Vérification de protocoles cryptographiques

Bruno Blanchet

INRIA Paris
bruno.blanchet@inria.fr

Septembre 2021

(partiellement inspiré de transparents de Stéphanie Delaune)

Protocoles cryptographiques

Protocoles cryptographiques

- Petits programmes conçus pour sécuriser les communications

Protocoles cryptographiques

Protocoles cryptographiques

- Petits programmes conçus pour sécuriser les communications

Propriétés de sécurité

Les protocoles cryptographiques cherchent à garantir des propriétés de sécurité :

- **Secret** : Un attaquant ne doit pas pouvoir apprendre un message secret entre deux participants honnêtes.
- Authentification : Si l'agent Alice pense parler avec Bob, alors Alice parle vraiment avec Bob.
- ...

Primitives cryptographiques

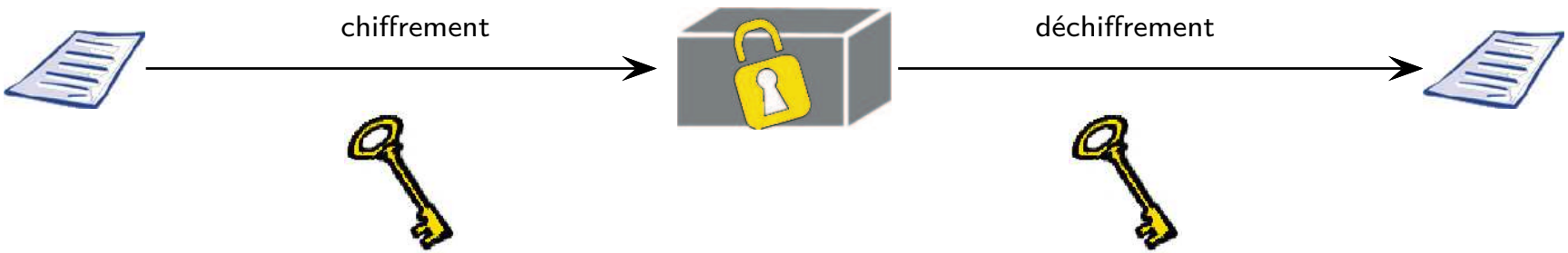
Primitives cryptographiques

Algorithmes de base fréquemment utilisés pour construire des protocoles cryptographiques. Ces routines incluent entre autres le chiffrement et les signatures.

Primitives cryptographiques

Algorithmes de base fréquemment utilisés pour construire des protocoles cryptographiques. Ces routines incluent entre autres le chiffrement et les signatures.

Chiffrement symétrique

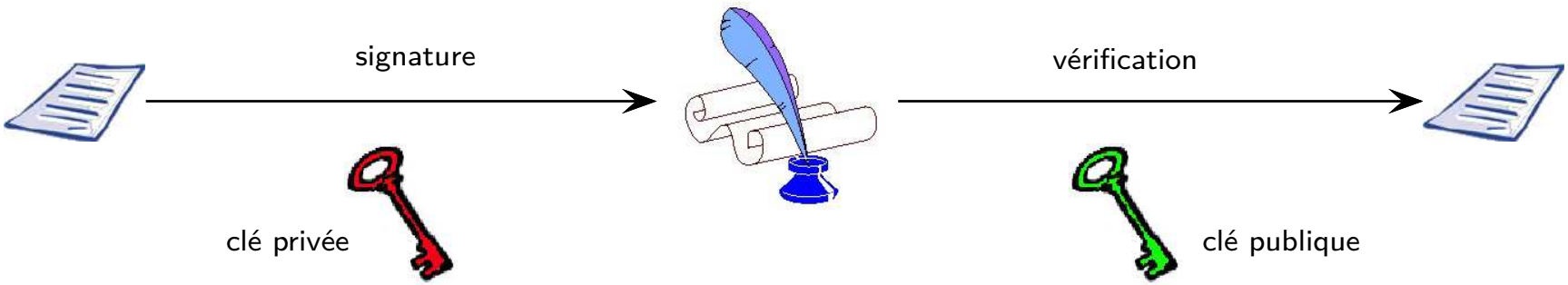


$\longrightarrow$ Exemples : DES, AES, ...

Primitives cryptographiques

Algorithmes de base fréquemment utilisés pour construire des protocoles cryptographiques. Ces routines incluent entre autres le chiffrement et les signatures.

Signature



Pourquoi vérifier les protocoles cryptographiques ?

Depuis les années 1990, la vérification des protocoles cryptographiques est un sujet de recherche très actif.

- Leur conception est délicate.
- Les erreurs de sécurité ne sont pas détectées par les tests :
  Elles apparaissent seulement en présence d'un attaquant.
- Les erreurs peuvent avoir de graves conséquences.

Exemple : paiement par carte de crédit

- Le client Cl met sa carte de crédit C dans le terminal T.
- Le marchand entre le montant M de la vente.

