



Yannick Perret – CNRS – CSSI LIRIS

UMR 5205 CNRS

Les certificats électroniques (X509)

Laboratoire d'InfoRmatique en Image et Systèmes d'information



INSA



UNIVERSITÉ
LUMIÈRE
LYON 2



Problématique

- Client-serveur : est-ce le bon serveur ?
 - (man-in-the-middle...)
- Serveur-client : est-ce le bon client ?
 - (mot de passe ; IP...)
- Échanges : interception des contenus ?
 - (mots de passe ; données sensibles...)
- Documents (mails, textes) : origine et intégrité ?
 - (faux mail ; document falsifié...)

Besoins

Il faut pouvoir :

- Prouver son identité / vérifier une identité
- Permettre l'échange sécurisé de données
- Permettre la signature de données :
 - Non modification
 - Non répudiation
 - Non réutilisation
 - (+ identification / non falsifiable)

Chiffrement

Rendre non lisibles des données (à transférer) sans la possession de la « clé » de déchiffrement.

Rendre non modifiable les données chiffrées (en tout cas pas sans conséquence : la modification doit être détectable, ou détruire l'ensemble des données).

→ de nombreuses propriétés spécifiques sont nécessaires (chiffrement $\neq$ codage)

Chiffrement symétrique

Une même clé permet de chiffrer **et** de déchiffrer.

