TIW4 : SÉCURITÉ DES SYSTÈMES D'INFORMATION

INTRODUCTION À L'AUTHENTIFICATION

romuald.thion@univ-lyon1.fr

http://liris.cnrs.fr/~rthion/dokuwiki/enseignement:tiw4

Master « Technologies de l'Information »

Introduction

Définition

L'authentification désigne le processus visant à *vérifier* qu'un entité (personne, service, machine) est bien légitime pour accéder au système.

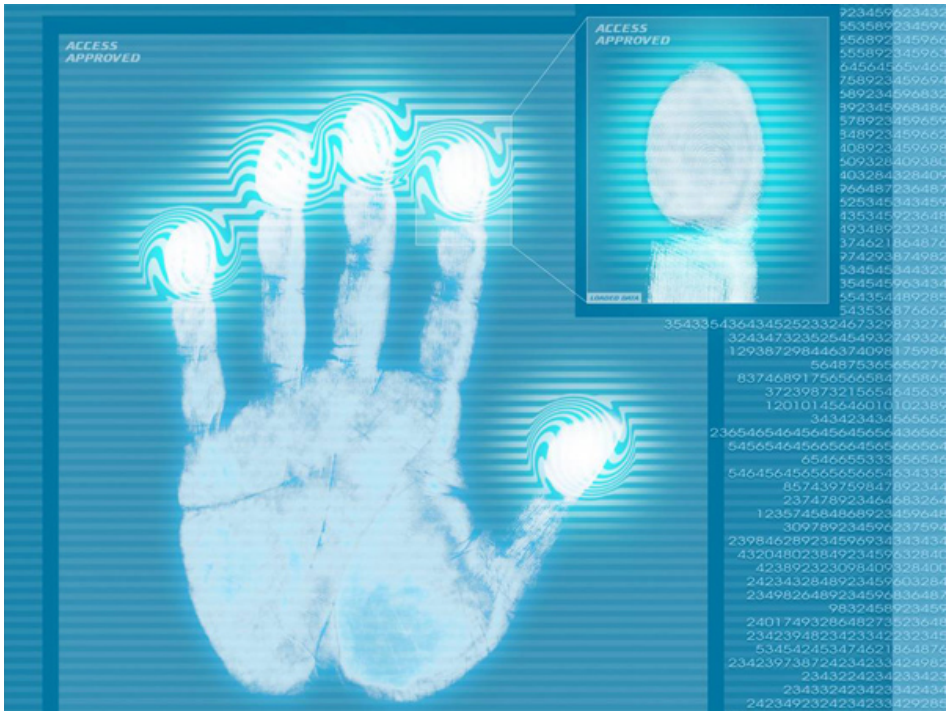
Différences entre

Identification dire qui je suis (e.g., login, empreinte)

Authentification vérifier/prouver qui je suis (e.g., password)

Autorisation vérifier si j'ai le droit (contrôle d'accès)

Authentification et autorisation



Authentification

Autorisation

Introduction

Les facteurs de validation

- ce que je sais (*knowledge factors*) : PIN, mot de pass, *passphrase*, question de sécurité
- ce que je possède (*ownership factors*) : carrte, clef usb, téléphone, jeton (logiciel), dongle
- ce que je suis (*inherence factors*) : empreintes digitales, rétine, voix, visage, ADN. On peut classer en *morphologique*, *comportementale* ou *biologique*.

Protocoles d'authentification

Un protocole cryptographique utilise les briques :

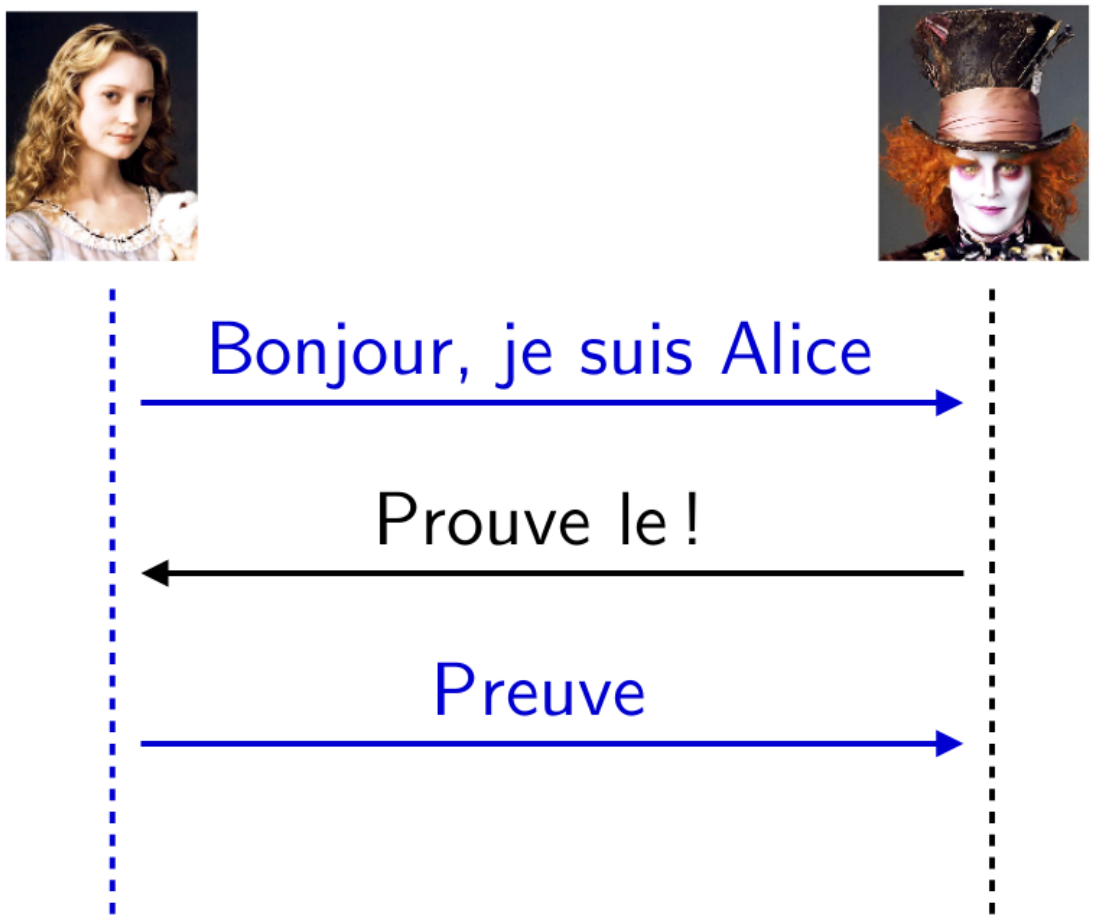
- chiffrement symétrique (dit à clef secrète)
- chiffrement asymétrique (dit à clefs publiques)
- fonction de hachage (dites à sens unique)
- génération de nombres aléatoires

