

Sécurisation des communications

1. Rappels : communication sur internet

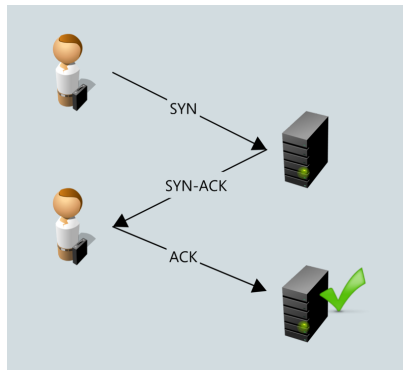
Que se passe-t-il lorsque nous tapons dans la barre d'adresse de Firefox une URL, telle que `http://www.zonensi.fr/` ? Entre le cours de SNT de seconde, et celui de NSI, nous pouvons décrire l'enchaînement des communications ainsi :

1. L'URL est décodée par le navigateur, qui isole :

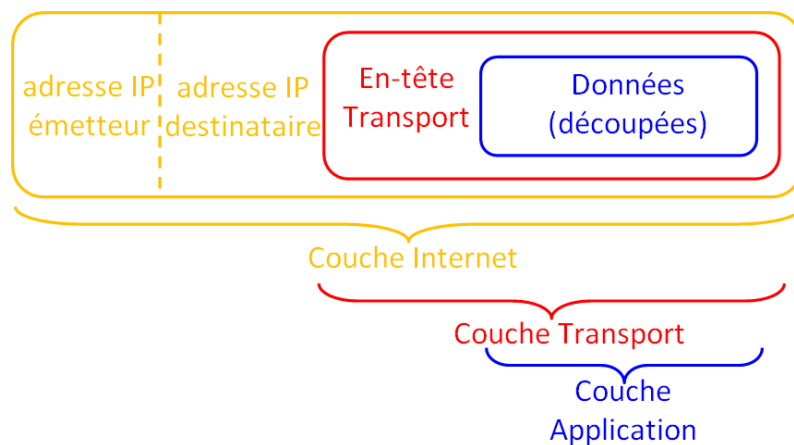
- le protocole utilisé : HTTP
- le nom de domaine : `www.zonensi.fr`
- le chemin vers la ressource : `/`, la racine du site.

2. Le navigateur effectue une résolution de nom, soit en se connectant à un serveur DNS, soit dans son propre cache DNS, ce qui lui donne l'adresse IP de la ressource cherchée.

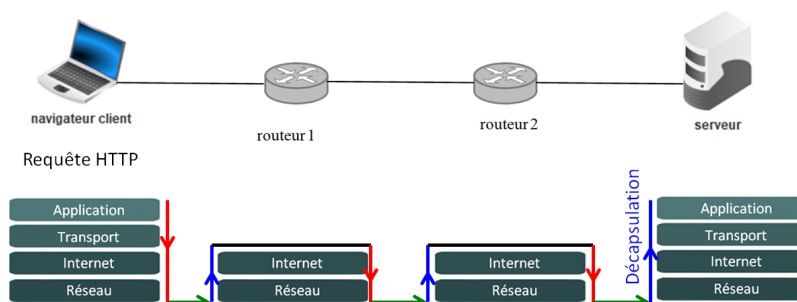
3. Le navigateur peut établir une connexion TCP vers l'adresse IP, via un **handshaking en trois temps**, comme montré sur l'image ci-dessous :



4. Une fois la connexion établie, client et serveur échangent des données en utilisant le protocole HTTP, tout en découpant les données en paquets TCP, eux-mêmes encapsulés dans des paquets IP (on pourra se rappeler du modèle OSI).



5. Les paquets sont transmis de routeurs en routeurs, en étant à chaque fois désencapsulés pour inscrire l'adresse IP du prochain routeur.



⚠ Des limites

Ce système, en place depuis l'invention de TCP/IP dans les années 1970, a vu l'intégration de chaque nouveau protocole (FTP, SMTP, etc) au sein de la couche application. Mais avec la démocratisation d'Internet, des problèmes sont rapidement apparus : si on utilise tel quel le modèle écrit ci-dessus pour effectuer des opérations bancaires ou échanger des données confidentielles, on se rend compte qu'un grand nombre d'intermédiaires (en particulier les routeurs) sont en possibilité de lire les données transmises.

On souhaite donc sécuriser les connexions afin que seul l'émetteur et le destinataire puissent avoir connaissance du contenu. D'où un questionnement sur trois aspects :

- Comment chiffrer le contenu des communications afin qu'elles ne soient lisibles que par la source et la destination (garantie de **confidentialité**) ?
- Comment garantir que le serveur auquel on se connecte est bien celui auquel on pense se connecter (garantie d'**authenticité**) ?
- Comment s'assurer que le message transmis n'a pas été modifié par un tiers (garantie d'**intégrité**) ?

Le tout devant bien entendu se faire dans le cadre d'une communication en utilisant l'infrastructure d'Internet, à savoir les communications TCP/IP ?

2. Quelques définitions nécessaires

🔗 Coder/Décoder

Coder, c'est représenter l'information par un ensemble de signes prédéfinis. **Décoder**, c'est interpréter un ensemble de signes pour en extraire l'information qu'ils représentent.

Coder et décoder s'emploient lorsqu'il n'y a pas de secret. Par exemple on peut coder/décoder des entiers relatifs par une suite de bits par un «codage en complément à deux».

🔗 Cryptographie

La **cryptographie** est une discipline veillant à protéger des messages (pour en assurer la confidentialité, l'authenticité et l'intégrité), par l'intermédiaire de **clés de chiffrements**.

La cryptographie est utilisée depuis au moins l'antiquité.

Son pendant est la **cryptanalyse**, qui est la technique qui consiste à déduire un texte en clair d'un texte chiffré **sans posséder la clé de chiffrement**. Le processus par lequel on tente de comprendre un message en particulier est appelé **une attaque**.

Chiffrer un message, c'est rendre une suite de symboles incompréhensible au moyen d'une **clé de chiffrement**.

Déchiffrer ou **décrypter**, c'est retrouver la suite de symboles originale à partir du message chiffré. On utilise **déchiffrer** quand on utilise la clé de chiffrement pour récupérer le texte original, et **décrypter** lorsqu'on arrive à retrouver le message original sans connaître la clé de chiffrement.

