M1 MEEF NSI — Sécurité des communications informatiques

PoID, Signatures & TLS

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://membres-ljk.imag.fr/Pierre.Karpman/tea.html

2024–02–16

Preuve d'identité interactive

Signatures

TLS

Objectif : preuve publique d'identité

Contexte du moment :

- Une personne $\mathscr{A}$ souhaite prouver son identité à une personne $\mathscr{B}$ sur un canal fiable
- Face à des adversaires passifs

Comment faire si $\mathscr{A}$ & $\mathscr{B}$:

- Possèdent un petit *élément public partagé* ?

Protocole à base de fonction de hachage

Un protocole à usage unique :

1. Soit $\mathcal{H}$ une fonction de hachage, $\mathcal{R}$ un ensemble « suffisamment grand »
2. $\mathscr{A}$ tire $x \twoheadleftarrow \mathcal{R}$, calcule $X = \mathcal{H}(x)$ et le partage publiquement avec $\mathscr{B}$
3. Pour prouver son identité, $\mathscr{A}$ révèle x

$\rightsquigarrow$ Sécurité réduite à la résistance aux préimages de $\mathcal{H}$

Variante à usage multiple :

1. $\mathscr{A}$ tire $x \twoheadleftarrow \mathcal{R}$, calcule $X = \mathcal{H}^N(x)$ et le partage publiquement avec $\mathscr{B}$
2. Pour prouver son identité à l'étape $i \leq N$, $\mathscr{A}$ révèle $\mathcal{H}^i(x)$ et $\mathscr{B}$ redéfinit $X = \mathcal{H}^{i-1}(x)$

L'idée se généralise et mène notamment aux *signatures à base de fonction de hachage*

Protocole à base de logarithme discret

1. Soit $\mathbb{G} = \langle g \rangle$ un groupe cyclique de taille N
2. $\mathscr{A}$ tire $x \twoheadleftarrow [\![0, N-1]\!]$, calcule $X = g^x$ et le partage publiquement avec $\mathscr{B}$
3. Pour prouver son identité, $\mathscr{A}$ tire $r \twoheadleftarrow [\![0, N-1]\!]$, envoie $R := g^r$ à $\mathscr{B}$
4. $\mathscr{B}$ choisit c et l'envoie à $\mathscr{A}$
5. $\mathscr{A}$ calcule $a := (r + cx) \% N$ et l'envoie à $\mathscr{B}$
6. $\mathscr{B}$ vérifie que $RX^c = g^a$

$\rightsquigarrow$ Sécurité réduite à la difficulté de calculer des logarithmes discrets dans $\mathbb{G}$

$\rightsquigarrow$ Propriété "zero-knowledge" : le protocole ne fuite aucune information ! (cf. TD)

Le protocole peut être exécuté plusieurs fois, mais avec des valeurs de r différentes (et toujours uniformes) !

PoID v. Signatures

Un protocole de preuve d'identité :

- Est interactif
- N'« inclut pas » de message

Si on change ces deux points $\rightsquigarrow$ *signature*

Preuve d'identité interactive

Signatures

TLS

Objectif : signature

Contexte du moment :

- Une personne $\mathscr{A}$ souhaite certifier un message auprès d'une $\mathscr{B}$ sur un canal fiable
- Face à des adversaires *actifs*

Comment faire si $\mathscr{A}$ & $\mathscr{B}$:

- Possèdent un petit *élément public partagé* ?

Schéma de signature : fonctionnalité

Un schéma de signature comprend typiquement :

- Un algorithme de génération de clefs
  Keygen : $\{0,1\}^\kappa \to \mathcal{K}_s \times \mathcal{K}_v$
- Un algorithme de signature Sig : $\mathcal{K}_s \times \mathcal{M} \to \mathcal{S}$
- Un algorithme de vérification Ver : $\mathcal{K}_v \times \mathcal{M} \times \mathcal{S} \to \{\top, \bot\}$

Avec l'exigence que pour tout $(s_k, v_k) = \text{Keygen}(*)$, pour tout $m \in \mathcal{M}$, $\text{Ver}(v_k, m, \text{Sig}(s_k, m)) = \top$ (avec très forte proba.)

Signatures : remarques

- Contrairement à un MAC, une signature est *asymétrique* : les clefs de Sig et Ver sont (a priori) distinctes
- Mais pour vérifier une signature avec s_k, il faut connaître v_k
  $\rightsquigarrow$ comment faire en présence d'adversaire actif ?
- Une approche : les *public-key infrastructures* (PKI) (cf. plus bas)

Schéma de signature : sécurité

Sécurité d'une signature par ex. analysée relativement au jeu EUF-CMA : un adversaire capable de demander des signatures de messages de son choix pour une clef s_k inconnue n'arrive pas à produire une signature valide pour un autre message :

1. $(s_k, v_k) = \text{Keygen}(\$)$; on donne v_k à l'adversaire
2. L'adversaire envoie des requêtes m_i et obtient $\text{Sig}(s_k, m_i)$
3. L'adversaire retourne (σ_f, m_f) et gagne si $m_f \neq_i m_i$ et $\text{Ver}(v_k, m_f, \sigma_f) = \top$

$\rightsquigarrow$ Comme pour un MAC, en adaptant à la fonctionnalité

Signatures de Schnorr

Le protocole de PoID à base de logarithme discret peut être transformé en :

- Supprimant l'interaction (le choix de c par $\mathscr{B}$)
- Faisant dépendre la signature d'un message

$\rightsquigarrow$ Transformée de Fiat-Shamir : utiliser le message pour générer c !

