Sécurisation

Ce qu'il faut savoir faire à l'issue du chapitre :

- Décrire les principes de chiffrement symétrique (clé partagée) et asymétrique (clé privée/-clé publique)
- Décrire l'échange d'une clé symétrique en utilisant un protocole asymétrique pour sécuriser une communication HTTPS

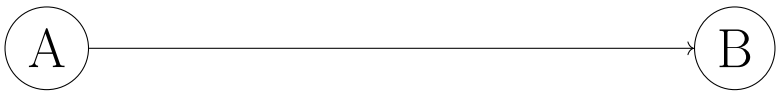
1 Intérêt

Histoire

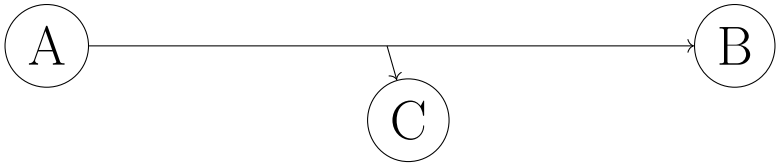
Il faut savoir que l'idée de chiffrer des messages (de les rendre illisibles pour des personnes non autorisées) ne date pas du début de l'ère de l'informatique !
En effet, dès l'antiquité, on cherchait déjà à sécuriser les communications en chiffrant les messages sensibles.
Citons par exemple :

- Le chiffrement de césar : http://www.cryptage.org/chiffre-cesar.html
- Le chiffrement de Vignere : http://mathweb.free.fr/crypto/poly/vigenere.php3
- La machine enigma
  : http://www.irem.unilim.fr/fileadmin/user_ upload/Convergences/tpe_ enigma.pdf

Imaginons que A envoie un message à B :



Si le message est non sécurisé, il pourra être intercepté et lu par C.



Pour ce faire, A va chiffrer le message.

Si C intercepte le message, et si C ne posséde pas le moyen de déchiffrer le message chiffré, il sera impossible pour lui de le lire le contenu.

Définition 14.1

Le **chiffrement** d'un message consiste à le modifier et le rendre illisible par une personne qui n'est pas autorisée.

Définition 14.2

Le chiffrement (et le déchiffrement) est réalisé en appliquant au message initial une fonction mathématique, basée sur une donnée convenue entre les deux extrémités, appelée **clés de chiffrement**.

Il existe 2 grands types de chiffrement : le chiffrement symétrique et le chiffrement asymétrique.

Nous étudierons ici le chiffrement de message 'informatique'. Comme toute "donnée informatique" peut être vue comme une suite de zéro et de un. Nous chercherons donc à chiffrer une suite de zéro et de un !

2 Le chiffrement symétrique

2.1 Un exemple de chiffrement symétrique

2.1.1 Codage du mot

Je souhaite envoyer "Hello", qui donne en binaire :
01001000011001010110110001101100011011111

Afin de convertir du texte en binaire, et inversement , il est possible d'utiliser des sites comme :
Conversion en binaire du texte : https://www.rapidtables.com/convert/number/ascii-to-binary.html
ou mieux, une fonction python ! (cf. ce qui a été réalisé en première à ce sujet)

Exercice 14.1
Créer une fonction **ascii_to_binary(mot)** qui prend en argument une chaîne de caractères, et qui renvoie une chaine de caractère composée de 0 et de 1, et qui correspond à la conversion en binaire du mot entré en argument. Chaque lettre devra être codé sur 8 bits.

```
>>> ascii_to_binary('Hello')
'01001000011001010110110001101100011011111'
```

2.1.2 Décodage

Exercice 14.2
Créer une fonction **binary_to_ascii(mot)** qui prend en argument une chaîne de caractères constitutée de 0 et de 1 (multiple de 8), et qui renvoie le mot correspondant.

```
>>> binary_to_ascii('01001000011001010110110001101100011011111')
'Hello'
```

2.1.3 Chiffrement

Choisissons maintenant un "mot" qui nous servira de clé de chiffrement. Je choisis par exemple "JB" qui donne en binaire :
0100101001000010
On va ensuite "recopier" autant de fois que nécessaire la clé pour avoir exactement autant de bits que le mot à coder :

mot à coder	01001000011001010110110001101100011011111
clé	01001010010000100100101001000010100101 0