3. Cryptographie symétrique

📄 Cryptographie symétrique

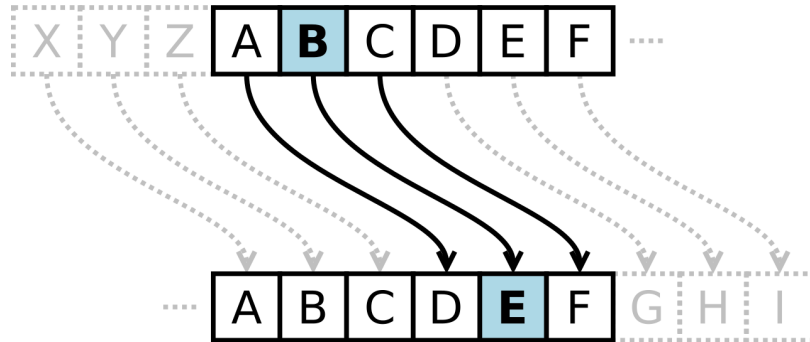
On parle de **cryptographie symétrique** lorsque la même clé est utilisée pour chiffrer et déchiffrer un message.

3.1. Code de César

📄 Le code (ou chiffre) de César : chiffrement par décalage

Le chiffre de César est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes (ce qui explique le nom « chiffre de César »).

Le texte chiffré est obtenu en remplaçant chaque lettre du texte original par une lettre obtenue par un décalage à distance fixe, toujours du même côté, dans l'ordre de l'alphabet.



On a ainsi, pour un chiffre de César avec un clé de 3, les correspondances suivantes :

```
>>> alphabet_clair = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
>>> alphabet_chiff = "DEFGHIJKLMNOPQRSTUVWXYZABC"
```

? Exercice

Afin de pouvoir chiffrer un message avec le code de César, il faut avoir un texte ne comportant que des lettres majuscules de l'alphabet latin, donc nettoyer le texte de tous les accents, espaces, signes de ponctuation, etc.

1. Créer une fonction `formate_texte(texte : str) -> str` qui prend en argument un texte non formaté, et renvoie le texte sans accents, sans signes de ponctuations ni espaces, et en majuscule
2. Créer alors une fonction `code_Cesar(texte :str, cle : int) -> texte` qui renvoie le chiffre de César d'un texte qui lui est passé en argument, avec une clé sous la forme d'un entier entre 1 et 26 qui représente le décalage devant être obtenu. On rappelle les éléments suivants :

```
>>> ord('A')
65
>> chr(66)
'B'
>>> chr(((ord('Z')-65)+3)%26+65)
'C'
```

🔍 Cryptanalyse d'un chiffre de César

Énoncé

Proposer un texte long chiffré au professeur. Combien de temps va-t-il mettre pour décrypter celui-ci ?

Attaque par force brute

Une attaque par force brute consiste à attaquer en testant toutes les possibilités. Avec le chiffre de César, il n'existe que 26 clés différentes, la méthode par force brute est donc particulièrement adaptée, même dans le cas d'un texte long.

```
def brute_force_Cesar(texte : str) -> str :
    decrypte = []
    for k in range(1,26) :
        decrypte.append(code_Cesar(texte, k))
    return decrypte
```

Attaque par analyse de fréquences

Une étude des textes écrits en français montre que la lettre la plus fréquemment utilisée est «E» (minuscule ou majuscule). Si on sait que le texte est écrit en français, il est probable que la lettre la plus fréquente dans le code de César soit celle qui remplace le «E». Il est alors possible de tester en premier la clé correspondant à ce décalage.

```
def get_freq_texte(texte : str) -> dict :
    dico = dict()
    for letter in texte :
        if letter in dico :
            dico[letter] += 1
        else :
            dico[letter] = 1
    return dico

def attaque_Cesar_analyse_frequence(texte : str) -> str:
    dico = get_freq_texte(texte)
    max_dico = []
    maxi = 0
    for letter in dico :
        if dico[letter]> maxi :
            max_dico=[letter]
            maxi = dico[letter]
        elif dico[letter] == maxi :
            max_dico.append(letter)
    possible_keys = [ord(k)-ord('E') for k in max_dico]
    possible_texts = [ f"Clé {k} :\n\n"+code_Cesar(crypto, 26-k) for k in possible_keys]
    return possible_texts
```

🔍 Chiffre de Vigenère



3.2. Chiffrement XOR

☰ XOR (OU exclusif)

L'opérateur binaire XOR, où *OU Exclusif*, noté \oplus , est un opérateur dont la table de vérité est :

\oplus	0	1
0	0	1
1	1	0

L'opérateur XOR, en plus d'être *commutatif* ($A \oplus B = B \oplus A$) possède la propriété de **réversibilité**, ce qui signifie que si $A \oplus B = C$, alors on a les égalités suivantes :

- $A \oplus C = B$
- $B \oplus C = A$

☰ Chiffrement XOR

Étant donné un message, par exemple «UN MESSAGE TRÈS SECRET», et une clé de chiffrement, par exemple «NSI», on recopie plusieurs fois la clé sous le message :

```
UN MESSAGE TRÈS SECRET
NSINSINSINSINSINSINSIN
```

Chaque caractère du message est associé à une valeur numérique entière (par exemple son Unicode) :

```
85 78 32 77 69 83 83 65 71 69 32 84 82 200 83 32 83 69 67 82 69 84
78 83 73 78 83 73 78 83 73 78 83 73 78 83 73 78 83 73 78 83 73 78
```

On effectue ensuite l'opération \oplus entre chaque nombre du message et de la clé. par exemple pour le premier caractère :

$$\begin{array}{r}
 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \quad 85 \\
 \oplus \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \quad 78 \\
 \hline
 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \quad 27
 \end{array}$$

Le code obtenu dans notre exemple est donc :

```
27 29 105 3 22 26 29 18 14 11 115 29 28 155 26 110 0 12 13 1 12 26
```

L'opérateur \oplus étant réversible, la répétition de l'opération sur le message chiffré rendra le message original.

🔍 Chiffré/déchiffré

1. Créer une fonction `get_unicode(chaine : str) -> list` qui prend en argument une chaîne de caractère et renvoie la liste des codes unicode correspondant aux caractères.
2. Créer une fonction `get_string(liste : list) -> str` qui fait l'opération inverse de la fonction `get_unicode`.
3. Créer une fonction `chiffre_XOR(texte : list, cle : list) -> list` qui renvoie une liste de valeurs entières correspondant à l'application d'un XOR sur chacun des valeurs des deux listes `texte` et `cle`.