Pour signer m avec $(s_k := x, v_k := X := g^x)$:

1. $\mathscr{A}$ choisit $r \twoheadleftarrow [\![0, N-1]\!]$ et calcule $R = g^r$
2. Calcule $c = \mathcal{H}(R, X, g, m)$
3. Calcule $a = (r + cx) \ \% \ N$ et renvoie $(m, (c, a))$

Pour vérifier une signature :

1. $\mathscr{B}$ calcule $\hat{R} = g^a / X^c = g^a / g^{cx}$
2. Vérifie que $c = \mathcal{H}(\hat{R}, X, g, m)$

Sécurité réduite à DLOG, dans le ROM...

Preuve d'identité interactive

Signatures

TLS

Sécurisation des communications : exemple de TLS

Un client $\mathscr{C}$ veut communiquer de façon sûre avec un serveur $\mathscr{S}$:

- $\mathscr{S}$ doit prouver à $\mathscr{C}$ qu'il est le bon serveur
  - En utilisant une signature (par ex. DSA $\approx$ sig. de Schnorr)
- $\mathscr{C}$ et $\mathscr{S}$ doivent échanger un secret
  - En utilisant un protocole d'échange de clef (par ex. DH)
- $\mathscr{C}$ et $\mathscr{S}$ peuvent utiliser ce secret pour communiquer
  - Par exemple en utilisant un chiffrement en mode CTR couplé à un mécanisme d'authentification

Sur internet $\rightsquigarrow$ TLS

TLS : *Transport Security Layer*

- Évolution d'SSL (*Secure Socket Layer*) : 95-99
- Version : 1.3 depuis 2018
- Un protocole compliqué
  - Mélange crypto, réseau, implémentation
  - Cf. par ex. les RFCs ; Wikipédia...

TLS en très rapide

TLS utilise :

- Un protocole de *handshake*
  - Pour établir un secret partagé
- Un *record protocol*
  - Pour échanger des données

Il repose aussi fortement sur des *autorités de certification* (CA)

- Pour « initialiser » la confiance en les serveurs

Handshake rapide

Objectifs :

- (Exécuter le protocole d'échange de clef, généralement en prouvant l'identité de $\mathscr{S}$, possiblement (plus rare) en prouvant l'identité de $\mathscr{C}$)
- Négocier la version du protocole
- Négocier les algorithmes à utiliser ensuite

