ESIREM
ÉCOLE SUPÉRIEURE D'INGÉNIEURS
NUMÉRIQUE ET MATÉRIAUX

Programmation C++
Compte Rendu TP n°4

Martinez Roméo
Septier Aubin

27 janvier 2023

Dans ce TP, nous avons abordé la programmation orientée objets (POO) en C++ en créant différentes classes et en utilisant l'héritage afin de chiffrer des messages à l'aide de techniques de cryptage.

Rapport créé avec la template latex ESIREM.

Table des matières

1 Introduction

Pour ce second TP, nous avons décidé de travailler sur le TP4 qui consistait à implémenter des algorithmes de chiffrement de données en C++ (dont l'énoncé est disponible sur : GitHub). Nous avons choisi ce TP car après avoir fait le TP1 nous permettant de nous familiariser et d'apprendre à maîtriser les bases du C++, nous avons voulu prendre un sujet plus complexe, traitant de fonctionnalités plus poussées du langage comme l'Héritage. Pour le TP, nous avons également utilisé l'aide de cryptographie disponible avec les TP.

Comme pour le premier TP, l'ensemble de notre projet est disponible sur GitHub. Nous avons de nouveau utilisé l'environnement de développement intégré Visual Studio Code et son extension C/C++.

Dans ce deuxième TP, nous avons poursuivi notre exploration du C++ en nous concentrant sur les techniques de cryptage. L'objectif était de créer une application de chiffrement de message en utilisant différentes méthodes de cryptage telles que le chiffrement de Caesar et de Vigenere. Nous avons commencé par créer des classes de base pour le chiffrement, comme *BasicEncrypt* et *Encrypt*, qui ont servi de fondation pour les méthodes de chiffrement spécifiques. Ces classes comprenaient des méthodes pour encoder et décoder un message, ainsi que des méthodes pour récupérer et définir le message en clair et le message chiffré.

Nous avons ensuite développé des classes spécifiques pour le chiffrement de Caesar, telles que *Caesar* et *Caesar2*, ainsi que pour le chiffrement de Vigenere, *Vigenere*. Ces classes héritaient de la classe de base *Encrypt* et implémentaient des méthodes de chiffrement spécifiques pour chacun des algorithmes. Par exemple, la classe *Caesar* utilisait une méthode de décalage pour chiffrer le message, tandis que la classe *Vigenere* utilisait une méthode de substitution en utilisant une clé de chiffrement.

2 Main

Le code main présenté ici effectue des tests sur divers algorithmes de chiffrement, notamment les classes *BasicEncrypt*, *Encrypt*, *Caesar2* et *Vigenere*. Il utilise également des fonctions pour lire et écrire des fichiers pour stocker les messages cryptés et décryptés.

Tout d'abord, il teste les différentes classes pour encoder et décodere un message. Il teste aussi les fonctions qui permettent d'écrire dans des fichiers.

De plus nous avons décidé à l'instar du TP1 de réaliser un affichage graphique plus attrayant que de simples lignes dans le terminal.

En somme, ce code main permet de tester différentes méthodes de chiffrement en utilisant différents algorithmes, en utilisant les classes présentées juste après, et de montrer comment utiliser des fonctions pour lire et écrire des fichiers pour stocker les messages cryptés et décryptés.

3 Classes basiques de Cryptage

3.1 Classe *BasicEncrypt*

La classe *BasicEncrypt* est une base extrêmement simple possédant des méthodes permettant de "coder" un message en inscrivant la variable non encodée (plain) dans la variable encodée (cypher) et permettant d'afficher ces deux variables. Elle illustre le principe de base du chiffrement.

3.2 Classe *Encrypt*

La classe*Encrypt* est un des éléments clés de notre projet de cryptage. Elle permet de chiffrer et déchiffrer des messages en utilisant une méthode virtuelle. Elle contient des méthodes pour obtenir les différentes versions du message, à savoir la version originale (plain), la version chiffrée (cypher) et la version actuelle (message). Ce qui est intéressant dans cette classe, c'est qu'elle contient également des méthodes pour lire, stocker et écrire des messages dans des fichiers. Cela permet de gérer facilement les entrées et sorties de notre programme.

