Cryptographie 2ème partie

Guillaume Bienkowski — Braincube

Plan

- Fonctions de hachage (~1h)
- Signatures digitales (~1h)
- Certificats (~1h)
- TP tout le long (~1h)

Fonction de hachage

Définition

> Une fonction de hachage est une fonction qui convertit une entrée de taille arbitraire en sortie de taille fixe

Terminologie

Le résultat d'une fonction de hachage s'appelle un *hash* ou une *empreinte*, parfois un *condensat*

Fonction de hachage parfaite $f : X \to Y$:

$\forall x \in X, \forall x' \in X$ alors $(f(x) = f(x') \implies x = x')$

On dit que f est *injective*.

Implications: si X est grand alors Y sera au moins aussi vaste => ne rentre pas dans la RAM d'un ordinateur

En pratique: Y a une taille finie (énorme, mais finie).

Exemples de fonctions de hachage (imparfaites...)

- famille SHA (1, 256, ...) ($160 \leq d \leq 512$)
- MD5 ($d = 128$),
- Whirlpool ($d = 512$)
- BCrypt ($d = 192$), Scrypt, Argon2 ($d = 256$)
- $maFonction(x) = x \mod 32768$ (pas très efficace celle là)

Exemple sous linux

```
$ (echo "123456789" | shasum); (echo "123456779" | shasum)
                ^_____________________________^

6d78392a5886177fe5b86e585a0b695a2bcd01a05504b3c4e38bc8eeb21e8326
8f9d6dbc5c656b3fd63f25e72c3ec9d7738f198238a46eeb01875ee102c34860
```

Petit changement en entrée => grande différence en sortie

Collisions

Une collision, c'est quand deux valeurs en entrée d'une fonction de hachage donnent la même sortie.

Par exemple:

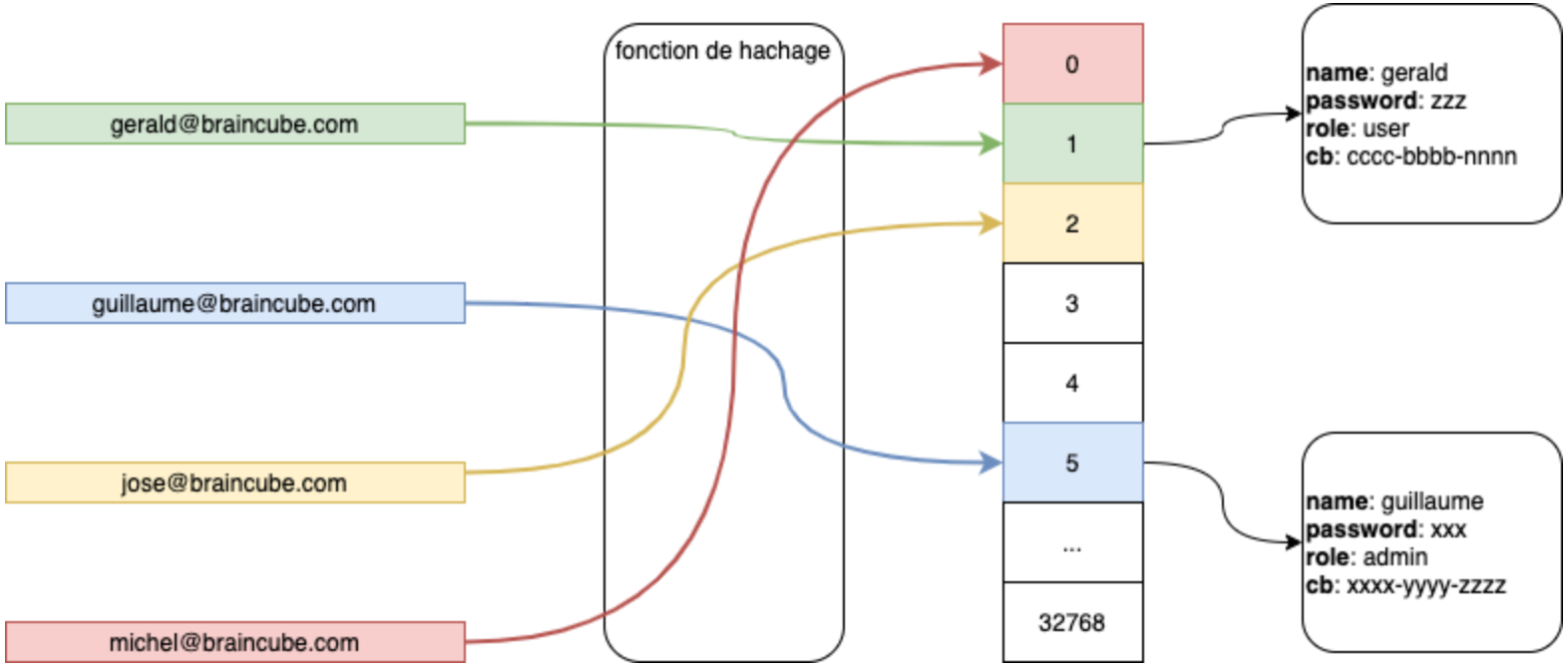
```
>>> def mafonctiondehachage(x):
...   return x % 32768
...
>>> mafonctiondehachage(10)
10
>>> mafonctiondehachage(32778)
10 # :-( je l'avais dit qu'elle n'était pas terrible cette fonction
>>>
```

Propriétés souhaitées d'une fonction de Hachage

- $f(x) = y$ est rapide à calculer
- Trouver x à partir de y est quasi impossible ("fonction à sens unique")
- Il doit être quasi impossible de trouver deux x et x' qui donnent le même *hash* (collision)
- Les valeurs retournées sont réparties uniformément dans l'espace de sortie Y