Indications :

- l'opérateur \oplus en python s'écrit de la manière suivante :

```
>> 85 ^ 78
27
```

- Il n'est pas nécessaire de créer une liste de la même longueur que le texte avec la clé. Une utilisation judicieuse de l'opération modulo doit vous permettre de vous en sortir.
4. Vérifiez que la fonction `chiffre_XOR` permet bien de chiffrer/déchiffrer un texte avec une clé donnée.

Les caractéristiques de l'opérateur XOR, et le fait qu'il puisse être implémenté directement dans le processeur, font qu'il est souvent utilisé dans les algorithmes de chiffrement modernes, comme AES ou ChaCha20. Bien sûr, ces algorithmes sont nettement plus complexes que la méthode naïve que nous avons utilisée, mais leurs principes reposent sur des fonctionnements similaires.

En plus d'être relativement sûrs (voir ci-dessous), ces algorithmes sont très efficaces et permettent de chiffrer très rapidement. On peut ainsi chiffrer en direct des communications audio ou vidéo en temps réel.

⚠️ Cryptanalyse : attention à la longueur de la clé !

Une clé trop courte peut compromettre la sécurité des données : dans le cas où un mot du message peut-être envisagé, et où la clé est de taille raisonnable (en pratique dans le code suivant, de taille maximale de 4), il est tout à fait possible de faire une attaque par force brute :

```
def all_possible(length : int) :
    """ crée une liste de toutes les clés possible de longueur donnée """
    if length == 0 :
        return ['']
    poss = []
    disp = all_possible(length-1)
    for uni in range(65, 65+26) :
        for d in disp :
            poss.append(chr(uni)+d)
    return poss

def cryptanalyse_XOR(chiffre : list, contain : str, taille_cle : int) :
    """ renvoie les clés possibles qui trouvent la chaîne contain dans le code chiffre, en testant toutes
    les clés possibles de taille taille_cle """
    poss_keys = []
    for k in all_possible(taille_cle) :
        decode = get_string(chiffre_XOR(chiffre, get_unicode(k)))
        if contain in decode :
            poss_keys.append(k)
    return poss_keys
```

⚠ Un point sur les mots de passe : entropie de Shannon

En informatique, la robustesse d'un mot de passe *aléatoire* est exprimée en terme d'entropie de Shannon, mesurée en bits.

D'après [wikipedia](#), « au lieu de mesurer la robustesse par le nombre de combinaisons de caractères qu'il faut tester pour trouver le mot de passe avec certitude, on utilise le logarithme en base 2 de ce nombre. Cette mesure est appelée l'entropie du mot de passe. Un mot de passe avec une entropie de 42 bits calculée de la sorte serait aussi robuste qu'une chaîne de 42 bits choisie au hasard.

En d'autres termes, un mot de passe de 42 bits de robustesse ne serait brisé de façon certaine qu'après $2^{42} = 4398046511104$ tentatives lors d'une attaque par force brute. L'ajout d'un bit d'entropie à un mot de passe double le nombre de tentatives requises, ce qui rend la tâche de l'attaquant deux fois plus difficile.»

L'entropie d'un mot de passe de taille L utilisant des caractères parmi N possibilités aura une entropie H calculée de la manière suivante :

$$H = L \cdot \log_2(N) = L \cdot \frac{\ln(N)}{\ln(2)}$$

Ce qui donne les résultats suivants :


Nombre de symboles	A-Z (26)	a-zA-Z(52)	a-zA-Z0-9 (62)	a-zA-Z0-9,;!... (95)
6 caractères	28	34	35	39
10 caractères	47	57	59	66
12 caractères	56	68	71	79
20 caractères	94	114	119	131

On constate donc qu'un mot de passe de 10 caractères latin majuscules est plus «résistant» qu'un mot de passe de 6 caractères utilisant n'importe quel symbole du clavier français... Cela signifie que le nombre de caractère est nettement plus important que la diversité de ceux-ci. On peut le voir grâce au tableau suivant :

**COMBIEN DE TEMPS FAUT-IL À UN PIRATE
POUR TROUVER VOTRE MOT DE PASSE 2024**

www.hivesystems.com/password

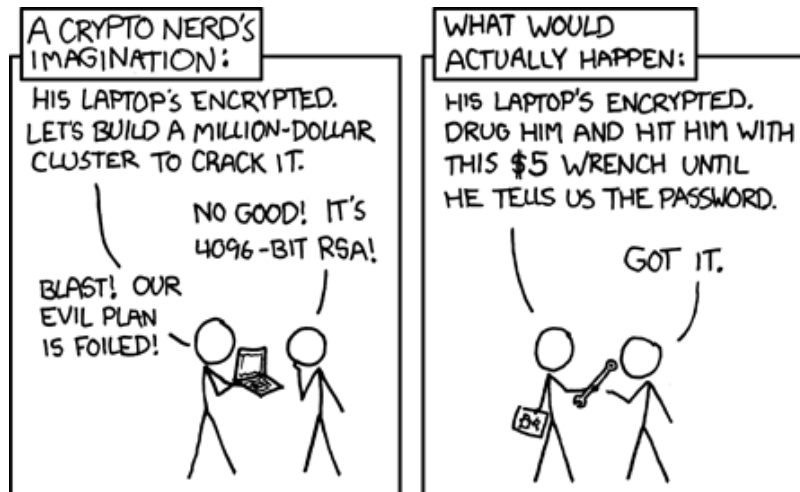
Nombre de caractères	Nombres seulement	Lettres minuscules	Lettres majuscules et minuscules	Nombres, lettres majuscules et minuscules	Nombres, lettres majuscules et minuscules, symboles
4	Immédiat	Immédiat	3 secs	6 secs	9 secs
5	Immédiat	4 secs	2 mins	6 mins	10 mins
6	Immédiat	2 mins	2 heures	6 heures	12 heures
7	4 secs	50 mins	4 jours	2 semaines	1 mois
8	37 secs	22 heures	8 mois	3 ans	7 ans
9	6 mins	3 semaines	33 ans	161 ans	479 ans
10	1 heure	2 ans	1k ans	9k ans	33k ans
11	10 heures	44 ans	89k ans	618k ans	2M ans
12	4 jours	1k ans	4M ans	38M ans	164M ans
13	1 mois	29k ans	241M ans	2Md ans	11Md ans
14	1 an	766k ans	12Md ans	147Md ans	805Md ans
15	12 ans	19M ans	652Md ans	9Bn ans	56Bn ans
16	119 ans	517M ans	33Bn ans	566Bn ans	3qd ans
17	1k ans	13Md ans	1qd ans	35qd ans	276qd ans
18	11k ans	350Md ans	91qd ans	2qn ans	19qn ans

 > 12 x RTX 4090 | bcrypt

Vous pouvez calculer l'entropie de vos mots de passe sur le [site suivant](#).