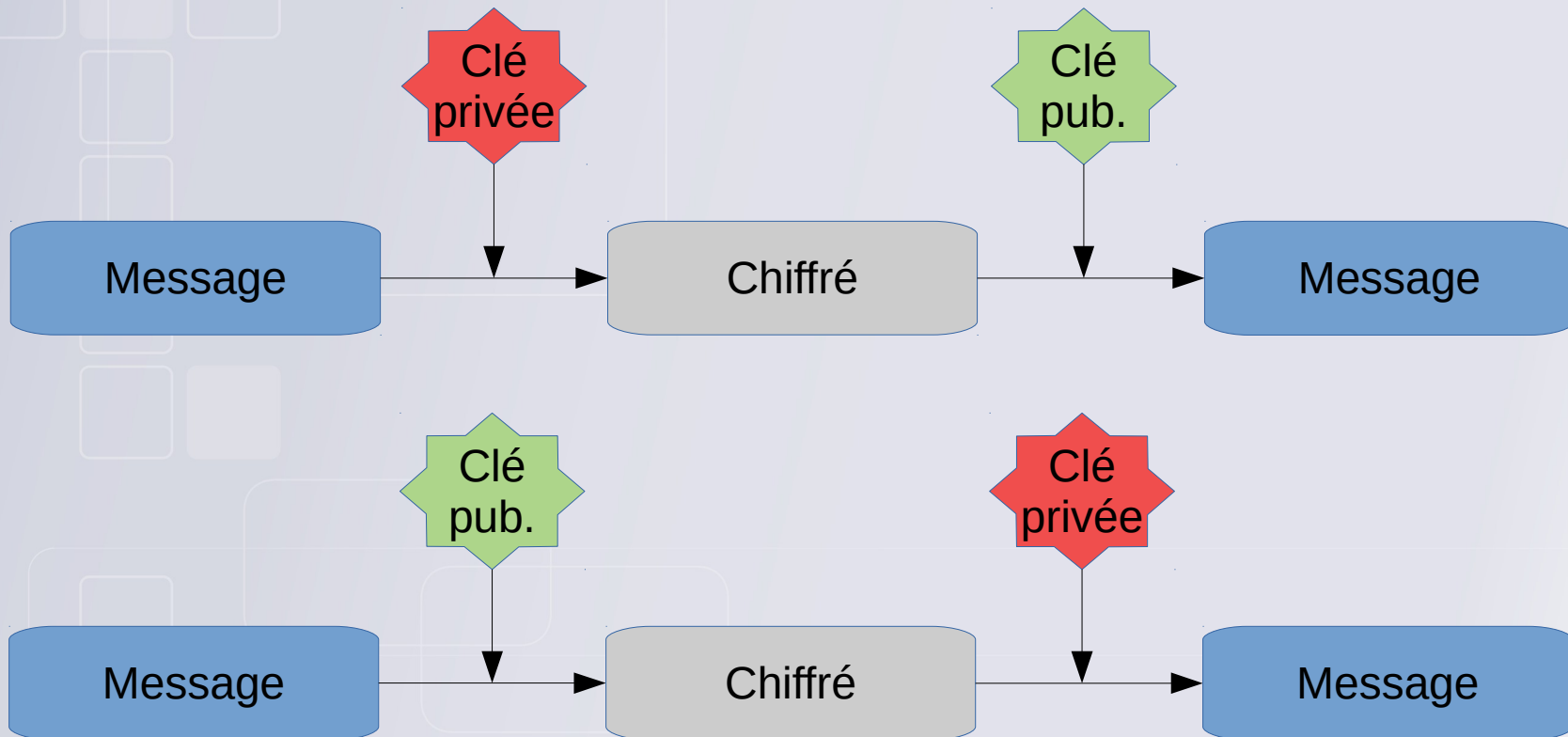


Avantage : simple (rapide)

Inconvénient : comment partager la clé avec son correspondant de façon sûre ?

Chiffrement asymétrique

Deux clés : une privée (C^{pr}), une publique (C^{pu}).



Chiffrement asymétrique (2)

Un message chiffré par clé privée :

- Ne peut être déchiffré que *via* la C^{pu}
- Assure qu'il a été émis par la C^{pr} corresp.

Un message chiffré par clé publique :

- Ne peut être déchiffré que *via* la clé privée

Mais :

- Tout le monde peut déchiffrer ce qui vient de C^{pr}
- Rien n'indique qui a envoyé ce qui vient de C^{pu}

Bilan intermédiaire

Symétrique : efficace, mais comment partager une clé secrète (unique pour chaque paire de correspondants) dans un contexte mondial ?

Asymétrique : clé publique (partage plus simple) mais :

- Assurance qu'on a la bonne clé publique ?
- Tout le monde peut déchiffrer ce qui vient de C^{pr}
- Rien ne dit qui a chiffré avec C^{pu}

→ il faut un mécanisme de preuve des identités

Identité

Comme pour une carte d'identité : une autorité valide un ensemble d'informations permettant de définir une identité.

Dans le cadre du chiffrement asymétrique : une autorité valide une association identité $\leftrightarrow C^{pu}$

Cas des serveurs : seul le serveur doit prouver son identité (la plupart du temps) $\rightarrow$ certificat serveur

Comment faire ?

Certificat numérique X509

Suite d'informations, devant contenir au moins :

- L'identité (nom, mail, entrée DNS...)
- La clé publique associée à l'identité
- + dates de validité, utilisations possibles...
- Une signature de l'autorité

La signature de l'autorité (**CA**) indique que :

- Les données fournies n'ont pas été modifiées
- Le signataire se porte garant du contenu

Signature

Utilise le principe clé publique / privée :

- L'ensemble des données est résumé (via MD5, SHA, ou autre méthode fiable)
- Le résumé est chiffré avec la clé privée du CA
- Le résultat est ajouté aux informations
- Le certificat du CA est ajouté aux informations

→ l'ensemble constitue un certificat

Vérifier un certificat

Les étapes sont :

- Récupérer le certificat du serveur
- Vérifier l'identité fournie (est-ce le bon serveur)
- Faire un résumé des infos présentes (MD5...)
- Déchiffrer la signature avec la clé publique du CA
- Comparer le résumé déchiffré avec celui calculé

Si les résumés sont identiques on a l'assurance que les informations sont telles que signées par le CA.

CA en cascade

Un CA peut être signé par un autre CA : chaîne de certificats, jusqu'à une **autorité racine**

Ceci permet la délégation de signature (en général pour un sous-ensemble des identités, par exemple un sous-domaine, les personnes d'une société...)

Mais : au bout de la chaîne il faut faire « confiance » à l'autorité. Comment ?

