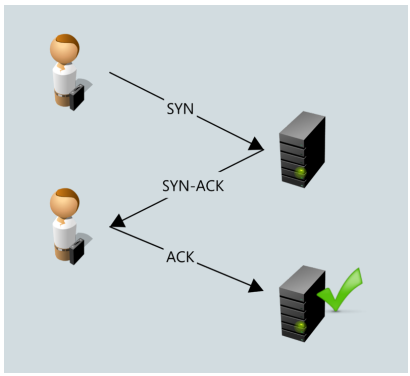
Sécurisation des communications

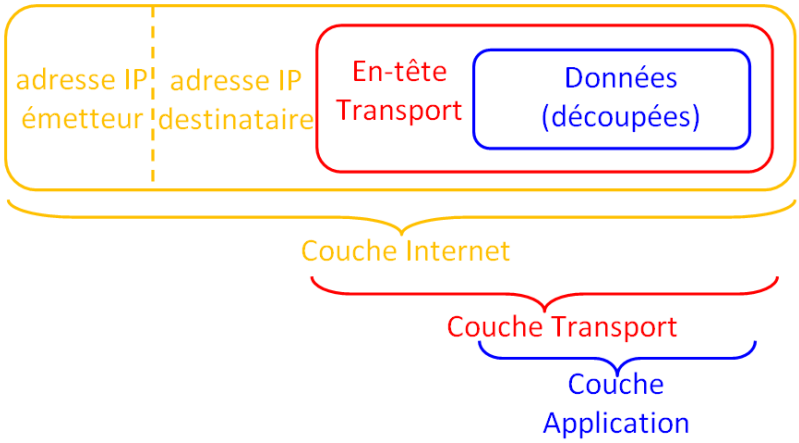
1. Rappels : communication sur internet

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'enchainement des communications ainsi :

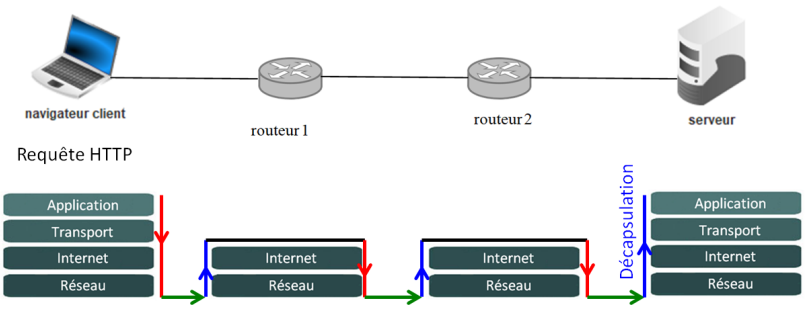
1. L'URL est décodée par le navigateur, qui isole :
   - le protocole utilisé : `HTTP`
   - le nom de domaine : `www.zonensi.fr`
   - le chemin vers la ressource : `/`, la racine du site.
2. Le navigateur effectue une résolution de nom, soit en se connectant à un serveur `DNS`, soit dans son propre cache `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 connaitre la clé de chiffrement.

3. Cryptographie symétrique

Cryptographie symétrique

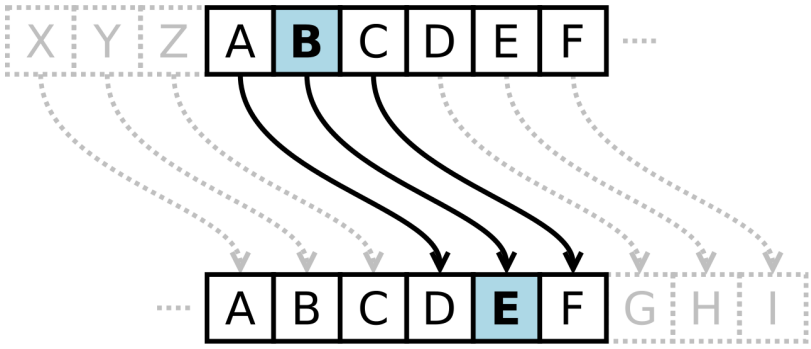
On parle de **cryptographie symétrique** lorsque la même clé est utilisée pour chiffrer et déchiffrer un message.

