Providing solutions for more secure exchanges

Stéphanie Delaune

November 15, 2011
Cryptographic protocols

- small programs designed to secure communication (various security goals)
- use cryptographic primitives (e.g. encryption, hash function, ...)
Cryptographic protocols

- small programs designed to secure communication (various security goals)
- use cryptographic primitives (e.g. encryption, hash function, ...)

The network is unsecure!

Communications take place over a public network like the Internet.
Cryptographic protocols

- small programs designed to secure communication (various security goals)
- use cryptographic primitives (e.g. encryption, hash function, ...)
Security properties (1)

- **Secrecy**: May an intruder learn some secret message between two honest participants?

- **Authentication**: Is the agent Alice really talking to Bob?

- **Fairness**: Alice and Bob want to sign a contract. Alice initiates the protocol. May Bob obtain some advantage?

- **Privacy**: Alice participate to an election. May a participant learn something about the vote of Alice?

- **Non-repudiation**: Alice sends a message to Bob. Alice cannot later deny having sent this message. Bob cannot deny having received the message.

...
Eligibility: only legitimate voters can vote, and only once

Fairness: no early results can be obtained which could influence the remaining voters

Individual verifiability: a voter can verify that her vote was really counted

Universal verifiability: the published outcome really is the sum of all the votes
Security properties: E-voting (3)

**Privacy:** the fact that a particular voted in a particular way is not revealed to anyone

**Receipt-freeness:** a voter cannot prove that she voted in a certain way (this is important to protect voters from coercion)

**Coercion-resistance:** same as receipt-freeness, but the coercer interacts with the voter during the protocol, (e.g. by preparing messages)
Cryptographic primitives

Algorithms that are frequently used to build computer security systems. These routines include, but are not limited to, one-way hash functions and encryption functions.
Cryptographic primitives

Algorithms that are frequently used to build computer security systems. These routines include, but are not limited to, one-way hash functions and encryption functions.

Symmetric encryption
Cryptographic primitives

Algorithms that are frequently used to build computer security systems. These routines include, but are not limited to, one-way hash functions and encryption functions.

Asymmetric encryption

- Encryption: Public key
- Decryption: Private key

Stéphanie Delaune (LSV)
Symmetric vs. asymmetric encryption

Symmetric encryption

- **efficient** in practice,
- agents have to **share a secret key**
  \[\rightarrow\] trusted third party, distribution key protocol

Asymmetric encryption

- **not efficient** in practice,
- agents do not have to share a secret
  \[\rightarrow\] often used in establishment key protocols
- **authentication** of public keys (certificate)
Digital signature: How does it work?

- similar to public key encryption
- everyone knows the key to verify the signature (**public** key)
- the key used to sign a message has to be **private** (**private** key)
Properties

- the signature has to authenticate the signer
- the signature “belongs to“ one particular document
- the signed document can not be modified afterwards
Properties and applications

Properties

- the signature has to authenticate the signer
- the signature “belongs to“ one particular document
- the signed document can not be modified afterwards

Applications

- certificate to authenticate a public key
- contract signing protocols
- E-voting protocols (blind signature)
  → allows someone to sign without knowing the message he is signing.
Hash function: What is it?

It is a reproducible method of turning some kind of data into a (relatively) small number that may serve as a digital "fingerprint" of the data (again substitutions and permutations).

Examples: MD5, SHA-1
Properties and Applications

Properties

- **deterministic function**
- **one-way function**: there is no practical way to retrieve $m$ from $\text{hash}(m)$
- **collision resistant**: difficult to find $m_1$ and $m_2$ such that $m_1 \neq m_2$ and $\text{hash}(m_1) = \text{hash}(m_2)$

Some applications

- to improve **efficiency**: we can sign $\text{hash}(m)$ instead of $m$
- use to **guarantee the integrity** of a message
- checksum to detect errors
Outline of the talk

1. Introduction

2. Some examples of security protocols
   - Credit Card payment
   - Needham Schroeder protocol

3. How can we verify them?
   - How protocols can be attacked?
   - How protocols can be proved secure?

4. Conclusion
Outline of the talk

1. Introduction

2. Some examples of security protocols
   - Credit Card payment
   - Needham Schroeder protocol

3. How can we verify them?
   - How protocols can be attacked?
   - How protocols can be proved secure?

4. Conclusion
La carte bleue est protégée par un grand nombre public dont on ne connait pas la factorisation.
La carte bleue est protégée par un grand nombre public dont on ne connait pas la factorisation.

Nombre de 96 chiffres

2135987035920910082395022704999628797051095341826417406442524165008583957746445088405009430865999
La carte bleue est protégée par un grand nombre public dont on ne connaît pas la factorisation.

**Nombre de 96 chiffres**

```
213598703592091008239502270499962879705109534182641740644252
4165008583957746445088405009430865999
```

**Affaire Serge Humpich (1997)**

Il factorise ce nombre de 96 chiffres et conçoit de fausses cartes bleues (les “YesCard”).
La carte bleue est protégée par un grand nombre public dont on ne connait pas la factorisation.

**Nombre de 96 chiffres**

2135987035920910082395022704999628797051095341826417406442524165008583957746445088405009430865999

**Affaire Serge Humpich (1997)**

Il factorise ce nombre de 96 chiffres et conçoit de fausses cartes bleues (les “YesCard”).