Nous allons ensuite, pour chiffrer le message, effectuer un XOR bit à bit :

mot à coder	01001000011001010110110001101100011011111
clé	01001010010000100100101001000010100101 0
XOR bit à bit	00000010001001110010011000101110001001 01

Exercice 14.3
Créer une fonction **Codage(mot,cle)** qui prend en argument une chaîne de caractères et une clé, et qui renvoie le mot codé.

```
>>> codage('Hello','JB')
"\x02'&.%"
```

C'est le message que nous allons envoyer ! Si C l'intercepte et le lit, il aura "\x02'& .% "! et ne pourra pas lire le contenu.

2.1.4 Déchiffrement

B reçoit le message chiffré, et il connait la clé (car A lui a donné) :

On fait de nouveau un XOR bit à bit :

mot à décoder	000000100010011100100110001011100010 0101
clé	010010100100001001001010010000100 1001010
XOR bit à bit	010010000110010101101100011011000 1101111

Vous pouvez remarquer que nous avons bien retrouvé le code binaire d'origine ! ouf !

A vous de jouer ! 14.1
Imaginer un message court que vous souhaitez envoyer et convertissez-le en code binaire ASCII : chaque caractère est codé sur un octet.
Choisissez une clé et chiffrez votre message. Remettez à votre voisin le message chiffré et la clé : celui-ci doit retrouver le message initial.

☞ Le gros problème avec le chiffrement symétrique, c'est qu'il est nécessaire pour A et B de se mettre d'accord à l'avance sur la clé qui sera utilisée lors des échanges (et aussi se mettre d'accord sur l'algorithme de chiffrement). Le chiffrement asymétrique permet d'éviter ce problème.

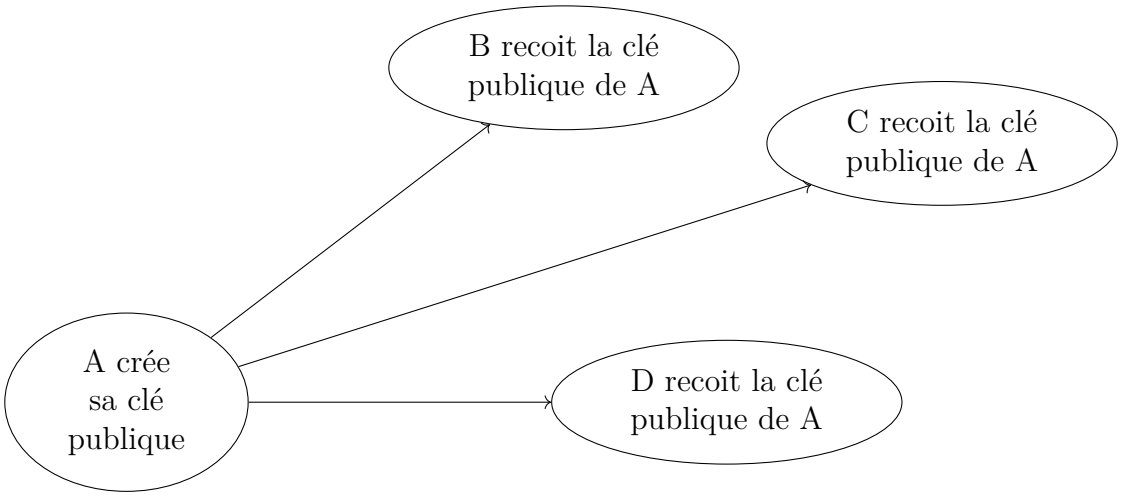
☞ Les avantages d'utiliser "XOR" :

- L'opération "ou exclusif" se calcule très rapidement.
- Le chiffrement fonctionne avec n'importe quel fichier binaire et pas seulement du texte.
- La même clef sert au chiffrement et au déchiffrement.