- Le terminal authentifie la carte de crédit.
- Le client entre son code.
  Si $M \geq 100\,€$, alors dans 20% des cas,
  - Le terminal contacte la banque B.
  - La banque donne son autorisation.

Plus en détail

La Banque B, le Client Cl, la Carte de crédit C et le Terminal T

Plus en détail

La Banque B, le Client Cl, la Carte de crédit C et le Terminal T

Banque

- une clé de signature privée – $\mathrm{priv}(B)$
- une clé publique pour vérifier une signature – $\mathrm{pub}(B)$
- une clé secrète partagée avec la carte de crédit – K_{CB}

Plus en détail

La Banque B, le Client Cl, la Carte de crédit C et le Terminal T

Banque

- une clé de signature privée – $\mathsf{priv}(B)$
- une clé publique pour vérifier une signature – $\mathsf{pub}(B)$
- une clé secrète partagée avec la carte de crédit – K_{CB}

Carte de crédit

- des *Données* : le nom du porteur, date d'expiration, ...
- une signature des *Données* – $\mathsf{sign}(\mathit{Données}, \mathsf{priv}(B))$
- une clé secrète partagée avec la banque – K_{CB}

Plus en détail

La Banque B, le Client Cl, la Carte de crédit C et le Terminal T

Banque

- une clé de signature privée – $\mathsf{priv}(B)$
- une clé publique pour vérifier une signature – $\mathsf{pub}(B)$
- une clé secrète partagée avec la carte de crédit – K_{CB}

Carte de crédit

- des *Données* : le nom du porteur, date d'expiration, ...
- une signature des *Données* – $\mathsf{sign}(\mathit{Données}, \mathsf{priv}(B))$
- une clé secrète partagée avec la banque – K_{CB}

Terminal

- la clé publique de la banque – $\mathsf{pub}(B)$

Protocole de paiement

Le terminal T lit la carte de crédit C :

1. $C \rightarrow T$: $\mathit{Données}, \mathsf{sign}(\mathit{Données}, \mathsf{priv}(B))$

Protocole de paiement

Le terminal T lit la carte de crédit C :

1. $C \rightarrow T$: $\mathit{Données}, \mathsf{sign}(\mathit{Données}, \mathsf{priv}(B))$

Le terminal T demande le code :

2. $T \rightarrow CI$: $\mathit{code?}$
3. $CI \rightarrow C$: 1234
4. $C \rightarrow T$: ok

Protocole de paiement

Le terminal T lit la carte de crédit C :

1. $C \rightarrow T$: $\mathit{Données}, \mathsf{sign}(\mathit{Données}, \mathsf{priv}(B))$

Le terminal T demande le code :

2. $T \rightarrow CI$: $\mathit{code?}$
3. $CI \rightarrow C$: 1234
4. $C \rightarrow T$: ok

Le terminal T demande l'autorisation de la banque B :

5. $T \rightarrow B$: $\mathit{auth?}$
6. $B \rightarrow T$: 4528965874123
7. $T \rightarrow C$: 4528965874123
8. $C \rightarrow T$: $\mathsf{enc}(4528965874123, K_{CB})$
9. $T \rightarrow B$: $\mathsf{enc}(4528965874123, K_{CB})$
10. $B \rightarrow T$: ok

Attaques contre les cartes de crédit

Cependant, il y a des attaques !

- attaque cryptographique : les clés de signature de 320 bits ne sont plus sûres,
- attaque logique : pas de lien entre le code à quatre chiffres et l'authentification,
- attaque matérielle : duplication de cartes.

Attaques contre les cartes de crédit

Cependant, il y a des attaques !

- attaque cryptographique : les clés de signature de 320 bits ne sont plus sûres,
- attaque logique : pas de lien entre le code à quatre chiffres et l'authentification,
- attaque matérielle : duplication de cartes.

→ « YesCard » fabriquée par Serge Humpich (1997).

La « YesCard » : Comment fonctionne-t-elle ?

Attaque logique

$$
\begin{aligned}
1.C &\rightarrow T &&: \mathit{Données}, \mathsf{sign}(\mathit{Données}, \mathsf{priv}(B)) \\
2.T &\rightarrow CI &&: \mathit{code}? \\
3.CI &\rightarrow C &&: 1234 \\
4.C &\rightarrow T &&: \mathit{ok}
\end{aligned}
$$

La « YesCard » : Comment fonctionne-t-elle ?

Attaque logique

$$
\begin{aligned}
1.\ C &\rightarrow T &&: \mathit{Données}, \mathsf{sign}(\mathit{Données}, \mathsf{priv}(B)) \\
2.\ T &\rightarrow \mathit{CI} &&: \mathit{code}? \\
3.\ \mathit{CI} &\rightarrow C' &&: 2345 \\
4.\ C' &\rightarrow T &&: \mathit{ok}
\end{aligned}
$$

La « YesCard » : Comment fonctionne-t-elle ?