Applications des hash

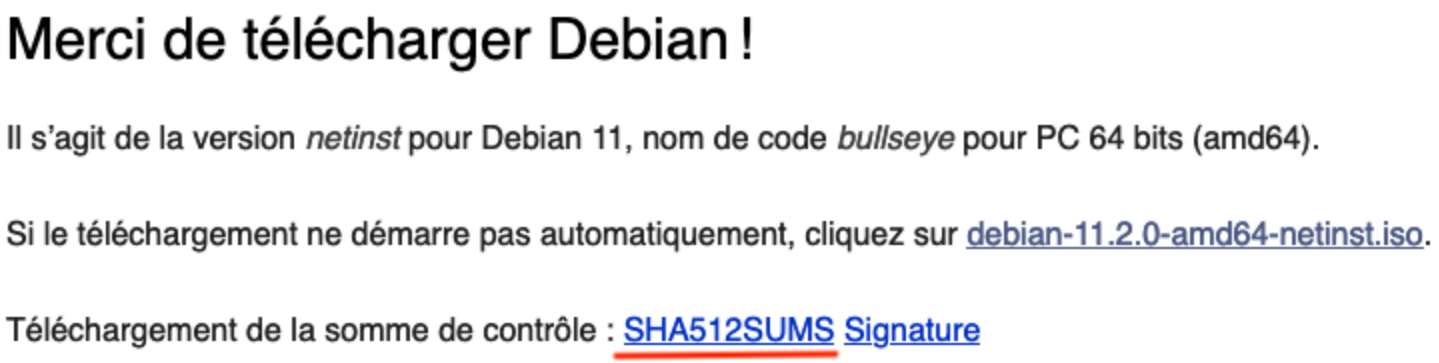
HashMaps, Dictionnaires (Programmation)



Avantages: $f(x)$ TRÈS rapide vs itération systématique du tableau

Vérification d'intégrité

Rappel: différences minimes => hash complètement différents



Merci de télécharger Debian !

Il s'agit de la version *netinst* pour Debian 11, nom de code *bullseye* pour PC 64 bits (amd64).

Si le téléchargement ne démarre pas automatiquement, cliquez sur debian-11.2.0-amd64-netinst.iso.

Téléchargement de la somme de contrôle : SHA512SUMS Signature

Permet de s'assurer que le fichier (gros) qu'on a téléchargé est bien le même que celui hébergé par le serveur.

```
c685b85cf9f248633ba3cd2b9f9e781fa03225587e0c332aef2063f6877a1f0622f56d44cf0690087b0ca36883147ecb5593e3da6f965968402cdbdf12f6dd74  debian-11.2.0-amd64-netinst.iso
f2da0996196f19585b464e48bfb6b8e4938eb596667d92a5ebd428e1a88a1a115c00f1d052f350eca44fa08f42f0500c63351763dfb47f1e1783f917590c175d  debian-edu-11.2.0-amd64-netinst.iso
9b5b0475fbb3235ebb7da71415f10921b02131b7debc9325403f85f9f6798a3e902d6257831a7ec9c7aef3256817fb76c4f01bb5d035bfcdb3dc24da24fa1bda  debian-mac-11.2.0-amd64-netinst.iso
```

Stockage obfusqué d'un password

On ne stocke jamais un mot de passe en clair dans une base de données:

Id	Login	Password	PasswordDate
1	guillaume@braincube.com	ÇeCi3stUnRég4l!	01/01/2022

=> *NON*

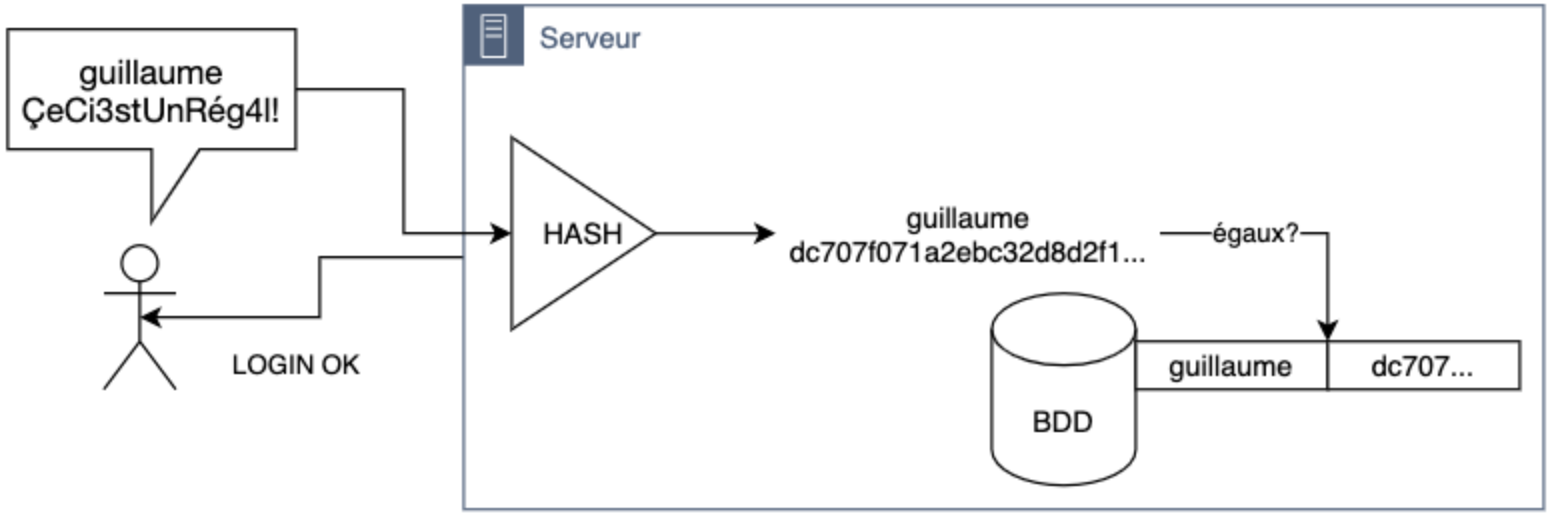
On stocke un hash:

Id	Login	PasswordHash	PasswordDate
1	guillaume@braincube.com	dc707f071a2ebc32d8d2f1128bb7565bc017573bed994f99100db1b14cd75c1b	01/01/2022
		=sha256sum(password)	

=> *Mieux*

Si la BDD est piratée, impossible de dériver (facilement) le mot de passe.

Vérification du mot de passe



L'utilisateur envoie son login et mot de passe, on le hash, et on vérifie que ce hash est le bon pour cet utilisateur.