⇒ Depuis, le nombre utilisé pour sécuriser les cartes bancaires comportent 232 chiffres.
Example: credit card payment

- The client \( C_l \) puts his credit card \( C \) in the terminal \( T \).

- The merchant enters the amount \( M \) of the sale.

- The terminal authenticates the credit card.

- The client enters his PIN.
  
  If \( M \geq 100 \)€, then in 20% of cases,
  
  - The terminal contacts the bank \( B \).
  - The banks gives its authorisation.
the Bank $B$, the Client $Cl$, the Credit Card $C$ and the Terminal $T$
the Bank $B$, the Client $Cl$, the Credit Card $C$ and the Terminal $T$

**Bank**

- a **private** signature key – $\text{priv}(B)$
- a **public** key to verify a signature – $\text{pub}(B)$
- a **secret** key shared with the credit card – $K_{CB}$
the Bank $B$, the Client $Cl$, the Credit Card $C$ and the Terminal $T$

**Bank**
- a private signature key – $\text{priv}(B)$
- a public key to verify a signature – $\text{pub}(B)$
- a secret key shared with the credit card – $K_{CB}$

**Credit Card**
- some *Data*: name of the cardholder, expiry date ...
- a signature of the *Data* – $\{\text{hash(Data)}\}_{\text{priv}(B)}$
- a secret key shared with the bank – $K_{CB}$
More details

the Bank $B$, the Client $C_l$, the Credit Card $C$ and the Terminal $T$

**Bank**
- a **private** signature key – $\text{priv}(B)$
- a **public** key to verify a signature – $\text{pub}(B)$
- a **secret** key shared with the credit card – $K_{CB}$

**Credit Card**
- some **Data**: name of the cardholder, expiry date ...
- a signature of the **Data** – $\{\text{hash(Data)}\}_{\text{priv}(B)}$
- a **secret** key shared with the bank – $K_{CB}$

**Terminal**
- the **public** key of the bank – $\text{pub}(B)$
the terminal $T$ reads the credit card $C$:

1. $C \rightarrow T : \text{Data}, \{\text{hash(Data)}\}_{\text{priv}(B)}$
the terminal $T$ reads the credit card $C$:

1. $C \rightarrow T : \text{Data, } \{\text{hash(Data)}\}_{\text{priv}(B)}$

the terminal $T$ asks the code:

2. $T \rightarrow Cl : \text{code}$?

3. $Cl \rightarrow C : 1234$

4. $C \rightarrow T : ok$
Payment protocol

the terminal $T$ reads the credit card $C$:

1. $C \rightarrow T : \text{Data, } \{ \text{hash(Data)} \}_{\text{priv}(B)}$

the terminal $T$ asks the code:

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

the terminal $T$ requests authorisation the bank $B$:

5. $T \rightarrow B : \text{auth}$?
6. $B \rightarrow T : 4528965874123$
7. $T \rightarrow C : 4528965874123$
8. $C \rightarrow T : \{4528965874123\}_{K_{CB}}$
9. $T \rightarrow B : \{4528965874123\}_{K_{CB}}$
10. $B \rightarrow T : \text{ok}$
Initialement la sécurité été assurée par :

- cartes difficilement réplicables,
- secret des clefs et du protocole.
Faille sur la carte bleue

Initialement la sécurité était assurée par :
- cartes difficilement réplicables,
- secret des clés et du protocole.

Mais il y a des failles !
- faille cryptographique : les clés de 320 bits ne sont plus sûres,
- faille logique : pas de lien entre le code secret à 4 chiffres et l’authentification,
- faille matériel : réplicabilité des cartes.

→ “YesCard” fabriquées par Serge Humpich (1997).
La « YesCard »: Comment ca marche ?

Faille logique

1. $C \rightarrow T : Data, \{\text{hash}(Data)\}_{\text{priv}(B)}$
2. $T \rightarrow Cl : code$
3. $Cl \rightarrow C : 1234$
4. $C \rightarrow T : ok$
Faille logique

1. $C \rightarrow T : \text{Data}, \{\text{hash(Data)}\}_{\text{priv}(B)}$

2. $T \rightarrow Cl : \text{code?}$

3. $Cl \rightarrow C' : 2345$

4. $C' \rightarrow T : \text{ok}$
La « YesCard »: Comment ça marche ?

Faille logique

1. $C \rightarrow T : \text{Data, } \{\text{hash(Data)}\}_{\text{priv}(B)}$
2. $T \rightarrow C_l : \text{code?}$
3. $C_l \rightarrow C' : 2345$
4. $C' \rightarrow T : \text{ok}$

Remarque : il y a toujours quelqu’un à débiter.
→ ajout d’un faux chiffrement sur une fausse carte (Serge Humpich).
La « YesCard »: Comment ca marche ?

Faille logique

1. $C \rightarrow T : \{\text{hash}(\text{Data})\}_{\text{priv}(B)}$
2. $T \rightarrow Cl : \text{code}$?
3. $Cl \rightarrow C' : 2345$
4. $C' \rightarrow T : \text{ok}$

Remarque : il y a toujours quelqu’un à débiter.
→ ajout d’un faux chiffrement sur une fausse carte (Serge Humpich).

