

Cryptographie 2ème partie

Guillaume Bienkowski — Brincube

Plan

Cours

- Fonctions de hachage
- Signatures digitales
- Certificats

→ Durée 1h30. 40 Slides. 2 minutes par slides, c'est *intense*.

Mise en pratique (TP)

→ Durée 1h30

Fonction de hachage

Définition

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

Terminologie

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

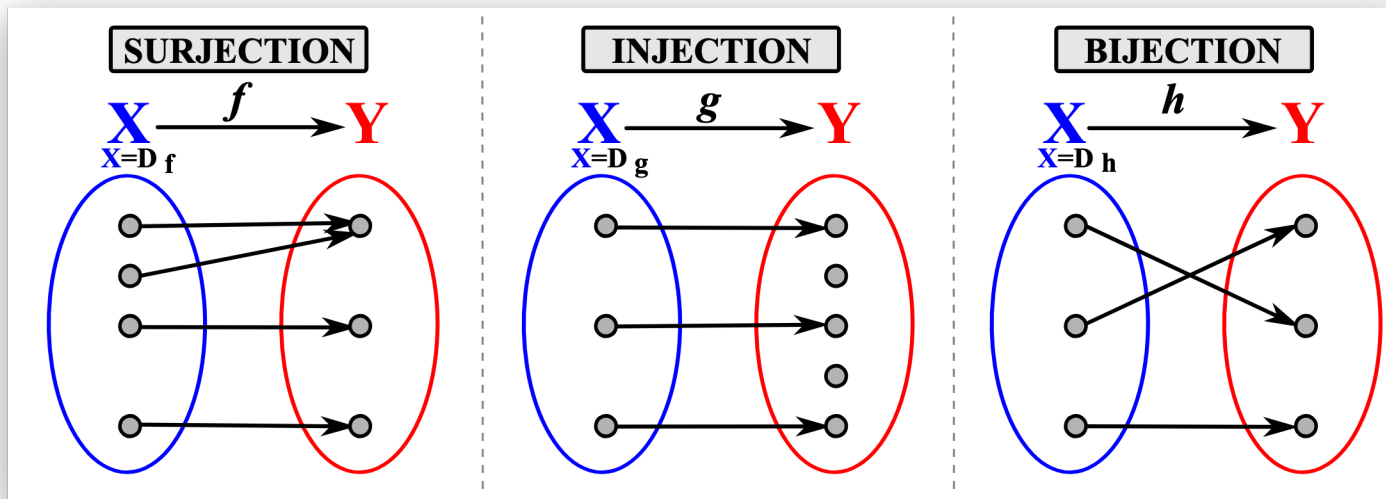
Fonction de hachage parfaite $f : X \rightarrow Y$:

$\forall x \in X, \forall x' \in X$ 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 \Rightarrow ne rentre pas dans la RAM d'un ordinateur

