Chapitre 04 : La couche Application

Table des matières

Objectifs....2
Rappel : modèles OSI et TCP/IP....2
Représentation des données sur la couche Application....3
- Sérialisation....3
- Compression....6

Protocoles courants....7
- DHCP et DNS....7
- HTTP et HTTPS....7
- FTP....7
- SSH et SFTP....8
- SMTP, IMAP et POP....8
- WebSocket....8
- BitTorrent....9

Le protocole HTTP....9
- Format des requêtes....10
- Format des réponses....10
- Les API Web....12

Chiffrement....12
- Chiffrement symétrique....12
- Chiffrement à clé publique....13
- TLS et HTTPS....14

Utilisation de QUIC par HTTP/3....15
Synthèse....16
Index....17
Références....18

© 2024 Pier-Luc Brault

Ce document est mis à disposition selon les termes de la licence Creative Commons - Attribution - Partage dans les Mêmes Conditions 4.0 International (CC BY-SA 4.0).

Objectifs

- Comprendre les notions de sérialisation et de compression des données.
- Connaître les principales méthodes de sérialisation textuelle.
- Pouvoir nommer plusieurs protocoles courants de la couche Application.
- Comprendre les bases du fonctionnement du protocole HTTP.
- Comprendre ce qu'est une API Web.
- Comprendre ce qu'est le chiffrement symétrique et le chiffrement à clé publique.
- Comprendre les garanties fournies par l'utilisation de TLS.
- Comprendre les avantages d'utiliser QUIC pour le protocole HTTP/3.

Rappel : modèles OSI et TCP/IP

Dans le premier chapitre, nous avons vu la correspondance suivante entre les couches des modèles OSI et TCP/IP:

<table>
<tr><th>OSI</th><th>TCP/IP</th></tr>
<tr><td>7 – Application</td><td rowspan="3">4 - Application</td></tr>
<tr><td>6 – Présentation</td></tr>
<tr><td>5 – Session</td></tr>
<tr><td>4 – Transport</td><td>3 - Transport</td></tr>
<tr><td>3 – Réseau</td><td>2 - Internet</td></tr>
<tr><td>2 – Liaison de données</td><td>1 - Liaison</td></tr>
<tr><td>1 – Physique</td><td></td></tr>
</table>

Pour le modèle OSI, la couche Session contrôle le début et la fin d'une communication, ainsi que sa reprise en cas d'interruption. On peut donc faire une analogie avec une conversation téléphonique : une session commence dès que la personne appelée répond à l'appel, et se termine dès que quelqu'un met fin à l'appel.

La couche Présentation, pour sa part, formate les données de façon à ce qu'elles soient compréhensibles par d'autres machines. Elle assure également le chiffrement des données s'il y a lieu.

Finalement, la couche Application du modèle OSI est celle qui est exploitée directement par les applications avec lesquelles interagissent les utilisateurs. Par exemple, le protocole HTTP permettant à un navigateur d'accéder à des pages Web est un protocole de la couche Application.

© 2024 Pier-Luc Brault

En ce qui concerne le modèle TCP/IP, les couches Session et Présentation n'existent pas, et leurs fonctions sont englobées dans la couche Application. Certains protocoles de la couche Application du modèle TCP/IP incluent des fonctions qui seraient typiquement associées à la couche Session ou à la couche Présentation du modèle OSI, tandis que d'autres protocoles ne comprennent simplement pas de telles fonctions.

Représentation des données sur la couche Application

L'échange de données entre différents ordinateurs pose certains défis, puisque la représentation des données peut varier d'un ordinateur à l'autre. L'exemple le plus simple concerne la représentation des caractères : les ordinateurs qui s'échangent des données textuelles doivent s'entendre sur la séquence de bits qui représente chaque caractère. La norme **ASCII** est celle qui a été la plus répandue historiquement, mais les caractères qu'elle permet de représenter se limitent à ceux utilisés par la langue anglaise. D'autres normes sont apparues en réponse à cette limitation, dont la norme **UTF-8** qui a pour objectif de pouvoir représenter les caractères de toutes les langues en usage dans le monde. Or, pour qu'un message textuel soit interprété de la même façon d'un bout en bout d'une communication réseau, il faut que l'hôte de destination ait un moyen de savoir quelle norme d'encodage a été utilisée par l'hôte source.

Il en va de même pour les données numériques : le nombre de bits utilisés pour représenter un *int* en C++, par exemple, peut différer selon le processeur ou le compilateur utilisé. Pire encore : lorsqu'une donnée est représentée sur plusieurs octets, l'ordre dans lequel les octets sont lus peut différer d'une architecture de processeur à une autre! Ainsi, un entier non signé encodé sur 16 bits avec une valeur de 54 000 sur une machine (1101001011110000) pourrait devenir 61 650 (1111000011010010) sur une autre machine. C'est ce qu'on appelle le **boutisme** ou ***endianness***[1]. Les données échangées à travers un réseau ne peuvent donc pas toujours simplement être transmises selon leur représentation native sur l'hôte source, et les hôtes doivent s'entendre sur une représentation intermédiaire.

Le problème de la représentation des données ne concerne pas la couche Transport : celle-ci ne se préoccupe pas de la signification des octets qu'elle transporte. Ce sont donc aux protocoles de la couche Application du modèle TCP/IP de déterminer la façon dont les données en transit sont représentées. Dans le modèle OSI, on dirait que ce rôle incombe à la couche Présentation.

Sérialisation

La **sérialisation** désigne la conversion de données dans un format approprié pour leur transport à travers un réseau ou leur écriture sur un support de stockage. Dans le cas du transport à travers un réseau, la sérialisation se préoccupe justement de la représentation des données dans un format qui sera interprété correctement par les différents hôtes.

Il existe différentes méthodes de sérialisation. Une méthode de sérialisation peut utiliser soit un format binaire, soit un format textuel.

1 https://fr.wikipedia.org/wiki/Boutisme

© 2024 Pier-Luc Brault

Une **sérialisation binaire** permet de représenter directement des types de données numériques, et détermine les détails de représentation (par exemple, le boutisme et le nombre de bits utilisé par chaque type) qui devront être respectés par chaque machine.