pour garantir des propriétés de sécurité.

On va s'intéresser aux protocoles cryptographiques pour l'authentification.

1. Authentification des clients par mot de passe
   - Principe de base
   - Attaque des mots de passe
   - Exemples
   - Recommandations
2. Protocoles d'authentification
   - L'authentification à clef publique
   - La preuve d'authentification et le SSO
3. Conclusion

Authentification des clients par mot de passe



Crédits LAURADOUX C.

Authentification des clients par mot de passe

Un protocole très simple

1. $A \to B : A, pw$
2. $B \to A : 1$ si $\langle A, h(pw) \rangle \in \text{passwordtable}$

La table des mots de passe passwordtable

Le vérificateur stocke les paires $\langle A_i, h(pw_i) \rangle$

```
maintainer:OYnL8rCBbf7rc
edgar:C8Z6wDFKm5bV6
patrick:58kk0mpCjpL9o
edwin:.8cBf8RFZqfvI
frank:gvEFH2KSvYZS2
jaap:NMS4yrkEfQz9c
```

Attaque des mots de passe

Principe

- on connaît h le hashé et l'algorithme [a] $hash$
- on tâche de trouver pw tel que $h = hash(pw)$

a. Rappel : principe de Kerckhoff

Typologie de l'attaque

- online : on peut limiter le nombre, mettre un captcha, bannir
- offline : il faut que l'attaque la plus difficile (longue) possible

Comment parcourir intelligemment l'espace des mots de passe ?
Est-ce rentable de pré-calculer les hashés ?

Attaque des mots de passe

Principe

- on connaît h le hashé et l'algorithme [a] $hash$
- on tâche de trouver pw tel que $h = hash(pw)$

a. Rappel : principe de Kerckhoff

Typologie de l'attaque

- online : on peut limiter le nombre, mettre un captcha, bannir
- offline : il faut que l'attaque la plus difficile (longue) possible

Comment parcourir intelligemment l'espace des mots de passe ?
Est-ce rentable de pré-calculer les hashés ?

Attaque des mots de passe

Méthodes de choix des mots à essayer

- force brute (*bruteforce*)
- dictionnaires (langue naturelle, jargon, *ad hoc*)
- heuristique (choix de mots probables vis-à-vis de règles)
- stockage de hashés pré-calculés
- stockage d'une partie des hashés pré-calculés (*rainbow tables*)

Quels sont les avantages et inconvénients de ces méthodes ?

Outils : www.openwall.com/john et hashcat.net

Attaque des mots de passe

Pour un alphabet Σ, combien il y a-t-il de mots de passe de longueur au plus n (inclus) ?

```
$ john --test
Will run 4 OpenMP threads
Benchmarking: descrypt, traditional crypt(3)
                [DES 256/256 AVX2]... (4xOMP) DONE
Many salts: 16990K c/s real, 4247K c/s virtual
Only one salt: 11190K c/s real, 2884K c/s virtual
```

Combien de temps pour *tout* explorer ?

Authentification des clients par mot de passe

Comment rendre plus difficile l'attaque *offline* ?

Définition du *sel*

Le sel est une données aléatoire *non confidentielle* ajoutée au mot de passe.

La table des mots de passe salés

Le vérificateur stocke les paires $\langle A_i, s_i, h(pw_i, s_i) \rangle$

```
maintainer:ef$OYnL8rCBbf7rc
edgar:01$C8Z6wDFKm5bV6
patrick:45$58kk0mpCjpL9o
edwin:a5$.8cBf8RFZqfvI
...
```

Authentification des clients par mot de passe

Comment rendre plus difficile l'attaque *offline* ?

Définition du *sel*

Le sel est une données aléatoire *non confidentielle* ajoutée au mot de passe.