Attaques: **Rainbow Tables** et autres attaques basées sur un dictionnaire

Exercice mental:

- longueur password = 8
- symboles alphanumériques (`0–9` , `a–z` , `A–Z`) => 62 symboles différents
- alors on a $62^8 = 218340105584896$ combinaisons possibles
- chaque sha256 prend.. 256bits (32 octets)
- Le stockage nécessaire pour stocker un dictionnaire inverse:
  $62^8 * 32bits = 6,2\ Pio$

À la portée de n'importe quelle Fortune 500, voire du commun des mortels pour quelques heures en louant du disque chez Amazon.

Solution: *saler* son hash.

Salage

On ajoute un petit peut d'aléatoire dans le hash pour tempérer les attaques par dictionnaires.

```
import hashlib

def sha256(s):
  return hashlib.sha256(s.encode('utf-8')).hexdigest()

password = '4gNnLar5'
salt = 'UneStringRandomDeLongueurEntre5Et16'

standard_hash = sha256(password)
# c36b9fc5e51d59f5179e9cc2a0e1f02ea6c2f12448e9e1dfe01f68786092a924

salted_hash = salt + '!' + sha256( salt + password )
# UneStringRandomDeLongueurEntre5Et16!93d4b242b475a73d8f2d1de1...
```

=> password de longueur 8, mais sha256 basé sur une longueur aléatoire.

Autres applications

- HaveIBeenPwnd
- Private Contact Discovery (Signal)
- Blockchain
- Digital Signature (ça tombe bien c'est la suite, mais d'abord...)

À vous!

Rendez-vous ici: https://masterind4.github.io

Ou directement: `git clone`

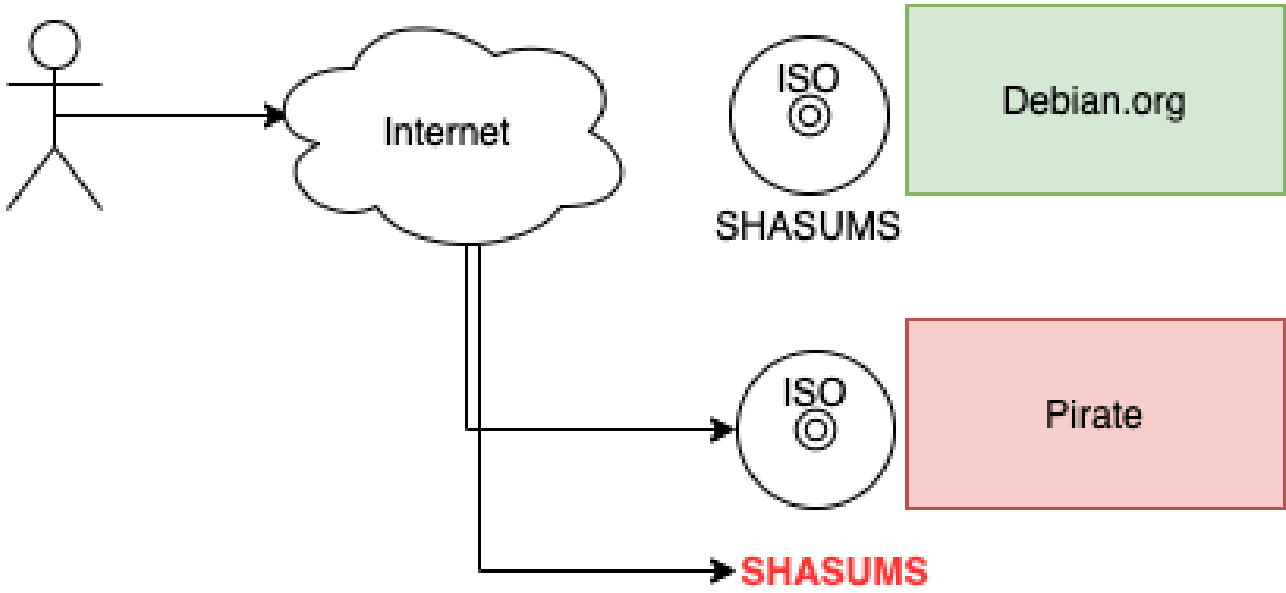
```
https://github.com/masterind4/masterind4.github.io.git
```

Signatures digitales

Revenons au téléchargement de notre fichier iso Debian:

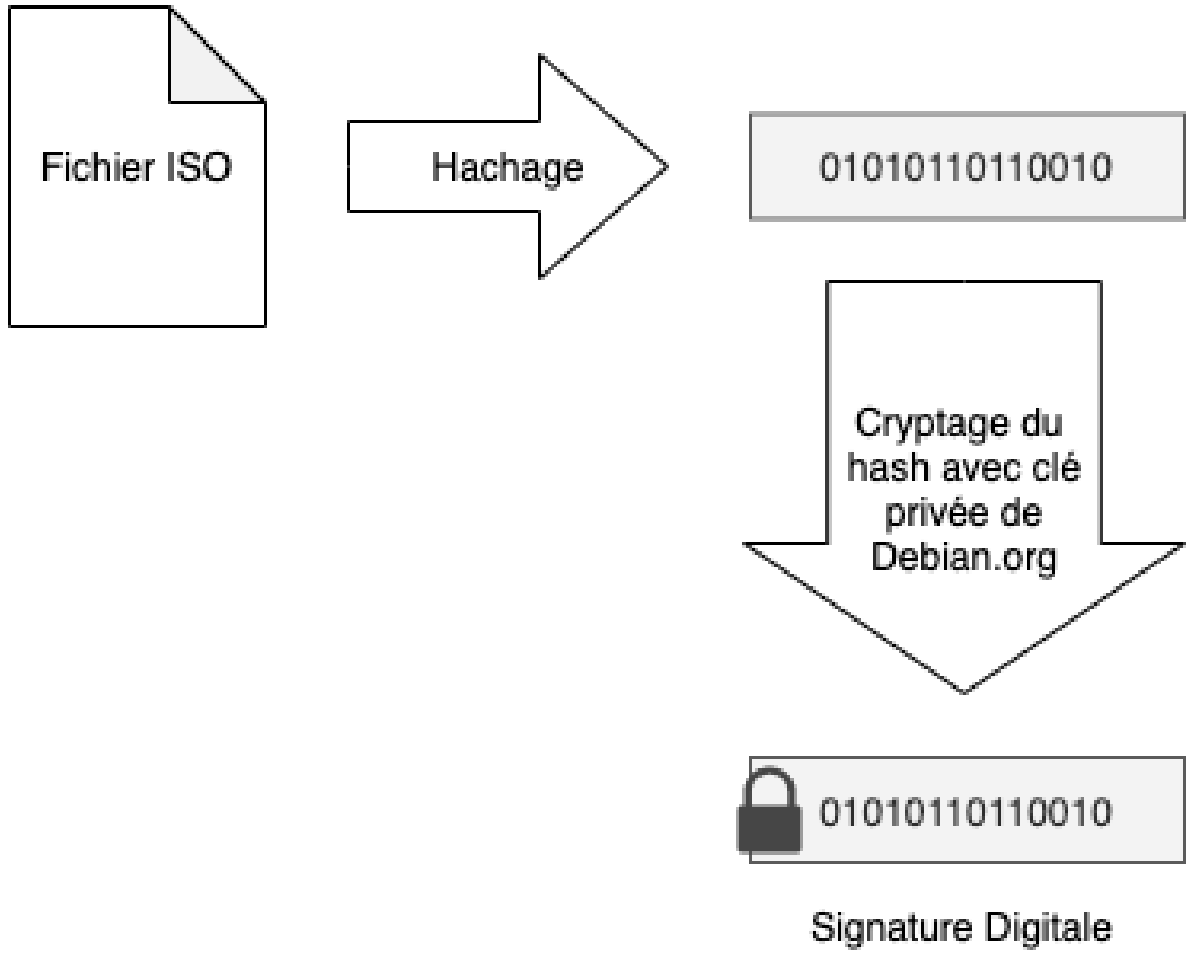
Si $sha256(fichier) = S_{Controle}$ alors je sais que le fichier est *intègre* (les octets sont les mêmes que sur le serveur)

Comment vérifier l'*authenticité* du fichier? (que c'est bien Debian qui me l'a fourni)
=> La signature digitale.



Principe

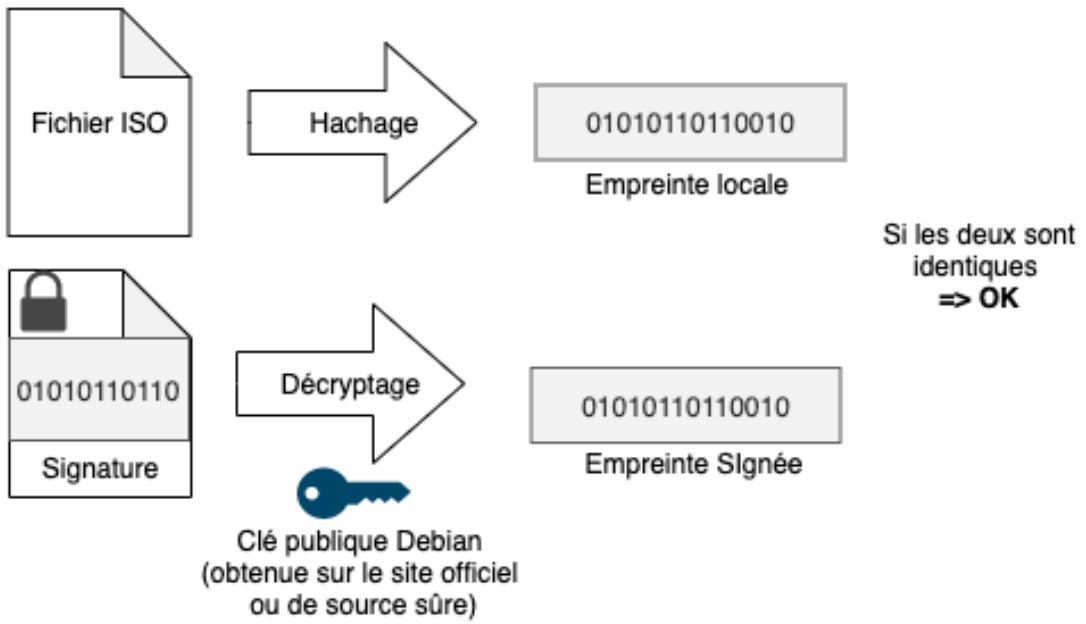
- on crypte *l'empreinte* avec une clé privée (c'est la *signature*)
- on diffuse la clé publique associée
- les gens sont capables de décrypter la signature, récupérer l'empreinte, et vérifier par eux mêmes



Vérification

On a un ISO et un fichier de signature

- on calcule le hash de l'iso
- on decrypte la signature et on obtient le hash théorique
- Si les deux sont identiques: c'est bien Debian qui a fabriqué cet ISO



On continue sur le TP, exercices sur la signature digitale.

Récapitulatif

- Un *hash* (ou empreinte) assure l'intégrité de la donnée
- Une signature assure *l'authenticité* de la donnée
- (Le chiffrement assure la *confidentialité* de la donnée, cf ce matin)

Certificats

Certificats

Les certificats sont à la base de:

- l'internet moderne
- vos apps Android, iOS
- la sécurité en entreprise
- l'interception et l'analyse du traffic dans les pays liberticides
- une myriade d'usages dès qu'il y a besoin d'authentification

Format standardisé: X.509

> Tout est question de confiance
> -- Guillaume B., 2022

Un certificat contient:

- une clé *publique*
- les informations du certificat (exemple: nom de domaine lié à ce certificat, date d'expiration, etc.)
- une signature de ce certificat (rappel: signature = `chiffrage( clé privée, HASH( contenu du certificat) )`) par un autre certificat tiers

La clé *privée* associée à la clé *publique* permet d'authentifier celui qui présente le certificat

La signature du certificat provient:

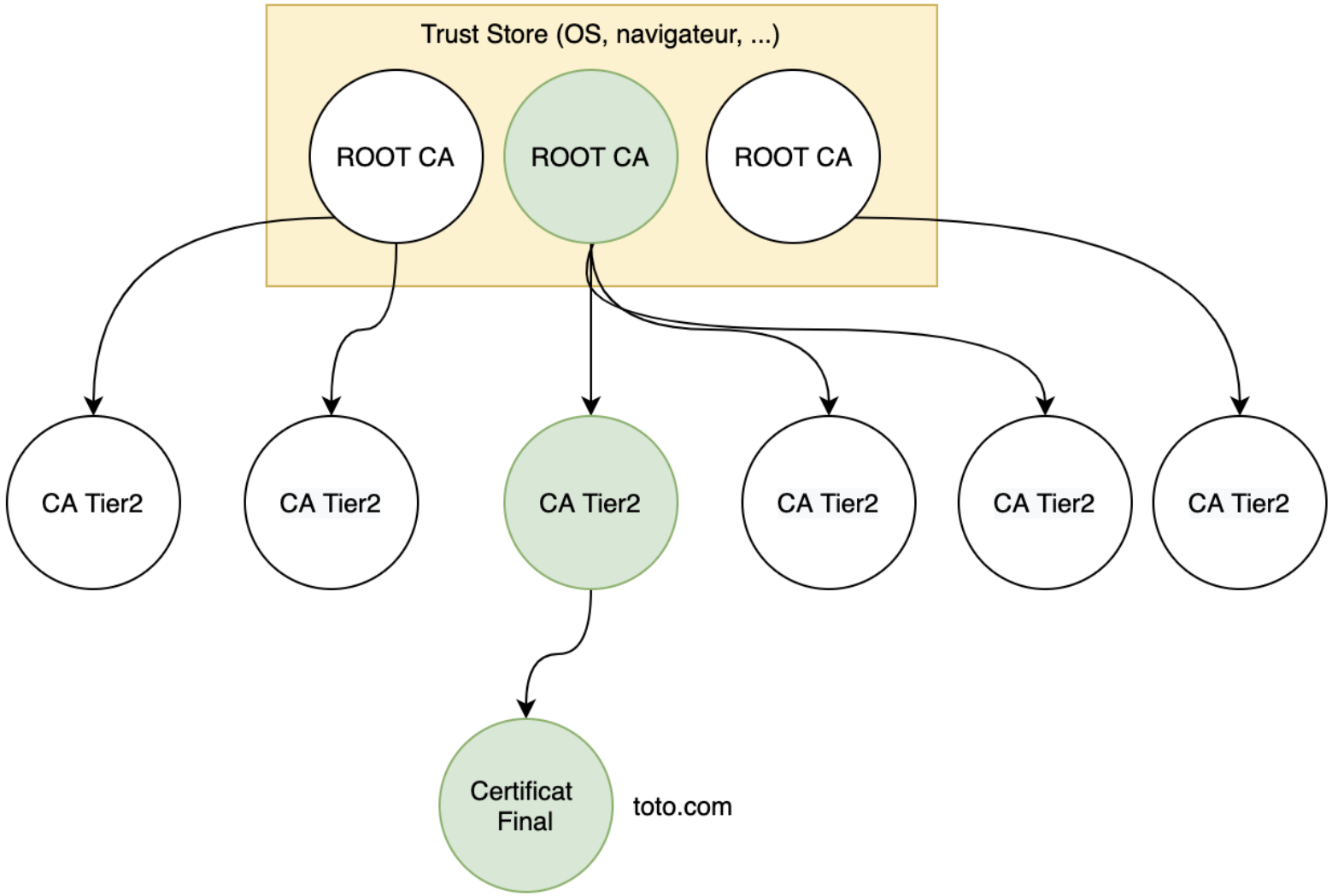
- d'une autorité de certification dont la clé publique est dans le porte clé de confiance (dans l'OS ou le navigateur)
- ou de la machine qui présente le certificat ("certificat autosigné"), dans ce cas la clé privée

Chaîne de certification

Exemple pour les certificats TLS utilisés sur internet:

- Les navigateurs n'embarquent que les certificats des autorités dite "Racines", qui sont de gros groupes commerciaux audités ou des gouvernements.
- Ces certificats racines signent des certificats intermédiaires, et les fournissent aux autorités de certification "Tier 2", qui peuvent à leur tour signer des certificats
- Moi, toto.com, demande à une de ces autorités de certification tier 2 de signer mon certificat avec sa clé privée, moyennant finances et preuves que je possède bien ce nom de domaine.
- La chaîne de confiance s'établit de proche en proche