Attaque logique

$$\begin{aligned} 1.C &\rightarrow T &&: \mathit{Données}, \mathsf{sign}(\mathit{Données}, \mathsf{priv}(B)) \\ 2.T &\rightarrow CI &&: \mathit{code}? \\ 3.CI &\rightarrow C' &&: 2345 \\ 4.C' &\rightarrow T &&: \mathit{ok} \end{aligned}$$

Remarque : il y a toujours quelqu'un à débiter.
$\rightarrow$ ajouter une fausse signature sur une fausse carte (Serge Humpich, attaque cryptographique contre le schéma de signature).

La « YesCard » : Comment fonctionne-t-elle ?

Attaque logique

$$\begin{aligned} 1.C &\rightarrow T &&: \mathit{Données}, \mathsf{sign}(\mathit{Données}, \mathsf{priv}(B)) \\ 2.T &\rightarrow CI &&: \mathit{code}? \\ 3.CI &\rightarrow C' &&: 2345 \\ 4.C' &\rightarrow T &&: \mathit{ok} \end{aligned}$$

Remarque : il y a toujours quelqu'un à débiter.
$\rightarrow$ ajouter une fausse signature sur une fausse carte (Serge Humpich, attaque cryptographique contre le schéma de signature).

$$\begin{aligned} 1.C' &\rightarrow T &&: \mathit{XXX}, \mathsf{sign}(\mathit{XXX}, \mathsf{priv}(B)) \\ 2.T &\rightarrow CI &&: \mathit{code}? \\ 3.CI &\rightarrow C' &&: 0000 \\ 4.C' &\rightarrow T &&: \mathit{ok} \end{aligned}$$

Modèles de protocoles : le modèle symbolique

Le modèle symbolique ou « modèle de Dolev-Yao » est dû à Needham et Schroeder (1978) et Dolev et Yao (1983).

- Les primitives cryptographiques sont des boîtes noires. enc
- Les messages sont des termes sur ces primitives. $\mathsf{enc}(\mathit{Hello}, k)$
- L'attaquant est restreint à calculer en utilisant seulement ces primitives.
  $\Rightarrow$ hypothèse de cryptographie parfaite

Ce modèle rend les preuves automatiques relativement faciles (Tamarin, ProVerif, ...).
Ce modèle détecte les attaques logiques.

Vérifier des protocoles dans le modèle symbolique

Idée principale (pour la plupart des vérificateurs) :

- Calculer la connaissance de l'attaquant.

Difficulté : les protocoles cryptographiques ont un ensemble d'états infini.

- L'attaquant peut créer des messages de taille non bornée.
- Le protocole peut être exécuté un nombre non borné de fois.

ProVerif : Idée principale

Représenter les protocoles par des formules logiques (clauses de Horn).

Cette représentation utilise le prédicat attacker :

attacker(M) *signifie* *"l'attaquant peut avoir M".*

Les actions de l'attaquant et des participants du protocole peuvent être modélisées grâce à ce prédicat.

Codage des primitives

Exemple : chiffrement à clé partagée $\mathrm{enc}(m, k)$

$\mathrm{attacker}(m) \wedge \mathrm{attacker}(k) \rightarrow \mathrm{attacker}(\mathrm{enc}(m, k))$

Exemple : déchiffrement à clé partagée $\mathrm{dec}(\mathrm{enc}(m, k), k) = m$

$\mathrm{attacker}(\mathrm{enc}(m, k)) \wedge \mathrm{attacker}(k) \rightarrow \mathrm{attacker}(m)$

En général, si $f(M_1, \ldots, M_n) = M$,
$\mathrm{attacker}(M_1) \wedge \ldots \wedge \mathrm{attacker}(M_n) \rightarrow \mathrm{attacker}(M)$

Codage du protocole

Si un participant a reçu les messages $M_1, \ldots, M_n$ et envoie le message M,

$$\text{attacker}(M_1) \wedge \ldots \wedge \text{attacker}(M_n) \rightarrow \text{attacker}(M).$$

Exemple

7. $? \rightarrow C : x$
8. $C \rightarrow ? : \text{enc}(x, K_{CB})$

A la réception d'un message x, la carte C répond $\text{enc}(x, K_{CB})$:

$$\text{attacker}(x) \rightarrow \text{attacker}(\text{enc}(x, K_{CB}))$$

L'attaquant envoie x à la carte C, et intercepte sa réponse $\text{enc}(x, K_{CB})$.

Preuve de secret

Théorème (Secret)

Si attacker(M) *ne peut pas être dérivé à partir des formules logiques, alors M est secret.*

Le terme M ne peut pas être construit par un attaquant.

L'algorithme de résolution déterminera si un fait donné peut être dérivé à partir des formules logiques.