Sécurisation des communications EXERCICES

Dernière mise à jour le : 16/04/2024

Exercice 1 : Chiffre de César

✍ **Question 1** : Chiffrez le mot `'NSI'` selon le chiffrement de César avec un décalage de 9.

🐍 **Question 2** : Écrivez une fonction `cesar` qui prend en paramètres une chaîne de caractères `texte` et un entier `c`, et qui renvoie le texte chiffrée par la méthode de César en utilisant le décalage `c`.
On supposera pour simplifier que le texte de départ ne contient que des majuscules non accentuées et qu'on ne chiffre ni les espaces ni les signes de ponctuation.

Rappels :

- la fonction `ord` renvoie le code ASCII du caractère passé en argument :

```
>>> ord('N')
78
```

- la fonction `chr` renvoie le caractère dont le code ASCII est passé en argument

```
>>> chr(78)
N
```

- `n%26` donne le reste dans la division euclidienne de `n` par 26

🐍 **Question 3 (attaque par force brute)** : Vous voulez attaquer par force brute le message chiffré par la méthode de César. Écrivez une fonction `force_brute_cesar` qui prend en paramètre une chaîne de caractères `texte_chiffre` qui a été chiffrée par la méthode de César, et qui décrypte cette chaîne par *force brute*.
On suppose toujours qu'il n'y a que des majuscules non acentuées et que les espaces et symboles de ponctuation ne sont pas chiffrés.

Exercice 2 : Masque jetable avec XOR

On considère la chaîne de caractères :

```
masque = "CETTEPHRASEESTVRAIMENTTRESTRESLONGUEMAISCESTFAITEXPRES"
```

🐍 **Question 1** : Écrivez une fonction `chiffre_xor(message: str, masque: str) -> str` qui chiffre `message` en faisant un XOR avec `masque`.
On supposera ici que la longueur du message est inférieure ou égale à celle du masque, comme c'est le cas pour la méthode dite du masque jetable ou (chiffrement de Vernam) qui prévoit (entre autres) que la clé ne doit pas être étendue pour rester sûre.

Exemples :

```
>>> chiffre_xor("hello", masque)
'+ 88*'
>>> chiffre_xor("nsi", masque)
'-6='
```

Aides :

- En Python, l'opérateur XOR s'écrit `^` et s'applique entre deux entiers (Python se charge de faire les conversions en binaire et d'exécuter le XOR bit à bit comme vu dans le cours.)

```
>>> 34 ^ 23  # 34 XOR 23
53
>>> bin(34), bin(23)
('0b100010', '0b10111')  # et 100010 XOR 010111 = 110101
>>> 0b110101  # vérification
53
```

- La fonction `ord` permet de renvoyer le code ASCII d'un caractère. La fonction `chr` fait l'opération inverse.

```
>>> ord('A')
65
>>> chr(65)
'A'
```

🐍 **Question 2** : Vérifiez que la fonction `chiffre_xor` permet **aussi** de déchiffrer le message chiffré.

Exercice 3 : Fiabilité de RSA

La fiabilité du système RSA est lié à la difficulté, en connaissant un entier n très grand, de trouver deux nombres premiers p et q tels que $n = pq$.

> *L'objectif de cet exercice est de le vérifier expérimentalement.*

Pour vérifier qu'un nombre est premier vous pourrez utiliser la fonction `isprime()` du modyle `sympy` (à installer si nécessaire avec `pip install sympy`). Cette fonction prend un entier en paramètre et renvoie `True` si cet entier est premier et `False` dans le cas contraire :

```
>>> from sympy import isprime
>>> isprime(17)
True
>>> isprime(27)
False
```

De plus, la bibliothèque `pycrypto` (à installer si nécessaire avec `pip install pycrypto`) permet de générer des nombres premiers de n'importe quelle taille :

```
from Crypto.Util import number
from Crypto.Random import get_random_bytes

# Taille des nombres (en bits)
n_length = 20

# Génération de deux nombres premiers p et q de cette taille
p = number.getPrime(n_length, randfunc=get_random_bytes)
q = number.getPrime(n_length, randfunc=get_random_bytes)
```

