Chiffrement symétrique
Christophe Viroulaud
Terminale NSI

Chiffrement symétrique

Christophe Viroulaud

Terminale NSI

Protocole TCP/IP

La communication sur internet est organisée en couches.

Couche application (Navigateur)
Couche TCP (Transport)
Couche IP (Internet)
Couche réseau (Matérielle)

1. calqué sur le modèle OSI (Open Systems Interconnection)
2. Réseau : Définit la forme dont les données sont physiquement transmises quelque soit le réseau (onde, impulsion électrique, lumière)
3. IP : Gère les chemins possibles à travers le réseau et achemine le message de l'expéditeur au destinataire.
4. TCP : S'assure de la bonne transmission des données. Découpe les messages en paquets.
5. Application : Protocole haut-niveau (http, imap…)

Protocole TCP/IP

La communication sur internet est organisée en couches.

Couche application (Navigateur)
Couche TCP (Transport)
Couche IP (Internet)
Couche réseau (Matérielle)

En théorie, rien n'interdit à un routeur d'inspecter un paquet et donc d'en connaître son contenu.
Comment chiffrer le contenu des communications ?

En théorie, rien n'interdit à un routeur d'inspecter un paquet et donc d'en connaître son contenu.

Comment chiffrer le contenu des communications ?

Sécurisation : deux étapes

- La source utilise une *fonction de chiffrement* pour coder un message m avec une clé de chiffrement k. La fonction produit en sortie un message chiffré s.

  `chiffrement(m, k)` $\rightarrow$ `s`

Sécurisation : deux étapes

- La source utilise une *fonction de chiffrement* pour coder un message m avec une clé de chiffrement k. La fonction produit en sortie un message chiffré s.

  `chiffrement(m, k)` $\rightarrow$ `s`

Sécurisation : deux étapes

- La source utilise une *fonction de chiffrement* pour coder un message m avec une clé de chiffrement k. La fonction produit en sortie un message chiffré s.
  `chiffrement(m, k)` $\rightarrow$ `s`
- Le destinataire utilise une *fonction de déchiffrement* pour décoder le message s avec la clé de chiffrement k. La fonction produit en sortie le message clair m.
  `déchiffrement(s, k)` $\rightarrow$ `m`

Sécurisation : deux étapes

- La source utilise une *fonction de chiffrement* pour coder un message m avec une clé de chiffrement k. La fonction produit en sortie un message chiffré s.

  `chiffrement(m, k)` $\rightarrow$ `s`

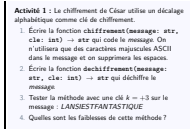
- Le destinataire utilise une *fonction de déchiffrement* pour décoder le message s avec la clé de chiffrement k. La fonction produit en sortie le message clair m.

  `déchiffrement(s, k)` $\rightarrow$ `m`

À retenir
Dans un chiffrement symétrique on utilise la même clé pour chiffrer et déchiffrer le message.

À retenir

Dans un chiffrement symétrique on utilise la même clé pour chiffrer et déchiffrer le message.


Activité 1 : Le chiffrement de César utilise un décalage alphabétique comme clé de chiffrement.

1. Écrire la fonction `chiffrement(message: str, cle: int) → str` qui code le *message*. On n'utilisera que des caractères majuscules ASCII dans le message et on supprimera les espaces.
2. Écrire la fonction `dechiffrement(message: str, cle: int) → str` qui déchiffre le *message*.
3. Tester la méthode avec une clé $k = +3$ sur le message : *LANSIESTFANTASTIQUE*
4. Quelles sont les faiblesses de cette méthode ?

Activité 1 : Le chiffrement de César utilise un décalage alphabétique comme clé de chiffrement.

1. Écrire la fonction **`chiffrement(message: str, cle: int)`** $\rightarrow$ **`str`** qui code le *message*. On n'utilisera que des caractères majuscules ASCII dans le message et on supprimera les espaces.
2. Écrire la fonction **`dechiffrement(message: str, cle: int)`** $\rightarrow$ **`str`** qui déchiffre le *message*.
3. Tester la méthode avec une clé $k = +3$ sur le message : *LANSIESTFANTASTIQUE*
4. Quelles sont les faiblesses de cette méthode ?

Correction

```
1  def chiffrement(message: str, cle: int) -> str:
2      sortie = ""
3      for lettre in message:
4          sortie += chr(ord(lettre)+cle)
5      return sortie
```

Correction

```
1  def dechiffrement(message: str, cle: int) -> str
       :
2      sortie = ""
3      for lettre in message:
4          sortie += chr(ord(lettre)-cle)
5      return sortie
```

```
k = 3
entree = "LANSIESTFANTASTIQUE"
m_chiffre = chiffrement(entree, k)
print(m_chiffre)
```
ODQVLHVWIDQWDVWLTXH

```
k = 3
entree = "LANSIESTFANTASTIQUE"
m_chiffre = chiffrement(entree, k)
print(m_chiffre)
```

ODQVLHVWIDQWDVWLTXH

ROT13

Correction
Quelle particularité si la clé est 13 ?

