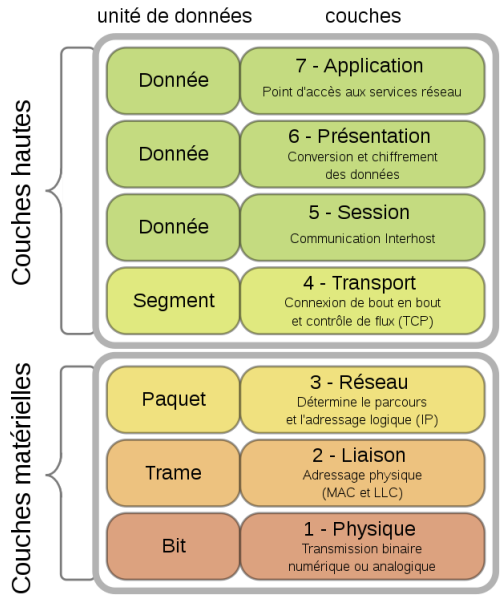
Chapitre :
Architecture et réseaux

I. Protocoles de routage

1. Rappels de première : le modèle OSI

Le modèle OSI (Open System Interconnection) est le modèle de référence pour l'interconnexion de systèmes ouverts. Le principe est la description des réseaux sous forme d'un ensemble de couches superposées les unes aux autres. L'étude du « tout » est réduit à celle de ses « parties », l'ensemble est donc plus facile à manipuler.



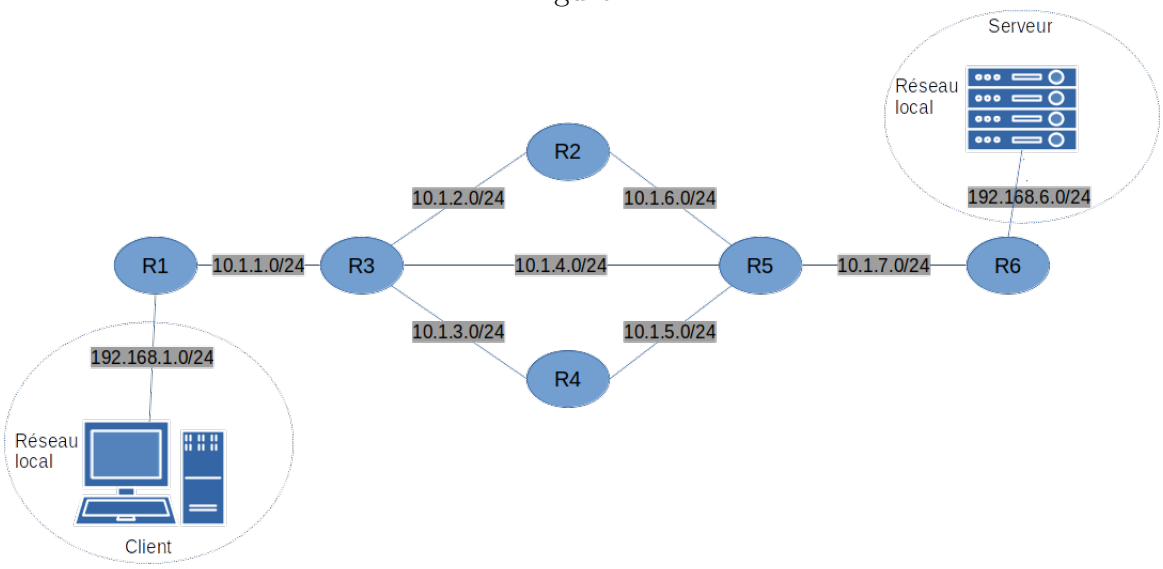
1. La couche « physique » est chargée de la transmission effective des signaux (émission/réception des bits).
2. La couche « liaison de données » gère les communications entre deux machines directement connectées entre elles.
3. La couche « réseau » gère les communications de proche en proche, généralement entre machines : routage et adressage des paquets.
4. La couche « transport » gère les communications de bout en bout entre processus.
5. La couche « session » gère la synchronisation des échanges et les « transactions ».
6. La couche « présentation » est chargée de la conversion entre données manipulées et chaînes d'octets transmises.
7. La couche « application » est le point d'accès aux services réseaux.

2. Routage

a. Topologie d'un réseau

Le **routage** est le mécanisme par lequel des chemins sont sélectionnés dans un réseau pour acheminer les données (ou paquets) d'un expéditeur jusqu'à leurs destinataires. Les **routeurs** sont les machines dont le rôle est de faire transiter les paquets d'une interface réseau vers une autre. L'interconnexion des routeurs forme la **topologie du réseau**.

Figure 1



Sur le schéma de la figure 1, il y a six routeurs : les routeurs `R1` et `R6` sont des routeurs d'accès, les quatre autres routeurs `R2`, ..., `R5` sont des routeurs internes.
Les adresses IP utilisées par les machines sont indiquées sous la forme sous-réseau/masque :

- a.b.c.d/8 indique que les adresses allant de a.0.0.0 jusqu'à a.255.255.255 peuvent être utilisées pour associer des adresses IP aux machines dans le sous-réseau,
- a.b.c.d/16 pour indiquer les adresses allant de a.b.0.0 jusqu'à a.b.255.255,
- a.b.c.d/24 pour indiquer les adresses allant de a.b.c.0 jusqu'à a.b.c.255.

Par exemple, le réseau local `Client` peut utiliser toutes les adresses allant de `192.168.1.0` à `192.168.1.255`.

b. Table de routage

Lorsqu'un routeur reçoit un paquet, en fonction de l'adresse de destination, il choisit vers quel routeur voisin le transmettre. Par exemple, si le `Client` veut envoyer un message au `Serveur`, il le transmet au routeur `R1` qui le transmet à son tour au routeur `R3` auquel il est connecté. Ce dernier à plusieurs possibilités, soit transmettre le message à `R2`, soit à `R4` ou encore `R5`, ce qui donne différent chemins possibles :

- `R1` → `R3` → `R2` → `R5` → `R6`
- `R1` → `R3` → `R4` → `R5` → `R6`
- `R1` → `R3` → `R5` → `R6`

Pour déterminer à quel routeur voisin envoyer les paquets reçu, chaque routeur possède une **table de routage**.
Sur le schéma de la figure 1, les tables (simplifiées) de `R1` et `R3` sont :

Table de routage de `R1`

Réseau destinataire	Passerelle	Interface
192.168.1.0/24		192.168.1.1
10.1.1.0/24		10.1.1.1
10.1.2.0/24	10.1.1.3	10.1.1.1
10.1.3.0/24	10.1.1.3	10.1.1.1
10.1.4.0/24	10.1.1.3	10.1.1.1
10.1.5.0/24	10.1.1.3	10.1.1.1
10.1.6.0/24	10.1.1.3	10.1.1.1
10.1.7.0/24	10.1.1.3	10.1.1.1
192.168.6.0/24	10.1.1.3	10.1.1.1

Table de routage de `R3`

Réseau destinataire	Passerelle	Interface
10.1.1.0/24		10.1.1.3
10.1.2.0/24		10.1.2.14
10.1.3.0/24		10.1.3.5
10.1.4.0/24		10.1.4.42
192.168.1.0/24	10.1.1.1	10.1.1.3
10.1.5.0/24	10.1.3.4	10.1.3.5
10.1.6.0/24	10.1.2.5	10.1.2.14
10.1.7.0/24	10.1.4.8	10.1.4.42
192.168.6.0/24	10.1.4.8	10.1.4.42

Pour le routeur `R1`, les deux premières lignes ne contiennent pas de passerelle, les réseaux `10.1.1.0` et `192.168.1.0` sont directement reliés au routeur.
L'interface connectant `R1` au réseau `192.168.1.0` a pour adresse `192.168.1.1` et celle le connectant au réseau `10.1.1.0` a pour adresse `10.1.1.1`.
Tous les paquets destinés aux autres réseaux passent par l'interface `10.1.1.1` à destination de `R3` dont l'interface connectée au réseau `10.1.1.0` a pour adresse `10.1.1.3`.

