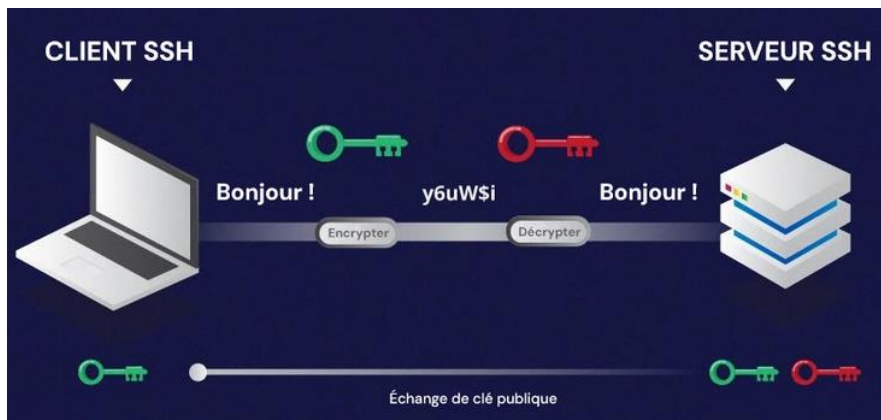


# Exercice -

# Le chiffrement RSA

Le **chiffrement RSA** (nommé par les initiales de ses trois inventeurs) est un algorithme de cryptographie **asymétrique**, très utilisé dans le commerce électronique, et plus généralement pour échanger des données confidentielles sur Internet. Il utilise une paire de clés (des nombres entiers) composée d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer des données confidentielles.



## 1- RAPPEL DE QUELQUES NOTIONS D'ARITHMETIQUES :



Définition : Les nombres **premiers** sont des entiers naturels supérieurs ou égal à 2, qui n'ont pas d'autres diviseurs qu'eux-mêmes et 1. Dans cette famille, on a les entiers suivants : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

Propriété : 2 nombres  $n$  et  $m$  sont **premiers entre eux** s'ils n'ont aucun diviseur en commun, autre que 1. Par exemple :

- 6 et 35 sont premiers entre eux :  $6 = 2 \times 3$  et  $35 = 7 \times 5$
- 6 et 27 ne sont pas premiers entre eux car ils ont 3 comme diviseur commun :  $6 = 2 \times 3$  et  $27 = 3 \times 3 \times 3$

## 2- DECOUVERTE AVEC UN CAS SIMPLIFIE :

a- Génération d'une clé publique  $(n, e)$  et d'une clé privée  $(n, d)$  :

Méthode	Application
1- On choisit au hasard deux nombres premiers $p$ et $q$	
2- On calcule les entiers $n$ et $m$ suivants : $n = p q$ et $m = (p - 1)(q - 1)$	
3- On choisit un entier $e$ appelé <i>exposant de chiffrement</i> qui respecte les conditions suivantes : $1 < e < m$ et $e$ <b>premier</b> avec $m$	
4- On choisit un entier $d$ appelé <i>exposant de déchiffrement</i> qui respecte la condition $(d \times e) \% m = 1$	

On suppose dans la suite que l'on utilise les clés publique ( $n, e$ ) et privé ( $n, d$ ) **du serveur**.

Lorsque le client réalise une demande au serveur (requête GET), celui-ci lui répond en lui envoyant « en clair » sa clé publique. Le client répond en utilisant cette clé pour chiffrer un premier envoi. Dans notre exemple ci-

dessous, ce premier envoi sera « Bonjour ! ». Le client va le chiffrer en utilisant la clé publique ( $n, e$ ). Par simplicité, on ne traite ci-dessous que le chiffrement de la première lettre qui est ici « B ».



b- Chiffrement du caractère « B » en utilisant la clé publique ( $n, e$ ) :

Méthode	Application
1- On convertit le caractère en nombre que l'on nommera ici $num$ en utilisant son code ASCII. En python, on peut utiliser la fonction <code>ord()</code> qui renvoie le code UNICODE du caractère : $num = ord(..)$	<pre>&gt;&gt;&gt; ord("B") 66</pre>
2- On calcule l'entier naturel $C$ égal à $C = (num^e) \% n$ Le caractère est chiffré avec le nombre $C$ .	