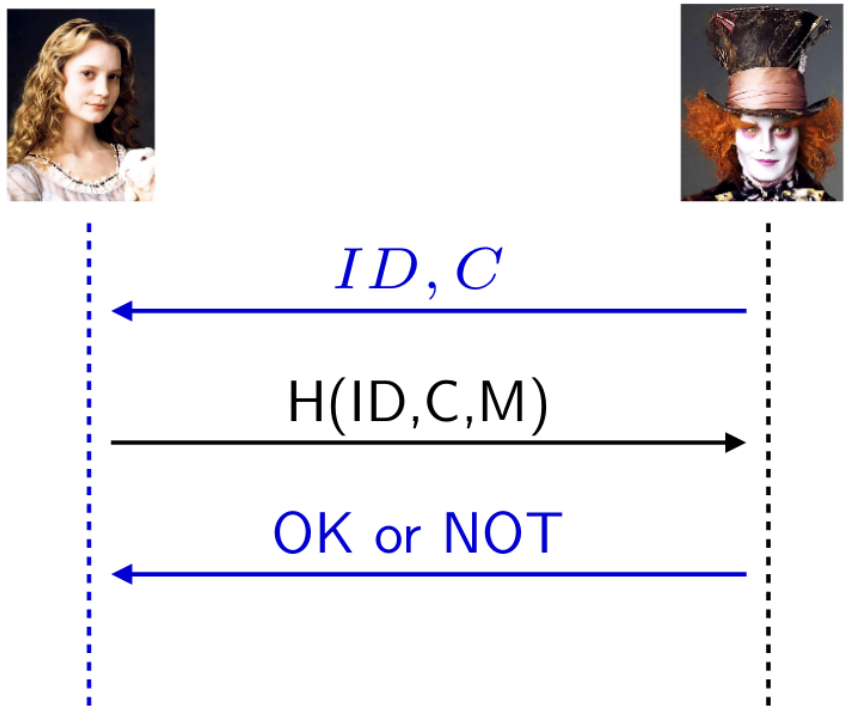
La table des mots de passe salés

Le vérificateur stocke les paires $\langle A_i, s_i, h(pw_i, s_i)\rangle$

```
maintainer:ef$OYnL8rCBbf7rc
edgar:01$C8Z6wDFKm5bV6
patrick:45$58kk0mpCjpL9o
edwin:a5$.8cBf8RFZqfvI
...
```

Challenge-response authentication

Principe général de Challenge-response authentication (dont CHAP, CRAM, SCRAM)



Crédits LAURADOUX C.

Challenge-response authentication : SCRAM

Méthode SCRAM (RFC5802) `scram-sha-256` sur PostgreSQL

```
SaltedPassword  := KeyDerive(password, salt, ic)
ClientKey       := HMAC(SaltedPassword, "Client Key")
ServerKey       := HMAC(SaltedPassword, "Server Key")
StoredKey       := H(ClientKey)
ServerRecord    := StoredKey,ServerKey,salt,ic
```

$C \to S$ `C, ClientNonce`

$S \to C$ `CombinedNonce, salt, ic`
`CombinedNonce := ClientNonce || ServerNonce`

$C \to S$ `ClientProof, CombinedNonce`
`ClientProof := ClientKey` $\oplus$ `ClientSign`
`ClientSign := HMAC(StoredKey, AuthMessage)`
Test `H(ClientSign` $\oplus$ `ClientProof) = StoredKey`

$S \to C$ `ServerSign := HMAC(ServerKey, AuthMessage)`

Challenge-response authentication : SCRAM

Improved Password-Based Authentication in MongoDB 3.0 : SCRAM (part. 1) (part. 2)

These attacks provide justification for SCRAM's design, as it is specifically intended to counter them.

- *Eavesdropping – The attacker can read all traffic exchanged between the client and server. To protect against eavesdropping, a SCRAM client never sends her password as plaintext over the network.*
- *Replay – The attacker can resend the client's valid responses to the server. Each SCRAM authentication session is made unique by random nonces, so that the protocol messages are only valid for a single session.*
- *Database Compromise – The attacker can view the contents of the server's persistent memory. A database compromise is mitigated by salting and iteratively hashing passwords before storing them.*
- *Malicious Server – The attacker can pose as a server to the client. An attacker is is unable to pose as server without knowledge of the client's SCRAM credentials.*

Exemple : authentification Linux

Le fichier de stockage

Hashés stockés dans `/etc/shadow`, différentes méthodes

- `crypt` (DES) : $h = DES(pass, 0...0 + sel)$, on stocke $sel + h$
- `$1$` (MD5) : $h = MD5(pass + sel)$, on stocke $sel + h$
- `$2$` à `$6$` : on utilise 6

Example

```
> mkpasswd -m sha-512  --salt abcdefgh
$6$u2bvcyi0$76I3KxGizdl/PENw[...]4FUhEBcKcg.

> openssl passwd -6 -salt abcdefgh
$6$u2bvcyi0$76I3KxGizdl/PENw[...]4FUhEBcKcg.
```

Etude de cas ';-have i been pwned?

Liste https://haveibeenpwned.com/Passwords
SHA1 11.1GB (24.5GB décompressés)

```
> head -n 5 pwned-sha1-ordered-by-count-v5.txt
7C4A8D09CA3762AF61E59520943DC26494F8941B:23547453
F7C3BC1D808E04732ADF679965CCC34CA7AE3441:7799814
B1B3773A05C0ED0176787A4F1574FF0075F7521E:3912816
5BAA61E4C9B93F3F0682250B6CF8331B7EE68FD8:3730471
3D4F2BF07DC1BE38B20CD6E46949A1071F9D0E3D:3120735

> wc -l  pwned-sha1-ordered-by-count-v5.txt
555278657  pwned-sha1-ordered-by-count-v5.txt
```

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
CREATE TABLE pwned (
  hash_id bigint  PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
  hash_content text UNIQUE NOT NULL,
  time_used bigint NOT NULL
);

\COPY pwned(hash_content,time_used)
  FROM 'pwned-passwords-sha1-ordered-by-count-v5-1000k.txt'
  WITH DELIMITER ':' CSV;
-- COPY 1000000
-- Time: 12903,211 ms (00:12,903)

CREATE OR REPLACE FUNCTION find_hash(text)
  RETURNS TABLE(id bigint, nb bigint) AS $$
    SELECT hash_id, time_used
    FROM pwned
    WHERE hash_content = upper(encode(digest($1, 'sha1'), 'hex'))
$$ LANGUAGE SQL STRICT STABLE;