Une **sérialisation textuelle** convertit toutes les données (numériques ou non) en une suite de caractères selon une norme d'encodage existante, telle qu'ASCII ou UTF-8. Bien entendu, les données demeurent ultimement représentées sous forme de bits. Il faut cependant plus de bits pour représenter un nombre au format textuel que directement au format binaire, puisque chaque chiffre est représenté sous la forme d'un caractère (et donc d'un octet). Par exemple, le nombre 42 prend un seul octet dans sa représentation binaire (00101010), mais en prend deux en représentation ASCII (00110100 pour le caractère « 4 », puis 00110010 pour le caractère « 2 »).

Formats de sérialisation textuels

Les formats de sérialisation textuels les plus connus aujourd'hui sont sans doute **XML** et **JSON**.

XML

XML est un langage semblable à HTML, mais qui permet de définir ses propres types de balises selon les besoins spécifiques de l'application. On pourrait par exemple utiliser XML pour définir le format de données d'une application de type « Liste de contacts » :

```
<Contact id="27">
    <name>
        <first>Harry</first>
        <middle>James</middle>
        <last>Potter</last>
    </name>
    <addresses>
        <address type="personal">
            <street>4 Privet Drive</street>
            <city>Little Whinging</city>
            <county>Surrey, England</county>
            <country>United Kingdom</country>
        </address>
        <address type="personal">
            <street>12 Grimmauld Place</street>
            <city>London</city>
            <country>United Kingdom</country>
        </address>
        <address type="school">
            <name>Hogwarts</name>
            <region>Highlands, Scotland</highlands>
            <country>United Kingdom</country>
        </address>
    </addresses>
</Contact>
```

JSON

JSON, pour *JavaScript Object Notation*, est un format de sérialisation basé sur la façon de représenter des objets dans le langage JavaScript. Voici à quoi ressembleraient les données de l'exemple précédant si on les convertissait en JSON :

```
{
  "id": 27,
  "name": {
    "first": "Harry",
    "middle": "James",
    "last": "Potter"
  },
  "addresses": [
    {
      "type": "personal",
      "street": "4 Privet Drive",
      "city": "Little Whinging",
      "county": "Surrey, England",
      "country": "United Kingdom"
    },
    {
      "type": "personal",
      "street": "12 Grimmauld Place",
      "city": "London",
      "country": "United Kingdom"
    },
    {
      "type": "school",
      "name": "Hogwarts",
      "region": "Highlands, Scotland",
      "country": "United Kingdom"
    }
  ]
}
```

JSON permet de représenter les types de données suivants :

- Nombre (ex : 27)
- Chaîne de caractères (ex : "Harry")
- Booléen (true ou false)
- Tableau ([…])
- Tableau associatif[2] ({ ... })

2 https://fr.wikipedia.org/wiki/Tableau_associatif

Le format JSON a supplanté XML dans la plupart des applications modernes.

Avantages et inconvénients de la sérialisation textuelle

Le principal avantage de la sérialisation textuelle par rapport à la sérialisation binaire est sa lisibilité. Des données formatées en JSON ou XML peuvent en effet facilement être lues et interprétées par des humains. Ce type de sérialisation rend donc plus facile le développement et le débogage des applications. C'est d'autant plus vrai lorsque les données sont destinées à être exploitées par différentes applications, ce qui est de plus en plus le cas à l'ère où de nombreuses applications permettent d'interagir avec d'autres applications ou d'intégrer des données provenant de celles-ci.

Les principaux inconvénients de la sérialisation textuelle sont au niveau de la taille des données et de la performance. D'une part, il faut souvent un plus grand nombre de bits pour représenter les mêmes informations de manière textuelle que dans un format binaire. D'autre part, les processus de sérialisation et de dé-sérialisation des données sont souvent plus lents avec un format textuel.

Comme pour beaucoup de choses en informatique, l'idée est de choisir le bon compromis selon le type d'application qu'on développe. Par exemple, si on développe un jeu multijoueur en ligne en temps réel, on voudra probablement utiliser un format binaire, puisque de nombreuses données devront être échangées entre les hôtes chaque seconde et que leur interprétation sur chaque hôte devra être très rapide. Par contre, si on développe plutôt un réseau social et qu'on veut permettre à des développeurs et développeuses de partout dans le monde d'intégrer des informations sur ses utilisateurs à leurs propres applications, on optera plutôt pour un format textuel. Certains types de données ne se prêtent par ailleurs tout simplement pas à un encodage textuel. Pensons par exemple à des données vidéo, qui ne peuvent être transmises que dans un format binaire.

Compression

La **compression** est une technique permettant de convertir une suite de bits en une suite de bits plus petite à l'aide d'un algorithme prévu à cet effet. Le processus inverse se nomme **décompression**, et permet de restituer les données d'origine. Il faut le plus souvent décompresser les données compressées avant d'être en mesure de les interpréter (l'exception la plus notable concerne les algorithmes de compression de données multimédia, mais certains d'entre eux, dits « avec perte », entraînent une perte de qualité du média d'origine).

Nous n'allons pas étudier les algorithmes de compression dans ce cours. Il importe cependant de savoir que certains protocoles de la couche Application compressent les données avant de les transmettre d'un hôte à un autre. L'intérêt de cette opération est d'utiliser moins de bande passante pour les échanges réseau, et donc d'accélérer la transmission des données. Celles-ci doivent cependant passer par des opérations de compression et de décompression, ce qui demande du temps également.

Protocoles courants

Les protocoles de la couche Application sont construits par-dessus les protocoles de la couche Transport (le plus souvent TCP ou UDP). Un protocole de la couche Application détermine donc quelles données sont échangées en TCP ou UDP (ou encore en QUIC), dans quels formats et selon quelles règles. Rappelons que les protocoles TCP et UDP ne se préoccupent aucunement de la nature des données qu'ils transportent : c'est la couche Application qui donne un sens à ces données.