On suppose que cette valeur chiffrée  $C$  est envoyée sur le réseau internet. Elle est reçue par le serveur qui va la déchiffrer en utilisant sa clé privée ( $n, d$ ) secrète. Seul le serveur la connaît car il l'a générée lui-même.



c- Déchiffrement de la valeur chiffrée  $C$  utilisant la clé privée ( $n, d$ ) :

Méthode	Application
1- On calcule l'entier naturel $num$ égal à $num = (C^d) \% n$ Si la paire de clés public-privée est correcte, on retrouve le code UNICODE envoyé par le client.	
2- On convertit ce code en caractère. En python, on utilise la fonction <code>chr()</code> qui renvoie le caractère en retour.	<pre>&gt;&gt;&gt; chr(66) 'B'</pre>

Le serveur a pu retrouver la lettre « B » avec sa clé privée qu’il est le seul à connaître. Le serveur est ainsi le seul à pouvoir déchiffrer un message qui a été chiffré avec sa clé publique.



### 3- LIMITES DE CETTE METHODE SIMPLIFIEE :

Si on utilise la même méthode sur tout le message « Bonjour ! », cela donne :

Chiffrement de « Bonjour ! » avec la clé publique  $(n, e) = (187, 3)$

	'B'	'o'	'n'	'j'	'o'	'u'	'r'	' '	'!'
$num = ord(..)$	66	111	110	106	111	117	114	32	33
$C = (num^e) \% n$			121	13	100	145	130	43	33

Déchiffrement de la suite de nombre chiffrés avec clé privée  $(n, d) = (187, 107)$

$C$			121	13	100	145	130	43	33
$num = (C^d) \% n$	66	111	110	106	111	117	114	32	33
caractère	'B'	'o'	'n'	'j'	'o'	'u'	'r'	' '	'!'

On peut mettre en avant les limites suivantes :

- Il y a plus de 100 000 codes UNICODE différents. Comme le caractère chiffré  $C$  est le résultat d'un calcul modulo  $n$  ( $\%n$ ), pour éviter que 2 caractères aient la même valeur chiffrée, il faut que  $n$  soit au-moins supérieur à 100 000.
- Le caractère 'o' de Bonjour se retrouve 2 fois dans le mot 'Bonjour !'. En réalisant une analyse fréquentielle sur la liste de nombre chiffrés, il serait possible de casser ce chiffrement. Pour palier à ce problème, on peut fixer un nombre de chiffres alloué à chaque caractère et ensuite regrouper les caractères. Par exemple si on chiffre des groupes de 2 caractères, chacun représenté par un UNICODE complété à 3 chiffres, cela donnerait pour le mot 'Bonjour !' :

: Chiffrement de « Bonjour ! » avec la clé publique  $(n, e) = (1592537, 16349)$

	'B'	'o'	'n'	'j'	'o'	'u'	'r'	' '	'!'
$num = ord(..)$	66	111	110	106	111	117	114	32	33
$C = (num^e) \% n$									

Déchiffrement de la suite de nombre chiffrés avec clé privée  $(n, d) = (1592537, 389)$

$C$									
$num = (C^d) \% n$									
$num$	66	111	110	106	111	117	114	32	33
caractère	'B'	'o'	'n'	'j'	'o'	'u'	'r'	' '	'!'

**3- ROBUSTESSE ET FAILLES POSSIBLES :**

Pour « casser » RSA il est nécessaire de pouvoir factoriser le nombre  $n$  pour retrouver le produit initial des nombres  $p$  et  $q$ . Avec les algorithmes classiques, le temps que prend cette factorisation croît exponentiellement avec la longueur de la clé. Ce qui fait le succès du RSA est qu'il n'existe pas d'algorithme connu de la communauté scientifique pour réaliser une attaque force brute avec des ordinateurs classiques. On peut trouver la factorisation d'une clé de taille inférieure à 256 bits en quelques minutes sur un ordinateur individuel, en utilisant des logiciels librement disponibles. Pour une taille allant jusqu'à 512 bits, il faut faire travailler conjointement plusieurs centaines d'ordinateurs. Par sûreté, il est couramment recommandé que la taille des clés RSA soit au moins de 2 048 bits .