SELECT find_hash('patate');
--(7334,15901)
--Time: 1,006 ms
```

Quelles recommandations ?

OWASP Authentication cheat sheet
OWASP : password storage cheat sheet

- Implement Proper Password Strength Controls
  Voir Calculer la « force » d'un mot de passe ou zxcvbn
- Implement Secure Password Recovery Mechanism
- Store Passwords in a Secure Fashion
  - Use a cryptographically strong credential-specific salt
  - Leverage an adaptive one-way function (e.g., Argon2, Bcrypt)
- Transmit Passwords Only Over TLS or Other Strong Transport

Chiffrement symétrique et asymétrique

- *Symétrique* ou *à clef secrète*
  - secret partagé : nécessite un canal sûr
  - algorithmes efficaces (e.g., AES-NI)
- *Asymétrique* ou *à clef publique*
  - pas de secret partagé : échange de clefs sans canal sûr
  - algorithmes peu efficaces

Conclusion : combiner les deux

- asymétrique : authentification, détermination d'une clef symétrique
- symétrique : chiffrement de la communication une fois établie

Et on renouvellera la clef symétrique (de session ou éphémère) à chaque nouvel échange (Perfect Forward Secrecy)

Chiffrement symétrique et asymétrique

- *Symétrique* ou *à clef secrète*
  - secret partagé : nécessite un canal sûr
  - algorithmes efficaces (e.g., AES-NI)
- *Asymétrique* ou *à clef publique*
  - pas de secret partagé : échange de clefs sans canal sûr
  - algorithmes peu efficaces

Conclusion : combiner les deux

- asymétrique : authentification, détermination d'une clef symétrique
- symétrique : chiffrement de la communication une fois établie