Correction

Quelle particularité si la clé est 13 ?

Correction
Quelle particularité si la clé est 13 ?
Les fonctions de chiffrement et déchiffrement sont identiques.

Correction

Quelle particularité si la clé est 13 ?
Les fonctions de chiffrement et déchiffrement sont identiques.

1. Si on ne prend pas en compte la clé qui transforme en la même lettre.
2. Si texte est assez long ; source : comptage dans une série de 20 livres ; **décryptage :** retrouver le message sans connaître la clé

Correction
▶ Il n'y a que 25 clés possibles.
1. source : apprendre-en-ligne

Correction

- Il n'y a que 25 clés possibles.

1. source : apprendre-en-ligne

Correction
- Il n'y a que 25 clés possibles.
- La fréquence d'apparition des lettres est une méthode simple à mettre en place pour décrypter un message.

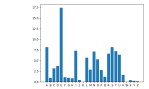


Figure – Fréquences d'apparition des lettres[1]

1. source : apprendre-en-ligne

1. Si on ne prend pas en compte la clé qui transforme en la même lettre.
2. Si texte est assez long ; source : comptage dans une série de 20 livres ; **décryptage :** retrouver le message sans connaître la clé

Correction

- Il n'y a que 25 clés possibles.
- La fréquence d'apparition des lettres est une méthode simple à mettre en place pour décrypter un message.

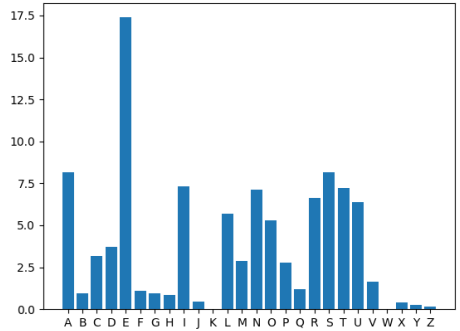
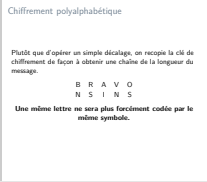


Figure – Fréquences d'apparition des lettres[1]

1. source : apprendre-en-ligne



Chiffrement polyalphabétique

Plutôt que d'opérer un simple décalage, on recopie la clé de chiffrement de façon à obtenir une chaîne de la longueur du message.

B R A V O
N S I N S

Une même lettre ne sera plus forcément codée par le même symbole.

Activité 2 :
1. Remplacer chaque lettre en son équivalent ASCII.
2. Écrire la fonction `int_en_bin(nb: int)` → `str` qui renvoie la représentation binaire de l'entier *nb*.
3. Convertir chaque entier en binaire.

on en cache un peu sous le tapis : on est toujours sur 7 bits

Activité 2 :

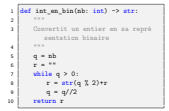
1. Remplacer chaque lettre en son équivalent ASCII.
2. Écrire la fonction `int_en_bin(nb: int)` → `str` qui renvoie la représentation binaire de l'entier *nb*.
3. Convertir chaque entier en binaire.

Correction

Correction

B	R	A	V	O
66	82	65	86	79
N	S	I	N	S
78	83	73	78	83

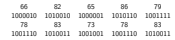
Correction



Correction

```
1  def int_en_bin(nb: int) -> str:
2      """
3      Convertit un entier en sa repré
           sentation binaire
4      """
5      q = nb
6      r = ""
7      while q > 0:
8          r = str(q % 2)+r
9          q = q//2
10     return r
```

Correction



Correction

66	82	65	86	79
1000010	1010010	1000001	1010110	1001111
78	83	73	78	83
1001110	1010011	1001001	1001110	1010011

On applique la porte logique *xor* entre chaque bit du message et de la clé. Une propriété intéressante de cette porte est qu'elle est réversible :

Si $A \oplus B = C$ alors $A \oplus C = B$ et $B \oplus C = A$

vigenere, enigma = substitution polyalphabétique

On applique la porte logique *xor* entre chaque bit du message et de la clé. Une propriété intéressante de cette porte est qu'elle est réversible :

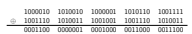
$$\text{Si } A \oplus B = C \text{ alors } A \oplus C = B \text{ et } B \oplus C = A$$

Activité 3 :
1. Appliquer le ou exclusif pour chaque bit du message.
2. Écrire la fonction `bin_en_int(paquet: str)` → `int` qui renvoie l'entier correspondant au paquet de bits.
3. Utiliser la fonction pour trouver l'entier correspondant à chaque paquet de sept bits.
4. Donner alors le message chiffré.

