

# TD 20 - Sécurisation des communications

## I - Chiffrement symétrique

---

### 1. Chiffrement par décalage

---

#### a. Le chiffre de César

---

a. FSRNSYV E XSYW. ZMZI PE QEXMIVI RWM!

b. Voir fichier Python.

c.

```
1 | >>> message_chiffre=cesar('BONJOUR A TOUS. VIVE LA MATIERE NSI!',4)
2 | 'FSRNSYV E XSYW. ZMZI PE QEXMIVI RWM!'
```

d.

```
1 | >>> cesar('FSRNSYV E XSYW. ZMZI PE QEXMIVI RWM!','-4')
```

#### b. Le chiffre de Vigenère

---

a.

```
1 | >>> vigenere_chiffrement('BONJOUR A TOUS. VIVE LA MATIERE NSI!','INFORMATIQUE')
2 | 'JBSXFGR T BEOW. DVAS CM MTBYYVM AXW!'
```

b. Voir fichier Python.

c. Voir fichier Python.

### 2. Chiffrement avec XOR bit à bit

---

a. Le XOR bit à bit est implémenté en python par `^`. Compléter la table de vérité du XOR ci-dessous.

| $x$ | $y$ | $x \wedge y$ |
|-----|-----|--------------|
| 0   | 0   | 0            |
| 0   | 1   | 1            |
| 1   | 0   | 1            |
| 1   | 1   | 0            |

b.

| $x$ | $y$ | $x \wedge y$ | $(x \wedge y) \wedge y$ |
|-----|-----|--------------|-------------------------|
| 0   | 0   | 0            | 0                       |
| 0   | 1   | 1            | 0                       |
| 1   | 0   | 1            | 1                       |
| 1   | 1   | 0            | 1                       |

On a donc  $(x \wedge y) \wedge y = x$

C'est donc un chiffrement symétrique et on peut utiliser la même fonction de chiffrement pour déchiffrer.

c.

$$65_{10} = 1000001_2$$

$$10_{10} = 1010_2 = 0001010_2$$

|                          |   |   |   |   |   |   |   |
|--------------------------|---|---|---|---|---|---|---|
| $65_{10}$                | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| $10_{10}$                | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $65_{10} \wedge 10_{10}$ | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

$$\text{Donc } 65_{10} \wedge 10_{10} = 1001011_2 = 75_{10}$$

On vérifie dans le shell:

```
1 | >>> 65^10
2 | 75
```

d . Voir fichier Python.

## II - Chiffrement asymétrique

---

## 1. Chiffrement RSA

---

a. Voir fichier Python. b. Voir fichier Python. c. Voir fichier Python. d. Voir fichier Python. e. Voir fichier Python. f. Voir fichier Python.

## 2. HTTPS

---

Les échanges vont se faire grâce à une clef symétrique.

Et pour échanger cette clef sans risque, un chiffrement asymétrique va être nécessaire.

Pour chaque nouveau client effectuant une requête, on a:

- Le client fait une requête auprès du serveur.
- Le serveur génère une clef publique ( `clef_publicque_serveur` ) et une clef privée ( `clef_privée_serveur` ) pour un chiffrement asymétrique. Il envoie la `clef_publicque_serveur` au client.
- Le client génère une clef partagée ( `clef_partagée` ) pour un chiffrement symétrique.
- Le client chiffre la `clef_partagée` avec la `clef_publicque_serveur` et l'envoie au serveur.
- Le serveur reçoit la version chiffrée de la `clef_partagée` du client, la déchiffre avec la `clef_privée_serveur` .
- Le client et le serveur sont donc en possession de la `clef_partagée` déchiffrée. Ils peuvent dorénavant échanger des données en les chiffrant et les déchiffrant avec la `clef_partagée` .