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



Collision

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  
>>>
```

Fonction de hachage cryptographique

Une fonction de hachage cryptographique idéale possède les six propriétés suivantes:

- un même message aura toujours la même valeur de hachage (**déterminisme**)
- le hash d'un message se calcule "facilement" (pour un ordinateur)
- impossible, pour un hash donné, de construire un message ayant cette valeur (**préimage**)
- impossible de trouver un second message ayant le même hash qu'un message donné (**seconde préimage**)
- impossible de trouver deux messages différents ayant le même hash (**collision**)
- modifier un tant soit peu un message modifie **considérablement** la valeur de hachage

Exemples de fonctions de hachage cryptographiques

d est la taille de l'ensemble de sortie Y en bits.

- famille SHA (1, 256, ...) ($160 \leq d \leq 512$)
- MD5 ($d = 128$), dépréciée! Ne jamais utiliser.
- Whirlpool ($d = 512$)
- BCrypt ($d = 192$), Scrypt, Argon2 ($d = 256$)
- $maFonction(x) = x \bmod 32768$ ($d = 15$) (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 trouvées pour:

- MD5 créé en 91, collision en 96.
- SHA1 créé en 95, fragile dès 2005, collision en 2017

PDF de résultats des **élection**
Américaines de 2008

We have prepared twelve different predictions, ten of which are shown in the table below.

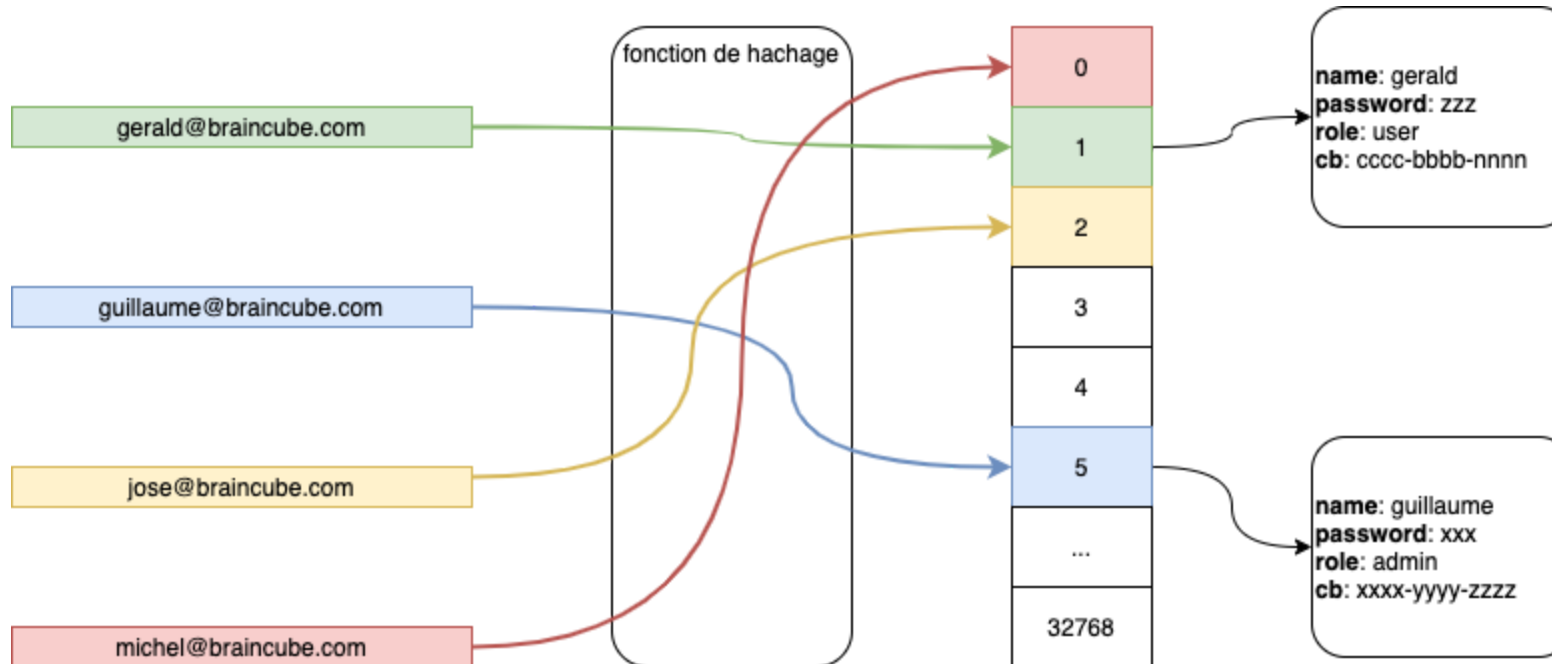
A:  John Edwards.pdf	G:  Fred Thompson.pdf
B:  John McCain.pdf	H:  (hidden)
C:  Mitt Romney.pdf	I:  Paris Hilton.pdf
D:  Ralph Nader.pdf	J:  Al Gore.pdf
E:  (hidden)	K:  Jeb Bush.pdf
F:  Barack Obama.pdf	L:  Oprah Winfrey.pdf

All twelve documents we prepared, the ten given above and two hidden ones, have the MD5 hash value

3D515DEAD7AA16560ABA3E9DF05CBC80.

Applications des hash

HashMaps, Dictionnaires (Programmation)



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

Vérification d'intégrité

Rappel: différences minimales => hash complètement différents

Téléchargement de Debian

Il s'agit de la version *netinst* pour Debian 12, nom de code *bookworm* pour PC 64 bits (amd64) [debian-12.4.0-amd64-netinst.iso](#).

Téléchargement de la somme de contrôle : [SHA512SUMS](#) [Signature](#)



Important : [Veiller à vérifier la somme de contrôle.](#)

Les ISO de l'installateur sont des images hybrides, ce qui signifie qu'elles peuvent être écrites directement sur des CD, DVD ou BD, ou sur des [clés USB](#).

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

0262488ce2cec6d95a6c9002cfba8b81ac0d1c29fe7993aa5af30f81cecad3eb66558b9d8689a86b57bf12b8cbeab1e11d128a53356b288d48e339bb003dace5e0da46ec9a0000c79f8c1abe08798afa674b71a4f232b4760c28bcb1717d3c7f6962fa1523769576a924c879710cde6f5ac82402633c4b84c8da6a48c37d61edfecfa87d9bcf555a2181078e1d629ebd11554b78fbd59888a438ec2c1b4f4cb27eb75013ebdb8f86cfac5fad19a07bca84dc1bc00b0787fc842d5e318748de84	debian-12.4.0-amd64-netinst.iso debian-edu-12.4.0-amd64-netinst.iso debian-mac-12.4.0-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	ÇeCi3stUnRég4ll	01/01/2022

=> *NON*

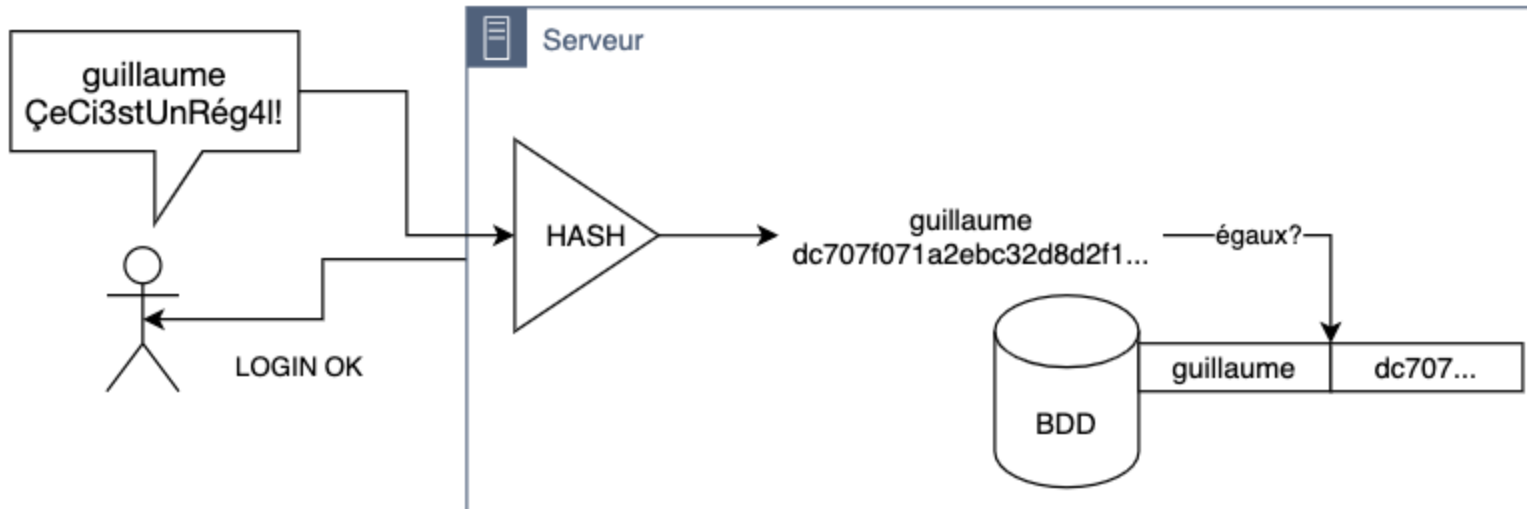
On stocke un hash:

Id	Login	PasswordHash	PasswordDate
1	guillaume@braincube.com	dc707f071a2ebc32d8d2f1128bb756 5bc017573bed994f99100db1b14cd7 5c1b	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

- 1 mot de passe de 8 caractères de long contient (0-9 , a-z , A-Z) => 62 symboles différents
- $62^8 = 218340105584896$ combinaisons possibles, un sha256 est stocké sur 32 bits
- $62^8 * 32\text{octets} = 6.2 \text{ } Pio$

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 - Problématique

User	Hash
alice	4420d1918..
jason	695ddccd9..
mario	cd5cb49b8..
teresa	73fb51a0c..
bob	4420d1918..
mike	77b177de4..

2 utilisateurs ont manifestement le même mot de passe. Qui sont-ils?

Salage - comment on fait

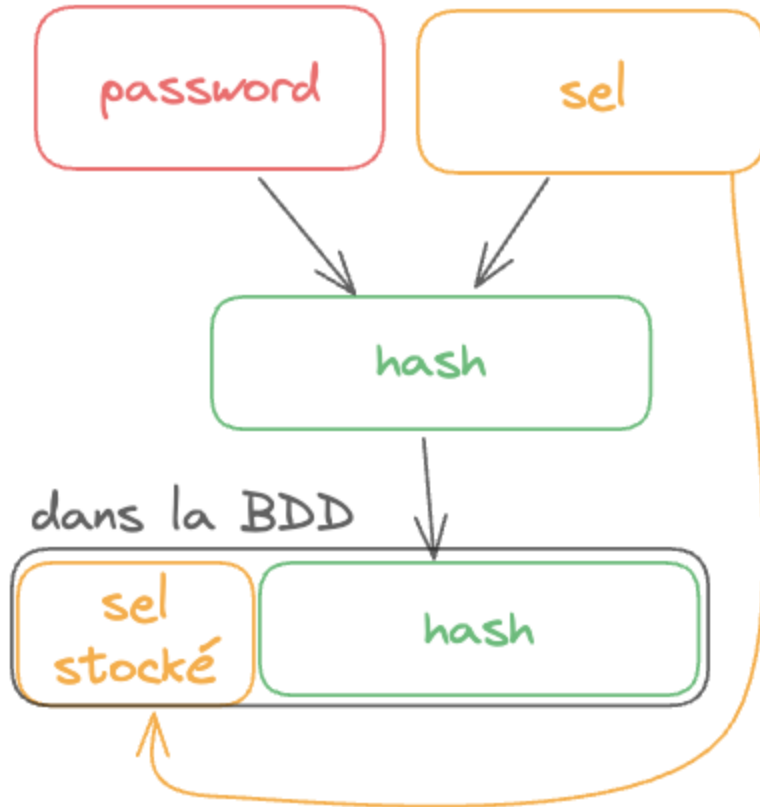
On ajoute un petit peu d'aléatoire dans le calcul du hash pour tempérer les attaques par dictionnaires.

```
password = '4gNnLar5'  
salt = getRandomString(length=math.randint(5,15))  
# par exemple salt='VmBgeaZ2PN', différent pour chaque password stocké!  
  
normal_hash = sha256(password)  
salted_hash = salt + '!' + sha256( salt + password ) # '!' sépare le sel du hash  
  
