GRENOBLE INP UGA Esisar VALENCE

CS312

Mini-Projet Java
Cryptographie

2023 – 2024

> *Alice et Bob ont besoin de vous !*
> *Ils veulent communiquer secrètement mais ils ont peur qu'Ève les espionne. Vous devez leur fournir un programme pour qu'ils puissent protéger leurs communications.*

Gestion des arguments du programme

Le programme doit pouvoir s'utiliser avec les arguments suivants :

```
-e|-d alg text [args ...]
```

Code 1: Arguments d'utilisation du programme.

Avec :

- `-e` signifiant de chiffrer le `text` avec l'algorithme `alg`;
- `-d` signifiant de déchiffrer le `text` avec l'algorithme `alg`.

Le résultat est simplement affiché dans la sortie standard.

Si les arguments ne sont pas passés correctement, le programme doit afficher un message reprenant ces informations.

Si l'`alg` passé ne correspond à aucun des algorithmes décrits ci-dessous, vous devez lever une exception `InvalidAlg`.

Pour certains des algorithmes, il faudra prendre une clef en argument, si cette clef n'est pas valide (en fonction de l'algorithme), vous devez lever une exception `InvalidKey`.

Création d'un programme Java

> *Alice et Bob ne veulent pas s'embêter avec Eclipse (VSCode) pour exécuter votre programme. Ils veulent pour l'utiliser comme tout programme : avec la ligne de commande.*

Le programme doit être fourni de façon à ce qu'il soit utilisable comme un exécutable, non pas comme une archive de fichiers source, ni comme une archive de fichier `.class`. Pour cela, vous utiliserez le format `.jar`.

Si vous utilisez un IDE (comme Eclipse) référez-vous à sa documentation pour créer un `.jar` et lui passer des paramètres.

Pour l'utilisation dans un shell, suivez les étapes (à la racine du « projet » Eclipse) :

1. Compilez les sources vers un répertoire existant :
   `javac -cp src -d build/ src/`
   avec
   - `javac` le compilateur,
   - `-cp src` un argument indiquant la base du package,

- `-d build` la destination (attention, `build` doit exister),
- `src/` la racine de l'arborescence de votre code.

2. Créer le fichier `.jar` :
   `jar -v -c -e fr.esisar.cs312.prjcrypto.Main -f Main.jar -C build .`
   avec
   - `jar` l'outil de création de création de `.jar`,
   - `-v` pour passer un mode verbeux,
   - `-c` pour indiquer de créer le fichier,
   - `-e fr.esisar.cs312.prjcrypto.Main` pour indiquer le nom de la classe possédant la méthode `main` à exécuter),
   - `-f PrjCrypto.jar` le nom du `.jar`,
   - `-C build` pour indiquer la racine du l'archive,
   - `.` pour indiquer de s'exécuter ici.
3. Exécuter le `.jar` :
   `java -jar PrjCrypto.jar ...`
   avec
   - `java` l'exécutable Java,
   - `-jar PrjCrypto.jar` l'option indiquant d'exécuter en `.jar` et le chemin vers celui-ci,
   - `...` les arguments à passer à la méthode `main`.

   Attention: Ne supprimez pas d'options ou d'arguments sans avoir compris la raison de leur présence avant !

Vous pouvez en faire un script shell :

```
javac -cp src -d build/ src/
jar -v -c -e fr.esisar.cs312.prjcrypto.Main -f Main.jar -C build .
java -jar Main.jar $@
```

Code 2: Script shell compilant, créant le `.jar` et passant les arguments ($@ correspond aux arguments passés au script lui-même).

Organisation du code

> *Comme tout bon clients, Alice et Bob ne seront jamais satisfaits, et vous devez vous préparer à ajouter de nouveaux algorithmes sur demande.*

Vous avez plusieurs algorithmes à implémenter, utilisez les principes de développement orienté objets pour organiser votre code au mieux : il doit être trivial d'ajouter ou supprimer un algorithme !

Par exemple, vous pouvez créer une classe par algorithme, et les faire implémenter une interface `Encrypter` de façon à pouvoir les utiliser de manière interchangeable.

Les algorithmes à supporter

> *Alice et Bob sont très exigeants, ils vous fournissent une liste d'algorithmes à implémenter obligatoirement.*

Pour chacun des algorithmes, vous devez fournir l'algorithme de chiffrement (`-e`) et de déchiffrement (`-d`) !

Cas particulier du code de César : ROT13

Vous devez commencer par un cas particulier des codes de César [1], ROT13 [2].
Le nom de l'algorithme, pour la ligne de commande, devra être `rot13`.

Codes de César

Fort de l'expérience d'implémentation de ROT13, vous devez implémenter l'ensemble des codes de César.
Le nom de l'algorithme, pour la ligne de commande, devra être `caesar`.
Remarquez que vous avez maintenant besoin de passer un argument supplémentaire : la clef de rotation.

Code de substitution

Pour une meilleure sécurité, vous implémentez les code du substitution [3].
Le nom de l'algorithme, pour la ligne de commande, devra être `sub`.
La clef pour cet algorithme sera une liste de 26×2 caractères, avec le premier caractères devant substituer la lettre 'a', le seconde la lettre 'b', le vingt-septième la lettre 'A', etc.
Bonus: Ajoutez le support des lettres à accents.

Chiffre de Vernam

Pour une sécurité absolue, vous implémentez le Chiffre de Vernam [4].
Le nom de l'algorithme, pour la ligne de commande, devra être `otp`.
Attention: Attention, la clef doit être au moins aussi longue que le message à chiffrer !

D'autres algorithmes

Bonus: Implémentez ce que vous voulez !
Mais avant ça, assurez-vous que votre code est bien organisé, clair, concis, correctement commenté, et complètement testé !

Références

[1] WIKIPEDIA. *Caesar cipher*. 2024. URL : https://en.wikipedia.org/wiki/Caesar_cipher (visité le 31/03/2024).

[2] WIKIPEDIA. *ROT13*. 2024. URL : https://fr.wikipedia.org/wiki/ROT13 (visité le 31/03/2024).

[3] WIKIPEDIA. *Substitution cipher*. 2024. URL : https://en.wikipedia.org/wiki/Substitution_cipher (visité le 31/03/2024).

[4] WIKIPEDIA. *One-time pad*. 2024. URL : https://en.wikipedia.org/wiki/One-time_pad (visité le 31/03/2024).