2.2 Principe du chiffrement symétrique

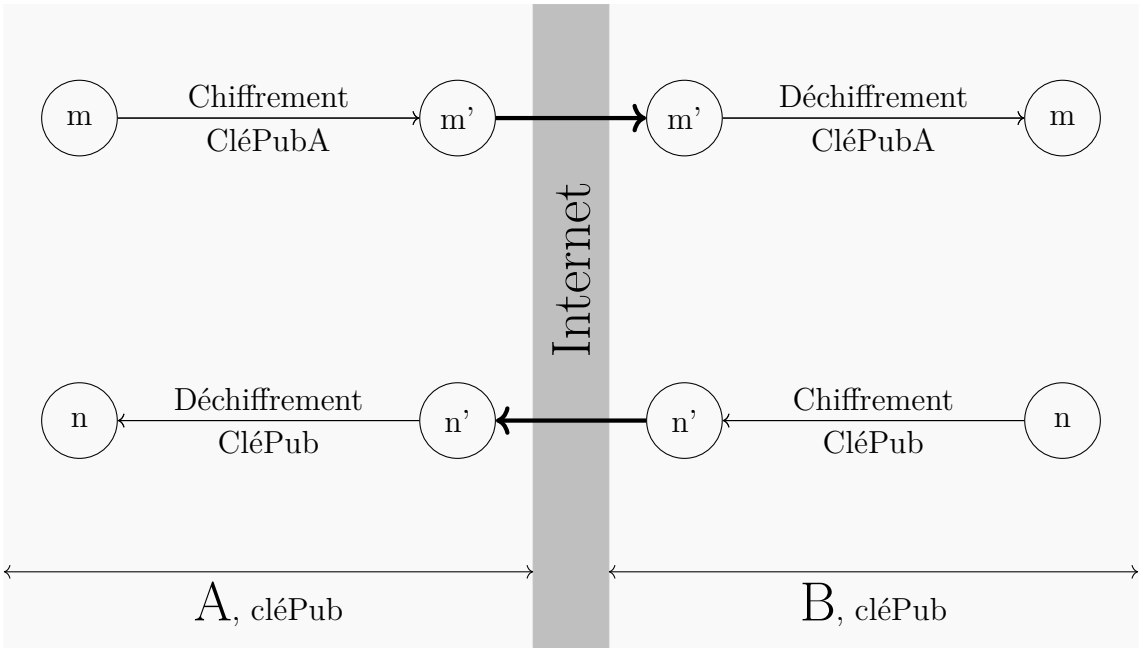
Supposons que (A) ait l'envie d'envoyer un message à (B), (C) et (D) de façon sécurisée. (A) doit créer une clé de chiffrement (clé publique) et la transmettre à (B) :

Génération des clés



La clé va servir à chiffre lors de l'envoi et à déchiffrer lors de la réception :

Envoi et réception de message avec XOR



2.3 Un autre algorithme

Analysez et testez ce programme :

```
def chiffrement(mess,cle):
    mess_code = ""
    position = 0
    for lettre in message:
        mess_code = mess_code + chr(ord(lettre) ^ cle[position]
        position = (position + 1) % len(cle)
    return mess_code
```

Exercice 14.4
Utilisez les fonctions précédentes pour chiffrer la phrase "Veuillez transférer 1000 € sur mon compte off-shore" en utilisant la clef "ŭšĠţŬǩĠųŹŭǩŴŲũűŵŧ".