# normal hash: c36b9fc5e51d59f5179e9cc2a0e1f02ea6c2f12448e9e1dfe01f68786092a924s  
# salted hash: VmBgeaZ2PN!93d4b242b475a73d8f2d1de1... c'est ce qu'on va stocker
```

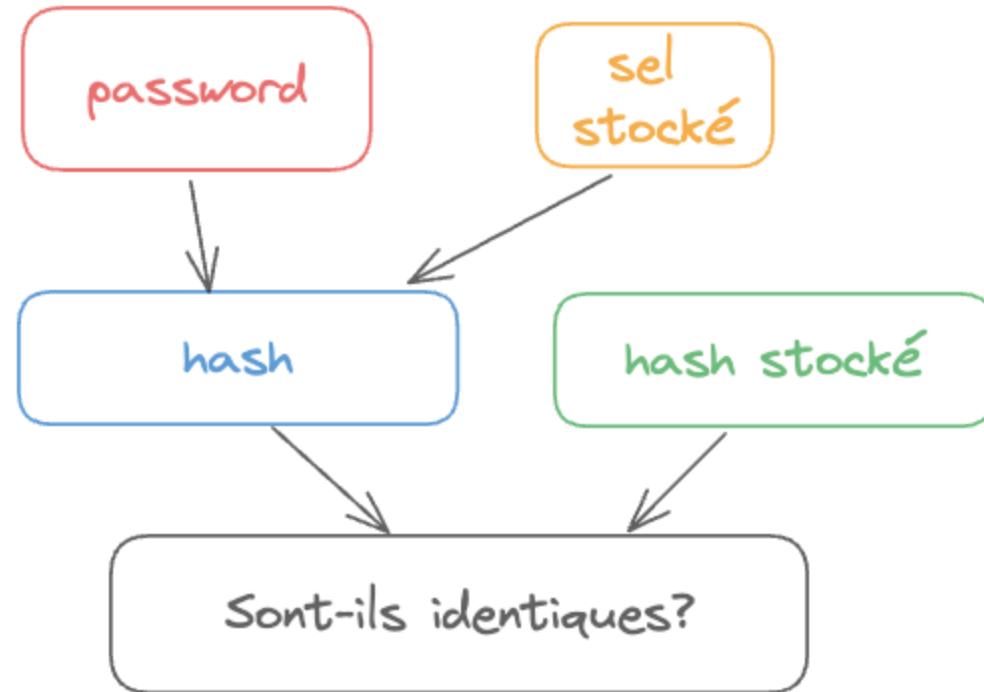
=> password de longueur 8, mais stocké hashé avec un random de taille aléatoire.

Salage - Vérification

À la création



À la vérification



Salage - pourquoi ça marche ?

User	Hash	Salted hash
alice	4420d1918..	3' r43d!3dae5f..
jason	695ddccd9..	4fTfcz!5g896da..
mario	cd5cb49b8..	Jeikdiucgh!de34a28..
teresa	73fb51a0c..	ùefs5FS!f290ff5..
bob	4420d1918..	èFGz4Ds!99g6bde..
mike	77b177de4..	DR"S2ç!22d45dd..

Impossible de dire qui a le même password qu'un autre.

Impossible d'utiliser une attaque à dictionnaire sur mdp de longueur 8.

Autres applications

- [HavelBeenPwnd](#)
- [Private Contact Discovery](#) (Signal)
- Blockchain
- Signatures digitales

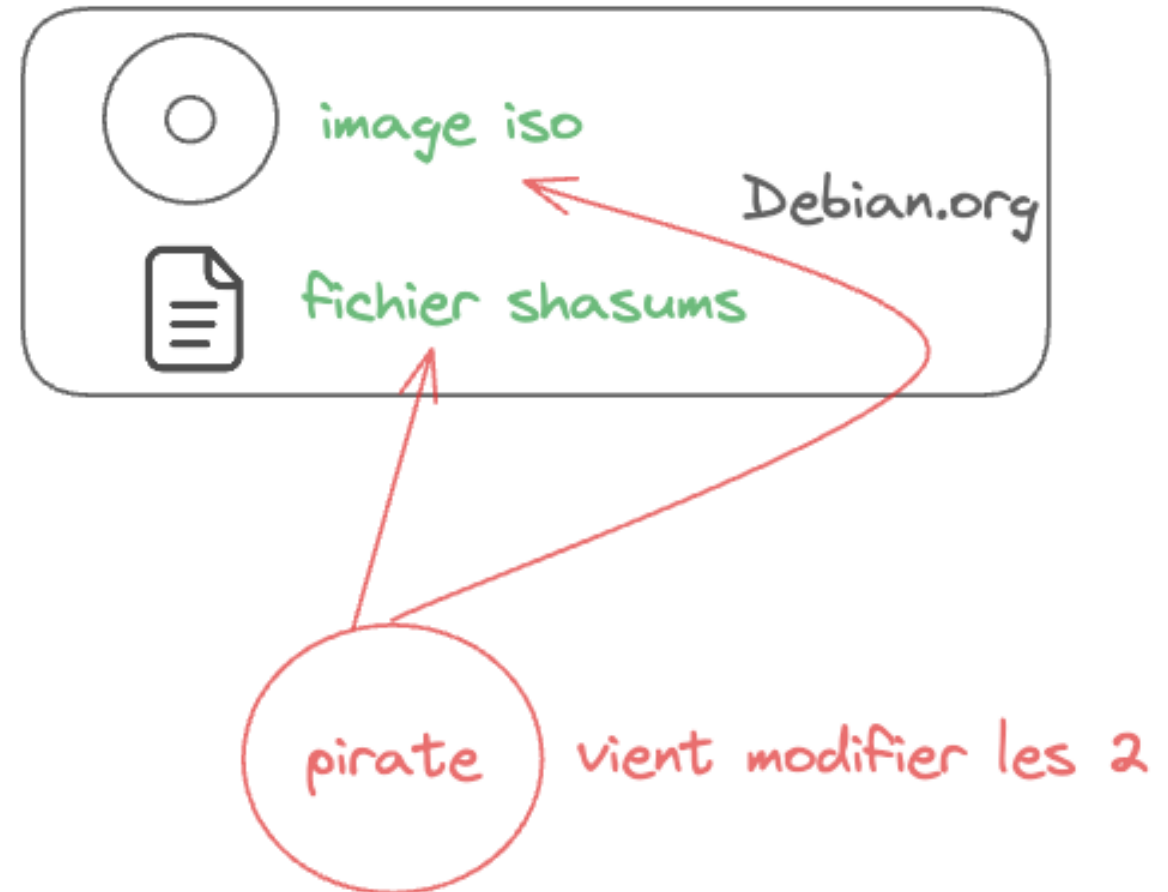
Signatures digitales

Revenons au téléchargement de notre fichier iso Debian:

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

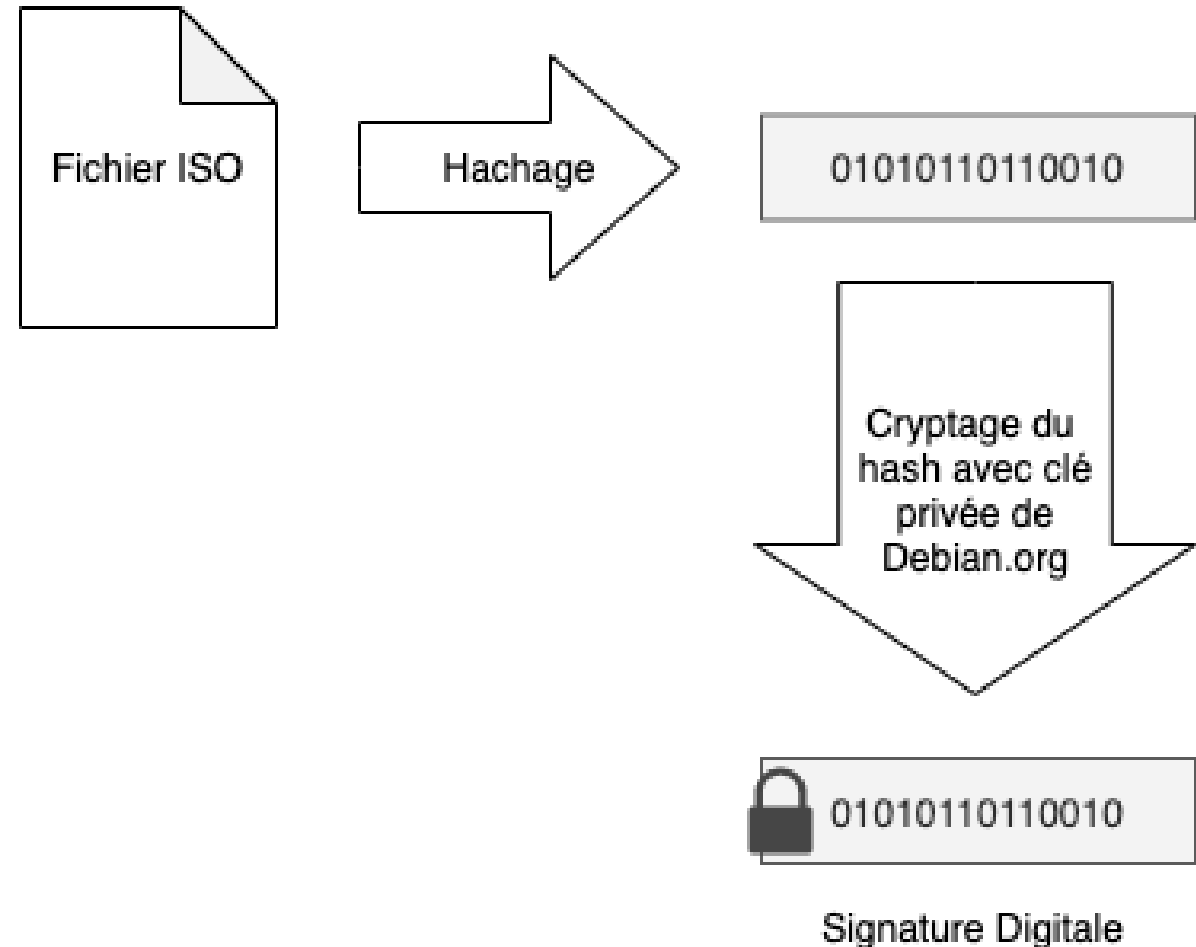
Comment vérifier l'*authenticité* du fichier? (que c'est bien Debian qui me l'a fourni)

=> La signature digitale, qui utilise la crypto asymétrique



Principe

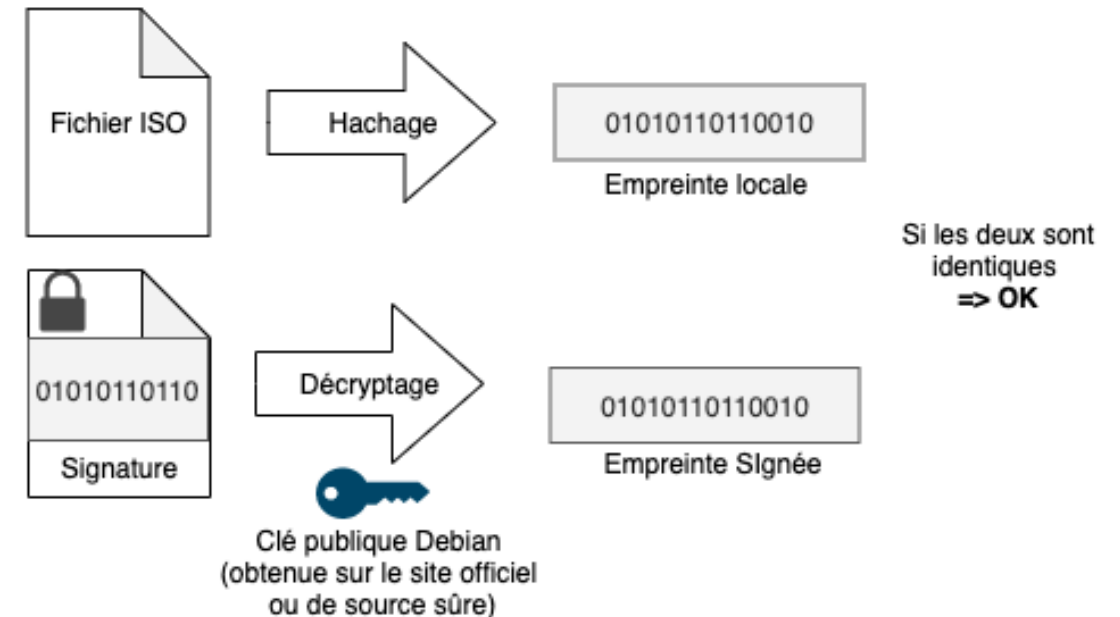
