



Le génie pour l'industrie

INF111-A19-T2-PARTIE-I

CRYPTOGRAPHIE ET CRYPTANALYSE

Auteur : Mathieu Nayrolles

Révision et mise en page : Pierre Bélisle

INTRODUCTION

Lors de ce second travail pratique, nous allons aborder des notions de cryptographie (partie 1) et cryptanalyse (partie 2) simple.

La cryptographie est une méthode qui permet de protéger l'information et les communications grâce à l'utilisation de code qui est possédé uniquement par ceux à qui l'information et les communications sont destinées.

En informatique, la cryptologie est une science pour sécuriser l'information et les communications grâce à des concepts mathématiques et des règles de transformations qui rendent l'information relativement difficile à décoder pour toute personne n'ayant pas le code. Elle est séparée en deux sous-domaines : la cryptographie et la cryptanalyse. La cryptographie chiffre le message tandis que la cryptanalyse vise à casser le cryptage (c.-à-d. décoder le message) sans avoir le code.

La cryptographie est présente dans l'intégralité des domaines informatiques et vous y êtes confrontés chaque jour via les sites internet sécurisés, le RPV (VPN) de l'ÉTS, les protocoles d'impression et de bureau distant ...

Au cours de ce travail pratique, nous allons implémenter des algorithmes simples de cryptographie et, dans un second temps, essayer de les décoder sans avoir le mot de passe secret en utilisant la cryptanalyse.

ARCHITECTURE

DÉFINITION D'ALGORITHMES

Les algorithmes que nous allons implémenter sont des algorithmes de cryptage et de décryptage. De ce fait, ils auront tous la méthode `public String encode(String)` et la méthode `public String decode(String)`. Afin de s'assurer que les algorithmes respectent tous ce contrat, nous créons l'interface Crypto suivante :

```
public interface Crypto {  
  
    public String decode(String message);  
    public String encode(String message);  
}
```

Cette interface définit le contrat que les algorithmes que nous allons implémenter doivent respecter. Pour implémenter une interface, il suffit d'ajouter `implements` `Crypto` à la fin de la déclaration de la classe, comme ceci :

```
public class ExempleAlgo implements Crypto {}
```

Comme vous pouvez le voir, `ExempleAlgo` est souligné car, présentement, la classe ne respecte pas le contrat Crypto. Les méthodes encode et decode ne sont pas implémentées. Voici une implémentation valide :

```
public class ExempleAlgo implements Crypto {  
  
    public String decode(String message) {  
        return message;  
    }  
  
    public String encode(String message) {  
        return message;  
    }  
}
```

Ici, notre algorithme se contente de retourner le texte qu'il reçoit sans le transformer. Le programme principal suivant montre que notre premier algorithme n'est pas vraiment sécurisé...

```
public class DemarrerCrypto {  
    public static void main(String[] args) {  
        ExempleAlgo algo = new ExempleAlgo();  
        String resultat = algo.encode("MONTEXTSECRET");  
        System.out.println(resultat);  
        resultat = algo.decode(resultat);  
        System.out.println(resultat);  
    }  
}
```

Voici le résultat de l'exécution :

- MONTEXTSECRET
- MONTEXTSECRET

Clé

La classe Clé permet de générer des clés aléatoirement pour crypter nos phrases. La classe contient un tableau **static** qui représente l'alphabet :

`public static char[] alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();`. Elle ne contient que 2 méthodes **static** :

- 1) Une méthode qui génère aléatoirement une clé de taille *nbCars* en utilisant l'alphabet. L'implémentation de cette méthode est laissée libre.
 - `private static char[] obtenirCleCar(int nbCars)`
- 2) Une méthode qui génère une clé numérique qui est au maximum *max*. L'implémentation de cette méthode est laissée libre.
 - `private static int obtenirCleNum(int max)`

***L'utilisation de la clé vous est décrite plus loin.

Serveur

Pour servir nos phrases encodées à nos utilisateurs, nous allons implémenter une classe nommée **Serveur**. Ci-dessous, vous trouverez la spécification de ses différentes méthodes :

Attribut :

Chaque ligne à traiter est ajoutée à un attribut du serveur, de la classe `ArrayList<String>`. D'autres attributs viendront s'ajouter.