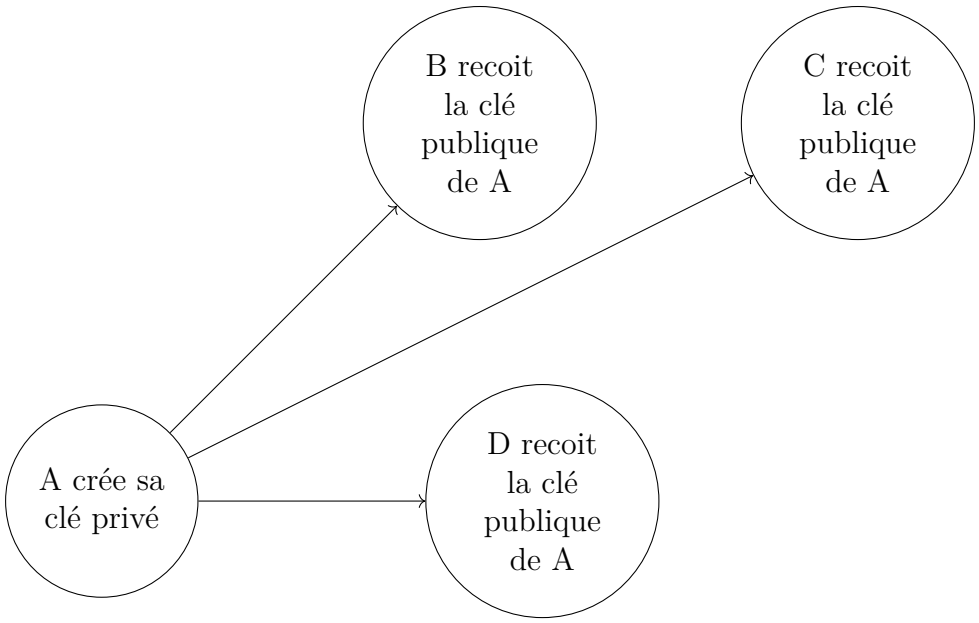
Cette clef s'obtient en considérant les caractères dont l'unicode est [365, 353, 288, 355, 364, 489, 288, 371, 377, 365, 489, 372, 370, 361, 369, 373, 357].

3 Chiffrement asymétrique

Supposons que (A) ait l'envie d'envoyer un message à (B), (C) et (D) de façon sécurisée. (A) doit créer une clé de chiffrement privé (clé privé) et le garde pour lui précieusement. A partir de sa clé privée, il crée une clé publique qu'il diffuse largement.

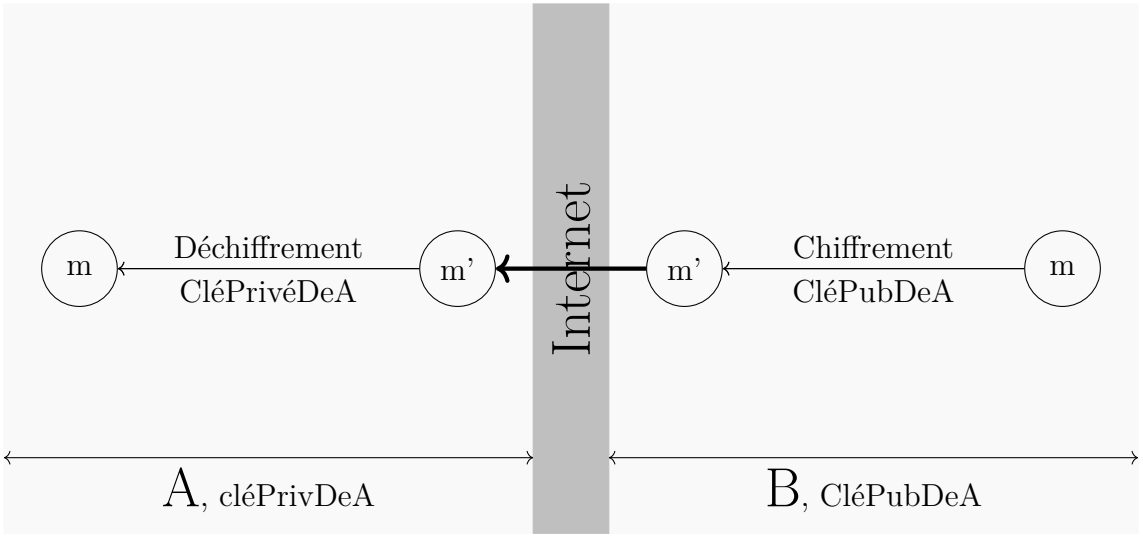
⚠ Bien évidemment, le système repose sur le fait qu'il est impossible de trouver la clé privé à partir de la clé publique.