1. $C' \rightarrow T : \{\text{hash}(\text{XXX})\}_{\text{priv}(B)}$
2. $T \rightarrow Cl : \text{code}$?
3. $Cl \rightarrow C' : 0000$
4. $C' \rightarrow T : \text{ok}$
Outline of the talk

1. Introduction

2. Some examples of security protocols
   - Credit Card payment
   - Needham Schroeder protocol

3. How can we verify them?
   - How protocols can be attacked?
   - How protocols can be proved secure?

4. Conclusion
Needham-Schroeder’s Protocol (1978)

\[ \begin{align*}
A & \rightarrow B : \quad \{ \langle A, N_a \rangle \}_{\text{pub}(B)} \\
B & \rightarrow A : \quad \{ \langle N_a, N_b \rangle \}_{\text{pub}(A)} \\
A & \rightarrow B : \quad \{ N_b \}_{\text{pub}(B)}
\end{align*} \]
Needham-Schroeder’s Protocol (1978)

\[
\begin{align*}
A & \rightarrow B : \quad \{ \langle A, N_a \rangle \}_{\text{pub}(B)} \\
B & \rightarrow A : \quad \{ \langle N_a, N_b \rangle \}_{\text{pub}(A)} \\
A & \rightarrow B : \quad \{ N_b \}_{\text{pub}(B)}
\end{align*}
\]
Needham-Schroeder’s Protocol (1978)

\[
\begin{align*}
A & \rightarrow B : \{\langle A, N_a \rangle \}_{\text{pub}(B)} \\
B & \rightarrow A : \{\langle N_a, N_b \rangle \}_{\text{pub}(A)} \\
\bullet & A \rightarrow B : \{N_b\}_{\text{pub}(B)}
\end{align*}
\]
Needham-Schroeder’s Protocol (1978)

\[
A \rightarrow B : \{ \langle A, N_a \rangle \}_{\text{pub}(B)} \\
B \rightarrow A : \{ \langle N_a, N_b \rangle \}_{\text{pub}(A)} \\
A \rightarrow B : \{ N_b \}_{\text{pub}(B)}
\]
Needham-Schroeder’s Protocol (1978)

\[
A \rightarrow B : \quad \{ \langle A, N_a \rangle \}_{\text{pub}(B)} \\
B \rightarrow A : \quad \{ \langle N_a, N_b \rangle \}_{\text{pub}(A)} \\
A \rightarrow B : \quad \{ N_b \}_{\text{pub}(B)}
\]

Questions

- Is \( N_b \) secret between \( A \) and \( B \) ?
- When \( B \) receives \( \{ N_b \}_{\text{pub}(B)} \), does this message really come from \( A \)?
Needham-Schroeder’s Protocol (1978)

\[ A \rightarrow B : \{\langle A, N_a \rangle\}_{\text{pub}(B)} \]
\[ B \rightarrow A : \{\langle N_a, N_b \rangle\}_{\text{pub}(A)} \]
\[ A \rightarrow B : \{N_b\}_{\text{pub}(B)} \]

Questions

- Is \( N_b \) secret between \( A \) and \( B \)?
- When \( B \) receives \( \{N_b\}_{\text{pub}(B)} \), does this message really come from \( A \)?

Attack

An attack was found 17 years after its publication! [Lowe 96]
Example: Man in the middle attack

Agent \( A \)

Intruder \( I \)

Agent \( B \)

**Attack**
- involving 2 sessions in parallel,
- an honest agent has to initiate a session with \( I \).

\[
\begin{align*}
A \rightarrow B & : \{\langle A, N_a \rangle\}_{\text{pub}(B)} \\
B \rightarrow A & : \{\langle N_a, N_b \rangle\}_{\text{pub}(A)} \\
A \rightarrow B & : \{N_b\}_{\text{pub}(B)}
\end{align*}
\]
Example: Man in the middle attack

\[\{\langle A, N_a \rangle\}_{\text{pub}(I)} \quad \{\langle A, N_a \rangle\}_{\text{pub}(B)}\]

Agent A \quad Intruder I \quad Agent B

A \rightarrow B : \{\langle A, N_a \rangle\}_{\text{pub}(B)}
B \rightarrow A : \{\langle N_a, N_b \rangle\}_{\text{pub}(A)}
A \rightarrow B : \{N_b\}_{\text{pub}(B)}
Example: Man in the middle attack

\[ \{\langle A, N_a \rangle\}_{pub(I)} \rightarrow \{\langle N_a, N_b \rangle\}_{pub(A)} \]

Agent A

Intruder I

Agent B

\[ \{\langle A, N_a \rangle\}_{pub(B)} \rightarrow \{\langle N_a, N_b \rangle\}_{pub(A)} \]

\[ A \rightarrow B : \{\langle A, N_a \rangle\}_{pub(B)} \]

\[ B \rightarrow A : \{\langle N_a, N_b \rangle\}_{pub(A)} \]

\[ A \rightarrow B : \{N_b\}_{pub(B)} \]
Example: Man in the middle attack

\[
\begin{align*}
\{ \langle A, N_a \rangle \}_\text{pub}(I) & \\
\{ \langle N_a, N_b \rangle \}_\text{pub}(A) & \\
\{ N_b \}_\text{pub}(I) & \\
\{ \langle A, N_a \rangle \}_\text{pub}(B) & \\
\{ \langle N_a, N_b \rangle \}_\text{pub}(A) & \\
\{ N_b \}_\text{pub}(B) &
\end{align*}
\]