Il existe une multitude de protocoles différents sur la couche Application. Les personnes qui développent une application peuvent même décider de créer leur propre protocole! C'est d'ailleurs ce que vous allez faire lorsque vous programmerez des applications réseau dans le cours : vous allez déterminer quel protocole de la couche Transport vous utiliserez, et de quelle façon vous l'utiliserez. Il demeure cependant important de connaître l'existence de certains protocoles standardisés, puisque vous devrez fort probablement les utiliser au cours de votre carrière, voire même dans d'autres cours du programme. Voici quelques-uns de ces protocoles :

DHCP et DNS

Les protocoles DHCP et DNS ont été vus au chapitre 2. Nous les mentionnons à nouveau ici simplement pour rappeler qu'ils appartiennent à la couche Application.

DNS utilise le port 53 en UDP. DHCP utilise les ports 67 et 68, également en UDP, et les clients envoient les demandes d'adresse IP en diffusion (*broadcast*).

Notons toutefois qu'il existe des variantes du protocole DNS qui n'utilisent pas UDP, telles que ***DNS over HTTPS*** (***DoH***)[3] et ***DNS over QUIC*** (***DoQ***).

HTTP et HTTPS

HTTP (ou **HTTPS**, sa version sécurisée) est sans aucun doute le protocole de la couche Application le plus utilisé de nos jours. D'abord conçu pour permettre à des navigateurs d'accéder à des pages Web, il est aujourd'hui utilisé également pour l'échange d'informations entre des applications. Nous l'explorerons plus en profondeur dans la prochaine section de ces notes de cours. L'acronyme HTTP signifie *Hypertext Transfer Protocol*, et HTTPS y ajoute un « S » pour *Secure*. HTTP utilise le port 80 par défaut, et HTTPS le port 443. Le protocole utilisé sur la couche Transport était TCP jusqu'à la dernière révision majeure de HTTP, appelée **HTTP/3**, qui utilise QUIC.

HTTP est un protocole de type requête-réponse : le client envoie des requêtes au serveur, et le serveur envoie une réponse pour chaque requête.

FTP

FTP, pour *File Transfer Protocol*, est un protocole permettant de téléverser des fichiers vers un serveur ou de télécharger des fichiers depuis celui-ci. Il permet aussi de créer des dossiers et renommer ou

3 https://fr.wikipedia.org/wiki/DNS_over_HTTPS

supprimer des fichiers et des dossiers, toujours à partir d'un client distant. Sa première version date de 1971, ce qui en fait un protocole plus ancien que le modèle TCP/IP, et donc plus ancien qu'Internet! Cela dit, sa version actuelle utilise une connexion TCP. Il existe une version sécurisée de FTP appelée **FTPS**, qu'il ne faut pas confondre avec **SFTP** (que nous verrons un peu plus bas).

Ce protocole tend à tomber en désuétude, et on lui préfère de plus en plus SFTP, jugé plus sécuritaire que FTP et même FTPS. Il demeure malgré tout utilisé par certains hébergeurs Web pour y téléverser les fichiers des sites à héberger.

FTP utilise par défaut les ports 20 et 21.

SSH et SFTP

SSH, pour *Secure Shell*, est un protocole dont la principale utilité est de permettre un accès à distance à une machine et le contrôle de celle-ci en ligne de commande. Il est surtout utilisé pour l'administration de serveurs Linux et Unix.

SSH permet aussi d'encapsuler une connexion TCP, et donc de rendre sécuritaires des protocoles qui ne le sont pas à la base. On dit alors qu'on crée un « tunnel SSH ».

Il existe aussi un protocole de transfert de fichiers appelé **SFTP**, pour *SSH File Transfer Protocol*. Comme on peut le deviner, celui-ci fonctionne au-dessus de SSH. SFTP n'a aucun lien avec FTP, ce sont deux protocoles complètement différents.

SSH (et donc SFTP) utilise par défaut le port 22 en TCP.

SMTP, IMAP et POP

Le protocole **SMTP**, pour *Simple Mail Transfer Protocol*, est utilisé à la fois pour envoyer des courriels à un serveur de messagerie électronique, et pour le transfert des courriels entre différents serveurs (par exemple, d'un serveur Gmail à un serveur Outlook). Il utilise les ports 25, 465 et 587, en TCP.

À l'inverse, les protocoles **IMAP** (pour *Interactive Message Access Protocol*) et **POP** (pour *Post Office Protocol*) permettent de consulter des courriels stockés sur un serveur de messagerie. Le protocole IMAP permet une véritable synchronisation avec le serveur, tandis que le protocole POP, plus ancien, permet uniquement de télécharger les courriels pour les gérer entièrement localement. IMAP utilise les ports 143, 220 et 993 en TCP, tandis que POP utilise les ports 110 et 995, également en TCP.

WebSocket

WebSocket est un protocole permettant la communication bidirectionnelle entre un navigateur et un serveur Web. Cela n'est pas le cas de HTTP : celui-ci permet à un client d'envoyer une requête à un serveur et de recevoir une réponse, mais ne permet pas au serveur d'envoyer des données au client de sa propre initiative. WebSocket est particulièrement utile pour intégrer des fonctionnalités en temps réel

à une page Web, comme un système de clavardage. WebSocket utilise les mêmes ports que HTTP et HTTPS, soit 80 et 443, en TCP.

Le protocole WebSocket a été normalisé en 2011. Auparavant, différentes techniques étaient utilisées pour simuler une communication bidirectionnelle en HTTP, notamment le *polling*, qui consiste à interroger le serveur à répétition pour savoir si de nouvelles données sont disponibles. De telles techniques demeurent utilisées aujourd'hui par certaines applications Web.

BitTorrent

BitTorrent est un protocole de partage de fichiers fonctionnant en **pair-à-pair** (*peer-to-peer* ou **P2P**), c'est-à-dire que chaque hôte (appelé « pair ») agit à la fois comme un client et un serveur, par opposition aux protocoles de type client-serveur. Ce protocole permet d'acquérir un fichier en téléchargeant des fragments de celui-ci provenant de plusieurs pairs. Un pair qui télécharge un fichier peut ensuite à son tour en partager des fragments aux autres pairs intéressés à le télécharger. Ainsi, plus un fichier est populaire, plus il y a de pairs qui offrent ce fichier, plus il est facile de le télécharger, alors que c'est plutôt le contraire qui se produit avec un modèle client-serveur traditionnel, où la popularité d'un fichier peut saturer les capacités du serveur. BitTorrent utilise principalement TCP et peut aussi utiliser UDP dans certains cas.