Génération des clés

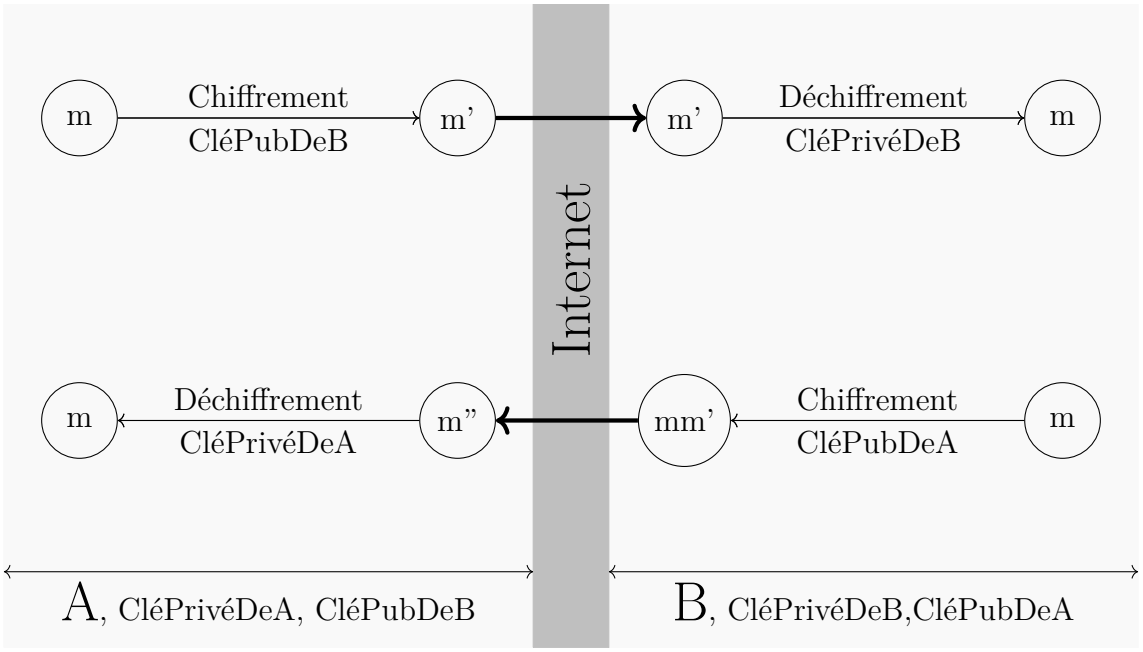


⚠ La clé privée va servir à chiffre lors de l'envoi et la clé publique à déchiffrer lors de la réception :

Envoi d'un message de (B) vers (A)