CA de confiance

Il faut une liste des CA racine, de confiance :

- Fournie par l'OS (liste acceptée par Windows, Debian, Apple...) et mise à jour régulièrement
- Fournie par l'outil (le navigateur par exemple)
- Fournie par l'utilisateur (import d'un CA de confiance, validé par l'utilisateur)

Failles possibles

- Brut-force (algorithmes trop faibles)
- Failles de conception (des algo. ou du code)
 - retrouver une clé privée, déchiffrer sans avoir le droit...
- Modifier les CA sur une machine (piratage)
- Modifier les CA chez le fournisseur (packages)
 - faire accepter un « faux » CA

Failles possibles (2)

- Voler un CA (signer des certificats soi-même)
- Faire signer illégitimement un CA (fraude)
 - obtenir un « vrai » CA/certificat
- Voler un certificat
 - se faire passer pour un serveur (par ex.)

Listes de révocations

Chaque autorité doit fournir un service de listes de révocations (CRL) pour indiquer les certificats qui ne sont plus valides :

- Certificat remplacé par un autre
- Certificat déclaré volé
- CA compromis

Problème : propagation fiable de ces informations !

Application : transaction HTTPS

Un navigateur se connecte à un serveur HTTPS :

- Le client demande une connexion sécurisée, en indiquant :
 - Le nom du serveur cible
 - Les protocoles supportés
- Le serveur répond en indiquant :
 - Qu'il accepte la connexion
 - Le protocole choisi dans la liste
 - Son certificat (+ certificats intermédiaires)

Application : transaction HTTPS (2)

- Le client valide le certificat
- Le client génère une clé de chiffrement symétrique
- Le client chiffre et envoie cette clé avec la clé publique du serveur : seul le serveur pourra la déchiffrer
- Le serveur récupère la clé symétrique

À partir de là client et serveur vont communiquer en chiffrement symétrique (plus rapide)

Certificat personnel

Même principe : une identité est certifiée par une autorité (CA), en général en incluant des vérifications directes et indirectes (présence dans l'entreprise, confirmation directe de la demande...)

Un certificat personnel peut être utilisé pour :

- Prouver son identité à un service (à la place de login+password / clé / adresse IP / ...)
- Signer des données (document, mail, formulaire)
- Chiffrer des données (document, mail...)

Certificat personnel (2)

Note : pour s'identifier ou signer le certificat (joint au document/mail) suffit.

Pour chiffrer un document il faut le **certificat du destinataire** (sinon tout le monde peut déchiffrer avec notre clé publique...)

→ nécessite un échange préalable de certificats

Note (2) : signer un mail nécessite la C^{pr} → difficile si on utilise un webmail

Certificat personnel (3)

Une fois la communication chiffrée établie :

- Le client fait suivre son certificat
- Le serveur vérifie la validité du certificat
- Le client signe une information avec sa C^{pr}
- Le serveur vérifie cette signature

→ le serveur est sûr qu'il parle avec celui qui possède la clé privée du certificat client reçu

Inconvénients

- Le chiffrement consomme des ressources sur les serveurs et les clients
- Il nécessite d'obtenir des certificats, généralement payants et à renouveler
- Il empêche le fonctionnement des serveurs de cache (proxy)
- Il peut générer des incompréhensions des utilisateurs (alertes sécurité...)

OpenPGP

X.509 est le plus répandu (grâce à HTTPS) mais il existe aussi le format OpenPGP :