**4- AUTRE EXERCICE DE CHIFFREMENT RSA SIMPLIFIE ET TRAITE A LA MAIN :**

**Alice** décide de choisir  $(n ; e) = (21 ; 5)$  comme clé publique.

a) Peut-elle faire ce choix ? Expliquer pourquoi ce choix de clé publique ne garantit pas la sécurité du message.

b) Rappeler les valeurs de  $p, q, m$ .

c) Parmi les clés suivantes déterminer celle qu'Alice peut choisir comme clé privée :

$(21 ; 15) ; (21 ; 17) ; (21 ; 19)$

Dans la suite on admet qu'Alice choisit la seule clé privée possible parmi les propositions précédentes.

**Bob** veut envoyer à Alice le message  $(M) : PAL$ . Il va utiliser la clé publique  $(21 ; 5)$ .

b) Associer à chaque lettre du message, sa **place** dans l'alphabet. Par exemple la lettre C est codée par 3. Obtenir ainsi trois entiers naturels  $M_1, M_2, M_3$ .

c) Déterminer les 3 nombres  $C_1, C_2, C_3$  qui chiffrent les nombres  $M_1, M_2, M_3$ .

d) En déduire le message chiffré ( $C$ ) que Bob envoie à Alice.

Alice reçoit le message ( $C$ ) : 4 1 3. Elle veut déchiffrer ce message avec sa clé privée (21 ; 17)

e) Déterminer  $M_1, M_2, M_3$  en déchiffrant les 3 nombres

f) En déduire le message en clair que Bob a envoyé à Alice.

## 5- GENERATION DE CLES RSA SUR PYTHON :

La bibliothèque *PyCryptodome* est une bibliothèque Python qui fournit des fonctions et des algorithmes cryptographiques. Il s'agit d'un package Python autonome offrant une large gamme d'opérations cryptographiques, notamment le chiffrement, le déchiffrement, le hachage et la vérification de signature.

⇒ Ouvrir un fichier nommé *rsa.py* qui sera à uploader sur répertoire *P:\stsIre\Devoir*

⇒ Installer la bibliothèque sur votre poste, en exécutant les commandes suivantes dans la console de pyzo :

- Mettre à jour pip car les ordinateurs du lycée Branly ne sont pas à jour. Pour cela exécuter :

```
>>> pip install --user --upgrade pip
```

- Installer la bibliothèque :

```
>>> pip install pycryptodome
```

- **Fermer** Pyzo et le **réouvrir** afin que l'installation soit prise en compte

⇒ Dans le fichier *rsa.py*, copier-coller le code ci-contre. Il permet de générer une paire de clés publique / privée qui sont écrites dans des fichiers au format *.pem*.

Exécuter ce script et répondre aux questions suivantes :

- a) Exécuter dans la console la commande `type(cle_rsa)`

On constate que cette variable *cle\_rsa* est une instance de la classe RSA qui contient une paire de clés de 2048 bits de longueur.

- b) Exécuter dans la console la commande

`type(cle_privee)`. On constate que cette variable est une instance de la class bytes. Les *bytes* sont des séquences immuables d'octets.

```
from Crypto.PublicKey import RSA
# Génération de la clé
cle_rsa = RSA.generate(2048)
# Export des clés publique et privée
cle_privee = cle_rsa.exportKey()
cle_publique = cle_rsa.publickey().exportKey()
# Écriture des clés dans des fichiers
fd = open("cle_privee.pem", "wb")
fd.write(cle_privee)
fd.close()
fd = open("cle_publique.pem", "wb")
fd.write(cle_publique)
fd.close()
```

- c) Exécuter dans la console `>>> cle_privee.decode("utf-8")`

```
puis >>> cle_privee
puis >>> cle_privee.hex()
```

Ces 3 exécutions permettent d'afficher les 2048 bits de la clé privée en format utf-8 ou hexadécimal.

## 6- UTILISATION CLES RSA SUR PYTHON :

⇒ Dans le fichier *rsa.py*, copier-coller à la suite le code ci-contre. Il permet d'utiliser les clés qui ont été générées pour chiffrer et déchiffrer un message.