Constructeur

Le constructeur de la classe **Serveur**, lit le fichier texte dont le nom est fourni en paramètre et l'ajoute, une ligne à la fois, dans le `ArrayList`. Pour vos tests, vous pouvez utiliser le fichier attaché au TP dont voici les dix premières lignes du fichier :

```
Bernardoclimbedthestairstothe castles ramparts
It was a bitter ly cold night
Hemad e his way carefully through the freezing fog to relieve
Francisco of his guard duty
Hes awa dim figure and challenged him
Who's there
```

*NoyouanswermeItwasFranciscosvoiceStopandidentifyyourself
 BernardostoppedLonglivetheKing
 Bernardo
 YesItsme
 FranciscorelaxedYouarerightontime
 ItsjustgonemidnightsaidBernardoGetofftobedFranciso
 ThankGodFranciscopreparedtoleaveItsfreezingandImdeadbored
 Hasitbeenquiet*







Le fichier contient 70 lignes tirées des œuvres de Shakespeare. Les lignes sont dépourvues de toute ponctuation ou espace pour rendre notre travail plus facile lors de l'implémentation de nos algorithmes.

Pour lire un fichier ligne par ligne, vous pouvez utiliser un `BufferedReader` dont voici un exemple d'utilisation :

```
File file = new File("./text.txt");
BufferedReader br = new BufferedReader(new FileReader(file));

String st;
while ((st = br.readLine()) != null) {
    System.out.println(st);
}
br.close();
```

Le fichier text.txt fourni avec le TP doit être placé au même niveau que le dossier **src** de votre projet Eclipse :

	.settings	2019-10-06 6:10 PM	File folder	
	bin	2019-10-10 9:26 AM	File folder	
	src	2019-10-10 7:24 AM	File folder	
	.classpath	2019-10-06 6:10 PM	CLASSPATH File	1 KB
	.project	2019-10-06 6:10 PM	PROJECT File	1 KB
	text.txt	2019-10-10 7:07 AM	Text Document	7 KB

Chargement du message et choix de l'algorithme

Cette méthode de la classe **Serveur** effectue plusieurs opérations. Premièrement, un des algorithmes décrits plus bas est choisi aléatoirement. Ensuite, les phrases contenues dans la ArrayList de phrases décodées sont encodées avec l'algorithme choisi et retourné à l'appelant dans une autre ArrayList.

```
public ArrayList<String> obtenirFichierEncode() ;
```

De plus, l'algorithme choisi doit avoir une clé de chiffrement. Cette clé de chiffrement doit être donnée à l'algorithme choisi via son constructeur. Cette clé peut être numérique ou alphabétique et doit être générée aléatoirement. Les

mécaniques de génération de clés sont dans la classe Cle décrite avant. Un appel à la classe Cle.obtenirCleCar() ou à Cle.obtenirCleNum() devra être fait dans la méthode obtenirFichierEncode().

La clé choisie doit être sauvegardée dans un membre de la classe serveur afin de pouvoir valider les tentatives de décryptage qui seront tentées via la cryptanalyse. La méthode soumettrePhrase valide que la phrase est bien celle à l'indice dans l'ArrayList de phrases décodées.

➤ `public boolean soumettrePhrase(String phrase, int indice)`

La méthode soumettreCle valide que la clé soumise est celle utilisée par les algorithmes utilisant des clés alphabétiques.

➤ `public boolean soumettreCle(String cle)`

La méthode soumettreCle surchargée, avec un int comme paramètre, valide que la clé soumise est celle utilisée par les algorithmes utilisant des clés numériques.

➤ `public boolean soumettreCle(int cle)`

ALGORITHMES DE CRYPTAGE

Les algorithmes spécifiés dans cette section implémentent tous l'interface Crypto et dispose donc tous des méthodes encode et decode. Un fichier qui contient les phrases encodées par algorithme est fourni avec le TP pour vous aider à valider votre implémentation. **Toutes les manipulations de chaînes de caractères doivent être réalisées via des vecteurs (Vector) de caractères.** Voici les bases de l'utilisation d'un vecteur de caractères en java :

```
Vector<Character> v = new Vector<Character>();  
v.add('a');  
v.add('b');  
v.add(0, 'c');  
System.out.println(v.get(0)); // imprime c  
System.out.println(v.get(1)); // imprime a  
System.out.println(v.get(2)); // imprime b
```