Agent A \quad \text{Intruder I} \quad \text{Agent B}

\[
\begin{align*}
A \rightarrow B & : \{ \langle A, N_a \rangle \}_\text{pub}(B) \\
B \rightarrow A & : \{ \langle N_a, N_b \rangle \}_\text{pub}(A) \\
A \rightarrow B & : \{ N_b \}_\text{pub}(B)
\end{align*}
\]
Example: Man in the middle attack

Agent A  Intruder I  Agent B

\(\{\langle A, N_a \rangle\}_{\text{pub}(I)}\)  \(\{\langle N_a, N_b \rangle\}_{\text{pub}(A)}\)  \(\{\langle A, N_a \rangle\}_{\text{pub}(B)}\)

\(\{N_b\}_{\text{pub}(I)}\)  \(\{\langle N_a, N_b \rangle\}_{\text{pub}(A)}\)  \(\{N_b\}_{\text{pub}(B)}\)

**Attack**
- the intruder knows \(N_b\),
- When B finishes his session (apparently with A), A has never talked with B.

\[\]
A \rightarrow B : \{\langle A, N_a \rangle\}_{\text{pub}(B)}
B \rightarrow A : \{\langle N_a, N_b \rangle\}_{\text{pub}(A)}
A \rightarrow B : \{N_b\}_{\text{pub}(B)}\]
Outline of the talk

1. Introduction

2. Some examples of security protocols
   - Credit Card payment
   - Needham Schroeder protocol

3. How can we verify them?
   - How protocols can be attacked?
   - How protocols can be proved secure?

4. Conclusion
accroître notre confiance dans les logiciels critiques
La recherche au LSV

→ accroître notre confiance dans les logiciels critiques

- **logiciel**: texte relativement long écrit dans un langage spécifique et qui sera **exécuté par un ordinateur**

- **critique**: une défaillance peut avoir des **conséquences désastreuses** en termes humains ou économiques
Ennemi public numéro 1: le bug ...
Ennemi public numéro 1: le bug …

… aussi connu sous le nom de bogue !
Dans la vie quotidienne !
Dans la vie quotidienne !

- MPlayer crashed. This shouldn't happen. It can be a bug in the MPlayer code _or_ in your drivers _or_ in your gcc version. If you think it's MPlayer's fault, please read DOCS/HTML/en/bugreports.html and follow the instructions there. We can't and won't help unless you provide this information when reporting a possible bug.
Ariane V - 4 juin 1996

Un crash après 40 secondes de vol dû ...

...
Un crash après 40 secondes de vol dû
... à un bug logiciel !

1. 189 vols réussis pour Ariane IV,
2. réutilisation du logiciel de
   lancement d’Ariane IV,
3. ajout du nécessaire pour la
   nouvelle fusée.

→ Le logiciel d’Ariane IV contenait un bug !
Perte de la sonde due ...
Perte de la sonde due à un problème d’unité de mesure !
Difficultés et enjeux de la vérification de programmes

→ une petite modification (quelques caractères) peut le transformer complètement.

Un besoin crucial de vérification

- pour des raisons économiques
  → Ariane 5, carte bancaire, . . .

- mais parfois il y a aussi des vies humaines en jeu
  → la machine Therac-25 dans les années 80
  → logiciels embarqués dans les voitures, les avions, . . .

- enjeux démocratiques
  → vote électronique
Tests

- à la main ou génération automatique;
- vérification d’un nombre fini de cas.
Comment fait-on ?

Tests

- à la main ou génération automatique;
- vérification d’un nombre fini de cas.

Accéder à l’infini: un rêve impossible?
Comment fait-on ?

Tests
- à la main ou génération automatique;
- vérification d’un nombre fini de cas.

∞ ∞ ∞ Accéder à l’infini: un rêve impossible ? ∞ ∞ ∞

Vérification (preuves formelles)
→ preuves mathématiques
- à la main ou à l’aide d’ordinateur;
- vérification de tous les cas possibles;
- plus difficile.
Comment vérifier ces programmes ?

Les mathématiques et l’informatique à la rescousse !
Les mathématiques et l’informatique à la rescousse !

Notre but:

1. faire des preuves mathématiques rigoureuses,
2. d’une façon automatique.

“Construire une machine à déteeter les bugs”
Comment vérifier ces programmes ?

Les mathématiques et l’informatique à la rescousse !

Notre but :

1. faire des preuves mathématiques rigoureuses,
2. d’une façon automatique.

“Construire une machine à détecter les bugs”

1936: une telle machine n’existe pas (Alan Turing)

... même dans le cas particulier des protocoles cryptographiques.
Mais alors, que faisons nous ?

Le problème n’a pas de solution ....
Mais alors, que faisons nous ?

Le problème n’a pas de solution .... mais seulement dans le cas général
Mais alors, que faisons nous ?

Le problème n’a pas de solution .... mais seulement dans le cas général

Différentes pistes:

- résoudre le problème dans de nombreux cas intéressants,
Mais alors, que faisons nous ?