Gardez toutefois en tête que la sécurité est maximale lorsque vous utilisez des mots de passe aléatoires de longueur suffisante, utilisant le maximum de caractères, et différents pour chaque site. Pour aider à retenir tous ces mots de passe, l'utilisation d'un gestionnaire de mots de passe, tel que [KeyPass](#) est nécessaire. celui-ci peut-être protégé grâce à une **Pass-Phrase**, c'est à dire une phrase composée de mots (aléatoires de préférence), garantissant une grande difficulté de décryptage.

Et le mot de la fin sera pour [xkcd](#)



4. Cryptographie asymétrique

Le gros problème des cryptographies symétriques est le suivant : **les deux protagonistes de l'échange doivent connaître la clé, et donc se l'échanger**. Or ils n'ont pas de moyens de communications sécurisés pour l'instant. Il leur reste donc deux solutions :

- soit ils échangent la clé par un moyen de communication non sécurisé, comme des mails ou du courrier, mais un attaquant pourrait alors s'emparer de la clé et compromettre la sureté des communications futures ;
- soit ils échangent la clé par un moyen plus sûr, mais moins pratique (sur un pont isolé par une nuit sans lune dans une mallette menottée au poignet, comme dans les films noirs des années 1950).

Pour résoudre ce problème, les scientifiques américains et britanniques dans les années 1970, puis la recherche académique publique, se sont tournés vers la **cryptographie asymétrique**. Il s'agit de méthodes utilisant des techniques de mathématiques avancées, dont on ne présentera pas ici les véritables tenants et aboutissants. On peut cependant présenter quelques méthodes et en expliquer sommairement le fonctionnement.

4.1. Les puzzles de Merkle

La méthode du puzzle de Merkle, créé en 1974 et pour la première fois publiée en 1978, est la première méthode de chiffrement asymétrique (**non top secrète**) à clé publique.

🔗 Déroulé d'un échange

Voici les étapes de la méthode des puzzles de Merkle, qui permette à Alice et Bob d'échanger des messages sans qu'Eve (diminutif de *eavesdropper*, ou oreille indiscreète, espion) puisse décrypter les messages :

Etape 1

Alice génère un fichier de très grande taille, par exemple 100 000 lignes, où chaque ligne consiste en un **identifiant unique** et une clé de longueur suffisante :

```
...
Id : 345768 Key : p(;;9a"ZMBz53P<6C5Q3
Id : 768453 Key : 8uQw(;e3SRHaN=]QsFp%
Id : 108943 Key : >ye5JH@%,%%J6<FsGWE,
...
```

Elle crypte ce fichier avec un chiffre XOR, mais en respectant les consignes suivantes :

- chaque ligne est chiffrée avec une clé différente ;
- les clés utilisées sont de petites tailles.

Elle transmet ensuite le fichier à Bob.

Le fichier est probablement intercepté par Eve.

Etape 2

Bob reçoit le message chiffré d'Alice. Il choisit au hasard une des lignes, et l'**attaque par force brute**. Comme la clé utilisée est de petite taille, l'attaque est possible en un temps raisonnable, disons 10 minutes.

Bob récupère donc une ligne avec un identifiant et une clé.

```
Id : 768453 Key : 8uQw(;e3SRHaN=]QsFp%
```

Bob transmet alors **en clair** l'identifiant 768453 à Alice.

Eve intercepte probablement cet identifiant.

Etape 3

Alice regarde dans son fichier non crypté la clé correspondant à l'identifiant transmis : 8uQw(;e3SRHaN=]QsFp%. Avec cette clé, la communication s'engage entre Alice et Bob en **utilisant un chiffrement symétrique**.

A aucun moment la clé n'a été transmise en clair entre les deux protagonistes, qui n'ont pas eu besoin de se rencontrer non plus pour entamer une communication sécurisée.

Et Eve ?

Eve a donc en sa possession un fichier crypté de 100 000 lignes, et un identifiant en clair. Mais pour faire correspondre cet identifiant à une clé, il faut qu'elle décrypte par force brute chacune des lignes du fichier jusqu'à trouver le bon identifiant. Pour décrypter la totalité du fichier, il lui faudra donc 100 000 fois 10 minutes, soit près de 12 jours. Donc, en moyenne, Alice et Bob peuvent communiquer sereinement pendant 6 jours avec la même clé.

4.2. Méthode de Diffie-Hellman

La méthode des puzzles de Merkle, bien que novatrice pour son époque, n'est plus jugée suffisante de nos jours pour garantir une véritable sécurité. Cependant, elle a posé les bases d'autres méthodes, comme la méthode de Diffie-Hellman, proposée en 1974 par les cryptologues américains Bailey W. Diffie et Martin Hellman.

Cette méthode repose sur l'utilisation d'une fonction mathématique à deux variables. Cette fonction, souvent nommée M (pour «mélange»), doit respecter les propriétés suivantes :

1. M est connue, ce qui signifie qu'on connaît l'algorithme ou la formule qui permet de calculer des images (toutes les fonctions ne sont pas calculables, voir [théorie de la calculabilité](#)).
2. Si on connaît $M(x; y)$ et x , il doit être **très difficile** de retrouver y . Par difficile, on entend le fait que pour trouver le y donnant à $M(x; y)$ une valeur donnée, il faudra essayer sur tous les entiers y possibles.
3. Pour tous entiers x, y et z , on a $M(M(x; y); z) = M(M(x; z); y)$, autrement dit y et z sont commutables.

Une analogie couramment utilisée pour expliquer le fonctionnement de cette fonction M est celle des pots de peinture :

Pots de Peintures

Les images suivantes ont été extraites et modifiées depuis [Idée originale : A.J. Han Vinck](#) [Version vectorielle : Flugaal](#) [Traduction : Dereckson](#), Public domain, via Wikimedia Commons

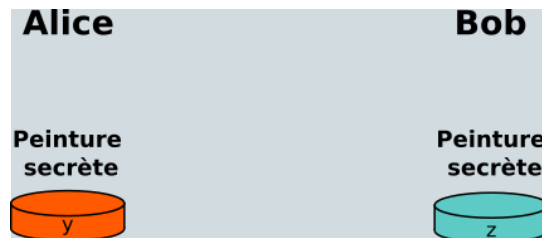
Etape 1

On dispose d'un très grand nombre de pots de peinture de couleurs différentes. Alice et Bob se mettent d'accord pour choisir une couleur commune x , qui sera «publique», puisqu'elle peut être interceptée.