Dans les deux sections qui suivent nous allons présenter deux algorithmes implémentés dans les routeurs qui permettent de configurer leur table de routage. Ces algorithmes, appelés *protocoles de routage*, sont à la fois décentralisés et dynamiques.

3. Protocole RIP

Le protocole RIP (*Routing Information Protocol*) rentre dans la catégorie des protocoles à vecteur de distance. Un vecteur de distance est un couple (adresse, distance). Le principe de ce protocole est de chercher à minimiser le nombre de routeurs à traverser pour atteindre la destination (on minimise le nombre de sauts).
Nous allons nous intéresser à l'évolution des tables de routage des routeurs `R1` et `R3` de la figure 1, sur lesquels on a activé le protocole RIP.

Étape 0

Initialement, les routeurs ne connaissent que leurs voisins proches. La table de `R1` ressemble à ceci :

Réseau destinataire	Passerelle	Interface	Distance
192.168.1.0/24		wlan0	1
10.1.1.0/24		eth0	1

Et celle de `R3` :

Réseau destinataire	Passerelle	Interface	Distance
10.1.1.0/24		eth1	1
10.1.2.0/24		eth0	1
10.1.3.0/24		eth3	1
10.1.4.0/24		eth2	1

Étape 1

Au bout de 30 secondes, un premier échange intervient avec les voisins immédiats de chacun des routeurs. Lorsqu'un routeur reçoit la table de routage d'un voisin, il y a quatre cas possibles :

1. Il découvre une route vers un nouveau sous-réseau inconnu : il l'ajoute à sa table.
2. Il découvre une route vers un réseau connu plus courte que celle qu'il possède dans sa table : il actualise sa table.
3. Il découvre une route vers un réseau connu, plus longue que celle, fournie par un autre voisin, déjà présente dans sa table : il ignore cette route.
4. Il reçoit une route plus longue vers un réseau connu que lui fournissait ce voisin : un problème est apparu sur son ancienne route, il met à jour sa table.

D'autre part, lorsqu'une liaison avec un voisin est perdue, il actualise sa table en effaçant les routes passant par ce voisin.

En appliquant ces règles, voici la table de routage de `R1` après une étape :

Réseau destinataire	Passerelle	Interface	Distance
192.168.1.0/24		wlan0	1
10.1.1.0/24		eth0	1
10.1.2.0/24	10.1.1.3	eth0	2
10.1.3.0/24	10.1.1.3	eth0	2
10.1.4.0/24	10.1.1.3	eth0	2

`10.1.1.3` est l'adresse IP du routeur `R3`. On ajoute à la table précédente les réseaux atteignables par `R3` en incrémentant la distance de 1. Si `R1` veut atteindre le réseau `10.1.2.0`, il s'adressera à `R3` et atteindra le réseau cible en 2 sauts.

Voici la table de `R3` qui s'enrichit des informations envoyées par `R1` afin d'atteindre le réseau local, mais aussi des informations en provenance de `R2`, `R4` et `R5`. Il découvre ainsi quatre nouveaux réseaux.

Réseau destinataire	Passerelle	Interface	Distance
10.1.1.0/24		eth1	1
10.1.2.0/24		eth3	1
10.1.3.0/24		eth2	1
10.1.4.0/24		eth0	1
192.168.1.0/24	10.1.1.1	eth1	2
10.1.6.0/30	10.1.2.5	eth3	2
10.1.5.0/30	10.1.4.8	eth0	2
10.1.7.0/30	10.1.4.8	eth0	2

Étape finale

En répétant l'étape précédente, les routeurs vont avoir la même vue du réseau. À ce moment-là, on dit que le protocole a **convergé**.

La table de routage finale de `R1`, sera :

Réseau destinataire	Passerelle	Interface	Distance
192.168.1.0/24		wlan0	1
10.1.1.0/24		eth0	1
10.1.2.0/24	10.1.1.3	eth0	2
10.1.3.0/24	10.1.1.3	eth0	2
10.1.4.0/24	10.1.1.3	eth0	2
10.1.5.0/24	10.1.1.3	eth0	3
10.1.6.0/24	10.1.1.3	eth0	3
10.1.7.0/24	10.1.1.3	eth0	3
192.168.6.0/24	10.1.1.3	eth0	4

Pour éviter les boucles de routage, le nombre de sauts est limité à 15. Au-delà, les routes sont supprimées de la table de routage.

Cette limite permet d'autre part d'éviter le problème du **comptage à l'infini**, qui peut se produire par exemple lorsque des liaisons disparaissent en rendant deux parties du réseaux incapables de communiquer entre elles. Dans un tel cas, les routeurs s'aperçoivent des ruptures des liaisons vers leurs anciens voisins directs et mettent temporairement (avant effacement) dans leur table une distance infinie (16) pour leurs voisins devenus inatteignables, mais croient, par les informations des routeurs voisins restés dans leur réseau, qui ignorent la coupure de certaines liaisons, avoir une route vers le réseau devenu inatteignable, avec une distance qui augmente à chaque itération du protocole, chaque routeur mettant à jour sa table avec l'application de la dernière règle d'actualisation.

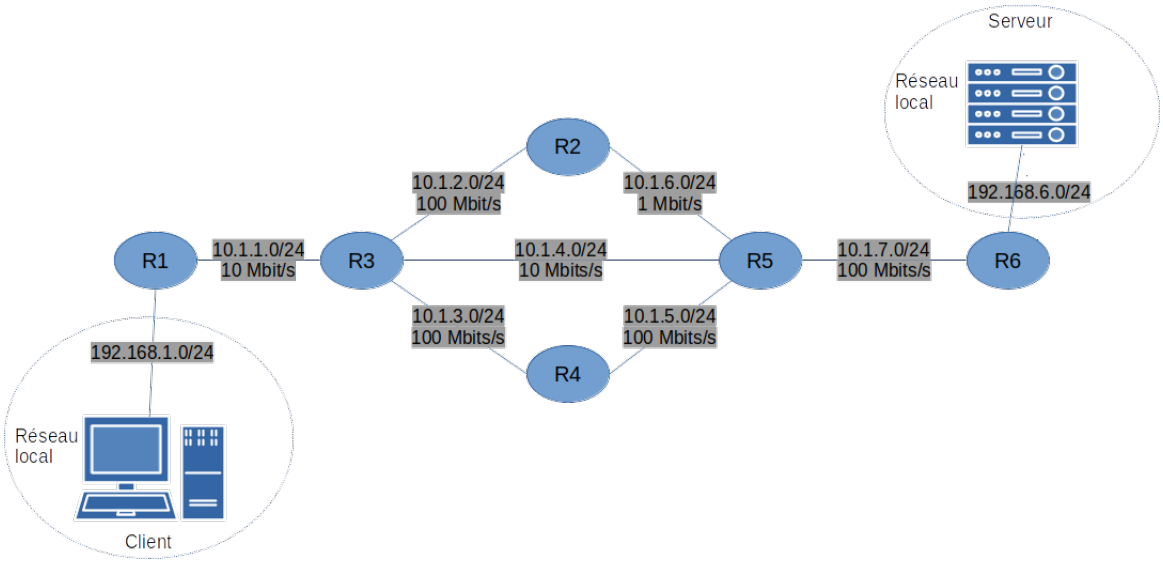
Un inconvénient de RIP est que la limite de distance fait que le protocole ne convient pas à de trop gros (longs) réseaux.

Un autre inconvénient de RIP est qu'il ne prend en compte que la distance entre deux machines en ce qui concerne le saut, mais il ne considère pas l'état de la liaison pour choisir la meilleure bande passante possible. Si l'on considère un réseau composé de trois routeurs A, B et C, reliés en triangle, la liaison directe A-B sera privilégiée pour aller de A à B même si la bande passante n'est que de 56 kbit/s alors qu'elle est de 10 Gbit/s entre A et C et C et B.