4 Chiffrement Caesar

4.1 Classe *Caesar*

La première classe *Caesar*, qui hérite de la classe *Encrypt*, permet de réaliser le chiffrement César c'est-à-dire chiffrer un message en minuscules en décalant d'un certain nombre de lettres celles composant le message. La méthode permettant d'encoder le message utilise deux alphabets, qui sont des chaînes de caractères afin de réaliser le décalage et de chiffrer le message à l'aide de boucle for.

Pour décoder le message, nous utilisons tout simplement de nouveau deux alphabets. Mais cette fois-ci, le second est à l'envers afin de reconstituer le message comme il l'était à l'origine. La variable message est de nouveau utilisée pour stocker l'état actuel du message

à chiffre/déchiffrer. Ainsi la variable plain sera toujours le message d'origine et cypher sera toujours le message encodé et la variable message sera le mot au moment où on l'affiche qu'il soit encodé ou décodé. Le chiffrement Caesar permet de crypter et décrypter aussi bien de simples mots comme des phrases.

4.2 Classe *Caesar2*

La seconde classe *Caesar2* fonctionne de la même façon que la première classe *Caesar*, héritant aussi de la classe *Encrypt*, à la seule exception que les alphabets, que l'on utilise afin de chiffrer et déchiffrer les messages, ne comportent cette fois-ci pas uniquement l'alphabet minuscule comme c'est le cas dans le chiffrement Caesar de base. En effet, cette fois-ci, les alphabets comportent l'intégralité des caractères ASCII (hors caractères de contrôle). Le chiffrement et le déchiffrement fonctionnent donc de la même façon à l'aide de deux alphabets qui sont dans l'ordre pour encoder et un qui est dans l'ordre et un dans le sens inverse pour décoder. Comment le premier chiffrement Caesar, cette version améliorée peut également crypter et décrypter des mots comme des textes.

5 Chiffrement Vigenere

5.1 Classe *Vigenere*

La classe *Vigenere* est une classe dérivée de la classe *Encrypt*, qui permet de chiffrer et de déchiffrer des messages en utilisant la méthode de chiffrement de *Vigenere*. Cette méthode repose sur l'utilisation d'une clé qui est utilisée pour décaler les lettres d'un message pour obtenir le message chiffré.

Elle possède aussi deux constructeurs : l'un prend en entrée le message à chiffrer et une clé sous forme de vecteur d'entiers, l'autre prend le message et une clé sous forme de chaîne de caractères. Cela permet de faciliter l'utilisation de la classe en fonction des besoins de l'utilisateur.

De plus, elle surcharge les méthodes encode() et decode() de la classe Encrypt pour implémenter la méthode de chiffrement de Vigenere. Elle possède également une méthode getKey() qui permet de récupérer la clé utilisée pour chiffrer le message.

La classe *Vigenere* possède également un attribut privé, `key`, qui stocke la clé utilisée pour chiffrer le message. Cette clé est utilisée par les méthodes encode() et decode() pour effectuer le chiffrement et le déchiffrement du message.

En utilisant cette classe, les utilisateurs peuvent chiffrer et déchiffrer des messages de manière sécurisée en utilisant la méthode de *Vigenere*. Cela permet de protéger des données sensibles contre les attaques de type "brute force" ou "analyse de fréquences" qui sont souvent utilisées pour craquer les chiffrements simples.

6 Conclusion

Au cours de ce TP, nous avons pu enrichir et construire nos connaissances en cryptographie en utilisant les concepts de base tels que la substitution et la transposition. Nous avons commencé par créer des classes de base telles que BasicEncrypt et Encrypt qui nous ont permis de comprendre les fondements de ces concepts. Nous avons ensuite utilisé ces classes pour implémenter des algorithmes de chiffrement tels que le chiffrement de Caesar et le chiffrement de Vigenere. Nous avons cependant rencontré des difficultés lors de l'implémentation de la classe Vigenere, en particulier pour gérer les clefs de chiffrement. Cependant, nous avons réussi à surmonter ces difficultés grâce à une analyse minutieuse de l'algorithme et à des tests répétés.