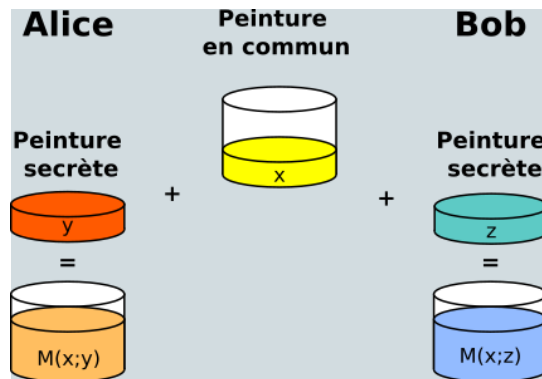
Etape 2

Alice et Bob choisissent alors chacun une couleur, respectivement y et z , qui resteront privées et secrètes et ne seront jamais échangées.



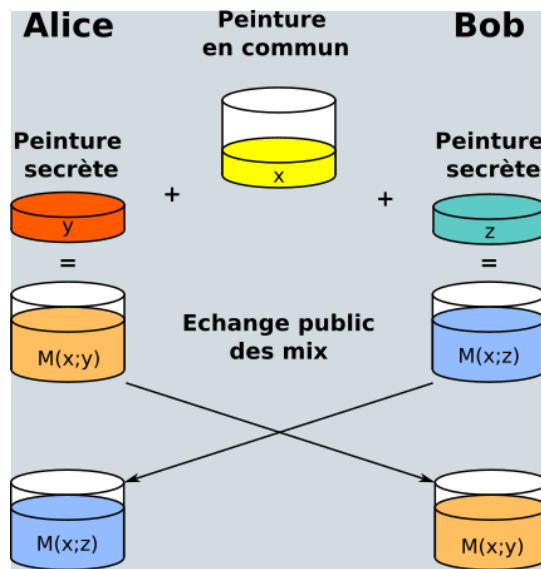
Etape 3

Alice et Bob élaborent alors leurs mélanges, en utilisant la couleur commune et leur propre couleur privée.



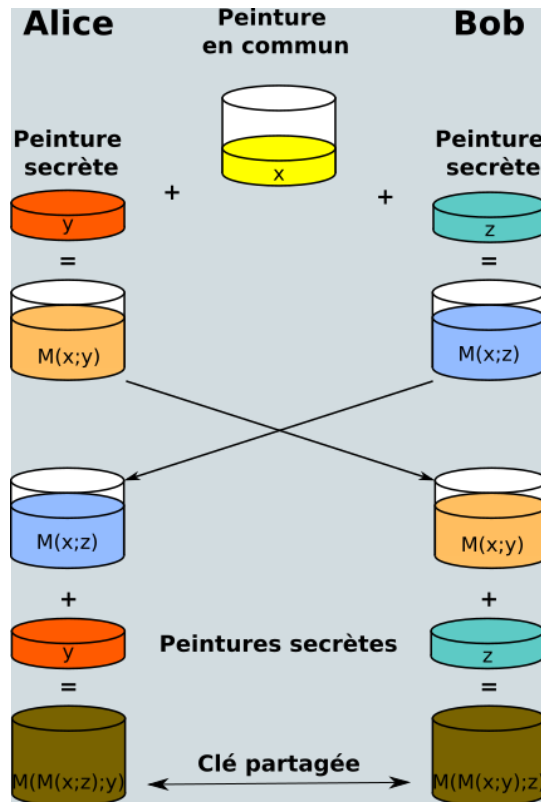
Etape 4

Alice et Bob échangent *en clair* leurs mélanges respectifs, qui peuvent donc être interceptés.



Etape 5

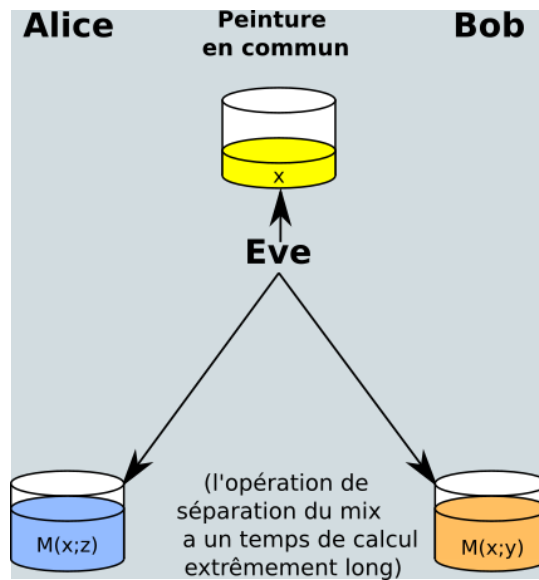
Alice et Bob ajoutent alors au mélange qu'ils ont reçu leur couleur secrète. La fonction de mélange est construite afin que les mix obtenus par Alice et Bob soient identiques. Ils ont ainsi une «couleur» commune secrète, qui peut alors leur permettre d'effectuer des échanges via un cryptage symétrique.



Et Eve ?

Eve peut connaître trois choses : la couleur commune, et chacun des mix ayant circulé en clair. Pour autant, la fonction de mélange est faite de telle manière qu'il soit extrêmement long et difficile d'extraire les couleurs secrètes même si la couleur commune est connue.

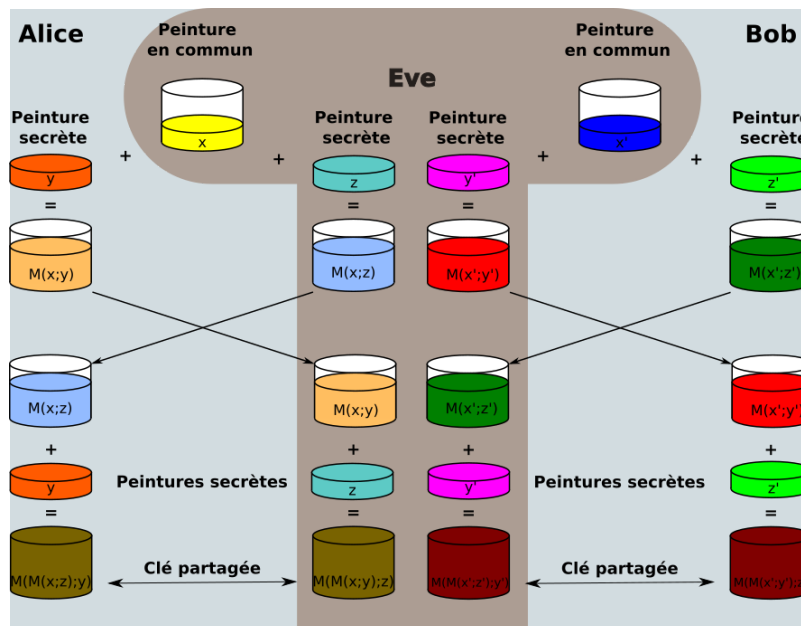
Par ailleurs, même en mélangeant les deux mix obtenus, on n'obtiendra pas la même couleur que celle obtenue par ALice et Bob, puisque la couleur commune sera deux fois plus présente.