Exécuter ce script et répondre aux questions suivantes :

Copier dans un fichier nommé *rsa.doc* qui sera à uploader sur répertoire *P:\stslrEr\Devoir* en fin d'activité :

- une copie d'écran du message en clair en hexadécimal (exécuter `message.hex()`)
- une copie d'écran du message chiffré en hexadécimal
- une copie d'écran du message déchiffré en utf-8 (exécuter `message.decode('utf-8')`)

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

#Lecture de la clé public
clePB = RSA.import_key(open('cle_publique.pem').read())

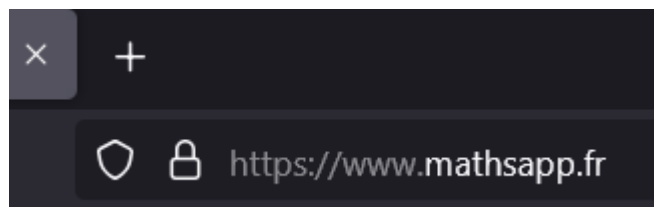
#Chiffrement d'un message avec la clé publique
message = b'Ceci est un message de test'
cipher = PKCS1_OAEP.new(clePB)
ciphertext = cipher.encrypt(message)

#Dechiffrement d'un message avec la clé privé
clePR = RSA.import_key(open('cle_privee.pem').read())
cipher = PKCS1_OAEP.new(clePR)
message_dechiffre = cipher.decrypt(ciphertext)
```

## 7- CLE PUBLIC DU SITE MATHSAPP.FR :

Les clés publiques des sites sont consultables. Il suffit de cliquer sur le cadenas de la barre d'URL de la page du site. Par exemple pour *mathsapp.fr*, le click renvoie la fenêtre ci-contre. Continuer sur Connexion sécurisée – Plus d'informations – Afficher le certificat.

On retrouve sur la page, dans la rubrique « Informations sur la clé publique » la valeur de l'exposant  $e$  et celle du module  $n$ .



- Copier dans le fichier *rsa.doc* une copie d'écran de  $e$  et de  $n$

- b) Le nombre  $n$  est donné en hexadécimal. Dans cette numération en base 16, 2 caractères représentent 1 octet, soit 8 bits. Compter le nombre d'octets de  $n$ . Cette clé a-t-elle vraiment une longueur de 2048 bits ?

## 8- HACHAGE DES MOTS DE PASS :

Le hachage des mots de passe consiste à convertir les mots de passe en une chaîne alphanumérique à l'aide d'algorithmes spécialisés.

⇒ Dans le fichier *rsa.py*, copier-coller à la suite le code ci-contre.

Exécuter ce script et répondre dans le fichier nommé *rsa.doc* aux questions suivantes :

- a) Donner une copie d'écran du mot de passe haché en hexadécimal
- b) Quelle est l'utilité de ce code ?
- c) Rechercher sur le net la réponse à la question suivante : « quelle différence y a-t-il entre le hachage et le chiffrement » ?

```
from Crypto.Hash import SHA256
#Le mot de pass est haché
mdpHach = SHA256.new(data=b'MonMotDePass')
mdpHach = mdpHach.hexdigest()
print("hach du mot de passe\n",mdpHach)

#Un utilisateur saisi un mot de passe
saisie = "MonMotDePass"
saisie = saisie.encode("utf-8")
hash_saisie = SHA256.new(data=saisie)
hash_saisie = hash_saisie.hexdigest()

if hash_saisie == mdpHach :
print("Le mot de pass saisi est le bon")
```

## 9- IMPLEMENTATION D'UNE FONCTION SIMPLIFIEE RSA:

⇒ Télécharger sur [mathsapp.fr/fr](http://mathsapp.fr/fr) le fichier *creerRsa.py* qui contient entre autres :

- La fonction *premiers(n)* qui retourne la liste de tous les nombres premiers inférieurs à  $n$ .
- La fonction *cles(p, q)* qui retourne 2 tuples  $(n, e)$  et  $(n, d)$  qui correspondent à une paire de clé publique / privée calculée à partir des nombres premiers  $p$  et  $q$  qui sont en paramètre.