Le protocole HTTP

Dans cette section, nous allons explorer les bases du fonctionnement du protocole HTTP. Rappelons d'abord que HTTP est un protocole de type requête-réponse, c'est-à-dire qu'un client envoie des requêtes à un serveur, et celui-ci lui envoie des réponses. Une réponse est envoyée par le serveur pour chaque requête reçue. Une réponse peut contenir des données textuelles, telles que du code HTML, CSS ou JavaScript, mais peut aussi contenir des données binaires, telles que le contenu d'un fichier image, vidéo, ou même de tout autre fichier.

Lors du chargement d'une page Web, un navigateur doit effectuer plusieurs requêtes HTTP. Une première requête est envoyée pour recevoir le code HTML de la page Web. Par la suite, en interprétant le code HTML, le navigateur peut identifier d'autres ressources à télécharger : des fichiers CSS, des fichiers JavaScript, des images, etc. Il doit donc effectuer une nouvelle requête HTTP pour chacune de ces ressources. Le navigateur peut utiliser une même connexion TCP ou QUIC pour envoyer toutes ces requêtes et recevoir toutes les réponses correspondantes.

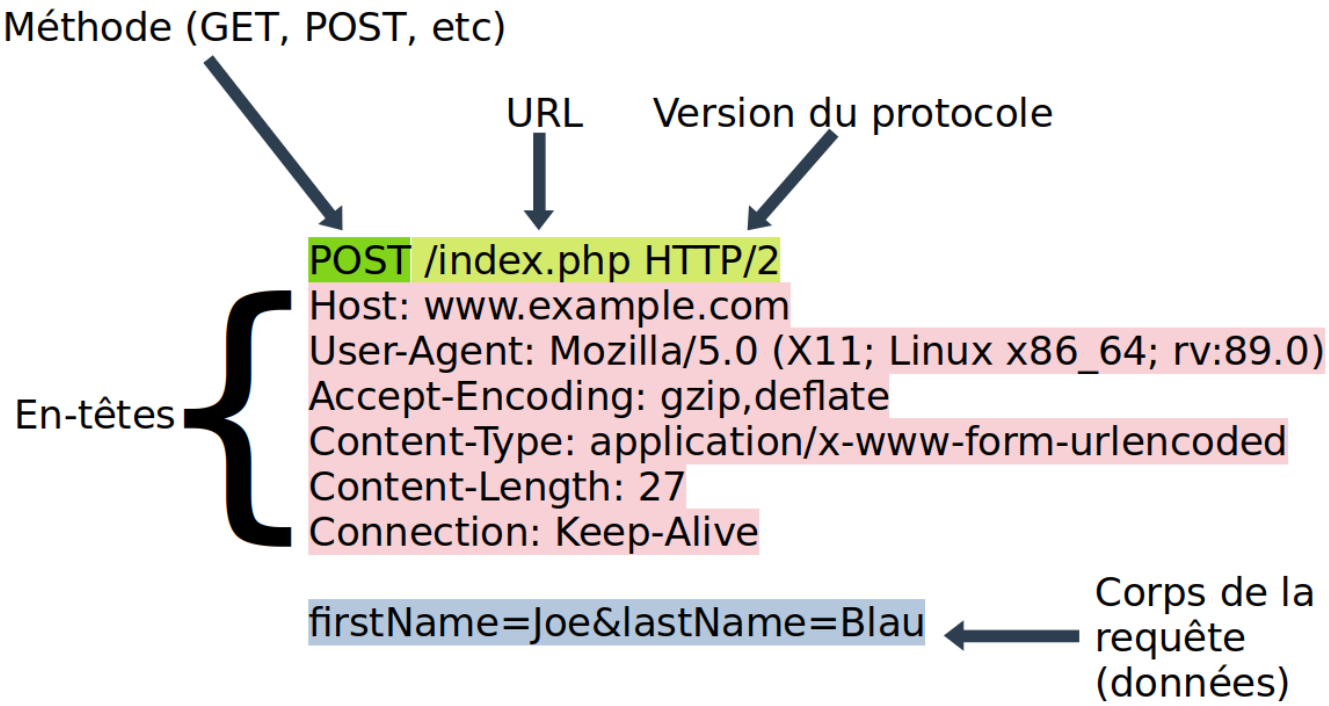
HTTP permet aussi au client d'envoyer des informations à destination du serveur. C'est par exemple ce qui se passe quand on envoie les données d'un formulaire Web.

Le protocole HTTP est inventé en même temps que le langage HTML par Tim Berners-Lee, alors employé au CERN[4], et sa première version est rendue publique en août 1991.

4 https://fr.wikipedia.org/wiki/Organisation_europ%C3%A9enne_pour_la_recherche_nucl%C3%A9aire

Format des requêtes

Une requête HTTP ressemble à ceci :



La **méthode** indique le type d'opération que le client veut accomplir. Les principales sont **GET**, qui sert à récupérer du contenu (par exemple, celui d'une page Web), et **POST**, qui sert à envoyer des données (par exemple, les valeurs saisies par l'utilisateur dans les champs d'un formulaire).

L'**URL** identifie la page ou la ressource à laquelle le client souhaite accéder, et elle est suivie de la version utilisée du protocole.