4. Protocole OSPF

Le protocole OSPF (*Open Shortest Path First*) est un protocole de routage « à état de liens ». Il corrige les limitations du protocole RIP, notamment en prenant en compte le débit des liaisons empruntées.

On considère le réseau de la figure 1 sur lequel on a ajouté le débit des liaisons :



Le coût d'une liaison est donné par la formule suivante :

$$\text{coût} = \frac{10^8}{\text{débit}}$$

où le débit est exprimé en bits/s (1 Gbit = 10^9 bits, 1 Mbit = 10^6 bits, 1 kbit = 10^3 bits).
Dans l'exemple précédent, on obtient le tableau suivant :

Topologie d'un réseau de routeurs OSPF

Liaison	R1-R3	R3-R2	R3-R4	R3-R5	R2-R5	R4-R5	R5-R6
Coût	10	1	1	10	100	1	1

Étape 0

Au démarrage du protocole, chaque routeur envoie un message (message `HELLO`) aux routeurs auxquels il est connecté pour construire sa *table de voisinage*.

Liaison	sous-réseau	coût
R3-R1	10.1.1.0/24	10
R3-R2	10.1.2.0/24	1
R3-R5	10.1.4.0/24	10
R3-R4	10.1.3.0/24	1

Table de voisinage de `R3`

Étape 1

Chaque routeur va ensuite échanger des messages `LSA` (*Link State Advertisement*) contenant sa table de voisinage avec *tous* les autres routeurs du réseau et ainsi pouvoir reconstituer la topologie complète du réseau.

Étape 2

Finalement chaque routeur calcule à l'aide de l'algorithme de Dijkstra (autre lien) le plus court chemin (en coût) pour chaque destination dans le réseau et construit ainsi sa table de routage.

R3	R1	R2	R4	R5	R6	Choix
0-R3						R3(0)
	10-R3	1-R3	1-R3	10-R3	∞	R2(1)
	10-R3		1-R3	10-R3	∞	R4(1)
	10-R3			2-R4	∞	R5(2)
	10-R3				3-R5	R6(3)
	10-R3					R1(10)

Algorithme de Dijkstra pour `R3`

On a ainsi le meilleur chemin depuis `R3` vers chacun des routeurs, donc les routeurs par lequel il faut passer, et en particulier la passerelle :

- vers `R1` : `R3` - `R1` (coût 10)
- vers `R2` : `R3` - `R2` (coût 1)
- vers `R4` : `R3` - `R4` (coût 1)
- vers `R5` : `R3` - `R4` - `R5` (coût 2)
- vers `R6` : `R3` - `R4` - `R5` - `R6` (coût 3)

On peut considérer les tables suivantes :

Destination	Passerelle	Coût
R1		10
R2		1
R4		1
R5	R4	2
R6	R4	3

Table de routage de `R3` (selon les routeurs)

Réseau destinataire	Passerelle	Interface	Coût
10.1.1.0/24		eth1	10
10.1.2.0/24		eth3	1
10.1.3.0/24		eth2	1
10.1.4.0/24		eth0	10
10.1.6.0/30	10.1.2.2 R2	eth3	1
10.1.5.0/30	10.1.3.4 R4	eth2	2
10.1.7.0/30	10.1.3.4 R4	eth2	3

Table de routage de `R3` (selon les réseaux)

5. Commandes système

a. La commande `ip`

La commande `ip` permet de montrer et manipuler les périphériques réseau et de routage.

La commande `ip addr` permet d'afficher la liste des périphériques réseau ainsi que les adresses IP qui leur sont associées.

```
login@linux:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state ...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc ...
    link/ether a8:5e:45:e2:9d:73 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.86/24 brd 192.168.1.255 scope global ...
       valid_lft 71649sec preferred_lft 71649sec
    inet6 fe80::e89b:6f37:960c:88e8/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
login@linux:~$
```

La sortie de la commande `ip addr` ci-dessus nous indique que la machine possède deux interfaces :

- L'interface `lo`, de type *loopback* (ou boucle locale, une interface fictive) est associé à l'adresse IPv4 `127.0.0.1` et l'adresse IPv6 `::1`.
- L'interface `enp4s0`, est de type `ethernet` (`en`). L'adresse MAC (adresse matérielle ou adresse physique) est `a8:5e:45:e2:9d:73`. Son adresse IPv4 est `192.168.1.86` et son adresse IPv6 est `fe80::e89b:6f37:960c:88e8`

Un affichage similaire peut être obtenue avec `ifconfig` (`ipconfig` sous Windows).

La commande `ip route` affiche la table de routage actuellement configurée.

```
login@linux:~$ ip route
default via 192.168.1.254 dev enp34s0 proto dhcp src 192.168.1.10 metric 100
192.168.1.0/24 dev enp34s0 proto kernel scope link src 192.168.1.10 metric 100
login@linux:~$
```

La *route par défaut* est la machine `192.168.1.254`, c'est le routeur d'accès (celui du réseau local). Tous les paquets qui ne sont pas à destination du réseau local seront transférés au routeur.
La dernière ligne indique que tout paquet à destination d'une adresse réseau `192.168.1.0/24` (c'est à dire toutes les adresses comprises entre `192.168.1.0` et `192.168.1.255`) sera diffusé directement par l'interface `enp34s0`.

b. La commande `ping`

La commande `ping` permet de tester l'accessibilité d'une autre machine à travers un réseau IP. La commande mesure également le temps mis pour recevoir une réponse.

```
login@linux:~$ ping -4 fr.wikipedia.org
PING  (185.15.58.224) 56(84) bytes of data.
64 bytes from text-lb.drmrs.wikimedia.org (185.15.58.224): icmp_seq=1 ttl=49 time=23.1
↪  ms
64 bytes from text-lb.drmrs.wikimedia.org (185.15.58.224): icmp_seq=2 ttl=49 time=23.0
↪  ms
^C
---  ping statistics ---
3 packets transmitted, 2 received, 33.3333% packet loss, time 2007ms
rtt min/avg/max/mdev = 314.330/315.483/316.636/1.153 ms
login@linux:~$
```

Dans l'exemple ci-dessus, on apprend que l'adresse IP de `fr.wikipedia.org` est `185.15.58.224` (c'est l'adresse IPv4, demandée explicitement avec l'option `-4`), la première requête a reçu une réponse en 23,1 ms après avoir été envoyée, la deuxième en 23,0 ms.
Le TTL (*Time To Live*) indique combien de routeurs un paquet peut encore traverser. À chaque passage dans un routeur, ce compteur est décrémenté. Quand un routeur reçoit un paquet avec un TTL de 0, il le détruit.

L'option `-c` permet de préciser le nombre de tentatives.

```
login@linux:~$ ping -c 1 fr.wikipedia.org
PING fr.wikipedia.org(text-lb.drmrs.wikimedia.org (2a02:ec80:600:ed1a::1)) 56 data
↪  bytes
64 bytes from text-lb.drmrs.wikimedia.org (2a02:ec80:600:ed1a::1): icmp_seq=1 ttl=54
↪  time=17.5 ms

--- fr.wikipedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 17.536/17.536/17.536/0.000 ms
login@linux:~$
```