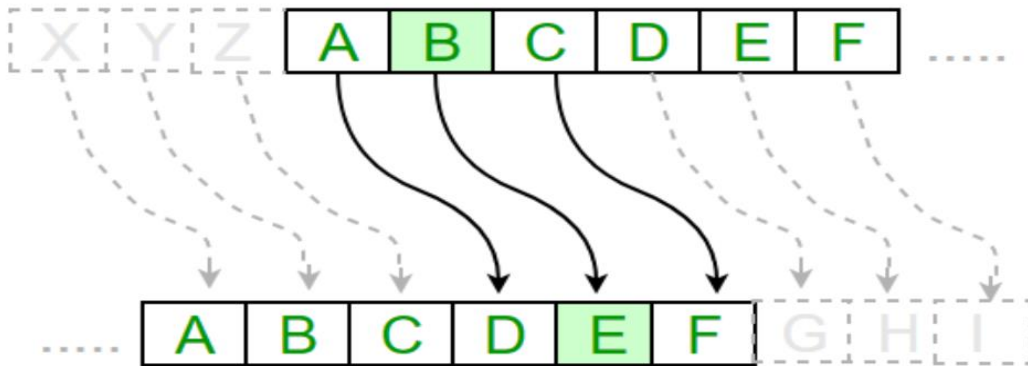
Premièrement un vecteur de caractères est instancié. Ensuite, nous utilisons la méthode add qui ajoute un caractère à la fin du vecteur de caractères. À ce moment-là, le vecteur contient uniquement la lettre a. Ensuite, nous invoquons à nouveau la méthode add avec 'b' comme argument. Le vecteur contient maintenant

les lettres 'a' et 'b'. Finalement, nous utilisons la méthode `add` avec deux arguments : un `int` et un caractère. L'entier, dans ce cas-ci 0, indique l'indice auquel nous souhaitons ajouter notre caractère. Les caractères se trouvant déjà dans le vecteur se déplacent sur la droite pour accommoder la nouvelle lettre. Ainsi notre vecteur contient les lettres 'c', 'a' et 'b'. Grâce à la méthode `get(int)`, nous pouvons récupérer le caractère qui se trouve à un indice donné. La spécification complète de la classe `vector` peut être consultée ici :

<https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html>.

Algorithme decaleADroite

Le premier algorithme, nommé `decaleADroite` substitue les lettres de la phrase décodée par une lettre que se trouve à `n`-place plus loin dans l'alphabet.



Dans l'exemple ci-dessus, les lettres sont substituées par les lettres qui sont 4 index plus loin dans l'alphabet. La clé est donc 4. Avec une clé de 4, la phrase suivante : *JECODEENJAVA* sera transformée en *NIGSHIIRNEZE*.

Afin de décoder *NIGSHIIRNEZE* dans sa version originale (*JECODEENJAVA*), l'algorithme doit être inversé.

Algorithme de decaleEnColonne

L'algorithme décaleEnColonne est une variante plus complexe de l'algorithme decaleAdroite. Pour chaque lettre à encoder, on sélectionne la colonne correspondante et pour une lettre de la clé on sélectionne la ligne correspondante.

	Lettre en clair																									
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Lettre de la clé	Lettres chiffrées (au croisement de la colonne <i>Lettre en clair</i> et de la ligne <i>Lettre de la clé</i>)																									
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Au croisement de la ligne et de la colonne on trouve la lettre chiffrée. La lettre de la clé est à prendre dans l'ordre dans laquelle elle se présente et on répète la clé en boucle autant que nécessaire.

Avec la table précédente, où la clé utilisée est l'alphabet, dans l'ordre, la version encodée de la lettre C serait E. Lorsque la clé n'est pas aussi longue que l'alphabet, alors on la répète autant de fois que nécessaire. Plus précisément, voici les opérations mathématiques qui permettent d'obtenir les versions encodées et décodées d'une lettre à la position i :

- $\text{msgEncode}[i] = \text{message}[i] + \text{cleCar}[\text{i modulo cleCar.length}]$
- $\text{msgDecode}[i] = \text{msgEncode}[i] - \text{cleCar}[\text{i modulo cleCar.length}]$

Algorithme du dictionnaire

L'algorithme du dictionnaire remplace les lettres de la phase décodée par en fonction d'une clé. Par exemple, avec l'alphabet suivant : 'Y', 'B', 'C', 'D', 'O', 'F', 'G', 'H', 'E', 'J', 'K', 'L', 'M', 'N', 'U', 'P', 'Q', 'R', 'S', 'T', 'A', 'V', 'W', 'X', 'I', 'Z' la lettre A sera remplacée par Y, la lettre B par B, la lettre C par C, la lettre D par D, la lettre E par O et ainsi de suite.

La phrase JECODEENJAVA deviendra JOCUDOONJYVY.

Algorithme du ou-logic

L'algorithme du ou-logic applique un «ou logique» entre la lettre à encoder et la valeur qui se trouve à la position $i \% \text{longueur de la clé}$. L'opérateur pour le ou logique en Java est \wedge . Ainsi, pour encoder la lettre d'un message à la position i , nous devons écrire : `message.charAt(i) ^ cle.charAt(i%cle.length())`

où message est la chaîne que nous souhaitons encoder, cle est notre clé et i l'indice de la lettre à encoder. La méthode `charAt` permet d'obtenir le caractère à une position donnée d'une chaîne.