Et on renouvellera la clef symétrique (de session ou éphémère) à chaque nouvel échange (Perfect Forward Secrecy)

L'authentification à clef publique

Problème de l'authentification

On peut transmettre les clefs publiques sur un canal non sûr, mais comment assurer effectivement qu'il s'agit du bon interlocuteur ?

Solution : faire confiance à un tiers (de confiance)

Certificats (X509) : l'autorité de certification (le tiers) produit un certificat où il atteste (signe) :

- l'identité du certifié et du certificateur
- la clef publique du certifié
- la date d'émission et la date limite de validité
- numéro de série, algorithmes utilisés etc.

L'authentification à clef publique

Problème de l'authentification

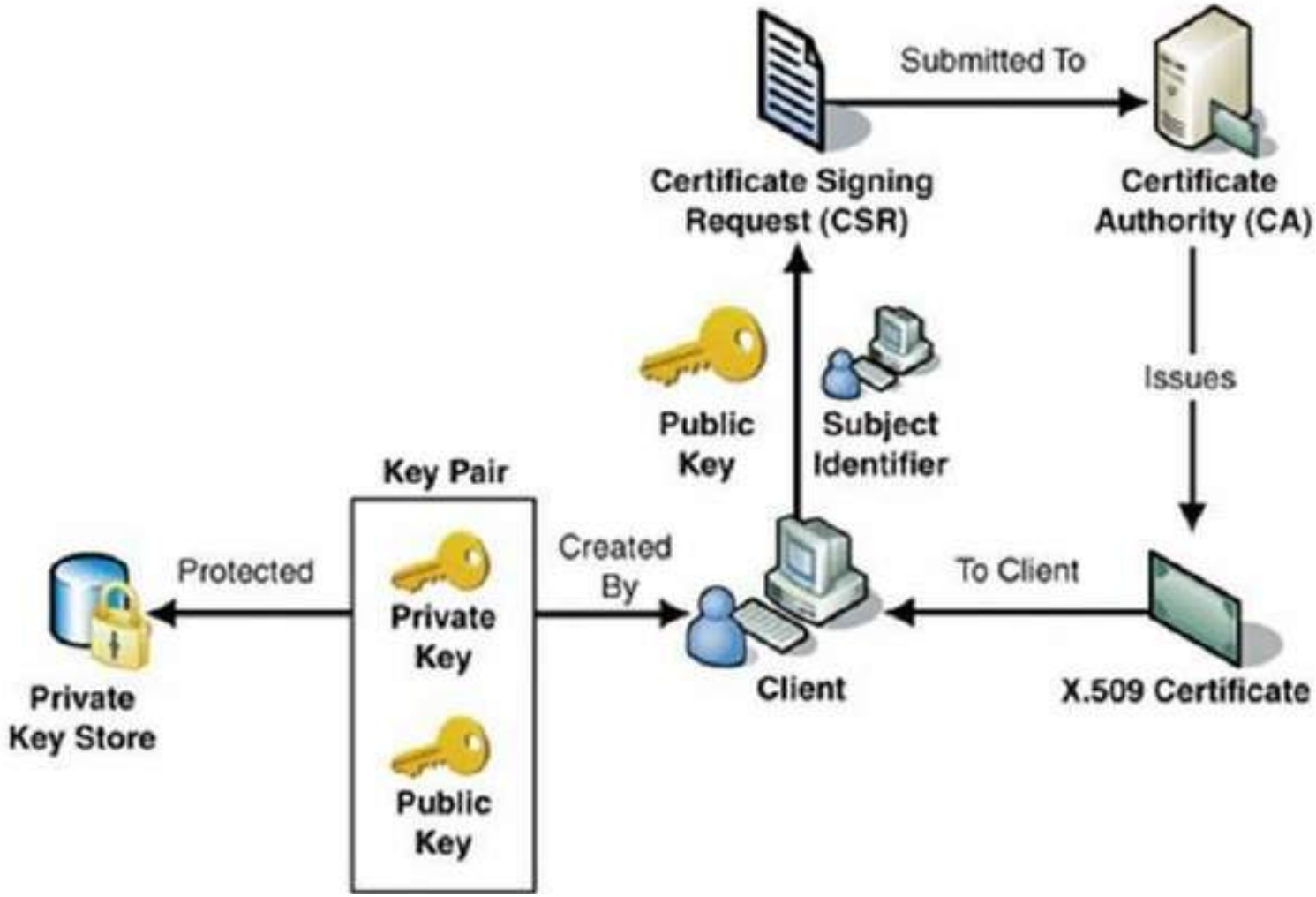
On peut transmettre les clefs publiques sur un canal non sûr, mais comment assurer effectivement qu'il s'agit du bon interlocuteur ?

Solution : faire confiance à un tiers (de confiance)

Certificats (X509) : l'autorité de certification (le tiers) produit un certificat où il atteste (signe) :

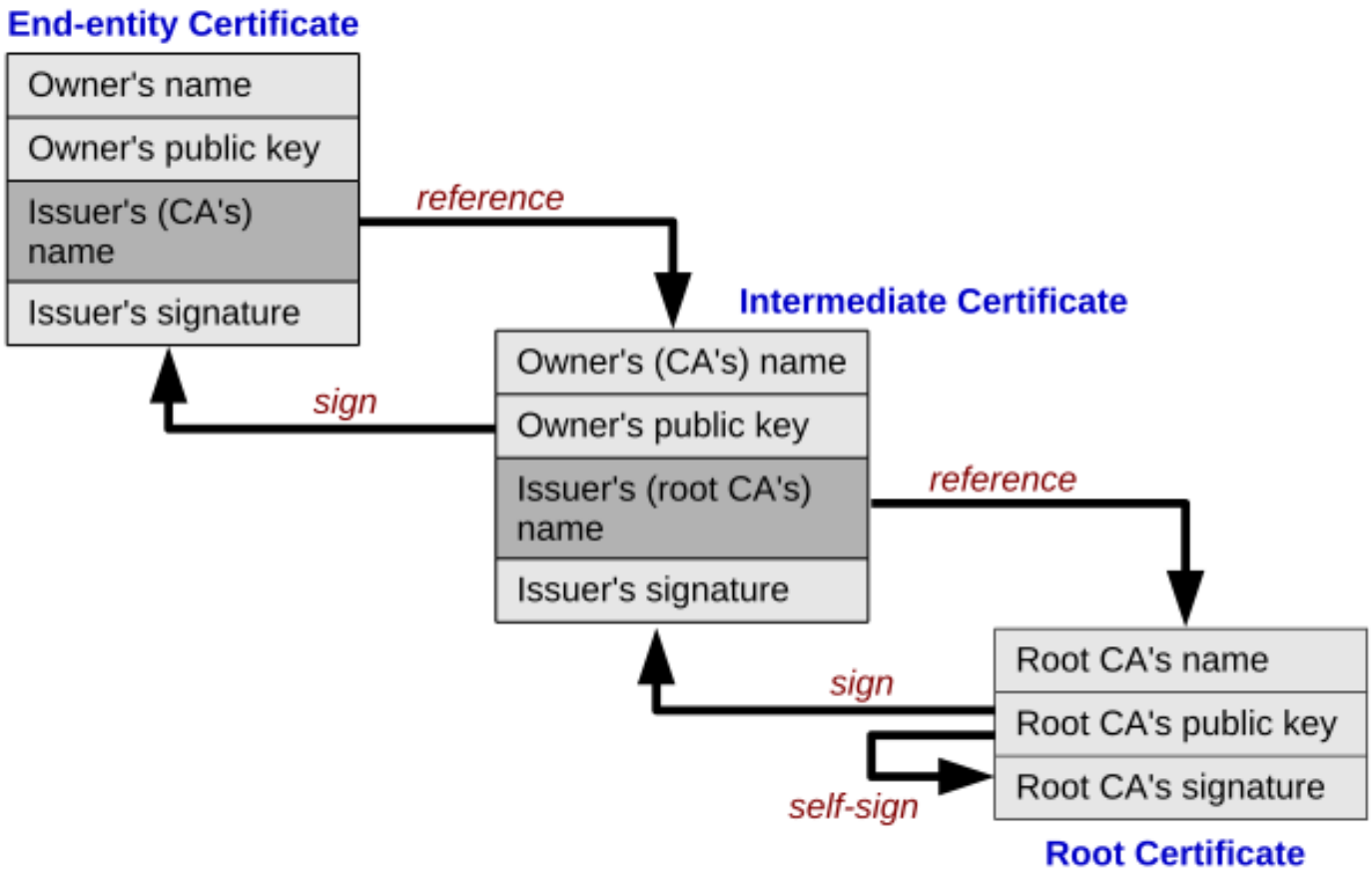
- l'identité du certifié et du certificateur
- la clef publique du certifié
- la date d'émission et la date limite de validité
- numéro de série, algorithmes utilisés etc.

L'authentification à clef publique : le certificat



Crédits techblognow – X.509 Certificates – Explained

L'authentification à clef publique : la chaîne de certificat



Wikipedia – chain of trust

L'authentification à clef publique : un certificat auto-signé (racine)