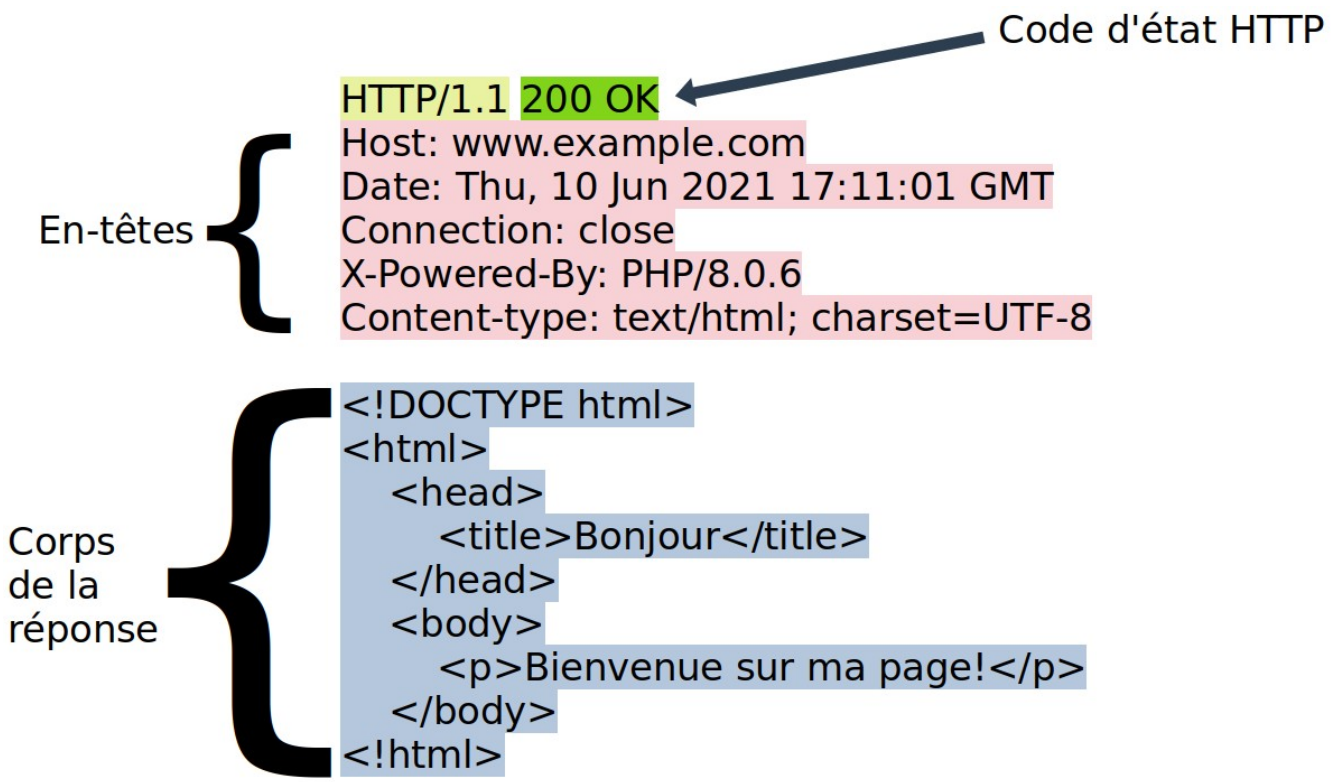
Après la version du protocole, on trouve les **en-têtes de la requête**. Ceux-ci servent à fournir au serveur différentes informations, dont certaines aident le serveur à répondre à la requête correctement.

Finalement, la requête peut contenir un **corps**, soit des données à transmettre au serveur. Une requête de type GET ne contient habituellement pas de corps, et une requête de type POST en contient généralement un.

Format des réponses

Une réponse HTTP ressemble à ceci :

© 2024 Pier-Luc Brault



Une réponse contient un **code d'état HTTP**. Celui-ci fournit de l'information sur le résultat de la requête. Voici les principaux :

Code d'état	**Description**[5]
200 *OK*	Requête traitée avec succès. La réponse dépendra de la méthode de requête utilisée.
301 *Moved Permanently*	Document déplacé de façon permanente.
304 *Not Modified*	Document non modifié depuis la dernière requête.
400 *Bad request*	La syntaxe de la requête est erronée.
401 *Unauthorized*	Une authentification est nécessaire pour accéder à la ressource.
403 *Forbidden*	Le serveur a compris la requête, mais refuse de l'exécuter. Contrairement à l'erreur 401, s'authentifier ne fera aucune différence. Sur les serveurs où l'authentification est requise, cela signifie généralement que l'authentification a été acceptée mais que les droits d'accès ne permettent pas au client d'accéder à la ressource.
404 *Not Found*	Ressource non trouvée.
500 *Internal Server Error*	Erreur interne du serveur.

Il existe de nombreux autres codes d'État qui sont utilisés moins fréquemment[6].

5 Les descriptions des codes d'état proviennent d'ici : https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP
6 Dont le fameux code 418 (*I'm a teapot*) — https://fr.wikipedia.org/wiki/Hyper_Text_Coffee_Pot_Control_Protocol

Les **en-têtes de la réponse** ont une fonction semblable à celle des en-têtes de la requête, et fournissent des informations qui peuvent parfois être utiles au client.

Finalement, la réponse contient elle aussi un corps. Celle ci peut contenir des données textuelles (par exemple du code HTML, CSS ou JavaScript) ou binaires (par exemple, le contenu d'un fichier multimédia ou de tout autre fichier binaire).

Les API Web

De nos jours, on n'utilise plus uniquement le protocole HTTP pour accéder à des pages Web : on s'en sert également pour échanger des données entre des applications. Pour ce faire, on programme la partie serveur d'une application Web de telle sorte que certaines de ses URL puissent retourner des données sérialisées plutôt que du code HTML. Aujourd'hui, le format de sérialisation JSON est le plus répandu pour cet usage. Anciennement, le format XML était plus populaire.

Le terme « **API Web** » désigne la partie du côté serveur d'une application Web qui permet d'échanger des données sérialisées avec celui-ci à travers le protocole HTTP. Le terme API signifie *Application Programming Interface* et désigne de façon plus générale une interface de programmation. On retrouve ce terme dans d'autres contextes en informatique, et le terme « API Web » peut lui-même aussi désigner une interface de programmation rendue disponible au langage JavaScript par le navigateur.

Les API Web seront vues plus en profondeur dans le cours *Développement d'applications Web* de votre programme.