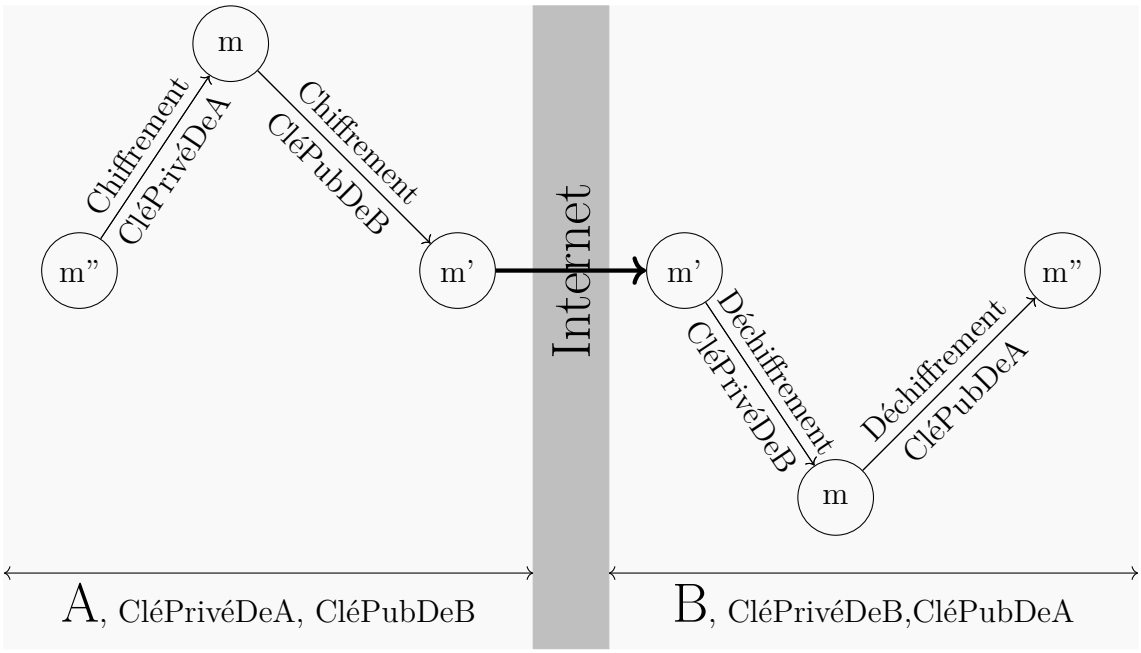


Envoi et réception de message



Le chiffrement asymétrique peut servir à authentifier l'expéditeur d'un message :

Authentification de l'émetteur



Si le récepteur (B ici) peut déchiffrer grâce à la clé publique de (A), c'est bien que le message a été envoyé par (A) (le seul à avoir ClePrivéDeA).

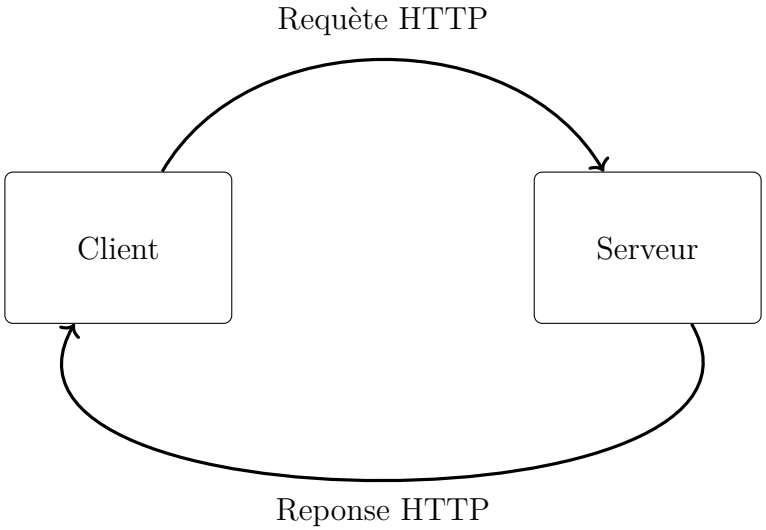
☞ Le concept de signature numérique est basé sur cette technique de chiffrement asymétrique.

☞ Le chiffrement asymétrique repose sur des problèmes très difficiles à résoudre dans un sens et faciles à résoudre dans l'autre sens.
Prenons un exemple : l'algorithme de chiffrement asymétrique RSA est très couramment utilisé, notamment dans tout ce qui touche au commerce électronique.
RSA se base sur la factorisation des très grands nombres premiers. Si vous prenez un nombre premier A (par exemple A = 16813007) et un nombre premier B (par exemple B = 258027589), il facile de déterminer C le produit de A par B (ici on a A x B = C avec C = 4338219660050123). En revanche si je vous donne C (ici 4338219660050123) il est très difficile de retrouver A et B. En tous les cas, à ce jour, aucun algorithme n'est capable de retrouver A et B connaissant C dans un temps "raisonnable". Nous avons donc bien ici un problème relativement facile dans un sens (trouver C à partir de A et B) est extrêmement difficile dans l'autre sens (trouver A et B à partir de C).

4 Le protocole HTTPS

Ne pas hésiter à reprendre le cours de première sur ce sujet.

Rappel échanges client-serveur protocole HTTP



HTTPS est la version sécurisée de HTTP.
HTTPS s'appuie sur le protocole TSL (Transport Layer Security), connu aussi sous le nom de SSL (Secure Sockets Layer).