1. Recopiez ce programme, le tester et le compléter afin de calculer $n = pq$.
2. Écrivez une fonction `factorise` qui prend en paramètre un entier `n` et renvoie deux nombres premiers `p` et `q` tels que `n = p*q`.
   Vous utiliserez l'algorithme très simple qui consiste à essayer toutes les divisions possibles de `n` jusqu'à en trouver une dont le reste est 0. Il faudra ensuite vérifier si le diviseur peut convenir...
3. Testez votre fonction `factorise` et vérifiez qu'elle permet de retrouver `p` et `q` à partir de `n`.
4. Modifiez la valeur du paramètre `n_length` et mesurez les temps d'exécution de la fonction `factorise` au fur et à mesure que `n_length` augmente.
   Attention : Ne pas choisir une valeur trop élevée pour `n_length`, prenez par exemple 21 puis 22, 23, ... et observez l'augmentation correspondante du temps de calcul.

5. Prévoyez le temps d'exécution pour `n_length = 100`.
6. La majorité des sites Web actuels utilisent actuellement des clés de longueur 1024. Quel serait le temps d'exécution prévisible de votre programme pour casser une de ces clés ?

Exercice 4 : Certificat et autorité de certification

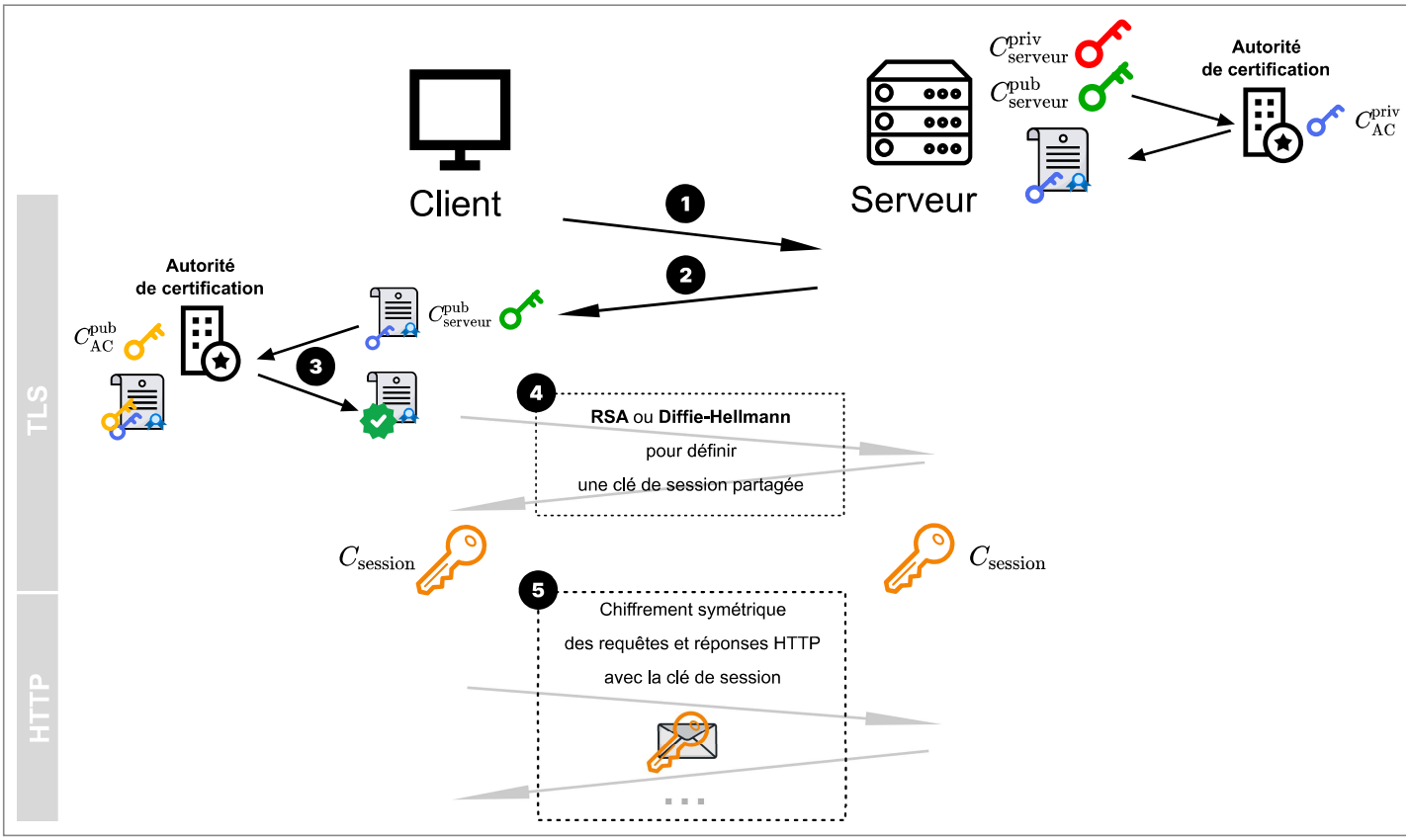
1. Lancez Firefox et ouvrez les outils de développement (raccourci F12) puis allez sur l'onglet *Réseau*.
2. Rendez-vous à l'adresse `www.wikipedia.fr` puis cliquez dans l'onglet *Réseau* sur la première requête envoyée et allez dans l'onglet *Sécurité* (tout à droite). Retrouver les informations suivantes :
   - La version du protocole TLS utilisé
   - Le nom de l'autorité de certification
   - La période de validité du cerfificat d'identité
   - L'algorithme utilisé pour la signature
   - L'algorithme utilisé pour la suite du chiffrement
3. Retrouvez ces informations à partir de l'icône *Cadenas* à côté de la barre d'URL.