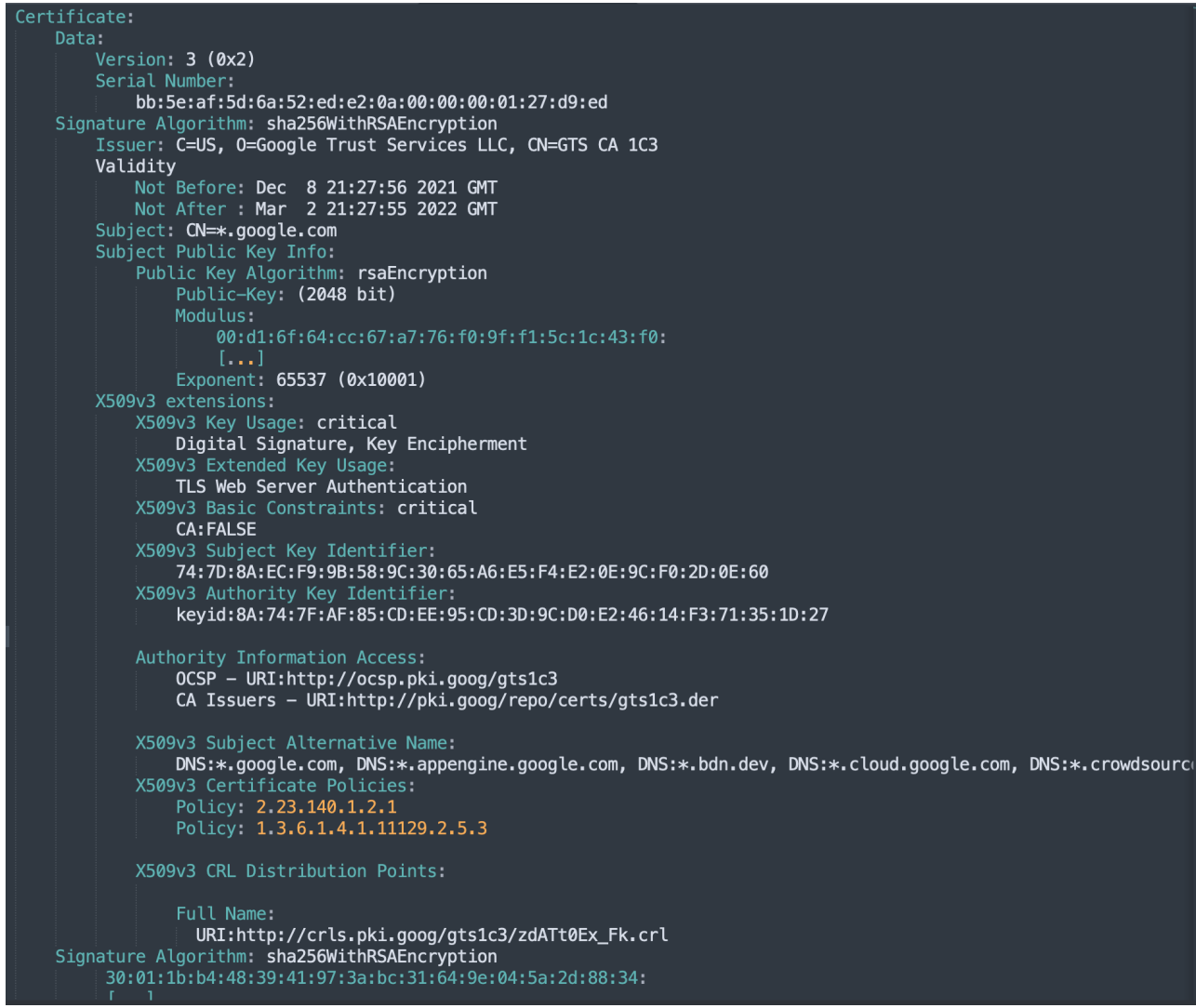
Chaîne de certification



Anatomie d'un certificat

```
$ echo \
    | openssl s_client -connect google.com:443  2>/dev/null \
    | openssl x509 -text
```

(lancez la commande dans votre terminal et admirez la liste de DNS Google)



Établissement de connection SSL/TLS

TLS 1.3

1. Le client établit un canal sécurisé avec le serveur (en utilisant un echange via Diffie-Helman)
2. Le serveur web présente son certificat au client
3. Le client vérifie la chaîne de certificats pour authentifier le serveur. Envoie un challenge encrypté avec la clé publique du serveur, et envoie sa clé publique
4. Le serveur web décrypte le challenge avec sa clé privée et le crypte avec la clé publique du client
5. Si le serveur arrive à renvoyer le challenge au client, c'est qu'il possède la clé privée.

Compromission d'un certificat

Si la clé privée associée à un certificat est exposée, alors ce certificat devient *compromis*: toute personne qui possède la clé privée peut l'utiliser pour se faire passer pour le serveur légitime.

Dans ce cas, on *révoque* le certificat

2 solutions:

- Certificate Revocation List
- Online Certificate Status Protocol (vulnérable à une attaque "replay")

Les urls de ces deux mécanismes sont intégrées aux certificats sous la forme d'attributs (tout comme la date ou le numéro de série)

Certificate revocation list

Une simple liste, signée cryptographiquement par l'autorité de certification, qui liste les certificats révoqués.

Avantages:

- simple à créer,
- simple à publier (même en HTTP)

Inconvénients:

Certificate revocation list

Une simple liste, signée cryptographiquement par l'autorité de certification, qui liste les certificats révoqués.

Avantages:

- simple à créer,
- simple à publier (même en HTTP)

Inconvénients:

- la liste grossit régulièrement => chaque réponse doit envoyer la CRL en entier
- 99% inutile: je ne suis intéressé que par mon certificat, pas le reste de la liste

OCSP

Méthode plus récente, basée sur une API:

- je veux savoir si le certificat présenté par toto.com est valide
- je contacte l'URL OCSP indiquée dans le certificat avec le Serial du certificat à vérifier
- l'URL répond avec un "Good" / "No good" **signé** pour dire si le certificat est bon ou pas