Chiffrement

Le **chiffrement**, ou *encryption* (qui est un anglicisme[7]) désigne l'ensemble des techniques utilisant les mathématiques pour rendre un message illisible aux personnes à qui il n'est pas destiné. On ne doit pas le confondre avec l'encodage, qui désigne simplement la représentation des informations dans un format donné. Lorsque des données échangées sur un réseau ne sont pas chiffrées, on dit qu'elles sont échangées **en clair**.

La sécurité des données repose en grande partie sur le chiffrement. Nous n'allons pas étudier en profondeur les techniques de chiffrement dans ce cours, mais nous allons en introduire quelques aspects.

Chiffrement symétrique

Le **chiffrement symétrique**, ou chiffrement à clé secrète, est une forme de chiffrement qui utilise une clé unique (en d'autres termes, un mot de passe) pour chiffrer et déchiffrer un message. Pour que le destinataire d'un message puisse le déchiffrer, il doit connaître la clé qui a été utilisée pour le chiffrer. Cela pose un problème de sécurité potentiel, car il faut bien transmettre la clé au destinataire d'une

7 https://vitrinelinguistique.oqlf.gouv.qc.ca/fiche-gdt/fiche/8387607/chiffrement

façon ou d'une autre. Elle peut donc être interceptée, permettant ainsi à un tiers de déchiffrer tous les messages.

Le chiffrement symétrique peut être utilisé par exemple pour chiffrer le contenu d'un support de stockage, ou pour sécuriser un réseau Wi-Fi domestique.

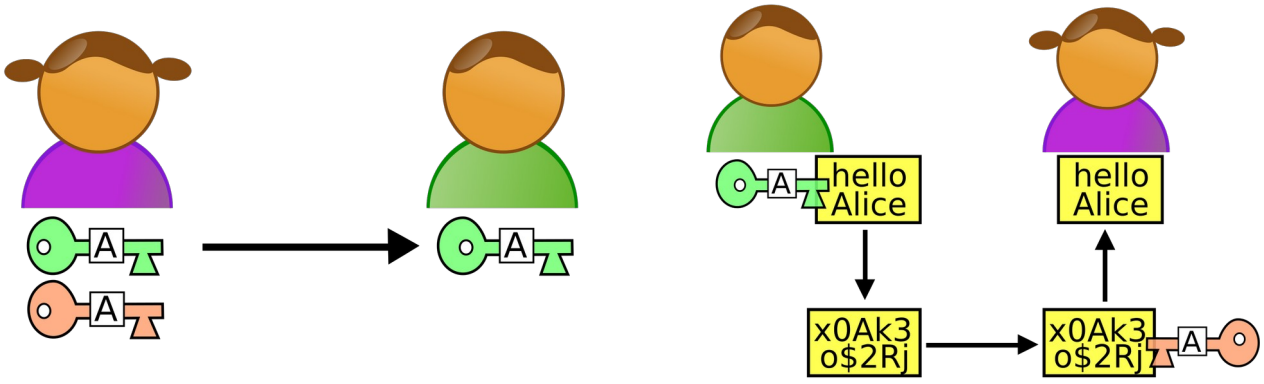
Chiffrement à clé publique

Le **chiffrement à clé publique**, ou **chiffrement asymétrique**, utilise deux clés : une **clé privée**, connue uniquement de son propriétaire, ainsi qu'une **clé publique**, qui peut être partagée à quiconque. La clé publique est générée à partir de la clé privée.

Les propriétés mathématiques sous-jacentes à ce type de chiffrement font en sorte qu'un message chiffré en utilisant la clé publique peut seulement être déchiffré en utilisant la clé privée.

Supposons ainsi que deux individus (appelons-les Alice et Bob) veulent s'échanger des messages chiffrés en utilisant le chiffrement à clé publique. Alice génère deux clés : une clé publique et une clé privée. Elle transmet sa clé publique à Bob, et conserve sa clé privée en sécurité. Bob génère aussi une clé publique et une clé privée, et transmet sa clé publique à Alice, et garde pour lui sa clé privée.

Si Bob veut envoyer un message à Alice, il le chiffre en utilisant la clé publique d'Alice. Alice déchiffre le message en utilisant sa propre clé privée.



Illustrations par l'utilisateur Odder sur Wikimedia Commons (Licence CC BY-SA 3.0)

Si Alice veut envoyer un message à Bob, elle le chiffre en utilisant la clé publique de Bob. Bob déchiffre le message en utilisant sa propre clé privée.

Cette forme de chiffrement permet d'éliminer le principal problème du chiffrement symétrique, qui nécessite de partager une clé secrète.

Ainsi, seule la clé privée associée à une clé publique permet de déchiffrer un message qui a été chiffré avec cette dernière. L'inverse est aussi vrai : un message chiffré avec une clé privée ne peut être déchiffré qu'à l'aide de la clé publique correspondante. Évidemment, cette propriété ne peut pas être utilisée pour assurer la confidentialité des données, puisque la clé publique est, justement, publique. Elle permet cependant de prouver qu'un hôte est bel et bien l'auteur d'un message. Pour ce faire, la source du message crée une **signature** en générant d'abord une valeur de petite taille (une **empreinte**

ou ***hash***) à partir du message en entier, qu'elle chiffre ensuite à l'aide de sa clé privée. La version chiffrée de l'empreinte constitue la signature. Celle-ci est transmise avec le message. Pour authentifier un message, il suffit à la destination de déchiffrer la signature en utilisant la clé publique de la source, puis de régénérer l'empreinte à partir du message en utilisant le même algorithme, pour enfin vérifier si les deux empreintes concordent. Si ce n'est pas le cas, cela signifie que le message a été falsifié.

