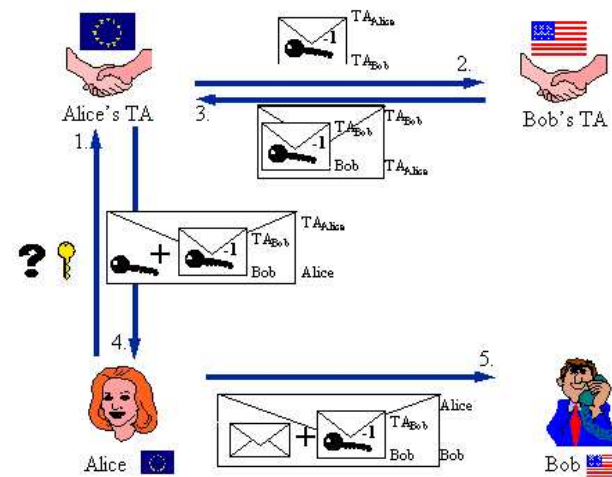


# Une fois qu'on n'a pas trouvé de preuve, comment le faire comprendre à un assistant de preuve?

Jean Goubault-Larrecq

<http://www.lsv.ens-cachan.fr/~goubault/>



Projet RNTL EVA

ACI VERNAM

ACI jeunes chercheurs “Sécurité info., protocoles crypto., et détection d'intrusions”.

## 1. La question abordée ici.

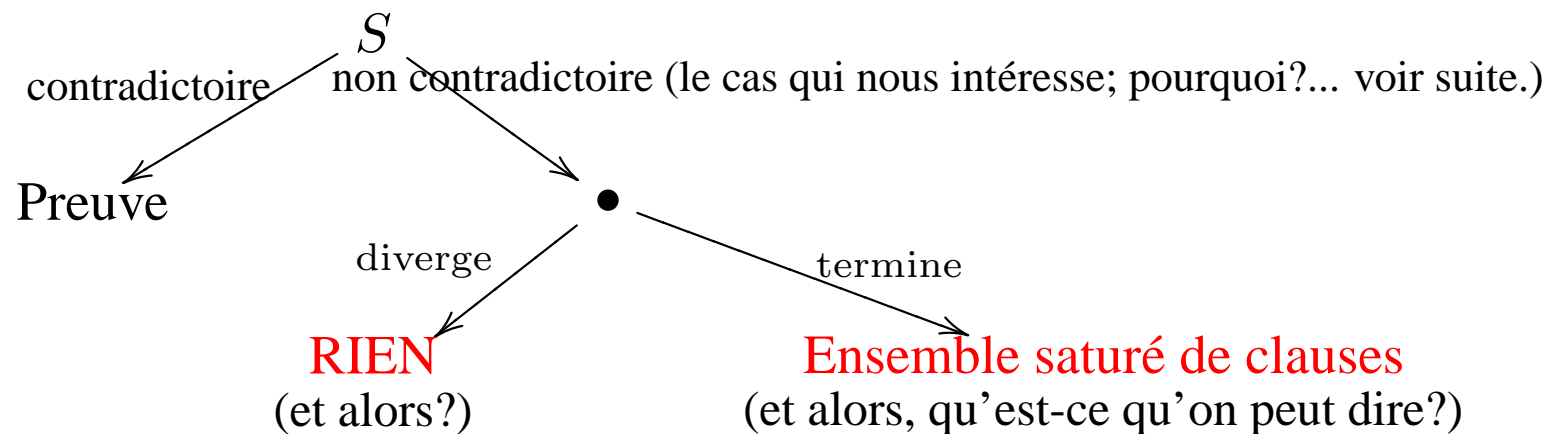
2. Les protocoles cryptographiques.
3. Modélisation en clauses de Horn.
4. Qu'est-ce qu'une preuve de sécurité?
5. Comment trouver une preuve de sécurité?
6. Qu'est-ce qu'une preuve de sécurité, constructivement?
7. Comment vérifier une preuve de sécurité (constructive)?
8. Conclusion.

## La question abordée ici

Soit  $S$  un ensemble de clauses (de Horn, du premier ordre).

Vérifier que  $S$  est non contradictoire est

- indécidable.
- co-r.e.: il existe des procédures (e.g., résolution [Robinson65]) qui
  - \* terminent sur “contradictoire” si  $S$  l’est;
  - \* ne terminent pas, ou terminent sur “non contradictoire” sinon.



1. La question abordée ici.
- 2. Les protocoles cryptographiques.**
3. Modélisation en clauses de Horn.
4. Qu'est-ce qu'une preuve de sécurité?
5. Comment trouver une preuve de sécurité?
6. Qu'est-ce qu'une preuve de sécurité, constructivement?
7. Comment vérifier une preuve de sécurité (constructive)?
8. Conclusion.



