui simule le trajet électrique).

Dans la case du premier rotor, à cet index, on trouve le décalage à appliquer pour connaitre l'index du rotor suivant. On applique ceci jusqu'au réflecteur puis au retour jusqu'à travers le rotor d'origine. Celui-ci nous fera décaler une dernière fois pour connaitre l'indice (0-25) de la lettre correspondante encryptée. Puis la fonction engendre une rotation 'rotate()' du premier rotor.

Enfin, elle imprime le message complet encrypté.

Mathématiquement, la fonction compute() utilise un 'count' contenant l'indice de départ, qu'elle va additionner avec le contenu du rotor suivant à cet indice (modulo 26).

Illustration simpliste : **count n+1 = count n + rotor n[count n]** (mod 26)

A diagram of a diagram

Description automatically generated with medium confidence

fig.4

Dans la figure 4, On peut comprendre comment la lettre L devient O en utilisant cette 'Key'.

***Fonctions d'impression***

Il y en a 3 : une permettant l'impression d'un tableau de chiffre a 2 lignes, une pour le tableau mon-ligne pour les chiffres, et une pour les lettres en tableau mono-ligne.

***Fonction main()***

Chronologiquement, la fonction main() va créer les 3 Rotors en utilisant les arguments 3 par 3, permettant de savoir quel est le rotor à chaque position, son sens de décalage et sa position de départ. Elle va ensuite préciser le rotor suivant 'next' de chacun (le 'next' du dernier est le premier).

Dans un deuxième temps, elle va imprimer à l'écran les différents rotors pour avoir un aperçu de la machine, et inviter l'utilisateur à écrire un message.

Une fois le message écrit, la fonction compute() est appelée, puis les rotors sont à nouveau imprimés.

**Conclusion**

L’implémentation est faite en fichier unique sans classe externe, pour la facilité d'utilisation et pour la lecture intuitive car le fichier n'est que de 400 lignes.

L'utilisation de plusieurs HashMap et plusieurs tableaux augmente la quantité mémoire utilisée (très peu car peu de données) mais permet ensuite une efficacité de lecture.

Le tout rend l'application très accessible et intuitive.

Quelques améliorations seraient à prévoir, comme une interface graphique (hors de mon champs de compétence actuel), une gestion plus élargie de classe externe pour permettre de personnaliser la machine en nombre ou en taille de rotor par exemple, ou encore en implémentant les cables servant de 'switch' entre 2 lettres, utilisé dans la vraie machine.