- on crypte *l'empreinte* avec une clé privée (c'est la *signature*)
- on diffuse la clé publique associée
- les gens sont capables de décrypter la signature, récupérer l'empreinte, et vérifier par eux mêmes



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



Récapitulatif

- **C** : Le chiffrement assure la **confidentialité** de la donnée, cf le cours précédent
- **I** : Un *hash* (ou empreinte) assure l'**intégrité** de la donnée
- **A** : Une signature assure l'**authenticité** de la donnée

CLA n'est pas qu'une grande agence de sécurité Américaine, mais aussi l'un des fondements de la sécurité informatique.

Certificats

Certificats

Les certificats sont à la base de:

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

Format d'échange de certificats standardisé: X.509

Ils utilisent tout ce qu'on vient de voir.

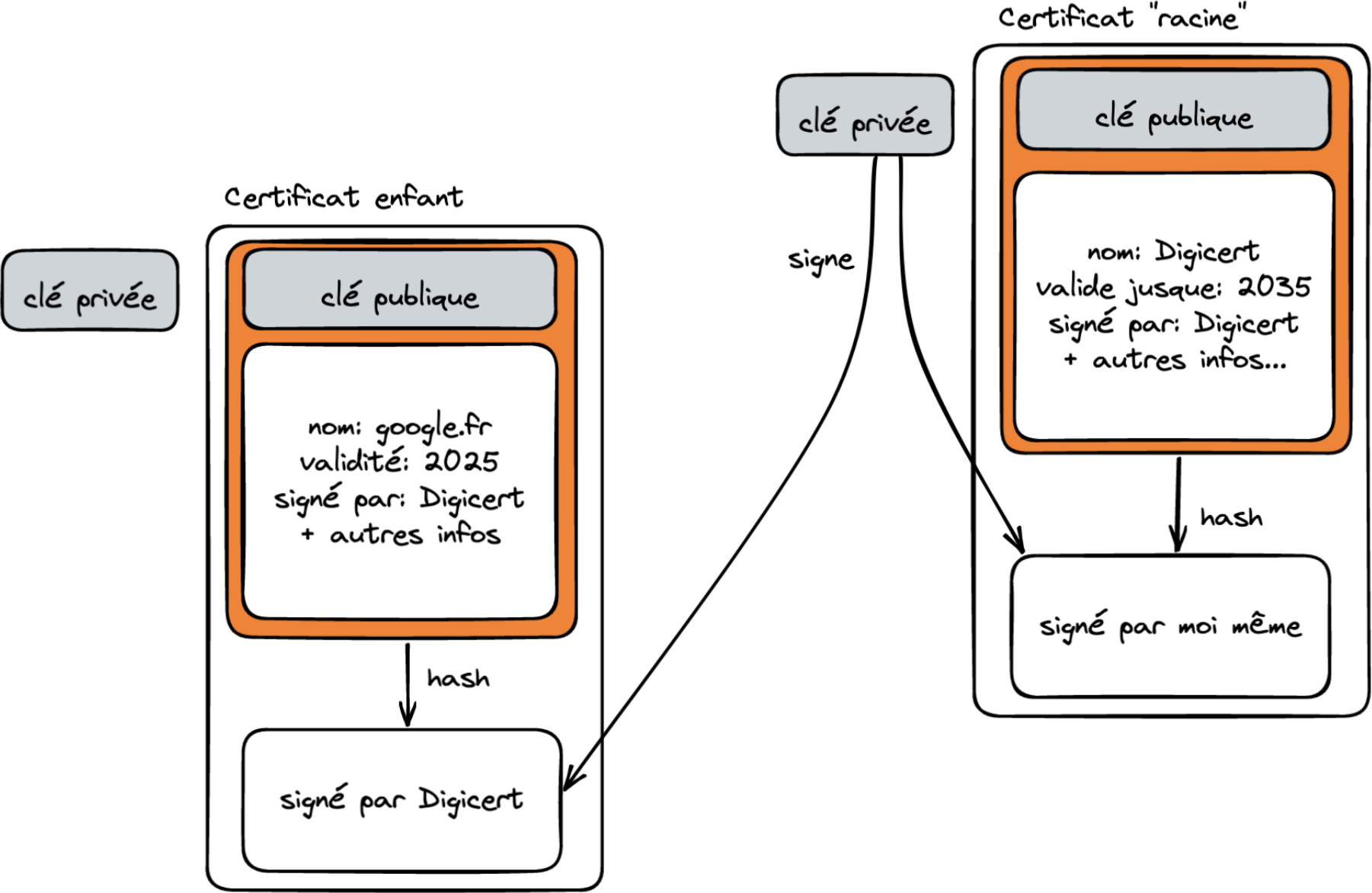
Un certificat contient:

- 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 = `chiffre(clé privée, HASH(contenu du certificat))`) par la clé d'un autre certificat, ou la sienne

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**

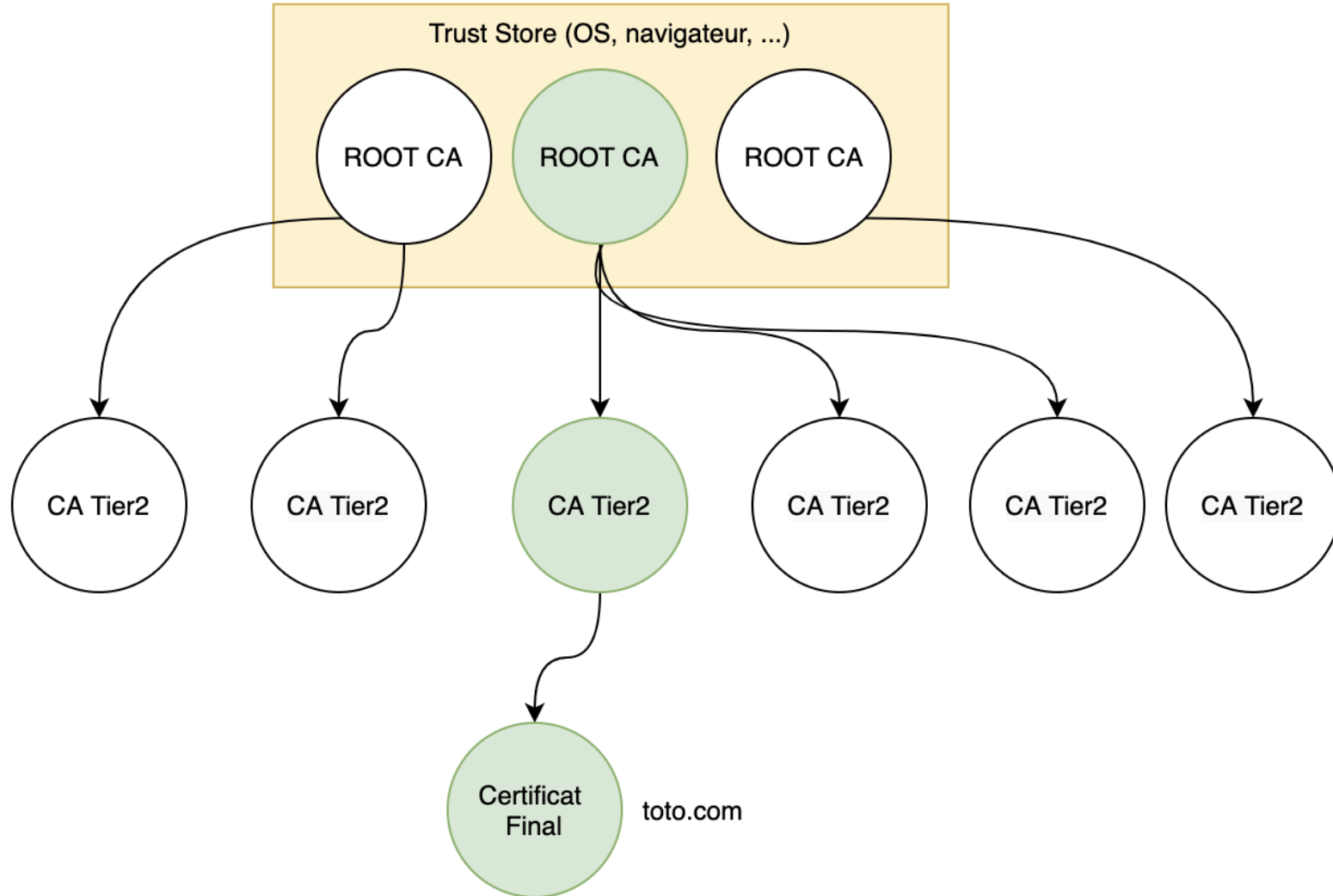


Chaîne de certification

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
```