On remarque ici, sans l'option `-4`, que c'est l'adresse IPv6 que l'on obtient.
L'option -t permet de préciser le TTL des paquets envoyés.

```
login@linux:~$ ping -c 1 -t 5 fr.wikipedia.org
PING fr.wikipedia.org(text-lb.drmrs.wikimedia.org (2a02:ec80:600:ed1a::1)) 56 data
↪  bytes
From prs-bb2-v6.ip.twelve99.net (2001:2034:1:c1::1) icmp_seq=1 Time exceeded: Hop limit

--- fr.wikipedia.org ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
login@linux:~$
```

c. La commande `traceroute`

La commande `traceroute` permet de déterminer la route empruntée par un paquet IP pour atteindre une machine cible. Pour cela la commande envoie des paquets IP vers la destination avec un TTL croissant. Chaque routeur se trouvant sur la route décrémente le TTL d'une unité avant de le transmettre si il est non nul. Si le TTL est nul, le routeur détruit le paquet et émet vers la source un paquet ICMP d'erreur signalant que le message a été détruit et n'a pas atteint sa source. Cela permet à traceroute de savoir quels sont les routeurs atteints à l'aller.

```
login@linux:~$ traceroute fr.wikipedia.org
traceroute to fr.wikipedia.org (185.15.58.224), 30 hops max, 60 byte packets
 1  _gateway (192.168.1.254)  0.190 ms  0.262 ms  0.486 ms
 2  hec76-1_migr-78-196-134-254.fbx.proxad.net (78.196.134.254)  2.609 ms  2.652 ms
 ↪  2.685 ms
 3  78.255.119.126 (78.255.119.126)  1.520 ms  1.577 ms  2.001 ms
 4  strasbourg-nx7k-1-po1001.intf.nro.proxad.net (78.254.242.61)  1.604 ms  2.056 ms
 ↪  2.094 ms
 5  strasbourg-crs8-1-be1010.intf.routers.proxad.net (194.149.162.189)  8.164 ms  8.201
 ↪  ms  8.238 ms
 6  p11-crs16-1-be1109.intf.routers.proxad.net (194.149.160.197)  7.885 ms *  6.714 ms
 7  freesas-svc080140-ic369938.ip.twelve99-cust.net (62.115.46.69)  6.947 ms * *
 8  prs-b3-link.ip.twelve99.net (62.115.46.68)  7.249 ms  7.499 ms  7.538 ms
 9  prs-bb1-link.ip.twelve99.net (62.115.118.58)  7.572 ms prs-bb2-link.ip.twelve99.net
 ↪  (62.115.118.62)  7.057 ms  7.161 ms
10  mei-b5-link.ip.twelve99.net (62.115.124.57)  17.331 ms mei-b5-link.ip.twelve99.net
↪  (62.115.124.55)  17.792 ms mei-b5-link.ip.twelve99.net (62.115.124.57)  17.540 ms
11  mei-b3-link.ip.twelve99.net (62.115.125.195)  17.576 ms  17.798 ms  17.977 ms
12  * * *
...
login@linux:~$
```

II. systèmes sur puce

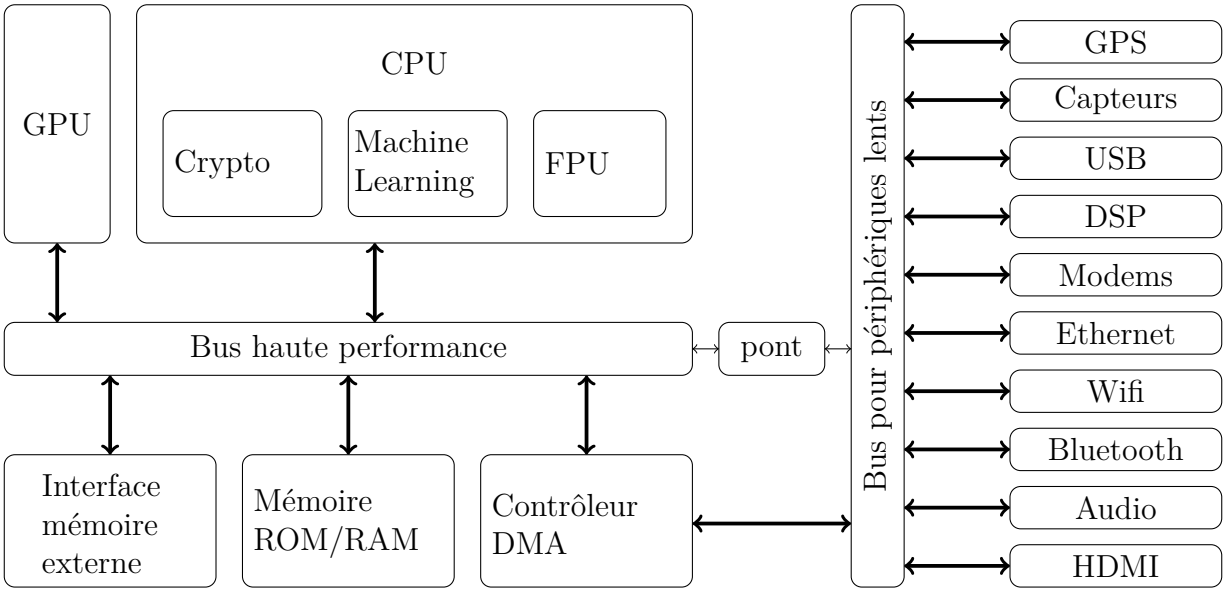
Les composants principaux trouvés sur la carte mère d'un ordinateur sont :

- Le processeur (CPU) qui se charge de réaliser les calculs les plus répandus, ceux qui permettent par exemple de faire tourner le système d'exploitation ou un navigateur web.
- La carte graphique (ou GPU) qui se charge d'afficher une image, qu'elle soit en 2D ou bien en 3D comme dans les jeux.
- Les barrettes de mémoire RAM (*Random Access Memory*) sur lesquelles sont stockées les données temporaires auxquelles un ordinateur doit pouvoir accéder rapidement.
- Les puces qui gèrent les interfaces réseau (Wifi et Ethernet).

La carte-mère relie entre eux tous les composants.
Mais depuis le début de l'ère des smartphones, on assiste à l'émergence de systèmes tout-en-un. Ainsi, presque tout le contenu d'un ordinateur se retrouve finalement dans une seule puce sur le smartphone : le *System on a Chip (SoC)*, ou système sur une puce en français.
Les caractéristiques d'un SoC sont de fait très proches de celles d'un ordinateur. Ils ont une puissance de calcul comparable qui repose sur des microprocesseurs avec plusieurs cœurs cadencés à plusieurs gigahertz, ainsi que des processeurs dédiés (graphique, sécurité). Leur capacité mémoire se mesure en gigaoctets et ils incluent des mémoires de type RAM et FLASH. Enfin ils contiennent de nombreux périphériques. Tout cela sur une puce d'environ 100 mm².

1. Architecture d'un système sur puce

L'architecture d'un SoC est habituellement comparable au schéma suivant :



Le CPU d'un SoC peut contenir des circuits dédiés à certaines opérations afin d'accélérer les calculs. Par exemple, on peut trouver aujourd'hui :

- Une unité de calcul pour les nombres flottants (FPU) simple ou double précision.
- Un circuit dédié aux opérations sur les matrices, par exemple pour accélérer les algorithmes d'apprentissage automatique (en anglais *Machine Learning*) très utilisés par les applications d'intelligence artificielle. Ce circuit peut par exemple exploiter les possibilités de parallélisation des opérations sur les structures.

- Un composant pour la sécurité qui implémente des opérations élémentaires utilisées par les algorithmes de cryptographie.

Ces CPU sont cadencés à des fréquences tellement élevées qu'ils sont fortement ralentis dès qu'ils lisent ou écrivent des données dans la mémoire ou qu'ils dialoguent avec des périphériques.
Ainsi, parce que les composants mémoire ont généralement une fréquence de fonctionnement et des débits de données bien plus importants que les autres périphériques, certaines architectures de SoC mettent en place un système de transfert d'informations reposant sur deux bus de communication :

- Un bus haute performance est dédié à l'échange entre le CPU, le GPU et les différents composants mémoire.
- Un deuxième bus est utilisé pour la communication avec les périphériques plus lent.

Ainsi, le CPU ne se retrouve pas bloqué en attente d'une réponse d'un périphérique lent et peut continuer à communiquer avec la mémoire par exemple.
Afin de limiter encore davantage la différence de vitesse entre CPU et périphériques, les SoC disposent d'un mécanisme d'accès direct à la mémoire (en anglais *Direct memory access*, DMA) qui permet directement de transférer des données de la mémoire vers ou depuis un périphérique sans intervention du CPU.
La liste des périphériques présents dans un système à puce peut fortement varier et dépend de l'espace physique disponible. Voici une liste non exhaustive des composants fréquemment présents :

- des modems (2G/3G/4G),
- des circuits radio (Wifi, Bluetooth),
- une puce GPS,
- des ports d'entrée-sortie (USB, Ethernet, HDMI, Audio),
- des capteurs (CCD).