## Pourquoi cette question: les protocoles cryptographiques

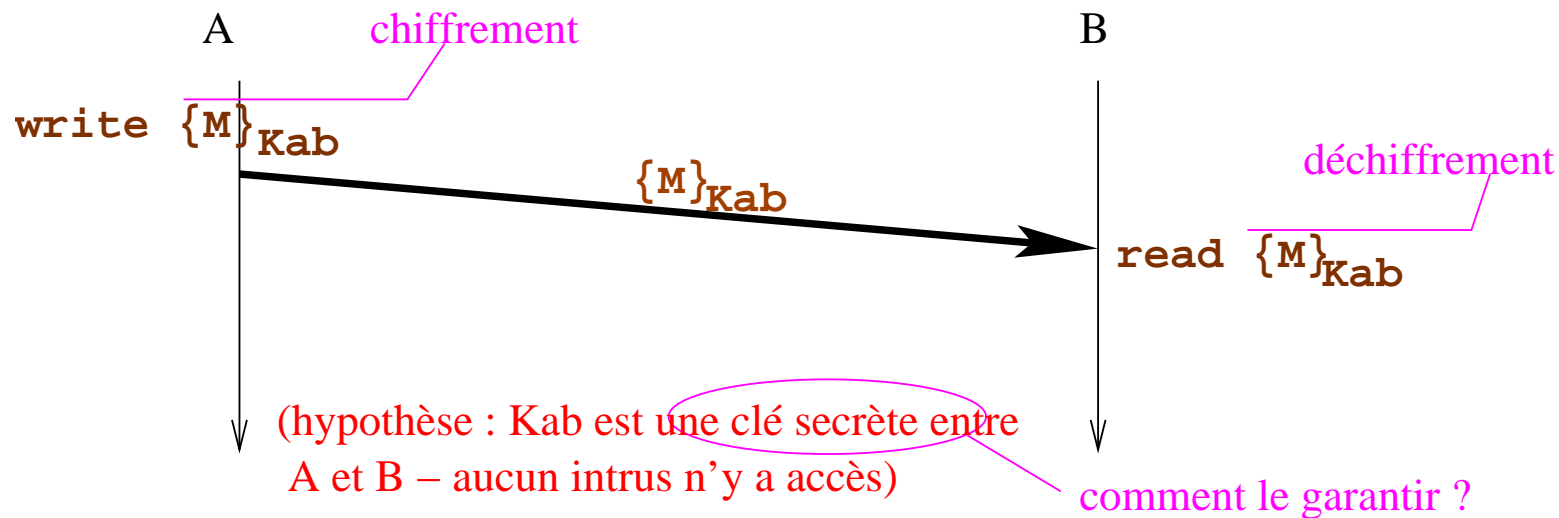
Besoin croissant en sécurité forte : cartes à puce, sécurité bancaire, commerce électronique, réseaux sécurisés ...



- secret** :  $M$  est secret si aucun intrus ne peut émettre  $M$ ;
- authenticité** : le seul processus qui peut fabriquer  $M$  est  $A$ ;
- fraîcheur** : le message  $M$  a été fabriqué récemment;
- non-duplication** : le message  $M$  ne peut être reçu qu'une fois (factures);
- non-répudiation** :  $A$  ne peut pas nier avoir émis  $M$  (commandes).

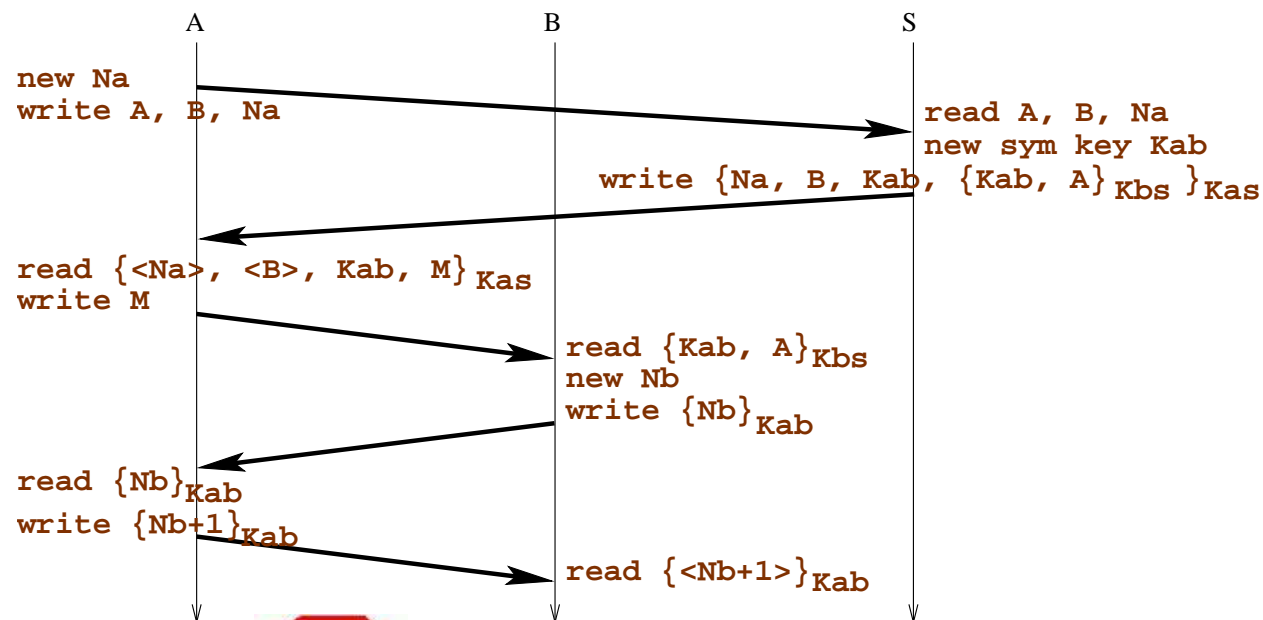
## La cryptographie ne suffit pas !