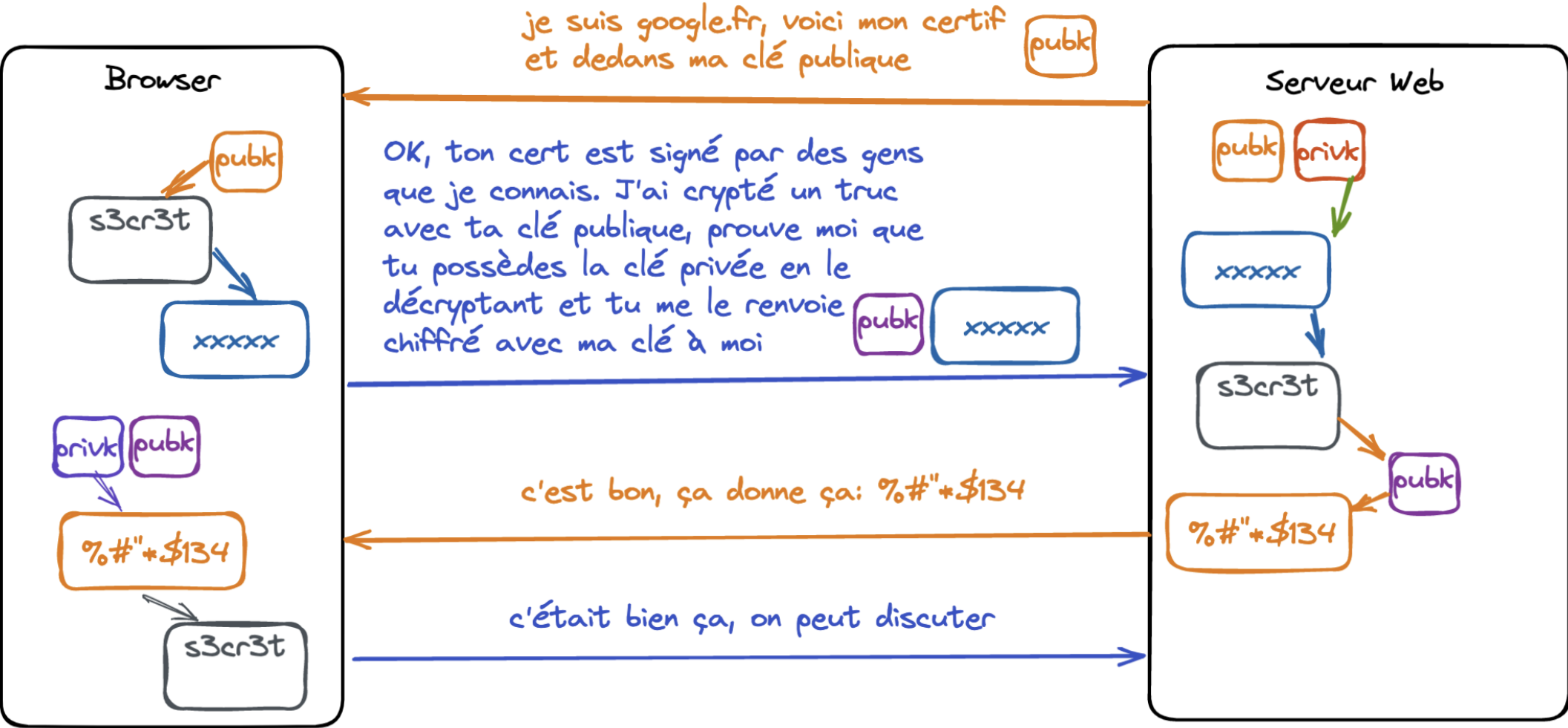
(lancez la commande dans votre terminal et admirez la liste de DNS Google)

```
1 Certificate:
2   Data:
3     Version: 3 (0x2)
4     Serial Number:
5       9c:db:54:99:1a:6c:4c:80:0a:8c:ef:d9:a2:87:35:f7
6     Signature Algorithm: sha256WithRSAEncryption
7     Issuer: C=US, O=Google Trust Services LLC, CN=GTS CA 1C3
8     Validity
9       Not Before: Dec 11 08:03:31 2023 GMT
10      Not After : Mar  4 08:03:30 2024 GMT
11     Subject: CN=*.google.com
12     Subject Public Key Info:
13       Public Key Algorithm: id-ecPublicKey
14       Public-Key: (256 bit)
15       pub:
16         04:f6:4e:21:8e:ef:59:20:88:46:d5:31:b0:1a:a1:
17         4a:75:cb:00:13:ba:1e:2c:50:a5:c6:45:91:4f:76:
18         ae:90:da:96:c0:7a:97:a3:11:67:c7:6d:e5:e0:cf:
19         0d:72:e9:a5:e3:40:c7:6a:84:64:c7:34:bf:1f:c6:
20         31:61:d1:66:b9
21       ASN1 OID: prime256v1
22       NIST CURVE: P-256
23     X509v3 extensions:
24       X509v3 Key Usage: critical
25         Digital Signature
26       X509v3 Extended Key Usage:
27         TLS Web Server Authentication
28       X509v3 Basic Constraints: critical
29         CA:FALSE
30       X509v3 Subject Key Identifier:
31         BF:2C:26:53:D2:00:F0:73:FA:B0:7C:60:90:8C:1B:6D:1D:05:A0:D9
32       X509v3 Authority Key Identifier:
33         8A:74:7F:AF:85:CD:EE:95:CD:3D:9C:D0:E2:46:14:F3:71:35:1D:27
34     Authority Information Access:
35       OCSP - URI:http://ocsp.pki.goog/gts1c3
36       CA Issuers - URI:http://pki.goog/repo/certs/gts1c3.der
37     X509v3 Subject Alternative Name:
38       DNS:*.google.com, DNS:*.appengine.google.com, DNS:*.bdn.dev, DNS:*.origin-test.bdn.dev, DNS:*.cloud.g
39       ogle.cl, DNS:*.google.co.in, DNS:*.google.co.jp, DNS:*.google.co.uk, DNS:*.google.com.ar, DNS:*.google.com.au, DNS:*.g
40       n, DNS:*.google.de, DNS:*.google.es, DNS:*.google.fr, DNS:*.google.hu, DNS:*.google.it, DNS:*.google.nl, DNS:*.google.
```