LIVRABLES PARTIE I

Pour cette première partie, vous devez :

- Implémenter la classe Serveur
- Implémenter l'interface Crypto
- Implémenter l'algorithme decaleADroite
- Implémenter l'algorithme du dictionnaire
- Implémenter l'algorithme du ou-logic
- Implémenter l'algorithme decaleEnColonne

Utiliser le programme principal suivant pour tester votre solution :

```
public class DemmarerCrrypto {  
    public static void main(String[] args) throws IOException {  
  
        Server s = new Serveur(("Shakespeare.txt");  
  
        for (String string : s.obtenirFichierEncode()) {  
            System.out.println(string);  
        }  
    }  
}
```

Le résultat attendu est une suite de chaînes de caractères encodées. À chaque exécution, la suite change, car l'algorithme est sélectionné aléatoirement. Par exemple, les premières lignes de la première exécution sont :

```
KnawjamxLurvknmcqncjrabxcqnljbcunbajvyjacbRcfjbjkrccnauhLxumwrpqcQnvjmnqrbfj  
hLjanoduuhcqaxdpqcqnoannirwpoxpcxanurnenOajwLrbLxxoqrbpdjammdchQnbjffjmrvorpdan  
jwmlqjuunwpmqrv  
Fqxbcqnan  
WxhxdjwbfnavnRcfjbOajwLrbLxbexrLnBcxyjwmrmnwcrohhxdabnuo  
KnawjamxbcxynmUxwpurencqnTrwp  
Knawjamx  
HnbRcbvn  
OajwLrbLxanujg
```

Et la seconde exécution donne

```
JEHFVCILBSGSNXZUKYCHJZVHV(BHUUVDYIDZPGYIWSWMSHFRWPD<BTUJGJHLKKLOSAPIHGKRNMYHY4  
GCQJZQZIKFRNDHQHKNRSOITB<VGVBETODSGKHXBSDHMBTSDH;QSWMVCIATAWNQLWXDXSAOZKJT8  
INTUCLQIDUEKPAEN  
EHEKOSJOD  
VOOGPLSPVLPSQBPXDMPFJEGXU6WSLGDNJPSVNGZWEHHDWOPCDW+AEBX  
JEHFVCILRAMVBXZMRHQZRMIIJ8SIDY  
JEHFVCIL  
GEIAODRB
```

PARTIE II

Lors de la seconde partie, nous nous attacherons à « casser » les cryptages que nous avons créés. Un second document explicatif vous sera délivré à la remise de la partie I.

Les détails et la date de remise partielle de la partie 1 vous seront annoncés par votre enseignant.

Vous ne pourrez obtenir la partie II que si la partie I a été rendue et jugée acceptable tenant compte du barème de correction suivant.

BARÈME DE CORRECTION

Exécution 40%

- Doit respecter le comportement décrit dans l'énoncé.

Qualité de programmation 60%

- Erreurs pénalisantes concernant le respect des normes suivantes :
 1. Identificateurs de fonction ou de procédure (nom + verbe) ne respectent pas la norme.
 2. Identificateurs non significatifs (variables, constantes, sous-programmes,...).
 3. Aération et/ou indentation et/ou impression laissent à désirer (**80 colonnes max**).
 4. Découpage en sous programmes insuffisants.
 5. Répétition inutile de code dû au manque de sous programmes.
 6. Répétition inutile de code dû à l'incompréhension de l'utilité du paramétrage.
 7. Constantes non définies.
 8. Constantes non utilisées lorsque possible (même dans les commentaires).
 9. Constantes en minuscules.
 10. Commentaire d'en-tête de programme manquant (explication, auteurs et version).
 11. Commentaires des constantes manquants.
 12. Commentaires des variables manquants.
 13. Commentaires de spécification des sous-programmes manquants (API).
 14. Commentaires non judicieux ou inutile.
 15. Commentaires manquants sur la stratégie employée dans chaque sous-programme dont l'algorithme n'est pas évident et nécessite réflexion (comment).
 16. Commentaires mal disposés.
 17. Non utilisation d'un sous-programme lorsque c'est possible ou exigé.
 18. Code inutile.
 19. Affichage dans une fonction de calcul.
 20. Qualité du français dans les commentaires.
 21. Non-respect des droits d'auteurs.
 22. Non utilisation de boucle lorsque possible.
 23. Non utilisation de switch-case lorsque possible.
 24. Non utilisation de l'opérateur ternaire lorsque possible.
 25. Autres (s'il y a une pratique non énumérée qui nuit à la bonne compréhension du code).

Bon travail!