Le problème n’a pas de solution .... mais seulement dans le cas général

Différentes pistes:

- résoudre le problème dans de nombreux cas intéressants,
- proposer des procédures approchées,

Exemple: si le vérificateur répond “oui” alors le logiciel est sûr, sinon on ne peut rien dire
Outline of the talk

1. Introduction

2. Some examples of security protocols
   - Credit Card payment
   - Needham Schroeder protocol

3. How can we verify them?
   - How protocols can be attacked?
   - How protocols can be proved secure?

4. Conclusion
Some specificities:

- protocol are executed in an **hostile** envrionment
  → a powerful attacker who controls the communication network

- **unbounded** number of sessions running concurrently

- the **cryptographic primitives** play an important role
  → we have to take them into account.
How cryptographic protocols can be attacked?

Breaking encryption

- Ciphertext-only attack,
- Known-plaintext attack, ...

Logical attack
Casser le chiffrement RSA

Les challenges RSA

- défis lancés par le laboratoire RSA Security
- récompenses importantes offertes
Casser le chiffrement RSA

Les challenges RSA
- défis lancés par le laboratoire RSA Security
- récompenses importantes offertes

<table>
<thead>
<tr>
<th>RSA-576</th>
<th>174 chiffres</th>
<th>réussi</th>
<th>2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-640</td>
<td>193 chiffres</td>
<td>réussi</td>
<td>2005</td>
</tr>
<tr>
<td>RSA-704</td>
<td>212 chiffres</td>
<td>non résolu – 30 000 dollars</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>RSA-2048</td>
<td>617 chiffres</td>
<td>non résolu – 200 000 dollars</td>
<td></td>
</tr>
</tbody>
</table>
Casser le chiffrement RSA

Les challenges RSA

- défis lancés par le laboratoire RSA Security
- récompenses importantes offertes

<table>
<thead>
<tr>
<th>RSA</th>
<th>Nombre de chiffres</th>
<th>Status</th>
<th>Année</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-576</td>
<td>174 chiffres</td>
<td>réussi</td>
<td>2003</td>
</tr>
<tr>
<td>RSA-640</td>
<td>193 chiffres</td>
<td>réussi</td>
<td>2005</td>
</tr>
<tr>
<td>RSA-704</td>
<td>212 chiffres</td>
<td>non résolu – 30 000 dollars</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>RSA-2048</td>
<td>617 chiffres</td>
<td>non résolu – 200 000 dollars</td>
<td></td>
</tr>
</tbody>
</table>

→ Ces challenges ont été retirés en 2007 !
Logical attacks

- can be mounted even assuming perfect cryptography,
  - replay attack, man-in-the middle attack, ...

- are numerous, see SPORE, Security Protocols Open REpository
  - [http://www.lsv.ens-cachan.fr/spore/](http://www.lsv.ens-cachan.fr/spore/)

- subtle and hard to detect by “eyeballing” the protocol

Examples:

- man in the middle attacks: *e.g.* Needham Schroeder protocol;

- replay attacks: electronic passport protocol (French version),
  electronic voting protocol (*e.g.* Helios).
Outline of the talk

1 Introduction

2 Some examples of security protocols
   - Credit Card payment
   - Needham Schroeder protocol

3 How can we verify them?
   - How protocols can be attacked?
   - How protocols can be proved secure?

4 Conclusion
Cryptographic models

Main features:

- Messages are bitstrings
- Protocols are programs that exchange messages
- **Real** algorithms for cryptographic primitives
- **Powerful attacker**: any probabilistic polynomial time Turing machine

→ quite realistic model
Cryptographic models

Main features:
- Messages are bitstrings
- Protocols are programs that exchange messages
- **Real** algorithms for cryptographic primitives
- **Powerful attacker**: any probabilisitic polynomial time Turing machine

→ quite realistic model

**Advantage**: Clear and quite strong security guarantee
**Drawback**: Proofs are difficult, tedious and error-prone.
Symbolic models

Main features:

- Messages are *abstracted by terms* (abstract objects)
- Protocols are programs that exchange messages
- Cryptographic primitives are *abstracted by function symbols*
- Idealized attacker: in particular, we have to describe what he can do.

→ very *abstract* model
Symbolic models

Main features:

- Messages are abstracted by terms (abstract objects)
- Protocols are programs that exchange messages
- Cryptographic primitives are abstracted by function symbols
- Idealized attacker: in particular, we have to describe what he can do.

→ very abstract model

**Advantage:** Security proofs are easier to do and they can be mechanized

**Drawback:** the security guarantees obtained are rather unclear.
Link between the two models

Computational soundness

Computational soundness aims to establish sufficient conditions under which results obtained using symbolic models imply security under computational models.

Seminal paper: Abadi & Rogaway, 2001

Many other papers have been obtained in this area.

A survey is available [Cortier et al., JAR 2010]
Symbolic model

Messages are abstracted by terms

- pairing \( \langle m_1, m_2 \rangle \),
- symmetric \( \text{senc}(m, k) \) and public key encryption \( \text{aenc}(m, \text{pub}(A)) \),
- signature \( \text{sign}(m, \text{priv}(A)) \).
Symbolic model

Messages are abstracted by terms

- pairing $\langle m_1, m_2 \rangle$,
- symmetric $\text{senc}(m, k)$ and public key encryption $\text{aenc}(m, \text{pub}(A))$,
- signature $\text{sign}(m, \text{priv}(A))$.