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

Enoncé

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

Le chiffre de César est donc très facilement déchiffrable, comme tout chiffrement monoalphabétique (où une lettre est toujours remplacée par le même symbole). Il existe d'autres méthodes de chiffrement par substitution, dits polyalphabétiques, pour lesquels la même lettre n'est pas forcément toujours remplacée par le même symbole, tels que le chiffre de Vigenère. Cependant de telles méthodes ne résiste pas forcément à des attaques par analyse de fréquences.

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 *commutatitf* ($A \oplus B = B \oplus A$) 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
NSINSINSINSINSINSIN
```

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 :

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

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 réitération 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 chaine 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 chaine 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 chaine 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 242 (4 398 046 511 104) 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 :

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

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 :

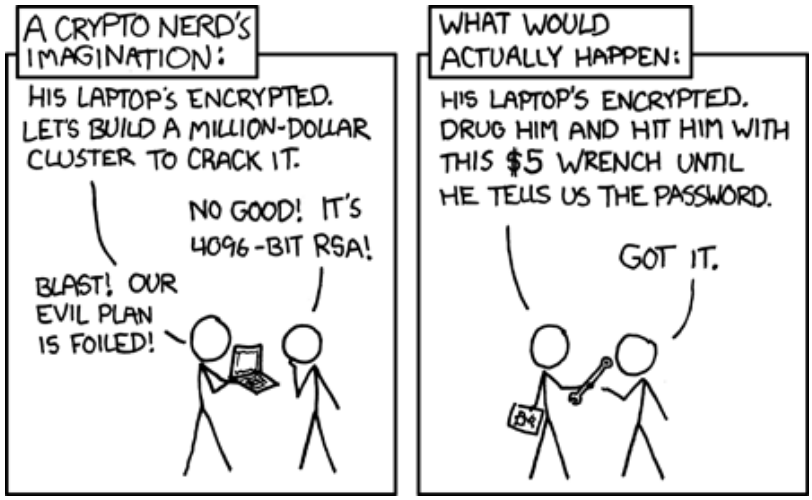
NOMBRE DE CARACTÈRES	UNIQUEMENT DES CHIFFRES	LETTRES MINUSCULES	LETTRES MINUSCULES ET MAJUSCULES	LETTRES MINUSCULES ET MAJUSCULES + CHIFFRES	LETTRES MINUSCULES ET MAJUSCULES + CHIFFRES + CARACTÈRES SPECIAUX
4	IMMÉDIATEMENT	IMMÉDIATEMENT	IMMÉDIATEMENT	IMMÉDIATEMENT	IMMÉDIATEMENT
6	IMMÉDIATEMENT	IMMÉDIATEMENT	IMMÉDIATEMENT	1 sec	5 sec
8	IMMÉDIATEMENT	5 sec	22 min	1 heure	9 heures
10	IMMÉDIATEMENT	58 min	1 mois	7 mois	5 ans
12	45 sec	3 semaines	300 ans	2000 ans	34 000 ans
14	41 min	51 ans	800 000 ans	9 millions d'années	200 millions d'années

*source : SCSP Community (Seasoned Cyber Security Professionnals)

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 Cozy Pass 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 connaitre 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 sûreté 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 indiscrète, 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 probalement 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 puzzle 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-Helman, 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 connait 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 connait $M(x;y)$ 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 sont tirées de Idée originale : A.J. Han VinckVersion vectorielle : FlugaalTraduction : Dereckson, Public domain, via Wikimedia Commons

Mise en place

On dispose d'un très grand nombre de pots de peinture de couleurs différentes

4.3. Cryptage RSA

5. Authentification des participants

6. Sources

http://www.monlyceenumerique.fr/nsi_premiere/archios_arse/a3_encapsulation_tcp_ip.php

https://fr.wikipedia.org/wiki/Three-way_handshake