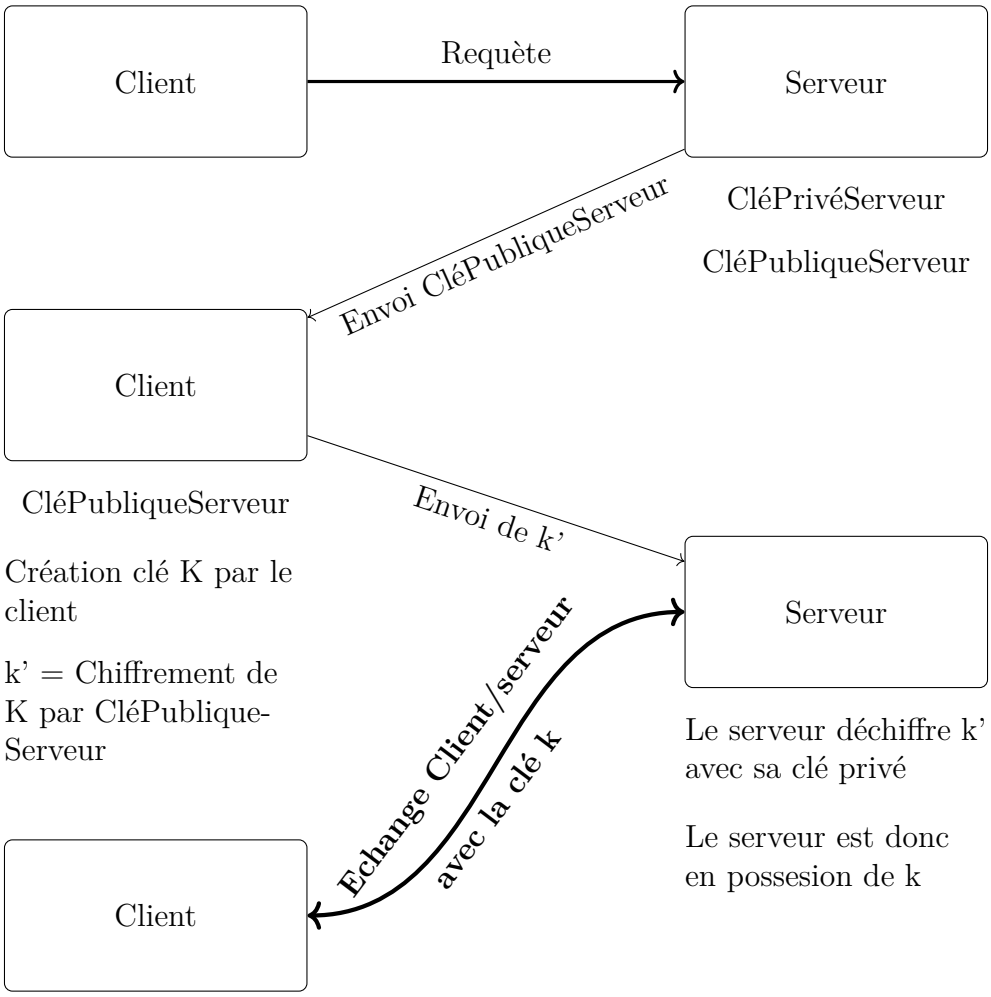
Principe :

Les communications vont être chiffrées grâce à une clé symétrique. Problème : comment échanger cette clé entre le client et le serveur ? Simplement en utilisant une paire clé publique / clé privée !

Voici le déroulement des opérations :

1. Le client effectue une requête HTTPS vers le serveur, en retour le serveur envoie sa clé publique (CléPubliqueServeur) au client
2. le client "fabrique" une clé k , chiffre cette clé k avec la clé publique CléPubliqueServeur et envoie la version k' chiffrée de la clé k au serveur
3. Le serveur reçoit la version chiffrée k' de la clé k et la déchiffre en utilisant sa clé privée (CléPrivéServeur).
4. À partir de ce moment-là, le client et le serveur sont tous les deux en possession de la clé k. Le client et le serveur peuvent échanger des données en les chiffrant et en les déchiffrant à l'aide de la clé K (chiffrement symétrique).

Principe du protocole TLS



À partir de ce moment-là, le client et le serveur sont en possession de la clé K le client et le serveur commencent à échanger des données en les chiffrant et en les déchiffrant à l'aide de la clé K (chiffrement symétrique).