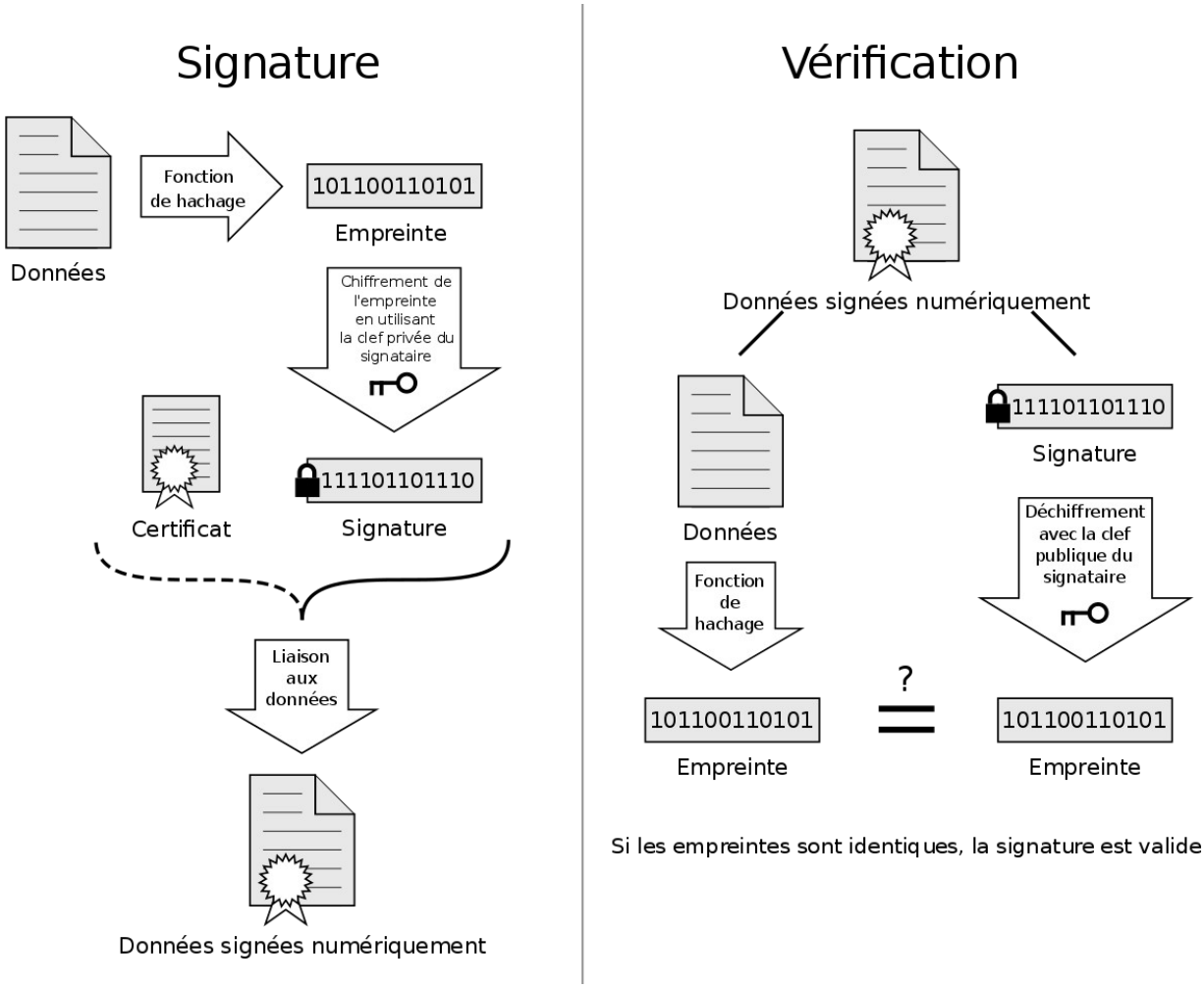


Illustration par l'utilisateur Guilieb sur Wikimedia Commons (Licence CC BY-SA 3.0)

TLS et HTTPS

TLS, pour *Transport Layer Security*, est un protocole permettant d'utiliser le chiffrement pour sécuriser des échanges sur un réseau. Le protocole HTTPS est simplement la combinaison de HTTP et de TLS.

TLS utilise à la fois le chiffrement symétrique et le chiffrement asymétrique. La clé publique du serveur est transmise au client au sein d'un **certificat** associé au nom de domaine. Celui-ci contient aussi des informations sur le serveur qu'il sécurise (ex : nom de domaine, coordonnées de l'organisation propriétaire, etc), et est signé par une autorité de certification, soit un tiers auquel le client (par exemple, un navigateur Web) fait confiance. Cela permet donc de prouver que le serveur

avec lequel le client communique est bien celui qu'il prétend être. Pour ce faire, les navigateurs utilisent des **certificats racine**, qui contiennent les clés publiques des autorités de certification.

Établissement d'une connexion TLS

L'utilisation de TLS pour sécuriser les échanges entre un client et un serveur implique un processus d'établissement de connexion (*handshake*) qui n'est pas sans rappeler celui de TCP. Ce processus suit les étapes suivantes :

1. Le client transmet au serveur une demande de connexion TLS.
2. Le serveur transmet son certificat TLS au client.
3. Le client valide la signature du certificat.
4. Le client génère une clé de chiffrement symétrique, puis la chiffre avec la clé publique du serveur.
5. Le client transmet au serveur la clé chiffrée qu'il a générée.

Une fois ces étapes complétées, le client et le serveur sont en mesure de communiquer entre eux de façon sécurisée à l'aide de la clé symétrique générée par le client. Cette façon de faire permet de sécuriser les échanges de données en utilisant le chiffrement symétrique, plus performant, tout en transmettant la clé de chiffrement de façon sécurisée via le chiffrement asymétrique, qui a l'avantage de ne pas nécessiter l'échange d'une information confidentielle.

En HTTPS, les échanges chiffrés qui suivent l'établissement de la connexion TLS sont des requêtes et des réponses HTTP.

Avantages de TLS

Lorsqu'on communique avec un serveur TLS (et donc, par exemple, avec un serveur HTTPS), on a les trois garanties suivantes[8] :

- Le serveur avec lequel on communique est bel et bien opéré par le propriétaire de son nom de domaine;
- Les données échangées avec le serveur, si elles sont interceptées par un tiers, ne peuvent pas être lues par celui-ci;
- Les données échangées avec le serveur ne peuvent pas être altérées par un tiers.