Dans une image

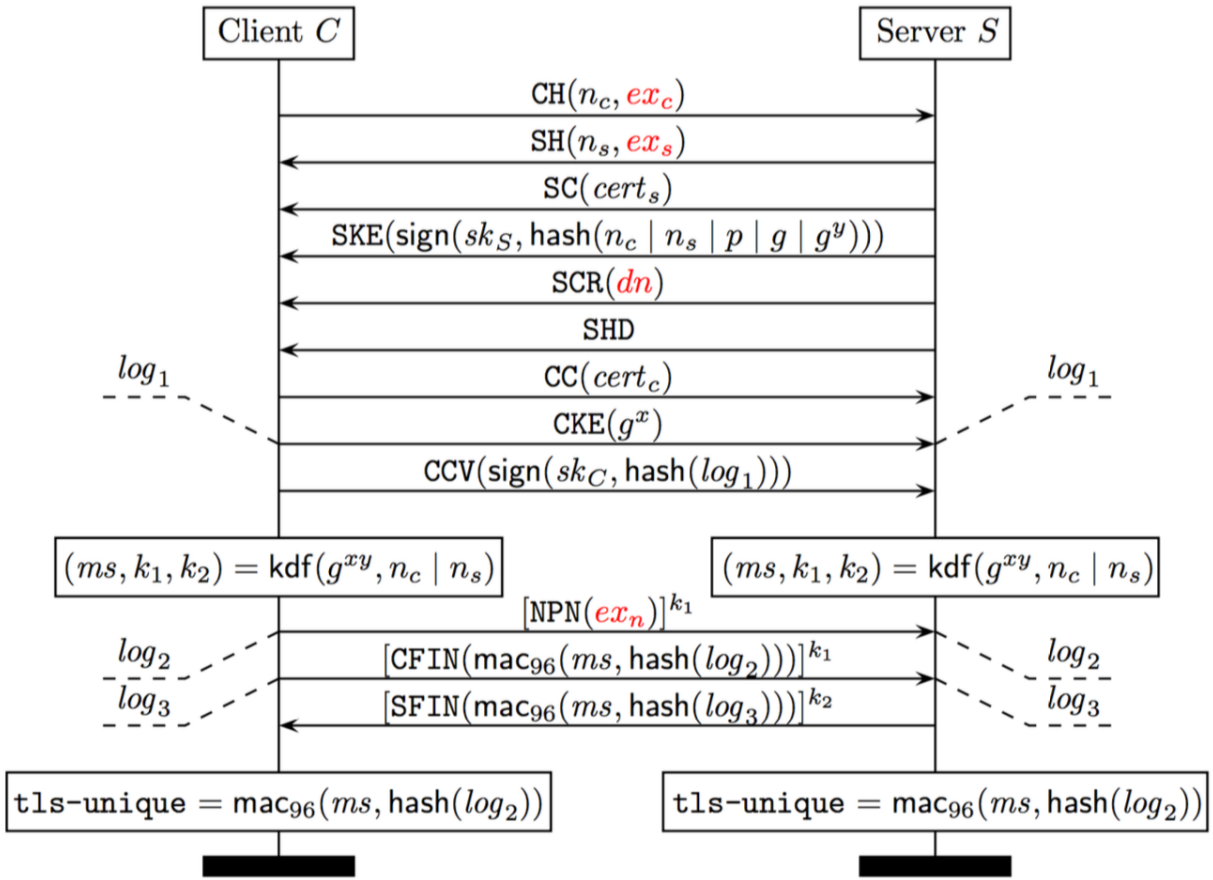

Figure – Handshake mutuellement authentifié avec DHE (Bhargavan & Leurent, 2016)

Remarques rapides

- Les paramètres de l'échange envoyés par le serveur sont signés
  - Prouve qu'il connaît (en principe) la clef de signature liée à son *certificat*
  - Protège des modifications (comme un MAC)
- L'échange se termine avec un échange de MACs authentifiant tout le transcript
  - Permet de vérifier que $\mathscr{C}$ et $\mathscr{S}$ sont d'accord sur le secret
- `tls-unique` peut être utilisé pour uniquement (ou pas...) identifier l'échange
  - Peut être utilisé à un niveau applicatif supérieur

Les certificats : quel rôle ?

Certificats X.509 : $\approx$ clefs de signature elle-même signées : contiennent

- Un numéro de série
- L'algorithme utilisé pour signer le certificat
- Les identité :
  - De l'émetteur (par ex. *Let's Encrypt*, typiquement une *Certification Authority*)
  - Du sujet (par ex. `univ-grenoble-alpes.fr`)
- Des dates de validité
- La clef de vérification du sujet (pour un certain algorithme)
- Si le sujet est lui-même une autorité de certification

Certification Authorities

CAs :

- Sont de confiance (par ex. ont la confiance des navigateurs Web)
- Permette de certifier des partis tiers
  1. Certifie que l'utilisateur $\mathscr{S}$ est qui il prétend être
  2. — qu'il connaît une paire de clef de signature et de vérification
  3. (Et signe un certificat contenant ces informations)

  - Un client faisant confiance à la CA peut maintenant (en principe) faire confiance au certificat de $\mathscr{S}$
- Peut déléguer la signature de certificats
  - $\rightsquigarrow$ Chaînes de certificats : CA *racine* $\rightarrow$ (CA *intermédiaires*)* $\rightarrow$ sujet terminal
  - (Il peut y avoir des restrictions sur la longueur des chaînes dans laquelle un CA peut intervenir)

(Les conséquences en cas de CA malicieux/attaqué peuvent être dramatiques)

Qui signe quoi ?

En fonction du contexte, les certificats utilisés dans TLS peuvent :

- N'être signé par aucun CA
  - Mais par ex. être auto-signés : protège des attaques si $\mathscr{C}$ « connaît déjà » $\mathscr{S}$
  - Similaire à l'utilisation courante de SSH ; ne passe pas à l'échelle
- Être signé par un CA gratuit
  - Par ex. `https://letsencrypt.org/`
- Être signé par un CA commercial/organisationnel (par ex. DigiCert/TERENA)

Gestion plus fine : *certificate pinning*

Un risque avec les CAs :

- Il y a *beaucoup* de CAs
  - Plus de 100 CAs racine, qui peuvent ensuite déléguer leur signature
- Possible pour des CAs d'émettre de faux certificats si malicieux/attaqué
  - Exemple : DigiNotar (en 2011)

Une tentative de contremesure : le *certificate/public key pinning*

- Un service/site web annonce (par ex. aux développeurs d'un navigateur) quel CA il utilise
- Lors d'une connexion, les certificats issus d'autres CAs sont rejetés
- Peut marcher pour quelques services critiques, mais ne passe pas à l'échelle

Autre approche : *certificate transparency*

Cf. `https://www.certificate-transparency.org` : liste géante de certificats

- Les CAs et utilisateurs peuvent ajouter des certificats à une liste géante monotone (*append-only*)
- Permet de tracer les erreurs/abus/attaques
- Possible de vérifier l'authenticité d'un certificat douteux
- (Assez lourd)

$\rightsquigarrow$ La *distribution de clef* à large échelle est toujours un problème compliqué !