Établissement de connection SSL/TLS

TLS 1.3

1. Le client établit un canal sécurisé (chiffrement symétrique) avec le serveur (en utilisant un échange 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 chiffré 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.



Le format x509 et PEM

Un certificat peut se transporter en binaire ou en texte.

x509 détermine le format binaire des information dans le certificat.

PEM est un format d'échange (un "container") au format texte des certificats.

En-tête: -----BEGIN CERTIFICATE-----

Bas de page: -----END CERTIFICATE-----

Exemple de certificat au format PEM

```
-----BEGIN CERTIFICATE-----
MIIEbjCCA1agAwIBAgISBCGGo0R9Ugg9lAvKt5Az74atMA0GCSqGSIb3DQEBCwUA
MDIxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw11ZXQncyBFbmNyeXB0MQswCQYDVQQD
EwJSMzAeFw0yMzExMjgyMTIwMjRaFw0yNDYyMTIwMjNaMBQxEjAQBGNVBAMT
CWxlbmNyLm9yZzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABEDrwsLt9ZbNyWiu
sDlheIZ9yWYB+lbgr3Q83qTsRAAySSaWxARYzaPDJzNxfp7T9RXsyN4qBo1ZQ1kz
/ShkMLGjggJlMIICYTA0BgNVHQ8BAf8EBAMCB4AwHQYDVVR0lBBYwFAYIKwYBBQUH
AwEGCCsGAQUFBwMCMAwGA1UdEwEB/wQCMAAwHQYDVVR0lBBYEFB1+tsin5/mdlGLL
u1i7zzHGh/qkMB8GA1UdIwQYMBaAFBQusxe3WfBLrLAJQ0Yfr52LFMLGMFUGCCsG
AQUFBwEBBEkwRzAhBggrBgEFBQcwAYYVaHR0cDovL3IzLm8ubGVuY3Iub3JnMCIG
CCsGAQUFBzAChhZodHRwOi8vcjMuaS5sZW5jci5vcmevMG8GA1UdEQRoMGaCCWxl
bmNyLm9yZ4IPbGV0c2VuY3J5cHQyY29tg9sZXRzZW5jcnldC5vcmeCDXd3dy5s
ZW5jci5vcmeCE3d3dy5sZXRzZW5jcnldC5jb22CE3d3dy5sZXRzZW5jcnldC5v
cmcwEwYDVVR0gBAwwCjAIBgZngQwBAgEwggEDBgorBgEEAdZ5AgQCBIH0BIHxA08A
dQA7U3d1Pi25gE6LMFsG/ka7Z9hPw/THvQANLXJv4frUFwAAAYwYB0mCAAEEAwBG
MEQCIgtNfADnkvEDmbw95yp7Eeb4muQspvosjgBEezSHOL3mAiA0PK1R0Pe6nlQp
k+k0Ux5RBWD24KRj2446dsG3U5GXBAB2AEiw42vapkc0D+VqAvqdM0scUgHLVt0s
gdm7v6s52IRzAAABjBgE6YcAAQDAECwRQIhAJ8AqaI2wH1+gEgT8PZmcybIbGk+
uAyRDQQEFC5i5HfEAiBRmZPJzUw5FY9Pjjz9LefoT3Qepa9N+yFDlLA960s+VjAN
BgkqhkiG9w0BAQsFAAOCQAQEABI4EmE8s1IeqtmjumuSqDUPWkjsypeFDWV8JShUF
rXD45gCHpcIgPHtCRR4PUkSLFdEpuMuS6xrwfDPwLLI02Ymcm7Cz0W3iEihpwPb5
w8Knjg3q/eCSRqwnGYuljh3jckB05giLU/QvvD2qoqET/esU+y+0wLjFJcakLMgv
fH/c5n7nbip/h9Ss0J6V/7QmD/taGhkgB+4AU+XLM0raqzTSYxy6TV1H5tb1ontg
JGQmiSP13DPJbA2FJvCR94PyoD2LI fzkFb9nHQw/PeZuMeHgveWd2f00l+8/6lN3
0+sR71jwSHjb1rvDUQeYtozIJavdMAKt4A8p04pHCpQ9vw==
-----END CERTIFICATE-----
```

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.

La liste est publique, et récupérable.