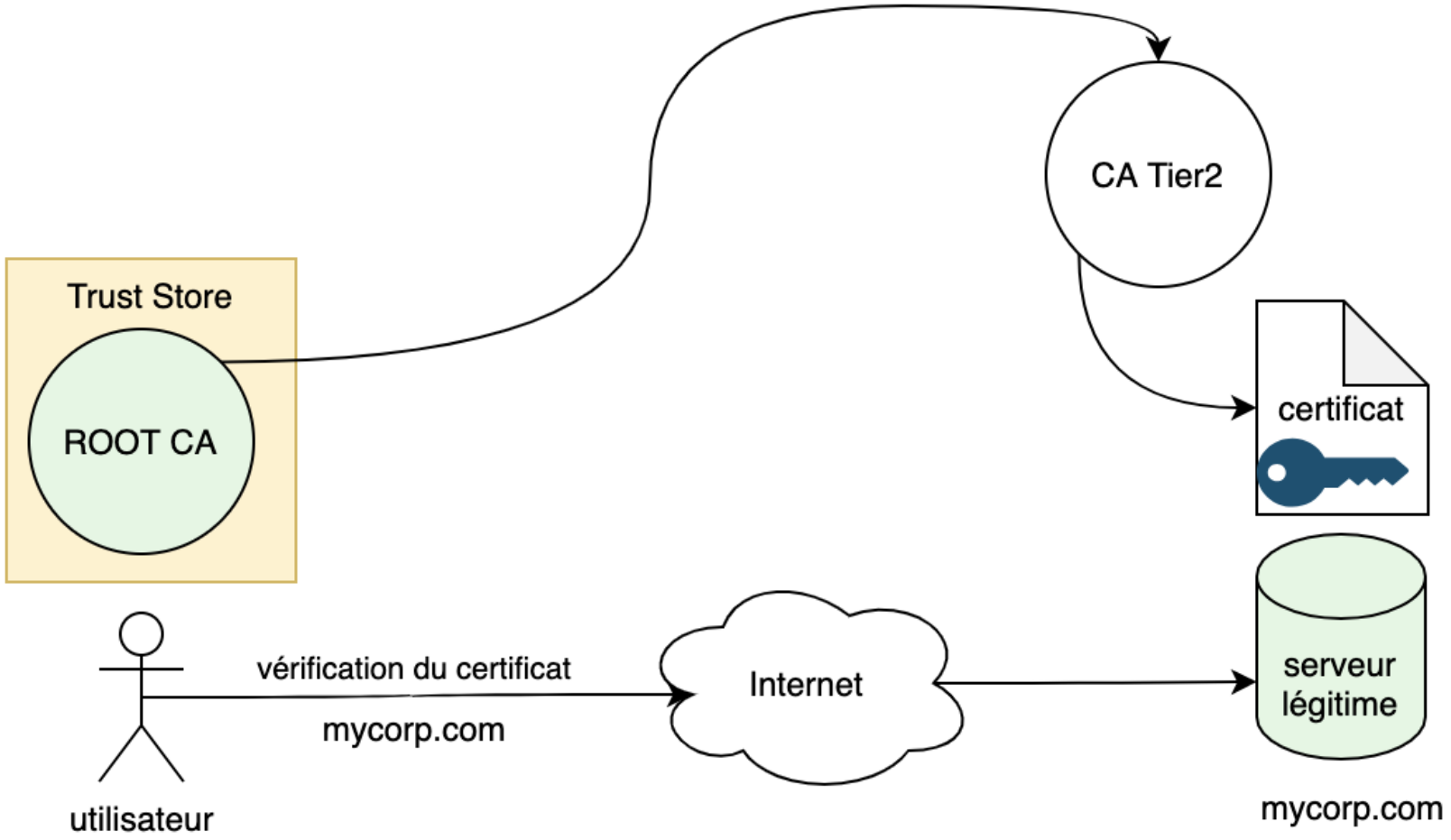
Avantages:

- très succinct: j'ai la réponse pour MON certificat, et seulement lui
- et donc très économe en resources et rapide à répondre

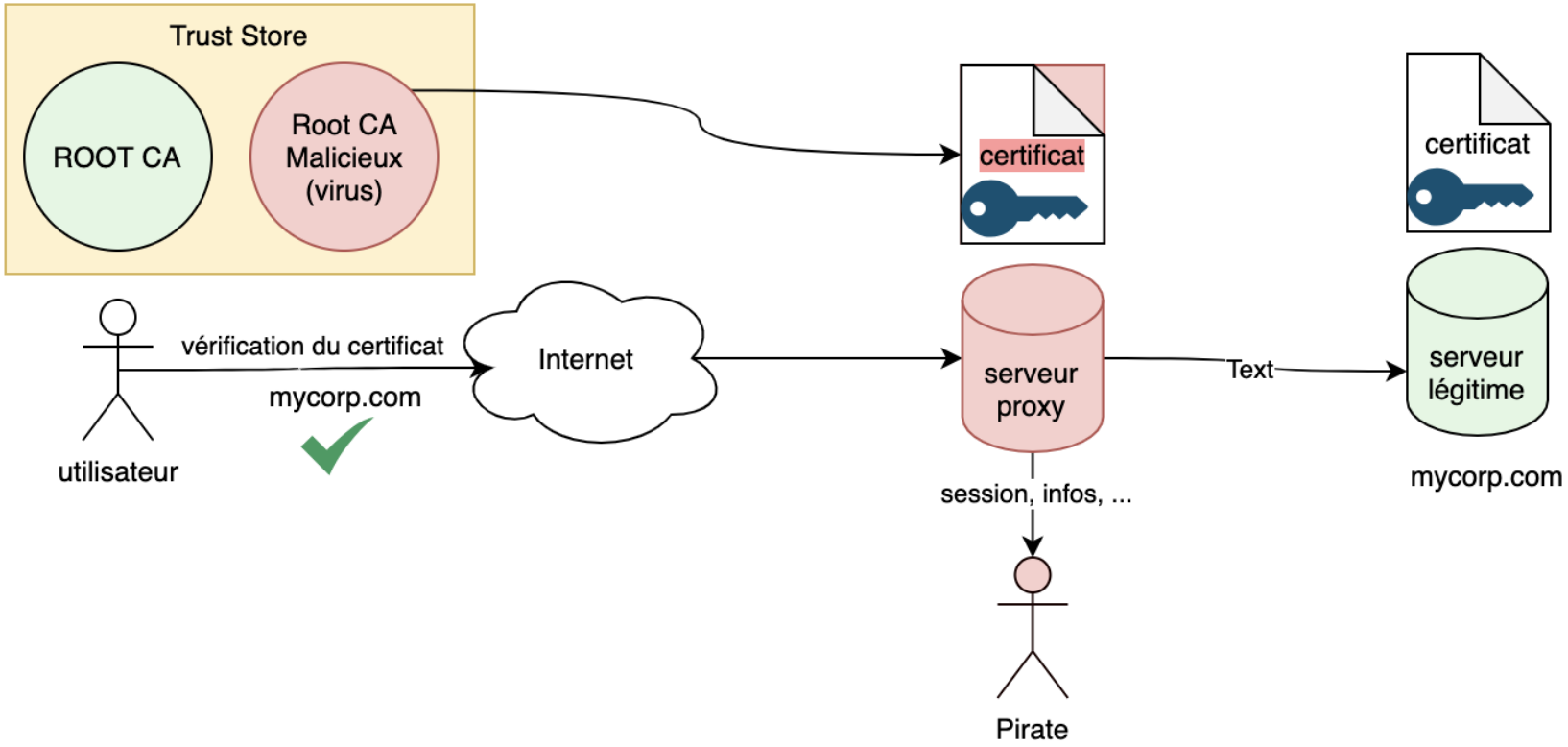
Inconvénients:

- réponses signées mises en cache, sans expiration ni *nonce*, et donc susceptibles d'être rejouées à un client.