2. Avantages et inconvénients

Les SoC sont aujourd'hui les composants incontournables de l'informatique nomade (smartphones, tablettes). Ils sont également de plus en plus utilisés dans le monde plus large des systèmes embarqués (voitures, robots, etc.). Ce succès est dû aux avantages suivants :

Énergie : Un grande partie de la consommation électrique d'un circuit est liée au câblage entre les composants. Comme les composants d'un SoC sont connectés sur des distances très petites, les gains sont très importants.

Cette faible consommation implique également une faible perte de chaleur qui évite de recourir à un ventilateur pour refroidir la puce. Les SoC sont donc silencieux.

Coût : Le prix d'un système sur puce est très petit si on le compare à celui d'une carte mère rassemblant les même composants. Même si les coûts d'ingénierie sont plus élevés sur la phase de conception, les coûts de matières premières et de fabrication sont eux réduits.

Les systèmes à puce ont également quelques inconvénients. Contrairement à un ordinateur équipé d'une carte mère, les SoC ne permettent pas de mise à jour, aucune extension n'est possible et, si un seul transistor est endommagé, il ne sera pas possible de réparer l'unité défaillante.

3. Exemples

https://en.wikipedia.org/wiki/Apple_A13
https://9to5mac.com/2019/09/19/a13-design-apple-ahead-of-the-competition/
https://versus.com/fr/apple-a13-bionic
Le système sur puce *A13 bionic*, disponible sur les téléphones et tablettes de la société Apple, est un SoC qui repose sur une architecture similaire à celle présentée. Cette puce électronique contient 8,5 milliards de transistors gravés avec une précision de 7 nanomètres. La liste ci-dessous résume ses principaux composants :

- Un CPU disposant de registres 64 bits. Il contient 6 cœurs dont deux rapides (2.65 GHz) et quatre plus lents (1.8 GHZ), ces derniers étant plus économes en énergie. Les cœurs rapides intègrent un module (AMX) pour accélérer les opérations de Machine Learning comme les multiplications de matrices. Ensemble, ces modules sont capables d'effectuer 1 000 milliards d'opérations 8 bits par seconde.
- Une carte graphique (GPU) avec quatre cœurs.
- Un processeur avec huit cœurs, appelé *Neural engine*, qui s'occupe de tous les traitements d'informations et calculs liés à l'intelligence artificielle comme la reconnaissance faciale et la réalité augmentée.
- Un module spécialisé pour les opérations cryptographiques (AES).
- Une capacité mémoire de 4Go de RAM directement incluse sur la puce.

4. Sources d'information

monlyceenumerique.fr (Thomas Lourdet, Johan Monteillet, Jean-Christophe Gérard, Pascal Thérèse)
https://pixees.fr (David Roche)

III. Processus

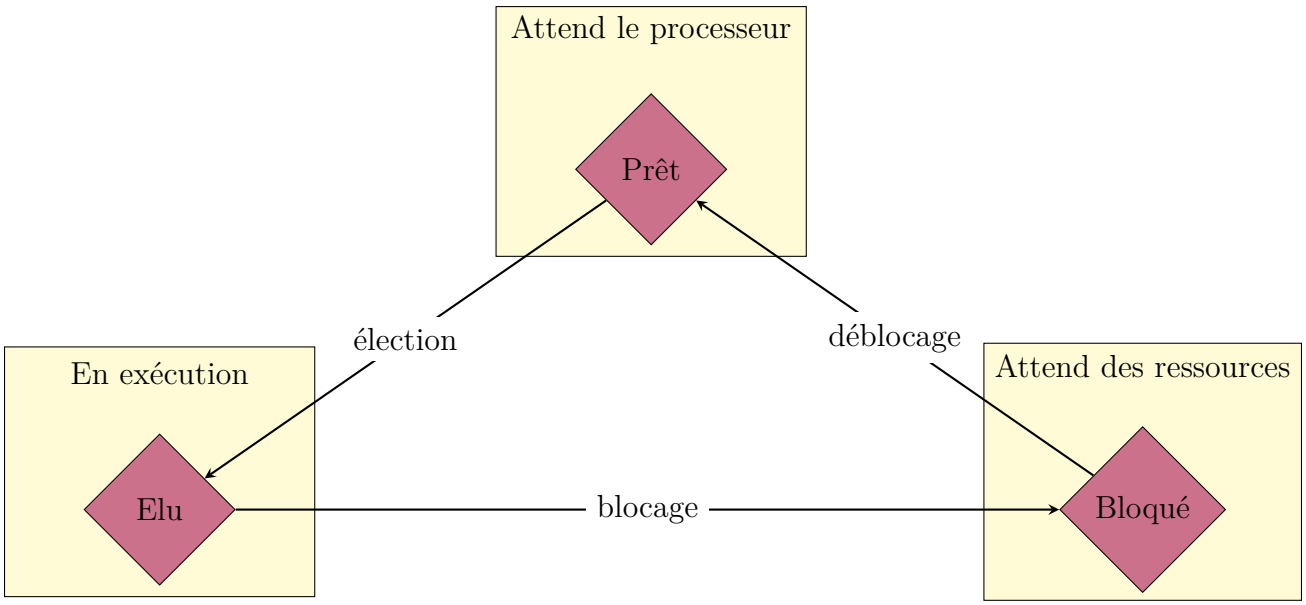
1. Définitions

En première nous avons vu le principe du modèle de von Neumann selon lequel un programme s'exécute de manière séquentielle par le processeur jusqu'à sa terminaison. Pourtant lorsque vous utilisez votre ordinateur, vous pouvez en même temps avoir un navigateur web ouvert avec plusieurs onglets, écouter de la musique et utiliser votre IDE Python préféré pour programmer votre prochain projet à rendre.

Définitions

- Un **processus** est une instance d'exécution d'un programme.
- Deux processus s'exécutent de manière **concurrente** si les intervalles de temps entre le début et la fin de leur exécution ont une partie commune.

Cette exécution concurrente de processus est l'une des fonctionnalités des systèmes d'exploitation modernes. On parle de systèmes d'exploitation multitâches.
voici les différents états d'un processus :



2. Commandes Unix sur les processus

a. Les commandes `ps` et `top`

Dans les systèmes POSIX, la commande `ps` (*process status*) permet d'obtenir des informations sur les processus en cours d'exécution.

```
$
$ ps -a -u -x
```

Les options `-a`, `-u` et `-x` permettent respectivement d'afficher les processus de tous les utilisateurs (et pas uniquement les vôtres), le nom des utilisateurs et les processus lancés ailleurs que depuis le terminal.
La commande affiche alors des informations sur les processus, comme par exemple :

```
USER       PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME COMMAND
root         1  0.0  0.0 225612  8920 ?     Ss   Dec05   0:03 /sbin/init splash
...
userid    2525  4.5  4.3 4101096 706008 ?   Sl   Dec05 142:48 /usr/lib/firefox/firefox
userid   20722  0.3  0.9 2969204 159756 ?   Sl   Dec06   6:51 texmaker
userid   26945  0.0  0.0  40904  3632 pts/0 R+   18:11   0:00 ps -a -u -x
```

- `USER` indique le nom de l'utilisateur qui a lancé le processus.
- `PID` donne l'identifiant numérique du processus.
- `%CPU` et `%MEM` indiquent respectivement le taux d'occupation du processeur et de la mémoire par le processus.
- `STAT` indique l'état du processus, `S` pour *sleeping*, le processus est en attente et `R` pour *running*, le processus est dans l'état prêt ou élu.
- `COMMAND` indique la commande utilisée pour lancer le processus.
- `START` et `TIME` indiquent respectivement l'heure ou la date à laquelle le programme à été lancé et le temps cumulé d'exécution du processus.