OCSP

Online Certificate Status

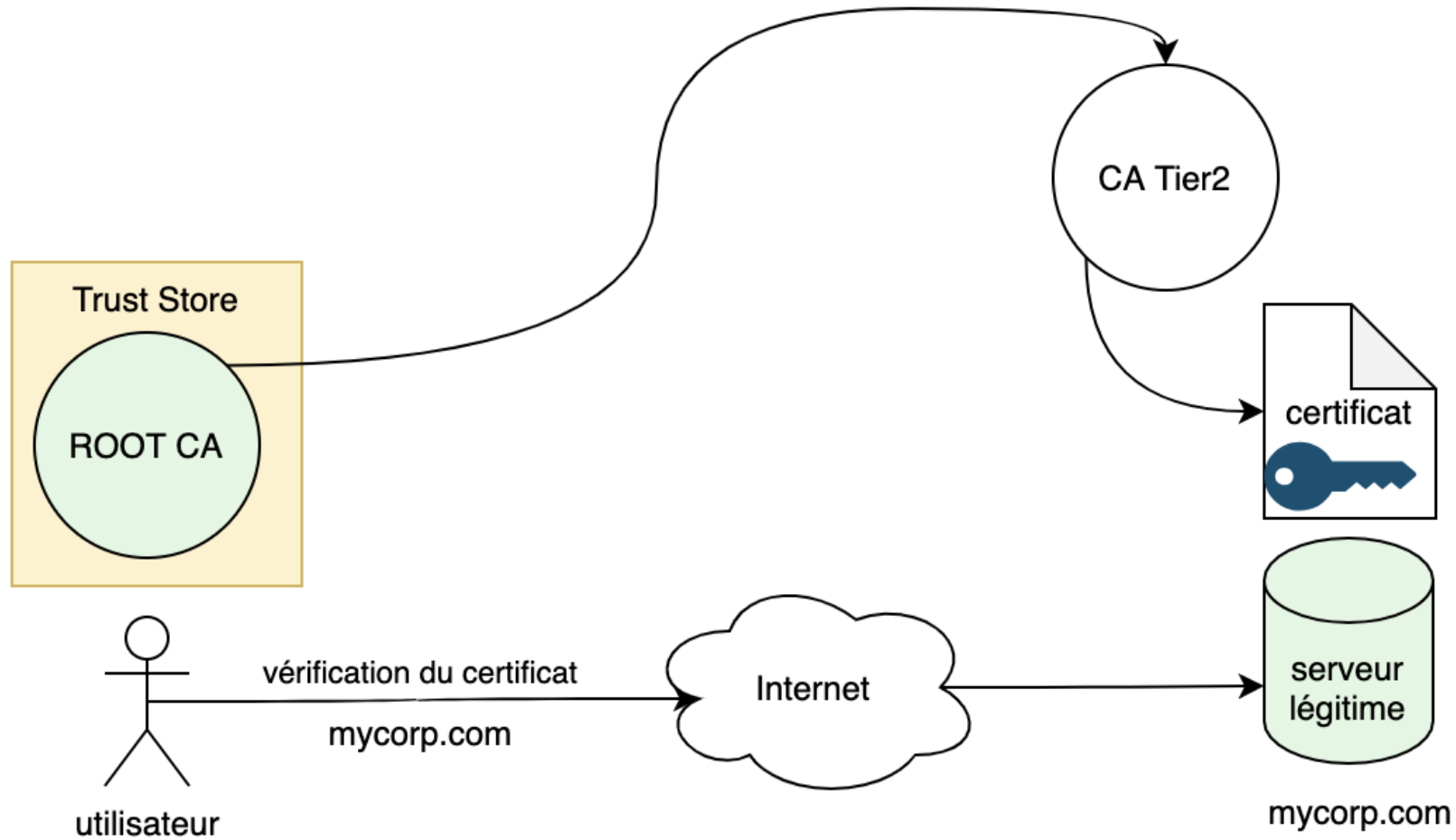
Protocol, basée sur une API:

Je demande à l'URL indiquée dans le certificat si celui-ci est toujours OK.

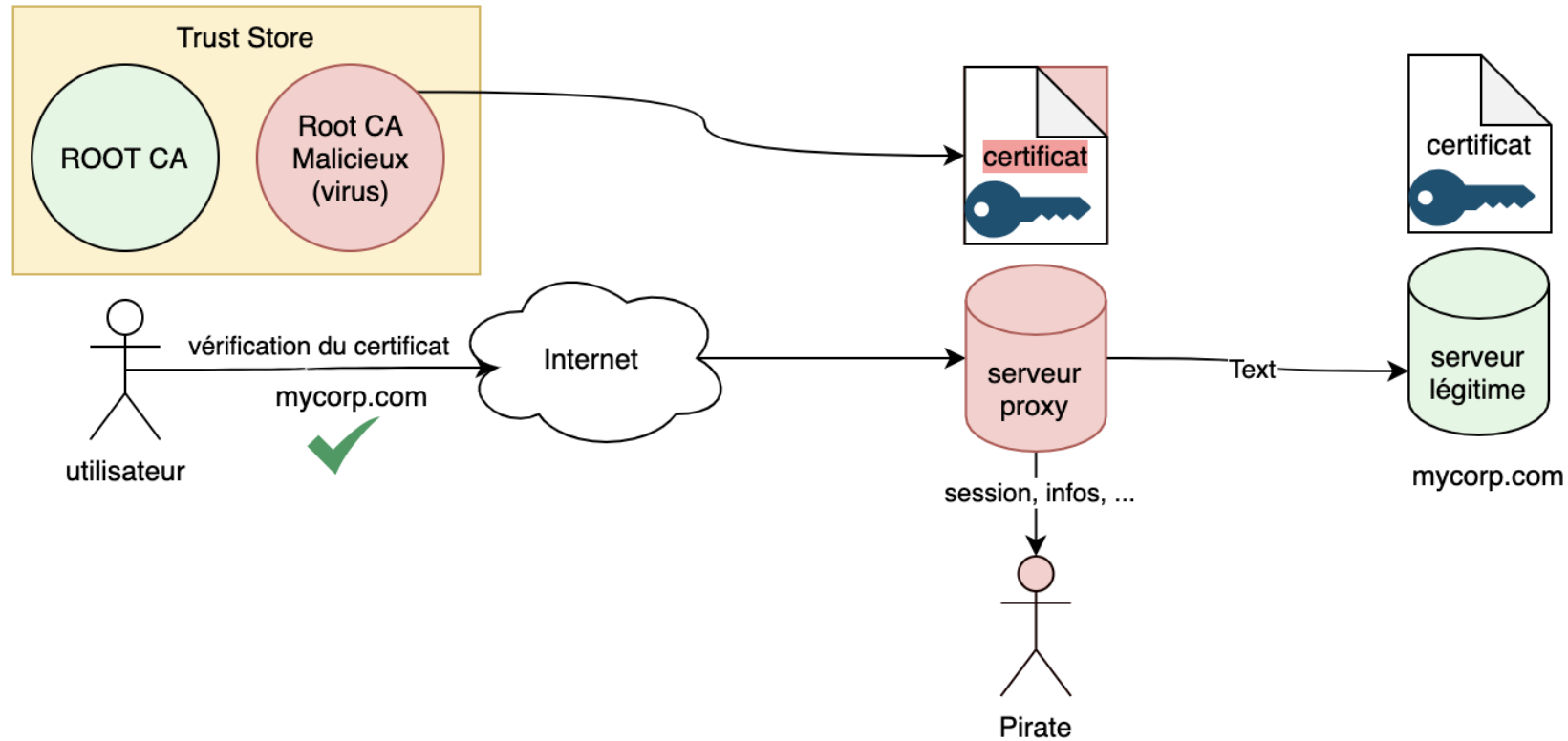
Réponse immédiate

```
+ tmp openssl crl -in cert.crl -text
Certificate Revocation List (CRL):
  Version 2 (0x1)
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: O=Sample Signer Organization, OU=Sample Signer Unit, CN=Sample Signer
  Last Update: Feb 18 10:32:00 2013 GMT
  Next Update: Feb 18 10:42:00 2013 GMT
  CRL extensions:
    X509v3 Authority Key Identifier:
      BE:12:01:CC:AA:EA:11:80:DA:2E:AD:B2:EA:C7:B5:FB:9F:F9:AD:34
    X509v3 CRL Number:
      3
  Revoked Certificates:
    Serial Number: 147947
      Revocation Date: Feb 18 10:22:12 2013 GMT
      CRL entry extensions:
        X509v3 CRL Reason Code:
          Affiliation Changed
      Invalidity Date:
        Feb 18 10:22:00 2013 GMT
    Serial Number: 147948
      Revocation Date: Feb 18 10:22:22 2013 GMT
      CRL entry extensions:
        X509v3 CRL Reason Code:
          Certificate Hold
      Invalidity Date:
        Feb 18 10:22:00 2013 GMT
    Serial Number: 147949
      Revocation Date: Feb 18 10:22:32 2013 GMT
      CRL entry extensions:
        X509v3 CRL Reason Code:
          Superseded
      Invalidity Date:
        Feb 18 10:22:00 2013 GMT
    Serial Number: 14794A
      Revocation Date: Feb 18 10:22:42 2013 GMT
      CRL entry extensions:
        X509v3 CRL Reason Code:
          Key Compromise
      Invalidity Date:
        Feb 18 10:22:00 2013 GMT
    Serial Number: 14794B
      Revocation Date: Feb 18 10:22:51 2013 GMT
      CRL entry extensions:
        X509v3 CRL Reason Code:
          Cessation Of Operation
      Invalidity Date:
        Feb 18 10:22:00 2013 GMT
  Signature Algorithm: sha1WithRSAEncryption
  Signature Value:
    42:21:be:81:f1:c3:79:76:66:5b:ce:21:13:8a:68:a8:b4:3c:
    be:16:c3:af:4b:dd:cb:78:35:92:90:d8:d7:4c:6f:fe:6c:68:
    27:ae:6d:da:42:98:01:ee:17:93:f0:bd:a8:ee:cd:90:b6:35:
```