Le protocole d'échange de clé de Diffie-Hellman propose donc une manière élégante de régler le problème d'échange de clé posé par le chiffrement symétrique. Cependant, un problème reste à régler, il s'agit du **problème de l'authentification** : la sûreté des communications dépend essentiellement sur le fait qu'Alice et Bob soient certains de communiquer avec la bonne personne.

⚠ Attaque de l'homme du milieu

Imaginons qu'Eve ne se contente pas d'intercepter la couleur commune, mais qu'elle se fasse passer pour Bob auprès d'Alice, et d'Alice auprès de Bob. Eve peut alors choisir une couleur commune avec Alice, et une autre avec Bob. Le protocole d'échange se poursuit normalement, mais Eve intercepte et décode chaque message transmis, avant de le retransmettre à son tour, modifié ou non :



Une telle technique s'appelle une **attaque de l'homme du milieu**, ou **man in the middle attack**, souvent abrégée en MITM.

4.3. Cryptage RSA

Système RSA

Le système RSA est un système de chiffrement asymétrique basé sur des paires de clés publiques et privées, pour la première fois publié en 1978. Son nom provient des initiales de ses trois inventeurs : **Ron Rivest, Adi Shamir et Len Adelman**.

Les mathématiques derrière le système RSA utilisent entre autres les congruences sur les entiers et le petit théorème de Fermat. Tous les calculs se font modulo un nombre entier n qui est le produit de deux nombres premiers, en général très grands, car les messages clairs et chiffrés sont des entiers inférieurs à l'entier n . Les opérations de chiffrement et de déchiffrement consistent à élever le message à une certaine puissance modulo n , ce qui donne des calculs très coûteux.

Globalement, le système consiste en la mise en place d'une **paire de clés publiques et privées** pour chaque participant :

- Alice possède une clé K_A^{pub} et une clé privée K_A^{pri} .
- Bob possède une clé K_B^{pub} et une clé privée K_B^{pri} .

On notera $K_P^x(m)$ le fait de chiffrer un message avec la clé x de la personne P .

La manière exacte de créer ces clés est complexe, mais l'essentiel est de comprendre que l'utilisation des deux clés d'une personne permet de déchiffrer un message. Par exemple pour Alice :

$$K_A^{pub} \left(K_A^{pri} (m) \right) = K_A^{pri} \left(K_A^{pub} (m) \right) = m$$

Ce qui signifie qu'un message chiffré avec la clé publique d'Alice peut être déchiffré avec sa clé privée, et réciproquement.

D'autre part, les propriétés des clés font que :

- il est impossible en connaissant K_A^{pub} de deviner K_A^{pri} ;
- il est impossible en connaissant $K_A^{pub}(m)$ ou $K_A^{pri}(m)$ de deviner m .

Fonctionnement d'une communication

Si Bob veut envoyer un message secret à Alice, les deux procèdent comme suit :

1. Alice met à disposition sa clé publique K_A^{pub} , en la mettant par exemple sur son site web ou en l'envoyant par mail.
2. Bob chiffre son message m avec la clé publique d'Alice et envoie le résultat $K_A^{pub}(m)$ à Alice.
3. Alice applique sa clé privée sur le message reçu $K_A^{pri} \left(K_A^{pub}(m) \right) = m$, et déchiffre ainsi le message de Bob.

L'inconvénient majeur de RSA est que les **chiffrements et déchiffrements sont très coûteux en temps de calcul**, et ne permettent pas des échanges sur des gros volumes de données, ou sur des flux de communications audio ou vidéo.

Cependant, il est possible d'utiliser RSA à la manière de Diffie-Hellman, afin d'échanger une clé pour un algorithme de chiffrement symétrique, qui est en général un fichier de quelques milliers de bits.

Un autre des avantages de RSA est qu'il est possible de l'utiliser comme **système d'authentification**.

4.4. Certificats et tiers de confiance

En France, l'État délivre aux citoyens une carte d'identité. Lorsque une personne se présente au bureau de poste pour retirer un colis, son identité est vérifiée par la personne au guichet par l'intermédiaire de cette carte d'identité. A priori, un bout de carton plastifié à lui seul ne permet pas de garantir réellement une authentification. Le système fonctionne parce que le bureau de poste **fait confiance** à l'État, qui a fait les vérifications nécessaires pour s'assurer de l'identité de la personne, et qui a mis en place une carte difficile à falsifier. L'État joue ici le rôle d'un **tiers de confiance**.

On retrouve le même système dans les communications sur Internet, où certains acteurs jouent le rôle de tiers de confiance. Ils fournissent des **certificats** numériques, créés à partir des clés RSA publiques des participants.

Exemple d'authentification

Imaginons que Bob veuille s'assurer que c'est bien Alice avec qui il va entrer en communication, via son site web.

1. Alice fait appel à Thierry, un tiers de confiance. Thierry vérifie qu'Alice est bien la propriétaire du site, en constatant qu'elle peut administrer le site, ou bien par l'intermédiaire de factures montrant qu'elle possède le nom de domaine ainsi que le serveur qui héberge le site. Une fois ces vérifications effectuées, Thierry crée un certificat avec sa clé privée et la clé publique d'Alice :

$$c = K_T^{pri}(K_A^{pub})$$

2. Quand Bob se connecte au site d'Alice, celui-ci envoie le certificat c et la clé publique d'Alice K_A^{pub} .
3. Bob se sert alors de la clé publique de Thierry sur le certificat :

$$K_T^{pub}(c) = K_T^{pub}(K_T^{pri}(K_A^{pub})) = K_A^{pub}$$

Il compare le résultat avec la clé que lui a fourni le site d'Alice. Si il y a correspondance, il est assuré d'être en communication avec Alice. Il peut alors démarrer un échange de clé soit en utilisant RSA, soit Diffie-Hellman.