La commande `top` permet de voir en temps réel des informations similaires à celle de `ps`. Cette commande peut servir à déterminer quel processus occupe le plus le processeur ou utilise le plus de mémoire.

```
$
$ top
```

```
PID   USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM  TIME+    COMMAND
21044 userid    20   0 3669456 768032 220892 S  43,0  4,7  35:34.35 Web Content
 2525 userid    20   0 4212372 806024 312864 S  10,9  4,9 152:27.52 firefox
 1586 userid     9 -11 2792428  21948  17244 S   7,0  0,1  32:47.90 pulseaudio
 1746 userid    20   0 3800000 310868  95660 S   0,7  1,9  24:45.39 cinnamon
27882 userid    20   0   45456   4116   3396 R   0,7  0,0   0:00.31 top
    1 root      20   0  225612   8920   6816 S   0,0  0,1   0:03.68 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.04 kthreadd
```

b. La commande `kill`

La commande `kill` permet d'interrompre un processus dont on connaît le PID.

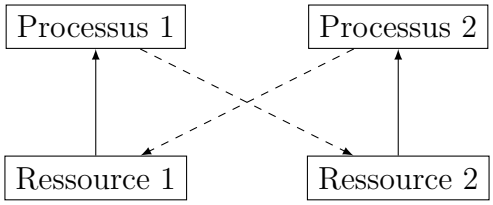
```
$ kill 21044 2525
$
```

La commande précédente va mettre fin aux processus dont les PID sont 21044 et 2639.
L'option `-9` de la commande `kill` permet d'immédiatement terminer un processus sans que celui-ci n'exécute la moindre opération supplémentaire. Cette option est uniquemenent à utiliser lorsqu'une application se comporte de manière anarchique.

3. Interblocage

L'**interblocage** (en anglais *deadlock*) est le blocage de processus en attente les uns des autres.

Cette situation peut survenir lorsqu'un premier processus attend une ressource mobilisée par le second qui attend une ressource mobilisée par le premier. On est dans un cas où les deux processus ont des accès exclusifs sur les ressources. Le blocage est alors définitif.
En fait, la plupart des ressources, matérielles comme logicielles, ne peuvent être utilisées que par un seul processus à la fois, par exemple une webcam, une carte graphique, un micro, une sortie son, etc. Le système d'exploitation fournit des mécanismes afin de permettre aux processus de contrôler l'utilisation de ressources partagées.
Le mécanisme le plus largement répandu est le **verrou**. Ce mécanisme permet d'assurer qu'une ressource est accédée au plus par un processus. Un processus souhaitant accéder à cette ressource doit devenir propriétaire du verrou. Tout autre processus désirant utiliser la ressource est alors bloqué, dans l'attente que le premier libère la ressource pour que le verrou puisse changer de propriétaire.
Dans le cas de l'interblocage, on a donc par exemple deux processus, chacun propriétaire du verrou d'une ressource, et chacun désirant utiliser en plus la ressource que l'autre utilise, sans qu'aucun ne relâche la ressource qu'il utilise. Les deux processus sont alors mis en attente, bloqués l'un par l'autre.



La plupart du temps, les systèmes d'exploitation ignorent ce type de problème d'interblocage, trop difficile et coûteux à résoudre. Il est préférable par exemple de traiter un interblocage en arrêtant un ou plusieurs processus concernés. Il est aussi parfois possible de retirer une ressource allouée à un processus pour l'affecter à un autre processus ou de retourner à un état antérieur. Mais il faut dans ce cas que le système d'exploitation soit capable de détecter un unterblocage. Il doit pour cela vérifier , en contrôlant l'affectation des ressources e manière régulière ou en observant l'utilisation du processus, qu'il n'y a pas d'interblocage.
Cependant il est compliqué d'éviter de telles situations. Il faudrait, avant d'allouer des ressources, disposer d'informations sur les ressources qui seront nécessaires par les processus et estimer un risque d'interblocage. Cette connaissance complète n'est généralement pas envisageable.

IV. Sécurisation des communications

L'objectif fondamental de la sécurité informatique est de permettre à deux personnes, appelées traditionnellement *Alice* et *Bob*, de communiquer à travers un canal peu sûr de telle sorte qu'un opposant, *Oscar*, qui a accès aux informations qui circulent sur le canal de communication, ne puisse ni comprendre et/ou modifier ce qui est échangé, ni se faire passer pour Alice ou Bob.
La **confidentialité** des informations est garantie par **les algorithmes de chiffrement** qui doivent en empêcher l'accès à ceux qui n'en sont pas destinataires. Ces derniers sont composés d'une fonction de chiffrement, qui transforme le message original pour le rendre incompréhensible, et d'une fonction de déchiffrement qui permet de retrouver le message original.
Deux techniques sont disponibles : le **chiffrement symétrique** et le **chiffrement asymétrique**.

1. Le chiffrement symétrique

Les **chiffrements symétriques** utilisent la **même clé** pour chiffrer et déchiffrer un message. La protection de cette clé est cruciale pour la confidentialité des informations échangées.
Un exemple simple est le chiffrement par décalage ou **chiffrement de César**, car déjà utilisé du temps des Romains. Le chiffrement du message est réalisé en décalant de n lettres dans l'alphabet chaque lettre du message initial (en recommençant à « A » si le l'on dépasse « Z »). Ici l'entier n constitue la clé de chiffrement. Par exemple, en utilisant un décalage de 5 lettres, le message

```
L INFORMATIQUE C EST SUPER
```

devient

```
Q NSKTWRFYNVZJ H JXY XZUJW
```

Une méthode de chiffrement un peu moins naïve est le **chiffrement par XOR** (*ou exclusif*). Celle-ci repose sur l'utilisation de l'opérateur binaire *ou exclusif* noté $\oplus$.
Étant donné un message m et une clé de chiffrement k, on recopie la clé de façon à obtenir une chaîne de la même longueur que le message (ici la clé est `NSI`) :

```
L'INFORMATIQUE C'EST SUPER
NSINSINSINSINSINSINSINS
```

Chaque caractère est ensuite converti en nombre (par exemple en son Unicode) :

```
76 39 73 78 70 79 82 77 65 84 73 81 85 69 32 67 39 69 83 84 32 83 85 80 69 82
78 83 73 78 83 73 78 83 73 78 83 73 78 83 73 78 83 73 78 83 73 78 83 73 78 83
```

On effectue ensuite l'opération $\oplus$ sur chaque nombre du message et de la clé. Par exemple, pour la lettre L et N, on obtient :