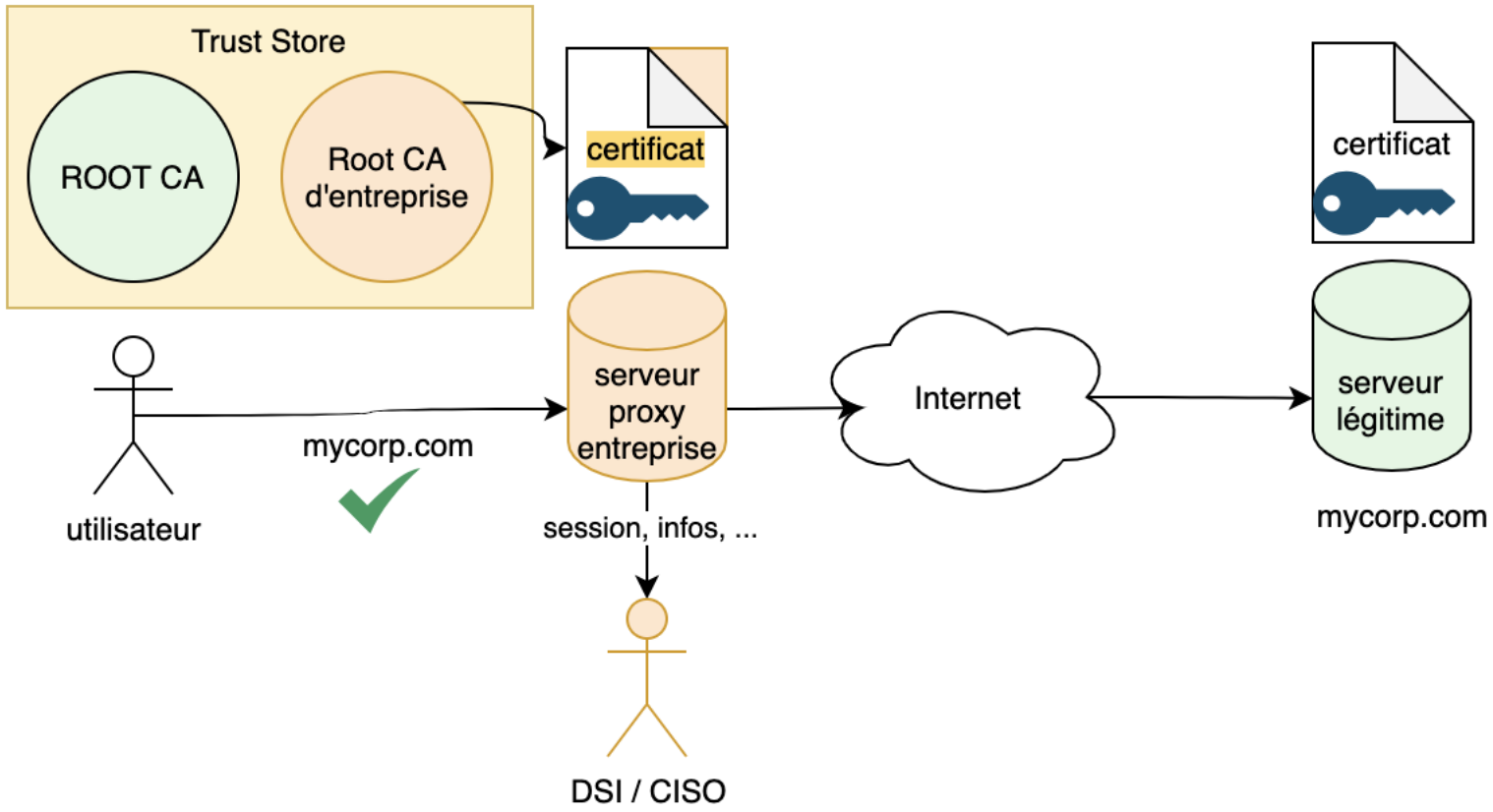
MITM (Man In The Middle)



MITM (Man In The Middle)



MITM (Man In The Middle)



Authentification par certificat client

Je demande à mon interlocuteur de me fournir un certificat. Pour cela:

1. Je crée un couple de clés, et je crée un certificat non signé avec la clé publique
2. J'envoie ce certificat pour signature ("CSR") au serveur web, qui me le renvoie signé.

Au prochain login, j'envoie mon certificat au serveur.

1. Il peut vérifier qu'il l'a bien signé en vérifiant la signature;
2. Il peut m'authentifier via l'authentification par clé publique, car ma clé est présente dans le certificat (Challenge/Réponse)
3. De mon côté je suis sûr de parler au même serveur car il m'a fourni sa clé publique.

Authentification très forte, résiste aux MITM.

Questions

Trivia

Certificats bloqués

Nom du certificat	Publié par	Type	Dimension de clé	SIG ALG	Numéro de série	Échéance	Politiqu EV
*.EGO.GOV.TR	TÜRKTRUST Elektronik Sunucu Sertifikası Hizmetleri	RSA	2 048 bits	SHA-1	08 27	7 h 07, 51 s, 6 juil 2021	Non EV
*.google.com	*.EGO.GOV.TR	RSA	1 024 bits	SHA-1	0A 88 90 40 CE 12 6E 65 57 AE C2 42 7B 4A C1 FB	19 h 43, 27 s, 7 juin 2013	Non EV

https://support.apple.com/fr-sn/HT212248

Trivia

Lenovo Superfish

- Embarque dans tous les ordinateurs Lenovo un certificat racine autosigné
- Embarque aussi la clé privée (protégée par mot de passe)
- Le mot de passe est trouvé facilement, et des certificats bidons peuvent être générés (pour google.com par exemple)
- TOUS les ordinateurs Lenovo pourront être dupés par ces certificats

Source

Biblio

Exemple d'échange de clé DH (Wikipédia)

Exemple de chiffrage clé publique (RSA) (Wikipédia)

Support TLS du browser

RFC Certificats x509

Décoder un pem (mais sinon, utiliser openssl)

Chiffrement RSA (Wikipédia)

Chiffrement par courbes elliptiques (Wikipédia)

Un site qui explique tout ça vraiment bien (et en français)

Si on a le temps

Blockchain, Proof Of Work, Bitcoin.