```
Data:
    Version: 3 (0x2)
    Serial Number:
        11:75:4f:19:da:09:eb:08:00:40:77:e9:6d:20:60:4f:39:73:60:1c
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: C = FR, ST = Some-State, O = UCBL, CN = TIW4
    Validity
        Not Before: Sep 12 12:10:30 2020 GMT
        Not After : Sep 12 12:10:30 2021 GMT
    Subject: C = FR, ST = Some-State, O = UCBL, CN = TIW4
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
            Public-Key: (384 bit)
            pub:
                04:98:5d:ad:00:f7:8d:cb:fd:29:07:5f:ae:a4:17:
                ...
            ASN1 OID: secp384r1
            NIST CURVE: P-384
     ...
```

Protocoles d'authentification

Et à la racine, à quels autorités fait-on confiance ?

Mozilla Included CA Certificate List

https://wiki.mozilla.org/CA/Included_Certificates

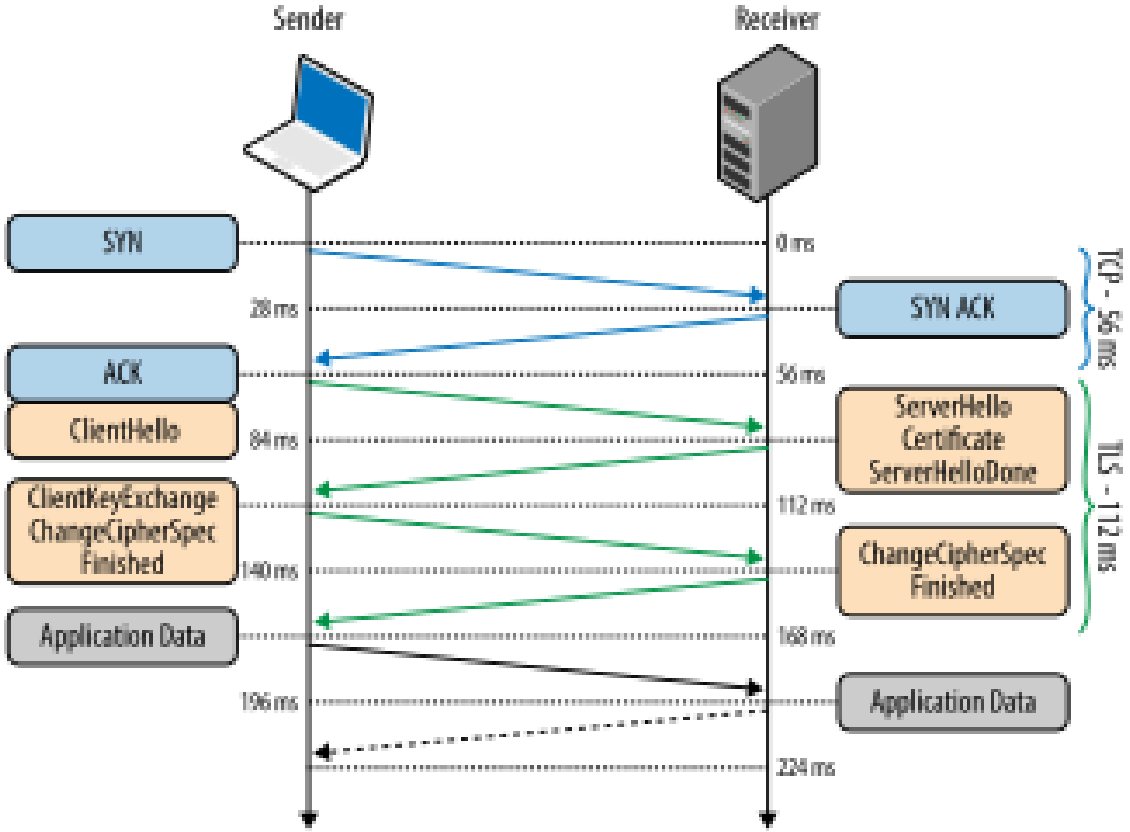
Protocoles d'authentification

Et à la racine, à quels autorités fait-on confiance ?

Mozilla Included CA Certificate List

https://wiki.mozilla.org/CA/Included_Certificates

L'authentification à clef publique : TLS



https://hpbn.co/transport-layer-security-tls/(crédits)

The First Few Milliseconds of an HTTPS Connection (TLS 1.0)

La preuve d'authentification et le SSO