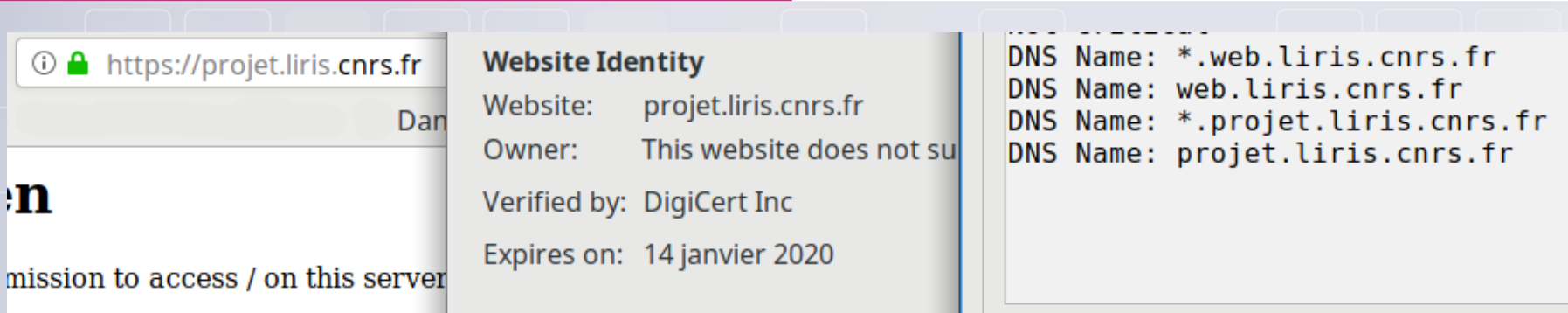
- Utilise les mêmes principes (chiffrement asymétrique + symétrique)
 - Peut contenir plusieurs identités
 - Chaque identité peut avoir plusieurs signatures
- ce dernier point permet de créer une « toile de confiance » : un modèle de confiance décentralisé

Obtenir un certificat

- Via un enregistreur de noms de domaines
- Via une institution (liris.cnrs.fr, univ-lyon1.fr...)
- Via Let's Encrypt (gratuit)
- Être son propre CA

Il faut créer une CSR, puis qu'elle soit signée par un CA

Examples



Could not verify this certificate because the issuer is unknown.

CRL Distribution Points :

URI: <http://crl3.digicert.com/ssca-sha2-g5.crl>

URI: <http://crl4.digicert.com/ssca-sha2-g5.crl>

(oui, en HTTP...)

Questions ?



Liens, ressources

https://fr.wikipedia.org/wiki/Certificat_%C3%A9lectronique

https://fr.wikipedia.org/wiki/Identit%C3%A9_num%C3%A9rique

<https://fr.wikipedia.org/wiki/X.509>

Let's Encrypt : <https://letsencrypt.org>

Créer une CSR :

<https://support.globalsign.com/customer/portal/articles/1221018-generate-csr---openssl>

Créer son propre CA (auto-signé) :

<https://www.linuxtricks.fr/wiki/creer-un-certificat-auto-signe-et-le-renseigner-dans-apache2-ou-nginx>

Obtenir un certificat CNRS (pour les personnels CNRS ou rattachés à une UMR) :

https://igc.services.cnrs.fr/CNRS2-Standard/?body=doc_user_certificate.html

Unix

Créer un CA auto-signé :

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert-ca-as.pem -days 365 -nodes
```

Option « nodes » = pas de passphrase (utilisation non interactive)

Il faut répondre aux questions. Important : ne pas mettre un nom de domaine, un nom de serveur... pour Common Name.

Créer un certificat signé par notre CA :

```
openssl req -new -newkey rsa:4096 -keyout server.key -out server.csr -nodes
```

(répondre aux questions. Le CN doit être le nom du serveur ex. www.mon-site.fr).

```
openssl x509 -req -days 360 -in server.csr -CA cert-ca-as.pem -CAkey key.pem -CAcreateserial -out server.crt
```

Ne mettre CAcreateserial que la 1ère fois (fichier de N° de série)

Let's Encrypt

Sur une Debian, Apache, site par défaut activé :

```
apt-get update ; apt-get install certbot
```

```
Certbot --apache
```

Certbot va regarder la configuration apache, détecter les VirtualHost existant, faire une (ou des) demande de certificat à Let's Encrypt moyennant un ensemble de *challenges* à réussir.

Certbot va ensuite stocker les certificats et proposer de modifier la configuration apache pour les utiliser.

Note : le package comprend une entrée *cron* qui s'occupe de renouveler les certificats lorsque nécessaire.

Pour des situations plus complexes voir [la doc de certbot](#).