Utilisation de QUIC par HTTP/3

Comme vu précédemment, HTTP/3 utilise QUIC comme protocole de couche Transport, alors que HTTP/1 et HTTP/2 utilisent TCP. Pour rappel, QUIC est aussi un protocole fiable et avec connexion, et

8 Il existe tout de même des vecteurs d'attaque qui peuvent compromettre ces garanties : par exemple, l'installation de faux certificats racine sur le poste de travail d'un utilisateur.

il est implémenté par-dessus UDP. Dans certaines situations, il présente des avantages au niveau de la performance par rapport à TCP, qui bénéficient particulièrement à HTTP/3.

Une des particularités de QUIC est qu'il intègre directement l'utilisation de TLS. C'est donc dire qu'avec QUIC, les données sont chiffrées sur la couche Transport plutôt que sur la couche Application, et donc que toute connexion HTTP/3 est une connexion HTTPS. Par rapport à TCP, cela réduit considérablement le nombre d'échanges réseau nécessaires à l'établissement de la connexion TLS, puisque le *handshake* de TLS se fait à même celui de QUIC – alors qu'avec TCP, deux *handshakes* ont lieu (un pour TCP, puis un pour TLS). QUIC permet même de sauter complètement le processus de *handshake* lorsqu'un client veut établir une connexion à un serveur avec lequel il a déjà communiqué récemment.

QUIC offre plusieurs autres améliorations de performance qui sont utiles à HTTP/3. L'une d'elles est la possibilité d'utiliser plusieurs flux de données au sein d'une même connexion, qui sont indépendants les uns des autres du point de vue de l'acquittement et de l'ordonnancement. Cela permet donc à HTTP/3 de télécharger plusieurs fichiers à la fois sans que la perte de segments ou la livraison de segments dans le désordre pour un fichier ne ralentisse le téléchargement des autres fichiers.

Synthèse

- La **sérialisation** permet de représenter des données dans un format qui sera interprété de la même façon par les différents hôtes.
  - Elle peut être soit **binaire** ou **textuelle**.
  - Les principaux formats de sérialisation textuels sont **JSON** et **XML**.
- Les données peuvent être **compressées** avant d'être transmises sur le réseau.
- Une application peut définir son propre protocole de la couche Application, mais il existe aussi plusieurs protocoles standardisés, dont le plus répandu est HTTP.
- Une **requête HTTP** contient une **méthode**, une **URL**, des **en-têtes** et un **corps** (des données).
  - Les méthodes les plus utilisées sont **GET** et **POST**.
- Une **réponse HTTP** contient un **code d'état**, des **en-têtes** et un **corps** (des données).
- Une **API Web** utilise le protocole HTTP pour transmettre des données sérialisées plutôt que des pages Web.
- Le **chiffrement symétrique** ou à clé secrète utilise une clé unique devant être partagée entre les hôtes.
- Le **chiffrement à clé publique** utilise une clé publique et une clé privée. L'une est nécessaire pour déchiffrer un message chiffré avec l'autre.
- Une **signature** utilise le chiffrement à clé publique pour authentifier l'auteur d'un message.

- **TLS** est un protocole de sécurisation des échanges réseau utilisant le chiffrement, et **HTTPS** est la combinaison de HTTP et de TLS.
- **TLS** nous donne trois garanties :
  - Le serveur avec lequel on communique est bel et bien opéré par le propriétaire de son nom de domaine;
  - Les données échangées avec le serveur, si elles sont interceptées par un tiers, ne peuvent pas être lues par celui-ci;
  - Les données échangées avec le serveur ne peuvent pas être altérées par un tiers.
- Avec **QUIC**, TLS est géré directement par la couche Transport, ce qui permet à **HTTP/3** de bénéficier d'améliorations de performance par rapport aux versions précédentes de HTTP, qui utilisent TCP.

Index

API Web..... 12
ASCII..... 3
boutisme..... 3
certificat..... 14
certificats racine..... 15
chiffrement..... 12
chiffrement à clé publique..... 13
chiffrement asymétrique..... 13
chiffrement symétrique..... 12
clé privée..... 13
clé publique..... 13
code d'état HTTP..... 11
compression..... 6
corps..... 10
décompression..... 6
DNS over HTTPS..... 7
DNS over QUIC..... 7
DoH..... 7
DoQ..... 7
empreinte..... 13
en clair..... 12
en-têtes de la réponse..... 12
en-têtes de la requête..... 10
endianness..... 3
FTP..... 7
FTPS..... 8
GET..... 10
hash..... 14
HTTP..... 7

HTTP/3....7
HTTPS....7
IMAP....8
JSON....5
méthode....10
P2P....9
pair-à-pair....9
POP....8
POST....10
sérialisation....3
sérialisation binaire....3
sérialisation textuelle....4
SFTP....8
signature....13
SMTP....8
SSH....8
URL....10
UTF-8....3
WebSocket....8
XML....4

Références

- TANENBAUM, Andrew et WETHERALL, David, *Réseaux, 5e édition*, Paris, Pearson Education France, 2011, p. 689-738, p. 817-930.
- https://en.wikipedia.org/wiki/Session_layer
- https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange
- https://fr.wikipedia.org/wiki/UTF-8
- https://www.sfml-dev.org/tutorials/2.5/network-packet-fr.php
- https://fr.wikipedia.org/wiki/Boutisme
- https://fr.wikipedia.org/wiki/S%C3%A9rialisation
- https://medium.com/testable-io/json-vs-binary-flatbuffers-5490e67f785f
- https://fr.wikipedia.org/wiki/Compression_de_donn%C3%A9es
- https://en.wikipedia.org/wiki/Secure_Shell
- https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
- https://fr.wikipedia.org/wiki/Internet_Message_Access_Protocol
- https://fr.wikipedia.org/wiki/Post_Office_Protocol
- https://fr.wikipedia.org/wiki/WebSocket

© 2024 Pier-Luc Brault

- https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP
- https://fr.wikipedia.org/wiki/Transport_Layer_Security
- https://fr.wikipedia.org/wiki/Certificat_%C3%A9lectronique
- https://en.wikipedia.org/wiki/Domain_Name_System
- https://en.wikipedia.org/wiki/QUIC
- https://fr.wikipedia.org/wiki/BitTorrent
- https://fr.wikipedia.org/wiki/Pair-%C3%A0-pair