Activité 3 :

1. Appliquer le ou exclusif pour chaque bit du message.
2. Écrire la fonction `bin_en_int(paquet: str)` → `int` qui renvoie l'entier correspondant au paquet de bits.
3. Utiliser la fonction pour trouver l'entier correspondant à chaque paquet de sept bits.
4. Donner alors le message chiffré.

Correction



Correction

	1000010	1010010	1000001	1010110	1001111
$\oplus$	1001110	1010011	1001001	1001110	1010011
	0001100	0000001	0001000	0011000	0011100

Correction
0001100 0000001 0001000 0011000 0011100
12 1 8 24 28

Correction

on ne retransforme pas en ASCII : pas d'intérêt et on est sur des caractères non imprimables

0001100	0000001	0001000	0011000	0011100
12	1	8	24	28

Avantages : sûreté

Une clé trop courte ne garantit pas une bonne sécurité
- algorithme DES (*Data Encryption Standard*) obsolète à cause d'une clé maximale de 56 bits.

1. à cause puissance de calculs qui augmentent
2. 2^{56} possibilités ; également lenteur pendant le chiffrage
3. minimum conseillé aujourd'hui : 80 bits
4. montré par Claude Shannon en 1949 ; père de la théorie de l'information
5. pendant guerre froide (1963) : téléphone rouge utilisait une clé de la taille du message

Avantages : sûreté

Une clé trop courte ne garantit pas une bonne sécurité

- algorithme DES (*Data Encryption Standard*) obsolète à cause d'une clé maximale de 56 bits.

1. à cause puissance de calculs qui augmentent
2. 2^{56} possibilités ; également lenteur pendant le chiffrage
3. minimum conseillé aujourd'hui : 80 bits
4. montré par Claude Shannon en 1949 ; père de la théorie de l'information
5. pendant guerre froide (1963) : téléphone rouge utilisait une clé de la taille du message

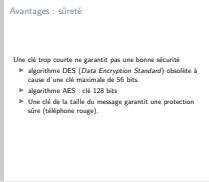
Avantages : sûreté

Une clé trop courte ne garantit pas une bonne sécurité
- algorithme DES (*Data Encryption Standard*) obsolète à cause d'une clé maximale de 56 bits.
- algorithme AES : clé 128 bits

Avantages : sûreté

Une clé trop courte ne garantit pas une bonne sécurité

- algorithme DES (*Data Encryption Standard*) obsolète à cause d'une clé maximale de 56 bits.
- algorithme AES : clé 128 bits



1. à cause puissance de calculs qui augmentent
2. 2^{56} possibilités ; également lenteur pendant le chiffrage
3. minimum conseillé aujourd'hui : 80 bits
4. montré par Claude Shannon en 1949 ; père de la théorie de l'information
5. pendant guerre froide (1963) : téléphone rouge utilisait une clé de la taille du message

Avantages : sûreté

Une clé trop courte ne garantit pas une bonne sécurité

- algorithme DES (*Data Encryption Standard*) obsolète à cause d'une clé maximale de 56 bits.
- algorithme AES : clé 128 bits
- Une clé de la taille du message garantit une protection sûre (téléphone rouge).

Avantages : rapidité
- Fonctionnement similaire à la méthode du *ou exclusif*.

Avantages : rapidité

- Fonctionnement similaire à la méthode du *ou exclusif*.

Avantages : rapidité
- Fonctionnement similaire à la méthode du *ou exclusif*.
- *xor* est une fonction implémentée dans les processeurs

Avantages : rapidité

- Fonctionnement similaire à la méthode du *ou exclusif*.
- *xor* est une fonction implémentée dans les processeurs

Avantages : rapidité
- Fonctionnement similaire à la méthode du *ou exclusif*.
- *xor* est une fonction implémentée dans les processeurs
- Possibilité de chiffrer en temps réel (données du disque dur par exemple)

Avantages : rapidité

- Fonctionnement similaire à la méthode du *ou exclusif*.
- *xor* est une fonction implémentée dans les processeurs
- Possibilité de chiffrer en temps réel (données du disque dur par exemple)

20 = étapes de mélange

Exemples

- *AES pour Advanced Encryption Standard :* choisi par l'institut de standardisation américain NIST (National Institute of Standards and Technology) en décembre 2001.
- *Chacha20 :* date de 2008 et améliore les performances d'un autre algorithme (Salsa20)

Exemples

- *AES pour Advanced Encryption Standard :* choisi par l'institut de standardisation américain NIST (National Institute of Standards and Technology) en décembre 2001.
- *Chacha20 :* date de 2008 et améliore les performances d'un autre algorithme (Salsa20)