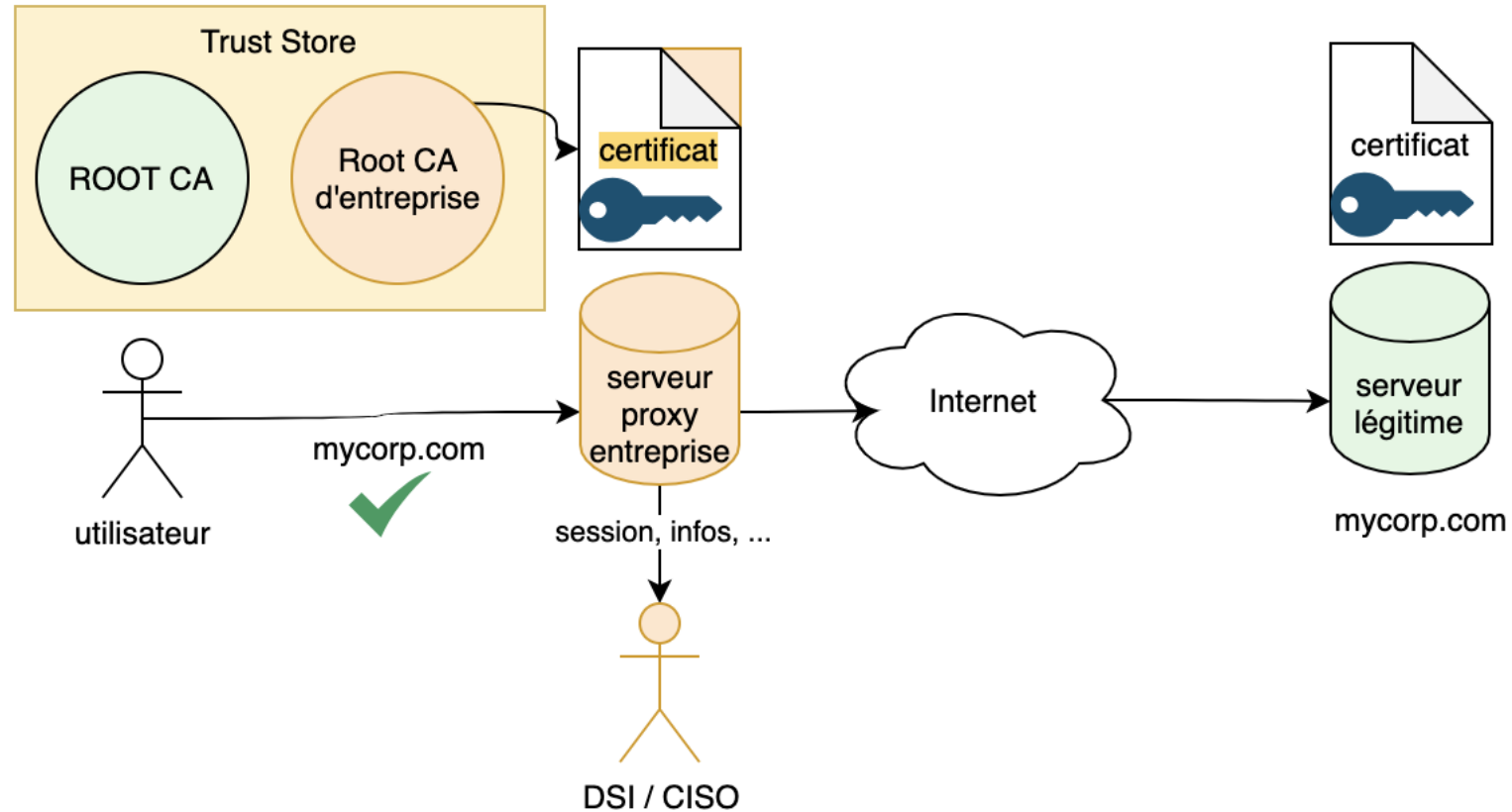
MITM (Man In The Middle)



MITM (Man In The Middle)



MITM (Man In The Middle)



Authentification par certificat client

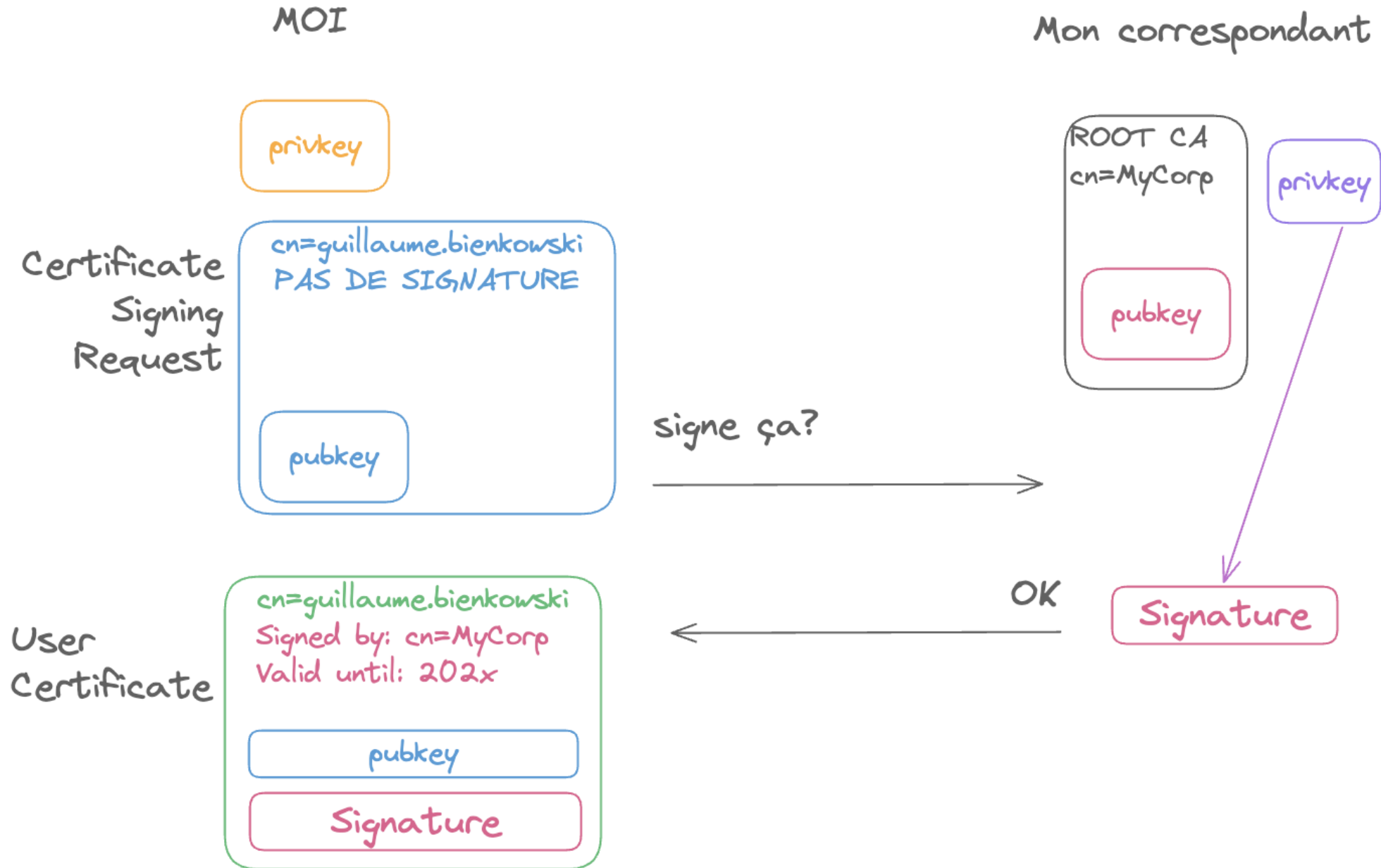
Je demande à mon interlocuteur de me fournir un certificat. Pour cela:

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.



C'est la fin du cours, on fait une PAUSE. Puis on se lance dans le TP.

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 chiffrement 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.