Presence of an idealized attacker

- may read, intercept and send messages,
- may build new messages following deduction rules (symbolic manipulation on terms).
Symbolic model

Messages are abstracted by terms

- pairing \( \langle m_1, m_2 \rangle \),
- symmetric \( \text{senc}(m, k) \) and public key encryption \( \text{aenc}(m, \text{pub}(A)) \),
- signature \( \text{sign}(m, \text{priv}(A)) \).

Presence of an idealized attacker

- may read, intercept and send messages,
- may build new messages following deduction rules (symbolic manipulation on terms).

Examples:

\[
\begin{array}{ccc}
  m & k & \text{senc}(m, k) \\
  \text{senc}(m, k) & k & \text{aenc}(m, \text{pub}(a)) \\
  & m & \text{priv}(a) \\
\end{array}
\]
Difficulties of the verification

Presence of an attacker ...
Difficulties of the verification

Presence of an attacker ...

who controls the communication network:

- may read every message sent on the network
- may intercept and send new messages
Difficulties of the verification

Presence of an attacker ...

who controls the communication network:
- may read every message sent on the network
- may intercept and send new messages

who has deduction capabilities
- encryption, decryption if he knows the decryption key,
- pairing, projection
Difficulties of the verification

Presence of an attacker ...

who controls the communication network:

- may read every message sent on the network
- may intercept and send new messages

who has deduction capabilities

- encryption, decryption if he knows the decryption key,
- pairing, projection

Secrecy problem for an unbounded number of sessions is undecidable.
Difficulties of the verification

Presence of an attacker ...

who controls the communication network:
- may read every message sent on the network
- may intercept and send new messages

who has deduction capabilities
- encryption, decryption if he knows the decryption key,
- pairing, projection

Secrecy problem for a fixed number of sessions is decidable.
Secrecy problem in presence of a passive attacker

Intruder deduction problem for a fixed inference system $\mathcal{I}$

**Input:** a finite set of ground terms $T$ (the knowledge of the attacker) and a ground term $s$ (the secret),

**Output:** Is $s$ deducible from $T$ in $\mathcal{I}$?
Secrecy problem in presence of a passive attacker

Intruder deduction problem for a fixed inference system $\mathcal{I}$

**Input:** a finite set of ground terms $T$ (the knowledge of the attacker) and a ground term $s$ (the secret),

**Output:** Is $s$ deducible from $T$ in $\mathcal{I}$?

**Example:** $T = \{\text{senc}(s_1, k_1); \text{senc}(s_2, k_2); \langle k_1, k_2 \rangle\}$ and $s = \langle s_1, s_2 \rangle$.

```
  x    y    〈x, y〉    〈x, y〉    x    y    senc(x, y)    y
  〈x, y〉    x    〈x, y〉    y    senc(x, y)    x
```

Stéphanie Delaune (LSV)  Providing solutions for more secure exchanges
Secrecy problem in presence of a passive attacker

Intruder deduction problem for a fixed inference system $\mathcal{I}$

**Input:** a finite set of ground terms $T$ (the knowledge of the attacker) and a ground term $s$ (the secret),

**Output:** Is $s$ deducible from $T$ in $\mathcal{I}$?

Example: $T = \{\text{senc}(s_1, k_1); \text{senc}(s_2, k_2); \langle k_1, k_2 \rangle\}$ and $s = \langle s_1, s_2 \rangle$.

\[
\begin{array}{cccc}
\langle x, y \rangle & \langle x, y \rangle & x & \text{senc}(x, y) \\
\langle x, y \rangle & \langle x, y \rangle & y & \text{senc}(x, y) \\
x & s & x & s \\
y & s & y & s
\end{array}
\]

**Results**

The intruder deduction problem is decidable in PTIME for the inference system given above (and some others).
Secrecy problem via constraint solving

\[ \rightarrow \text{ for a fixed number of sessions} \]

**Protocol rules**

\[
\begin{align*}
\text{in}(u_1); \text{out}(v_1) \\
\text{in}(u_2); \text{out}(v_2) \\
& \quad \cdots \\
\text{in}(u_n); \text{out}(v_n)
\end{align*}
\]

**Constraint System**

\[
C = \begin{cases}
T_0 \vdash u_1 \\
T_0, v_1 \vdash u_2 \\
\quad \cdots \\
T_0, v_1, \ldots, v_n \vdash s
\end{cases}
\]
Secrecy problem via constraint solving

for a fixed number of sessions

Protocol rules

\[
\begin{align*}
in(u_1); & \text{ out}(v_1) \\
in(u_2); & \text{ out}(v_2) \\
      & \cdots \\
in(u_n); & \text{ out}(v_n)
\end{align*}
\]

Constraint System

\[
\mathcal{C} = \left\{ \begin{array}{l}
\mathcal{T}_0 \vdash u_1 \\
\mathcal{T}_0, v_1 \vdash u_2 \\
      & \cdots \\
\mathcal{T}_0, v_1, \ldots, v_n \vdash s
\end{array} \right\}
\]

Solution of a constraint system in \( \mathcal{I} \)