5. Le protocole HTTPS

5.1. Autorités de certifications

Une **autorité de certification** (ou AC), est une entité habilitée à délivrer des certificats. Il s'agit de tiers de confiance, que l'on peut classer en trois catégories :

- les entreprises spécialisées ;
- les associations (comme *Let's Encrypt*, que nous croiserons plus tard);
- les entités étatiques.

Leur rôle est d'attester par l'intermédiaire de certificat qu'une entité est bien ce qu'elle prétend être. Elles sont soumises à des audits réguliers et pointilleux, dont les résultats sont publics, et il existe une hiérarchie des AC. En effet, une AC doit être elle-même certifiée par une autre AC, ce qui crée un arbre de certification jusqu'à des AC appelées **AC racines**. La fondation Mozilla reconnaît à ce jour (17 Mai 2024) [142 AC racines](#).

Le club des AC est donc très fermé, les OS et navigateurs ayant chacun les clés publiques des AC qu'ils reconnaissent (au passage, Google, Microsoft et Apple ne reconnaissent pas exactement les mêmes AC que Mozilla).

5.2. Normes de certifications X.509

Les AC suivent généralement le format standard de certificat qui est à l'heure actuelle le format **X.509**. Il s'agit d'un format de fichier binaire contenant entre autre :

- l'identifiant de l'AC qui **signe**(chiffre) le certificat ;
- les dates de validité du certificat (dates de départ et dates de fin);
- l'identité de l'entité certifiée ;
- la clé publique de l'entité certifiée ;
- l'algorithme utilisé pour la signature du certificat ;
- la signature du certificat par l'AS.

On retrouve la construction présentée précédemment, mais avec quelques points techniques supplémentaires : plutôt que de vérifier la signature de chaque ligne du fichier, on utilise une **fonction de hachage** qui génère une **somme de contrôle** du fichier. Cette

somme de contrôle est alors chiffrée avec la clé privée de l'AC.

Zonensi.fr

Voici un extrait du certificat de zonensi.fr :

```
Certificate:
  Data:
    Version: 3 (0x2)
    [...]
    Issuer: C = US, O = Let's Encrypt, CN = R3
    Validity
      Not Before: May 15 22:38:03 2023 GMT
      Not After : Aug 13 22:38:02 2023 GMT
    Subject: CN = zonensi.fr
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:ae:43:ac:a5:ae:80:f4:38:4c:52:32:f7:...
      Exponent: 65537 (0x10001)
    [...]
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
      59:17:d1:ff:e2:2f:1f:a1:e5:2f:71:b6:e3:4a:4d:e4:...
```

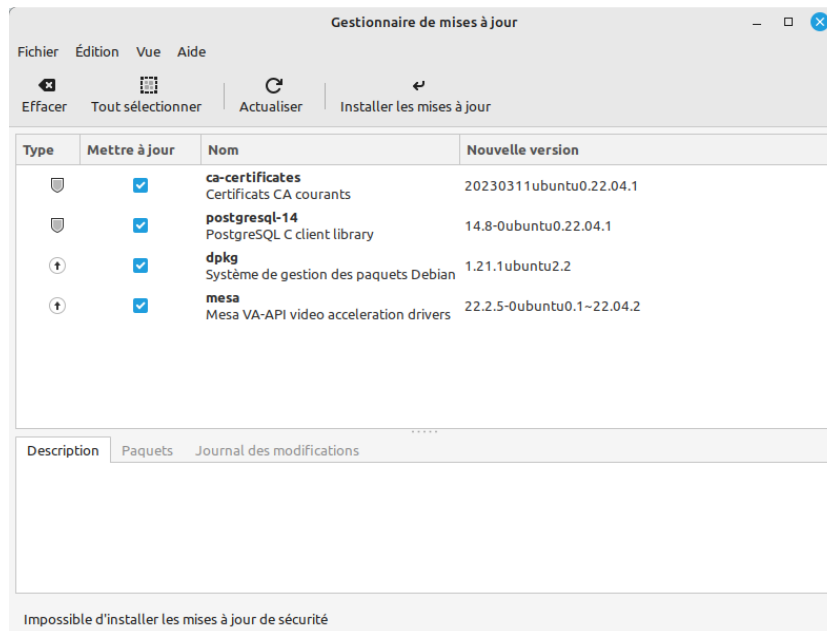
On constate donc que :

- l'AC qui a signé le certificat est [Let's Encrypt](#) ;
- celui-ci est valide du 15 mai 2023 au 13 Aout 2023 ;
- il certifie le domaine `zonensi.fr` ;
- la clé publique est une clé RSA, sur 2048 bits, et calculée à partir des informations `Modulus` et `Exponent` .
- l'algorithme utilisé pour créer la somme de contrôle du fichier est `sha256WithRSAEncryption` ;

Quand vous vous connectez à `zonensi.fr` en `https`, votre navigateur :

- récupère le certificat ;
- la signature est le résultat du chiffage de la somme de contrôle par la clé privée de l'AC.
- retire les deux dernières lignes et utilise l'algorithme `sha256WithRSAEncryption` pour calculer la somme de contrôle du certificat ;
- utilise la clé publique de *Let's Encrypt* sur la signature du certificat, et compare le résultat à la somme de contrôle calculée ;
- en cas d'égalité, l'identité est vérifiée et on peut commencer une transaction asymétrique.

Les certificats des AC sont régulièrement mis à jour, aussi bien au niveau du système d'exploitation que du navigateur : ici on voit une mise à jour des certificats d'AC sur une distribution LinuxMint :



5.3. HTTP+ SSL/TLS = HTTPS

📁 SSL/TLS

La **Transport Layer Security (TLS)** (et son prédécesseur **Secure Sockets Layer (SSL)**) est un protocole de sécurisation des échanges développé par l'*Internet Engineering Task Force (IETF)* (à la suite de la société Netscape Communication Corporation*, qui a développé SSL pour son navigateur).

On parle parfois de `SSL/TLS` pour désigner indifféremment `SSL` ou `TLS`.

Actuellement les versions les plus utilisées de `TLS` sont `TLS 1.2` (publiée en 2008) et `TLS 1.3` (publiée en 2018).

`TLS` est un protocole se rajoutant entre la couche `TCP` et la couche applicative (`HTML` par exemple). Ses rôles sont :

- de garantir l'authentification du serveur ;
- la confidentialité des données ;
- l'intégrité des données ;
- et de manière optionnelle, l'authentification du client.

Le protocole `HTTPS` est donc la mise en place d'une communication `HTTP` sécurisée par `TLS`, pour laquelle le port de communication est le port 443 à la place du port 80.