Les mécanismes d'authentification sont lents et ennuient l'utilisateur : il faut en minimiser le nombre pour un ou plusieurs services (*Single-Sign On* – SSO) en utilisant une autorité d'authentification (AA).

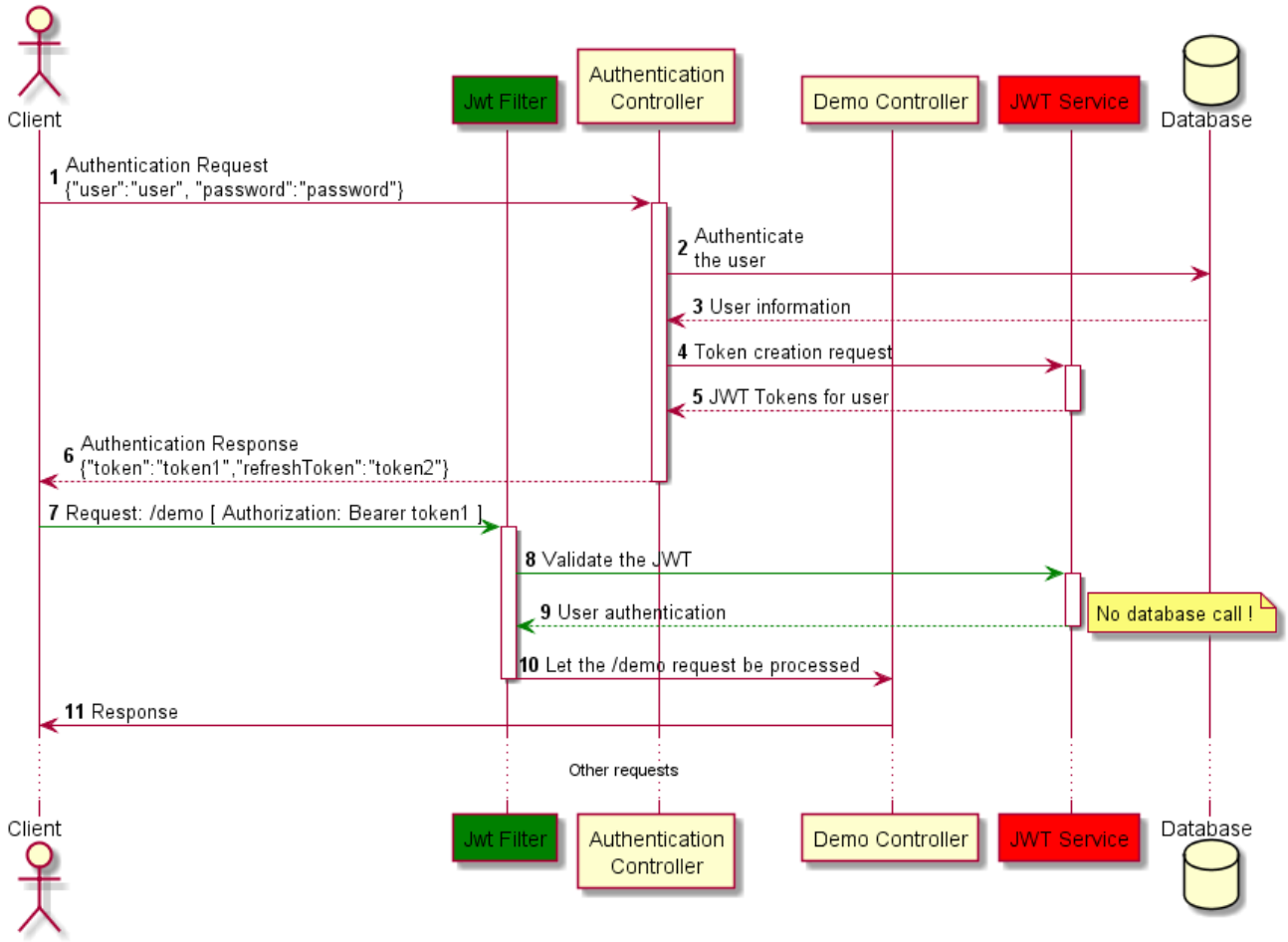
Principe : le client va s'authentifier de l'AA

Statefull AA donne un jeton (*token*) au client et maintient un état des clients authentifiés. L'AA répond aux services qui demandent de vérifier les clients. Exemple : CAS.

Stateless AA donne une preuve vérifiable. Le client présente sa preuve à chaque demande aux services qui y reconnaîtrons la signature de l'AA

Question : comment produire une preuve vérifiable ?

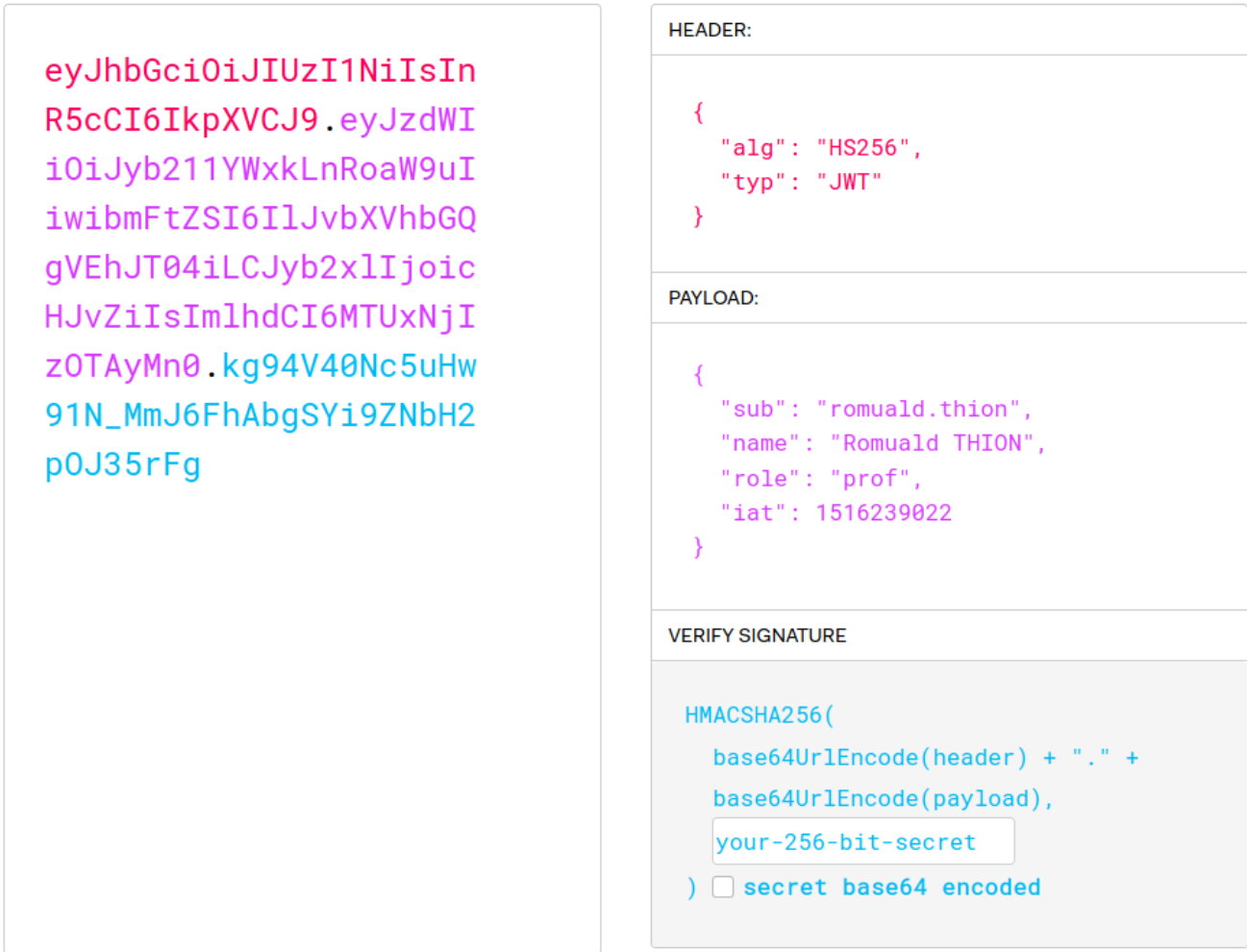
La preuve d'authentification et le SSO : cas JWT



https://blog.ippon.fr/2017/10/12/preuve-dauthentification-avec-jwt/

La preuve d'authentification et le SSO : cas JWT

JSON Web Token https://jwt.io/



La preuve d'authentification et le SSO : HMAC

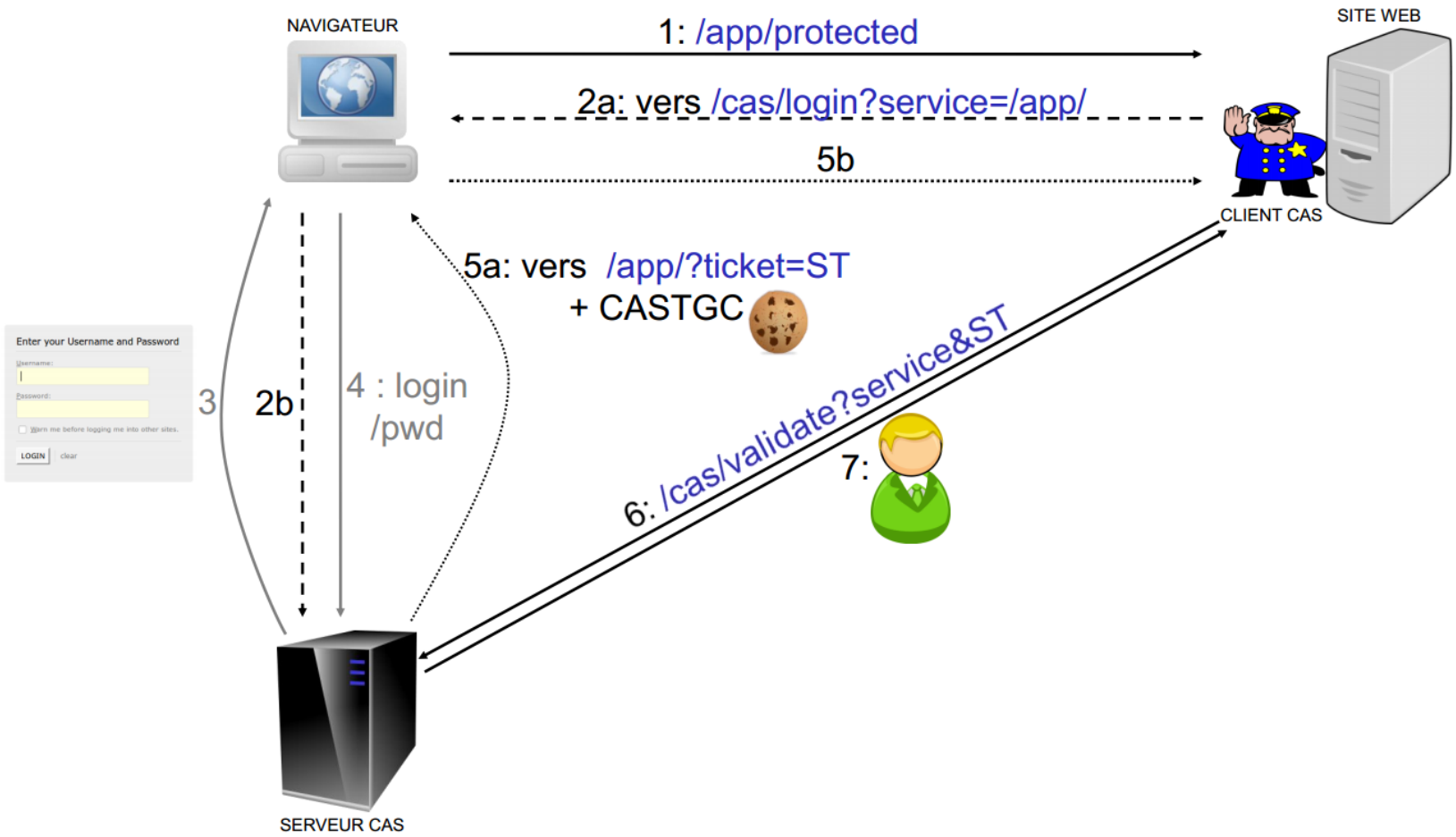
Hash-based message authentication code : on utilise une fonction de hash pour ajouter à un message une preuve (appelée *tag*) qui prouve l'authenticité d'un message, ici « *le serveur dit que . . .* ».

HMAC : Keyed-Hashing for Message Authentication (symétrique)