	1	0	0	1	1	0	0	76
$\oplus$	1	0	0	1	1	1	0	78
	0	0	0	0	0	1	0	2

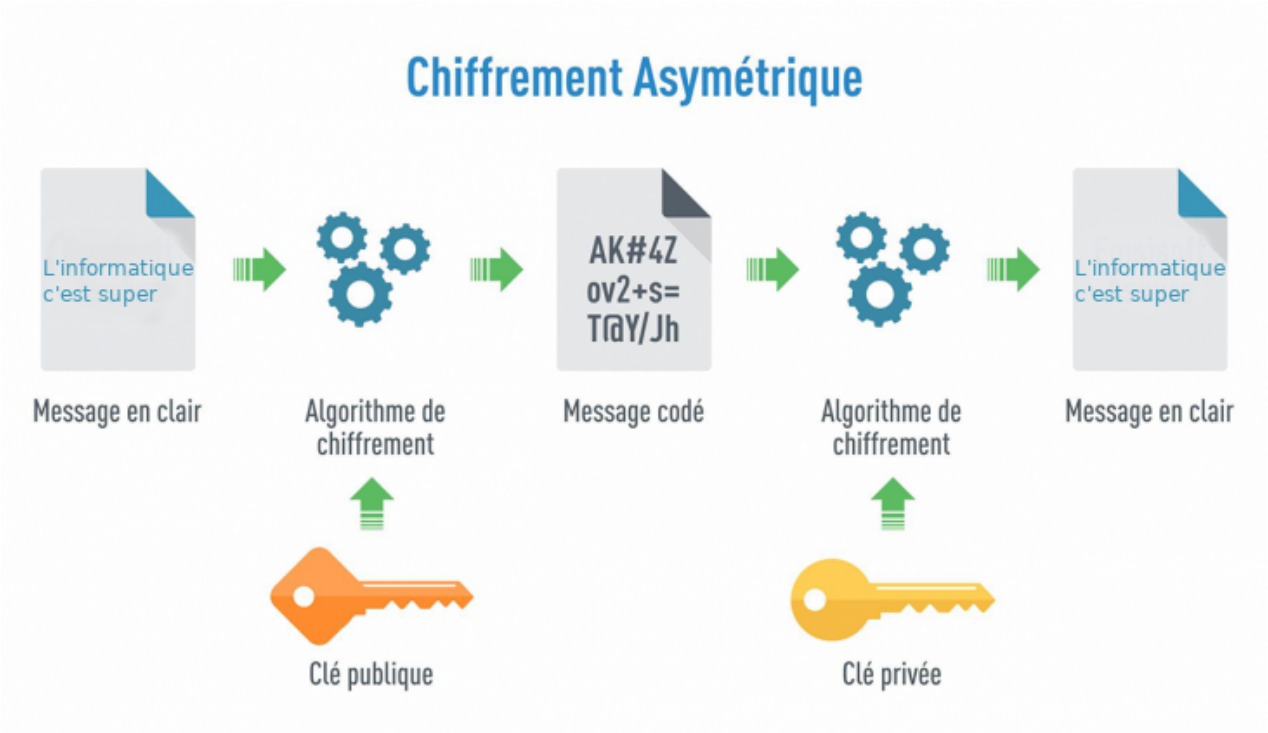
Le message chiffré sera alors :

```
2 116 0 0 21 6 28 30 8 26 26 24 27 22 105 13 116 12 29 7 105 29 6 25 11 1
```

Pour déchiffrer ce message, il suffit de remarquer que $A \oplus A = 0$. Pour un message M, une clé K et le message chiffré $C = M \oplus K$, on a alors $C \oplus K = M \oplus K \oplus K = M$. On peut donc déchiffrer le message avec la même opération que pour le chiffrer.
Le standard de chiffrement symétrique depuis 2000 est l'AES (*Advanced Encryption Standard*) qui est basé sur des mathématiques plus complexes. En plus d'être sûr, il est très efficace et permet de chiffrer de longs messages en temps réel.

2. Le chiffrement asymétrique

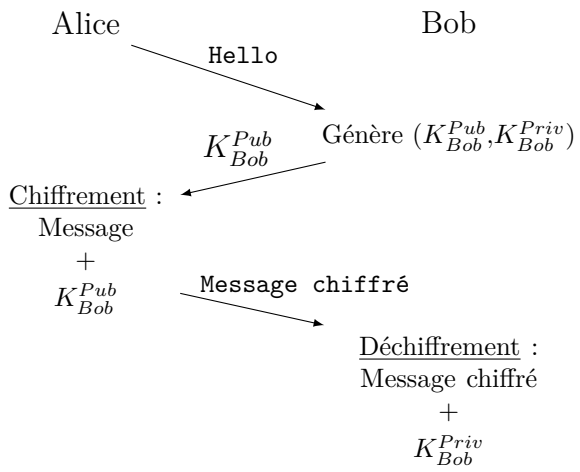
Un **chiffrement asymétrique** utilise une clé de chiffrement différente de la clé de déchiffrement. La clé de chiffrement est connue de tous, elle est appelée **clé publique**. La clé de déchiffrement est uniquement connue de l'expéditeur, elle est appelée **clé privée**.
Si Alice veut envoyer un message à Bob, elle utilise la clé public de Bob pour chiffrer ce message. Bob pourra ensuite déchiffrer le message avec sa clé privée.



Les chiffrements asymétriques reposent sur l'existence de fonctions mathématiques dites à **sens unique**, c'est-à-dire de fonctions qui sont facilement calculable mais dont la réciproque est en pratique impossible à calculer.
L'un des chiffrements les plus utilisés est le RSA (Rivest-Shamir-Adleman) inventé en 1978, il repose sur **le problème de la factorisation** :
Soit n un entier produit de deux grand nombres premiers p et q. Si l'on connaît p et q, il est très facile de calculer $n = p \times q$. Mais réciproquement, s'il on connaît n, il est très difficile de retrouver p et q.
Un autre exemple de chiffrement asymétrique est le chiffrement d'Elgamal, inventé en 1985, qui lui repose sur **le problème du logarithme discret** :
Étant connu p, g et u, il est facile de calculer le reste de la division euclidienne de g^u par p. Mais si l'on connaît p, g et g^u, il est difficile de retrouver u.

Le chiffrement asymétrique permet à Alice et Bob de s'échanger des messages de sorte que Oscar, qui observe uniquement les échanges (adversaire *passif*), ne puisse pas en déduire leur contenu.
Si Alice souhaite envoyer un message à Bob, les deux procèdent comme suit :

1. Bob génère une paire $(K^{\text{pub}}_{\text{Bob}}, K^{\text{priv}}_{\text{Bob}})$ et envoie la clé publique à Alice.
2. Alice chiffre son message m avec la clé publique de Bob, $K^{\text{pub}}_{\text{Bob}}(m)$, et l'envoie à Bob.
3. Bob utilise alors sa clé privée pour retrouver le message d'Alice, $K^{\text{priv}}_{\text{Bob}}(K^{\text{pub}}_{\text{Bob}}(m)) = m$.

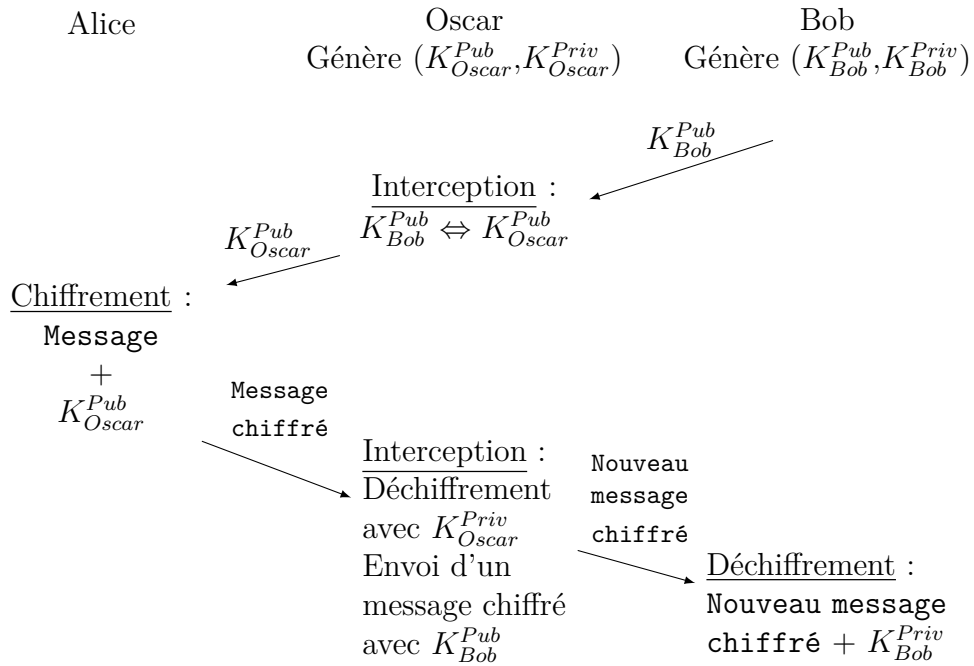


Le chiffrement asymétrique utilise des clés de grandes tailles et nécessite un temps de calcul long et plus de ressources que lors d'un chiffrement symétrique. On utilise alors le chiffrement asymétrique pour l'envoi de messages de petite taille, tels que l'envoi d'une clé de chiffrement symétrique.