A substitution \( \sigma \) such that

\[
\text{for every } \mathcal{T} \vdash u \in \mathcal{C}, \ u\sigma \text{ is deducible from } \mathcal{T}\sigma \text{ in } \mathcal{I}.
\]
Example: Needham-Schroeder’s protocol

**A(a, l) and B(b) (running in parallel)**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>({\langle n_a, x_{n_b} \rangle}_{\text{pub}(a)})</td>
<td>(\text{out}({\langle a, n_a \rangle}_{\text{pub}(l)}))</td>
</tr>
<tr>
<td>({\langle y_a, y_{n_a} \rangle}_{\text{pub}(b)})</td>
<td>(\text{out}({\langle y_{n_a}, n_b \rangle}_{\text{pub}(y_a)}))</td>
</tr>
</tbody>
</table>
Example: Needham-Schroeder’s protocol

\[ A(a, I) \text{ and } B(b) \text{ (running in parallel)} \]

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>in({\langle n_a, x_n_b \rangle }_{\text{pub}(a)}</td>
<td>out({\langle a, n_a \rangle }_{\text{pub}(I)})</td>
</tr>
<tr>
<td>3</td>
<td>in({\langle n_a, x_n_b \rangle }_{\text{pub}(a)}</td>
<td>out({\langle x_n_b \rangle }_{\text{pub}(I)})</td>
</tr>
<tr>
<td>2</td>
<td>in({\langle y_a, y_n_a \rangle }_{\text{pub}(b)}</td>
<td>out({\langle y_n_a, n_b \rangle }_{\text{pub}(y_a)})</td>
</tr>
</tbody>
</table>
Example: Needham-Schroeder’s protocol

\[ A(a, I) \text{ and } B(b) \text{ (running in parallel)} \]

1. \[ \text{out}(\{ \langle a, n_a \rangle \}_{\text{pub}(I)} \) \]

3. \[ \text{in}(\{ \langle n_a, x_n_b \rangle \}_{\text{pub}(a)} \) ; \text{out}(\{ x_n_b \}_{\text{pub}(I)} \) \]

2. \[ \text{in}(\{ \langle y_a, y_n_a \rangle \}_{\text{pub}(b)} \) ; \text{out}(\{ \langle y_n_a, n_b \rangle \}_{\text{pub}(y_a)} \) \]

Constraints System
### Example: Needham-Schroeder’s protocol

**A(a, I) and B(b) (running in parallel)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>out({\langle a, n_a \rangle }_{pub(I)})</td>
</tr>
<tr>
<td>3</td>
<td>in({\langle n_a, x_{n_b} \rangle }<em>{pub(a)}) ; out({x</em>{n_b} }_{pub(I)})</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>in({\langle y_a, y_{n_a} \rangle }<em>{pub(b)}) ; out({\langle y</em>{n_a}, n_b \rangle }_{pub(y_a)})</td>
<td></td>
</tr>
</tbody>
</table>

**Constraints System**

\[ T_0, \{ a, n_a \}_{pub(I)} \]
Example: Needham-Schroeder’s protocol

\[ A(a, I) \text{ and } B(b) \text{ (running in parallel)} \]

1. \[ \text{in} \left( \left\{ \langle n_a, x_n \rangle \right\}_{\text{pub}(a)} \right) ; \text{ out} \left( \left\{ \langle a, n_a \rangle \right\}_{\text{pub}(I)} \right) \]

2. \[ \text{in} \left( \left\{ \langle y_a, y_n \rangle \right\}_{\text{pub}(b)} \right) ; \text{ out} \left( \left\{ \langle y_n, n_b \rangle \right\}_{\text{pub}(y_a)} \right) \]

Constraints System

\[ T_0, \left\{ a, n_a \right\}_{\text{pub}(I)} \vdash \left\{ y_a, y_n \right\}_{\text{pub}(b)} \]
Example: Needham-Schroeder’s protocol

**A(a, l) and B(b) (running in parallel)**

1. \( \text{out}(\{\langle a, n_a \rangle\}_{\text{pub}(l)}) \)
2. \( \text{in}(\{\langle y_a, y_{na} \rangle\}_{\text{pub}(b)}) \); \( \text{out}(\{\langle y_{na}, n_b \rangle\}_{\text{pub}(y_a)}) \)
3. \( \text{in}(\{\langle n_a, x_{nb} \rangle\}_{\text{pub}(a)}) \); \( \text{out}(\{\langle x_{nb} \rangle\}_{\text{pub}(l)}) \)

**Constraints System**

\[
T_0, \quad \{a, n_a\}_{\text{pub}(l)} \vdash \{y_a, y_{na}\}_{\text{pub}(b)}
\]

\[
T_0, \quad \{a, n_a\}_{\text{pub}(l)}, \quad \{y_{na}, n_b\}_{\text{pub}(y_a)}
\]
Example: Needham-Schroeder’s protocol

A(a, l) and B(b) (running in parallel)

1

out(\{\langle a, n_a \rangle \}_\text{pub}(l))

3

in(\{\langle n_a, x_{n_b} \rangle \}_\text{pub}(a)) ; out(\{x_{n_b} \}_\text{pub}(l))

2

in(\{\langle y_a, y_{n_a} \rangle \}_\text{pub}(b)) ; out(\{\langle y_{n_a}, n_b \rangle \}_\text{pub}(y_a))