Exercice 5 : Questions diverses (pour réviser)

1. Expliquer en une phrase la différence entre un chiffrement *symétrique* et un chiffrement *asymétrique*.
2. Pour chacun des sigles ou noms suivants, le décrire en une phrase *sans donner la signification du sigle*.
   - TLS
   - RSA
   - HTTPS
   - Diffie-Hellman
   - AES
   - Autorité de certification
3. Un chiffrement symétrique est-il nécessairement plus faible qu'un chiffrement asymétrique ?

4. Bianca souhaite échanger un message chiffré avec Alan en utilisant un chiffrement *asymétrique*. Alan possède une clé publique $C_{\mathrm{A}}^{\mathrm{pub}}$ et une clé privée $C_{\mathrm{A}}^{\mathrm{priv}}$. Bianca possède une clé publique $C_{\mathrm{B}}^{\mathrm{pub}}$ et une clé privée $C_{\mathrm{B}}^{\mathrm{priv}}$. Bianca souhaite chiffrer un message m afin de pouvoir l'envoyer à Alan :
   - Quelle clé va être utilisée par Bianca pour chiffrer le message m ?
   - Quelle clé va être utilisée par Alan pour déchiffrer le message m ?

5. On se place dans le contexte du schéma ci-dessous vu dans le cours et qui présente le fonctionnement du protocole HTTPS.
   - Identifiez les phases :
     - d'authentification
     - de création d'une clé symétrique
     - d'échanges sécurisés selon un chiffrement symétrique
   - Indiquez si chacune de ces phases est implémentée par un chiffrement symétrique ou asymétrique en précisant lequel (ou lesquels) est utilisé.



Fonctionnement du protocole HTTPS

6. On se place toujours dans le contexte du schéma précédent. Dans chacune des situations suivantes, donnez le numéro de l'étape qui échoue.

- Le serveur Web n'est pas configuré pour supporter le protocole HTTPS et ne sert que des pages en HTTP
- Le fichier contenant le certificat du serveur est périmé.
- L'administration du serveur a créé une paire de clé publique et privée, a signé lui-même le certificat que le serveur a envoyé aux clients et effacé les clés.
- Le navigateur commence à afficher la page de garde du site. Le câble connectant le serveur au réseau est coupé.

Références :

- L'exercice 1 est une adaptation d'un exercice de Mireille Coilhac diffusé sous licence CC-BY-NC-SA : Le code de César
- L'exercice 2 est repris des cours de Gilles Lassus (Cryptographie) et de Mireille Coilhac (Sécurisation des communications) diffusés respectivement sous licences CC-BY-SA et CC-BY-NC-SA.
- Les exercices 3 et 4 sont adaptés d'exercices proposés par Fabrice Nativel dans son cours Sécurisation des communications diffusé sous licence CC-BY.
- Certaines questions de l'exercice 5 sont adaptées du manuel *Numérique et Sciences Informatiques, 24 leçons, Terminale*, T. BALABONSKI, S. CONCHON, J.-C. FILLIATRE, K. NGUYEN, éditions ELLIPSES.

Germain Becker, Lycée Emmanuel Mounier, Angers.

Voir en ligne : info-mounier.fr/terminale_nsi/archi_se_reseaux/securisation-communications-exercices