$$\mathsf{HMAC}_k(m) = h(\bar{k} \oplus \mathsf{opad} \| h(\bar{k} \oplus \mathsf{ipad} \| m))$$

- h une fonction de hashage cryptographique
- b taille de bloc en bits utilisée par la fonction de hashage
- $\bar{k}$ la clef (complétée avec des 0 si $|k| < b$)
- $\|$ la concaténation et $\oplus$ le XOR bit à bit
- ipad et opad deux block de b bits (`0x36` et `0x5c` répétés)

La preuve d'authentification et le SSO : cas CAS



Crédits Gauthier Perrineau (source)

Diagramme de séquence complet officiel

Conclusion

Le diable se cache dans les détails

- BEAST CVE-2011-3389
- HEARTBLEED CVE-2014-0160 `https://heartbleed.com/`
- POODLE CVE-2014-3566
- SWEET32 CVE-2016-2183 `https://sweet32.info/`
- Algorithme `none` dans JWT voir sur Auth0

Remarques finales

- JWT est plus général que l'authentification, on peut faire de l'autorisation en mettant ce qu'on veut dans le JSON.
- Attention, l'utilisation de la cryptographie n'est pas facile : c'est technique et la moindre erreur rend inexploitable.
- Conserver ses secrets : `https://keepassxc.org/`

Références

- Supports de Cédric Lauradoux (dont illustrations)
- A Graduate Course in Applied Cryptography
- OpenSSL PKI Tutorial v1.1
- OpenSSL Cookbook
- Security Protocol Open Repository