3. Authentification des participants

a. Attaque de « l'homme du milieu »

L'inconvénient majeur de l'échange précédent est qu'il est sensible à une attaque dite de « l'homme du milieu » (*Man in the middle*). Un adversaire *actif*, Oscar, placé entre Alice et Bob peut intercepter les communications, et ainsi se faire passer pour Bob auprès d'Alice (et réciproquement).



Oscar peut alors non seulement obtenir tous les échanges entre Alice et Bob, mais est aussi capable de les modifier.
La faille permettant cette attaque est l'absence d'**authentification** : Oscar, qui imite le comportement de Bob, n'a pas eu à prouver qu'il était Bob.

b. Certificats et tiers de confiance

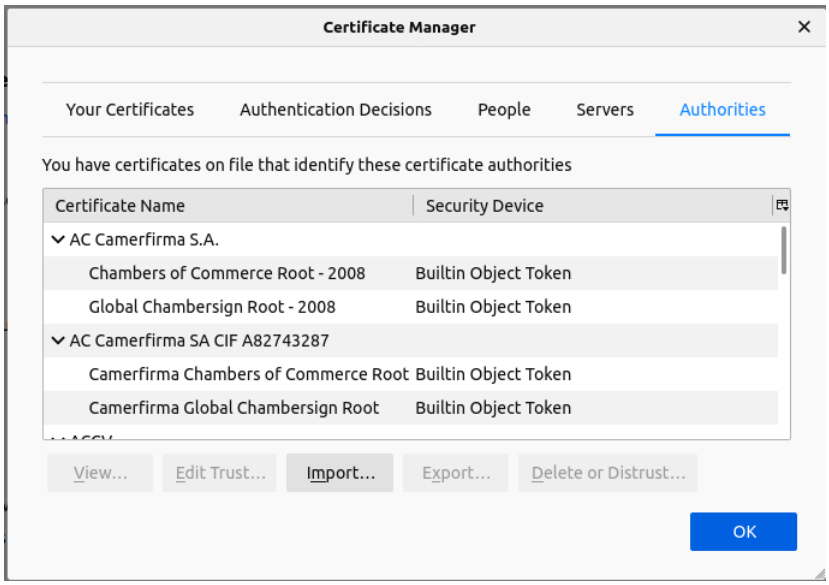
Une solution pour se prémunir de l'attaque de « l'homme du milieu », est d'introduire une tierce personne de *confiance*. De la même manière qu'un citoyen français s'authentifie grâce à sa carte d'identité délivrée par l'État Français en qui l'on fait confiance, les entités sur Internet sont authentifiées grâce à un *certificat* délivré par un *tiers de confiance*. Ces certificats numériques sont créés à partir des clés publiques et privées des participants.
Si Théo est le tiers de confiance d'Alice et Bob, il procède de la manière suivante :

1. Bob et Théo se rencontrent. Théo vérifie l'identité de Bob et utilise ensuite sa clé privée $K^{\text{priv}}_{\text{Theo}}$ pour chiffrer la clé de Bob :
$$s = K^{\text{priv}}_{\text{Theo}}(K^{\text{pub}}_{\text{Bob}})$$
Le fichier s sera alors le *certificat*. On dit que Théo *signe* la clé publique de Bob.
2. Alice, qui veut envoyer un message à Bob, reçoit la clé publique de Bob $K^{\text{pub}}_{\text{Bob}}$ et le certificat s.
3. Alice récupère la clé publique de Théo $K^{\text{pub}}_{\text{Theo}}$ en qui elle a confiance et calcule :
$$K^{\text{pub}}_{\text{Theo}}(s) = K^{\text{pub}}_{\text{Theo}}(K^{\text{priv}}_{\text{Theo}}(K^{\text{pub}}_{\text{Bob}})) = K^{\text{pub}}_{\text{Bob}}$$
Si la clé envoyée par Bob correspond au résultat du calcul, Alice peut authentifier Bob comme étant son interlocuteur.
4. Alice peut alors utiliser la clé publique de Bob pour lui envoyer une clé symétrique et ainsi pouvoir initier une communication sécurisée.

4. Le protocole HTTPS

a. Autorités de certifications et format X.509

Un tiers de confiance sur Internet est appelé une **autorité de certification** (CA) qui délivre des certificats. Les navigateurs web possèdent les clés publiques des CA. Elles sont en nombres relativement restreint, de l'ordre de la centaine.



Liste des autorités de certifications de Firefox

Le format standard de certificat est le format X.509 qui contient entre autres :

- le nom du CA émetteur,
- la période de validité,
- le nom du sujet certifié,
- la clé publique du sujet certifié.

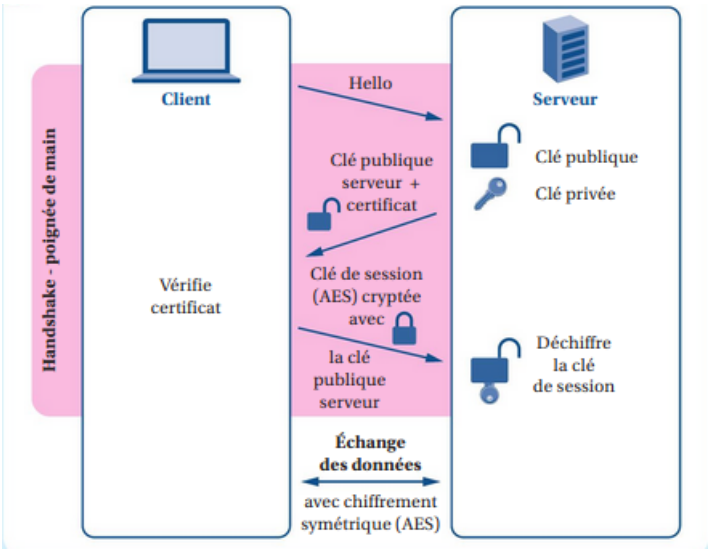
Le tout est signé grâce à la clé privée du CA.
Sous Unix, la commande `openssl` permet d'obtenir des informations sur un certificat. Par exemple, sur le certificat de https://www.cnrs.fr/, téléchargé depuis le navigateur au format .pem sous le nom www-cnrs-fr.pem, la commande donne les informations suivantes (en janvier 2023) :

```
nsi@linux:~$ openssl x509 -in www-cnrs-fr.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1e:c6:c9:20:cc:23:d3:37:a0:82:15:20:38:89:e6:42
        Signature Algorithm: sha384WithRSAEncryption
        Issuer: C = NL, O = GEANT Vereniging, CN = GEANT OV RSA CA 4
        Validity
            Not Before: Aug  2 00:00:00 2022 GMT
            Not After : Aug  2 23:59:59 2023 GMT
        Subject: C = FR, postalCode = 75016, ST = \C3\8Ele-de-France, L =
        ↪  Paris,
        street = "3, rue Michel-Ange", O = Centre national de la recherche
        ↪  scientifique, CN = www.cnrs.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:c4:33:04:56:d7:ef:5a:cb:0b:50:e9:c0:2c:6f:
                    ...
                Exponent: 65537 (0x10001)
                ...
nsi@linux:~$
```

Le CA ayant produit ce certificat est « *GEANT Vereniging* ». Le certificat est valide du 2 août 2022 au 2 août 2023 (inclus). Le site certifié est `www.cnrs.fr`. La clé publique est une clé RSA de 4096 bits et est donnée par deux champs `Modulus` et `Exponent`.

b. Détails du protocole HTTPS

Le protocole HTTPS (*HTTP Secure*) est le protocole HTTP sécurisé par le protocole TLS (*Transport Layer Secure*) qui permet la confidentialité et l'intégrité des messages échangés.
Le protocole TLS ajoute simplement une phase permettant l'authentification du serveur et la mise en place sécurisée d'une clé de chiffrement symétrique appelé **clé de session**. La figure ci-dessous décrit la phase de mise en place, appelée « poignée de main TLS » (*TLS handshake*).



Le Transport Layer Security