Une communication sécurisée par `TLS` est indiquée dans un navigateur par un cadenas dans la barre de navigation. Un clic sur le cadenas vous donnera les informations sur la sécurisation de la connexion. Par exemple pour `https://www.zonensi.fr` :

Informations sur la page - https://zonensi.fr/

Général Médias Permissions Sécurité

Identité du site web

Site web : zonensi.fr

Propriétaire : Ce site web ne fournit pas d'informations sur son propriétaire.

Vérifiée par : Let's Encrypt Afficher le certificat

Vie privée et historique

Ai-je déjà visité ce site web auparavant ? Oui, 76 fois

Ce site web conserve-t-il des informations sur mon ordinateur ? Oui, 208 octets de données de sites Effacer les cookies et les données de sites

Ai-je un mot de passe enregistré pour ce site web ? Non Voir les mots de passe enregistrés

Détails techniques

Connexion chiffrée (clés TLS_AES_128_GCM_SHA256, 128 bits, TLS 1.3)

La page actuellement affichée a été chiffrée avant d'avoir été envoyée sur Internet.

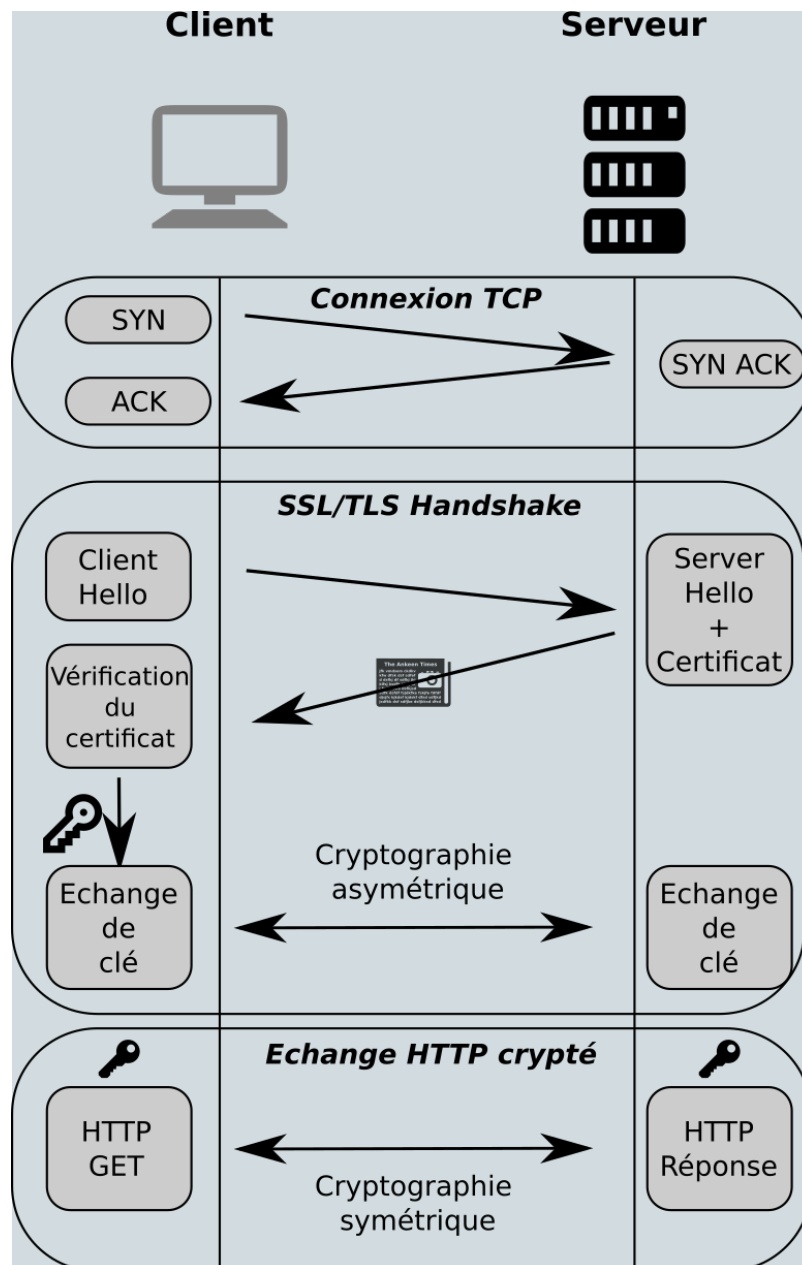
Le chiffrement rend très difficile aux personnes non autorisées la visualisation de la page durant son transit entre ordinateurs. Il est donc très improbable que quelqu'un puisse lire cette page durant son transit sur le réseau.

Aide

Détails de connexion

Autorité de certification

Fonctionnement d'un échange HTTPS



On peut décrire sommairement la brique `SSL/TLS` de la manière suivante :

1. Le client envoie un message `HELLO`, qui contient quelques options (versions de TLS supportée, algorithmes supportés).
2. Le serveur envoie un message `HELLO`, avec ses options, ainsi que son certificat au client.
3. Le client tente de vérifier la signature numérique du certificat, à l'aide des clés publiques des AC. Plusieurs solutions peuvent alors se produire
4. Si l'une d'entre elle fonctionne, le navigateur vérifie quelques autres points (entre autres les dates de validité), puis génère une clé de chiffrement symétrique, appelée **clé de session**, qu'il chiffre avec la clé publique du serveur. Une fois la clé reçue par le serveur, un échange `HTTP` chiffré est donc possible.
5. Si aucune des clés des AC ne fonctionne, le navigateur tente alors de vérifier la signature numérique du certificat avec la clé publique du serveur. En cas de réussite, cela signifie que **le serveur a généré lui-même son certificat**. Un message d'avertissement s'affiche alors sur le navigateur, prévenant que l'identité du serveur n'a pas pu être vérifiée, et qu'il peut s'agir potentiellement d'un site frauduleux (mais il s'agit parfois simplement d'un dépassement de validité du certificat, le propriétaire du site ayant oublié de renouveler son certificat auprès d'une AC).

6. Sources

- http://www.monlyceenumerique.fr/nsi_premiere/archios_arise/a3_encapsulation_tcp_ip.php
- https://fr.wikipedia.org/wiki/Three-way_handshake
- Numérique et Sciences Informatique Tle : 24 leçons avec exercices corrigés, Balabonski, Conchon, Filiâtre, Nguyen, éditions Ellipses
- [Wikipedia](#), pour une très grande partie des informations ci-dessus.