Constraints System

\[ T_0, \{a, n_a\}_\text{pub}(l) \vdash \{y_a, y_{n_a} \}_\text{pub}(b) \]

\[ T_0, \{a, n_a\}_\text{pub}(l), \{y_{n_a}, n_b \}_\text{pub}(y_a) \vdash \{n_a, x_{n_b} \}_\text{pub}(a) \]
Example: Needham-Schroeder’s protocol

\[A(a, I) \text{ and } B(b) \text{ (running in parallel)}\]

1. \(\text{out}([\langle a, n_a \rangle]_{\text{pub}(I)})\)
2. \(\text{in}([\langle y_a, y_n_a \rangle]_{\text{pub}(b)})\)
3. \(\text{in}([\langle n_a, x_n_b \rangle]_{\text{pub}(a)})\); \(\text{out}([\langle x_n_b \rangle]_{\text{pub}(I)})\)

Constraints System

\[T_0, \{a, n_a\}_{\text{pub}(I)} \vdash \{y_a, y_n_a\}_{\text{pub}(b)}\]
\[T_0, \{a, n_a\}_{\text{pub}(I)}, \{y_n_a, n_b\}_{\text{pub}(y_a)} \vdash \{n_a, x_n_b\}_{\text{pub}(a)}\]
\[T_0, \{a, n_a\}_{\text{pub}(I)}, \{y_n_a, n_b\}_{\text{pub}(y_a)}, \{x_n_b\}_{\text{pub}(I)}\]

Stéphanie Delaune (LSV)
Providing solutions for more secure exchanges
Example: Needham-Schroeder’s protocol

A(a, l) and B(b) (running in parallel)

1  out(\{\langle a, na \rangle \}_{pub(l)})

3  in(\{\langle na, xn_b \rangle \}_{pub(a)}) ;  out(\{x_{n_b} \}_{pub(l)})

2  in(\{\langle y_a, yn_a \rangle \}_{pub(b)}) ;  out(\{\langle yn_a, nb \rangle \}_{pub(y_a)})

Constraints System

\[ T_0, \quad \{a, na\}_{pub(l)} \vdash \{y_a, yn_a\}_{pub(b)} \]

\[ T_0, \quad \{a, na\}_{pub(l)}, \quad \{yn_a, nb\}_{pub(y_a)} \vdash \{na, xn_b\}_{pub(a)} \]

\[ T_0, \quad \{a, na\}_{pub(l)}, \quad \{yn_a, nb\}_{pub(y_a)}, \quad \{xn_b\}_{pub(l)} \vdash nb \]
Example: Needham-Schroeders’s protocol

\( A(a, I) \) and \( B(b) \) (running in parallel)

1
\[
\text{out}\left(\{\langle a, n_a \rangle\}_\text{pub}(I)\}\right)
\]

3
\[
\text{in}\left(\{\langle n_a, x_{n_b} \rangle\}_\text{pub}(a)\}\right) \; ; \; \text{out}\left(\{x_{n_b}\}_\text{pub}(I)\}\right)
\]

2
\[
\text{in}\left(\{\langle y_a, y_{n_a} \rangle\}_\text{pub}(b)\}\right) \; ; \; \text{out}\left(\{\langle y_{n_a}, n_b \rangle\}_\text{pub}(y_a)\}\right)
\]

Constraints System

\[
\begin{align*}
T_0, \quad \{a, n_a\}_\text{pub}(I) & \vdash \{y_a, y_{n_a}\}_\text{pub}(b) \\
T_0, \quad \{a, n_a\}_\text{pub}(I), \quad \{y_{n_a}, n_b\}_\text{pub}(y_a) & \vdash \{n_a, x_{n_b}\}_\text{pub}(a) \\
T_0, \quad \{a, n_a\}_\text{pub}(I), \quad \{y_{n_a}, n_b\}_\text{pub}(y_a), \quad \{x_{n_b}\}_\text{pub}(I) & \vdash n_b
\end{align*}
\]

Solution
\[
\sigma = \{y_a \mapsto a, \; y_{n_a} \mapsto n_a, \; x_{n_b} \mapsto n_b\}
\]
Decision procedure

There exists an algorithm (actually a set of simplification rules) to decide whether such kind of constraint systems have a solution or not.
There exists an algorithm (actually a set of simplification rules) to decide whether such kind of constraint systems have a solution or not.

Main idea of the procedure:
Outil de vérification AVISPA

Outil disponible en ligne: http://www.avispa-project.org/

→ Projet Européen (France, Italie, Allemagne, Suisse)
Outline of the talk

1. Introduction

2. Some examples of security protocols
   - Credit Card payment
   - Needham Schroeder protocol

3. How can we verify them?
   - How protocols can be attacked?
   - How protocols can be proved secure?

4. Conclusion
Conclusion

Cryptographic protocols

- numerous, various security goals
- can be attacked even if the primitives are secure
  \[\text{http://www.lsv.ens-cachan.fr/spore/}\]

How to verify them?

- modelling the protocol, the security properties
- manually / automatically
  \[\text{the problem is undecidable in general (some tools exist)}\]

It remains a lot to do

- modelling security properties is a difficult task
- does a suitable E-voting protocol exist?
- take into account the algebraic properties of the primitives
- analyse the source code of the protocol instead of its specification