Même en utilisant des algorithmes de chiffrements parfaits (incassables), il n'est pas si facile de préserver le **secret** ou l'**authenticité** des messages :

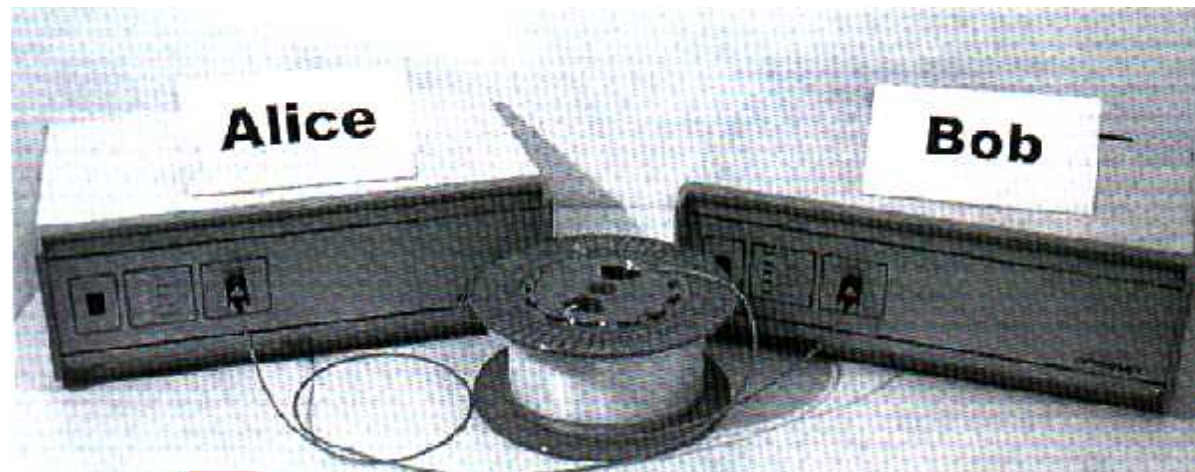
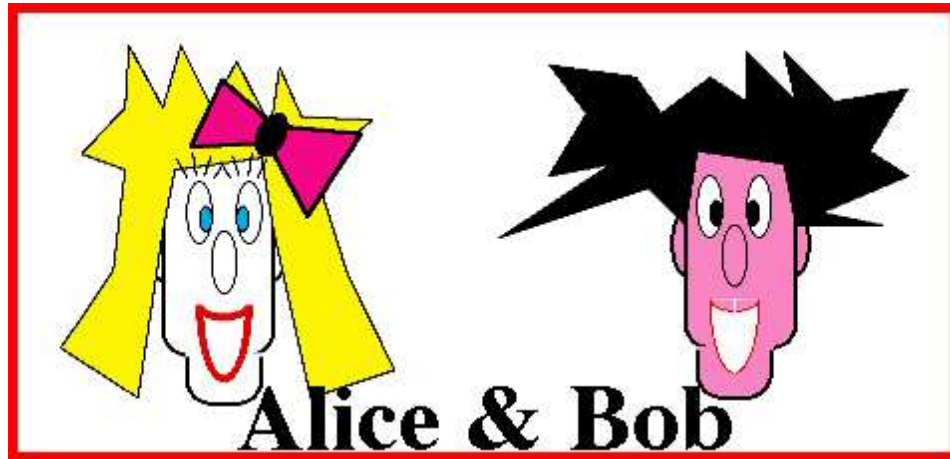


## Ex : Needham-Schroeder à clés symétriques

1.  $A \longrightarrow S : A, B, N_a$
2.  $S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3.  $A \longrightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4.  $B \longrightarrow A : \{N_b\}_{K_{ab}}$
5.  $A \longrightarrow B : \{N_b + 1\}_{K_{ab}}$

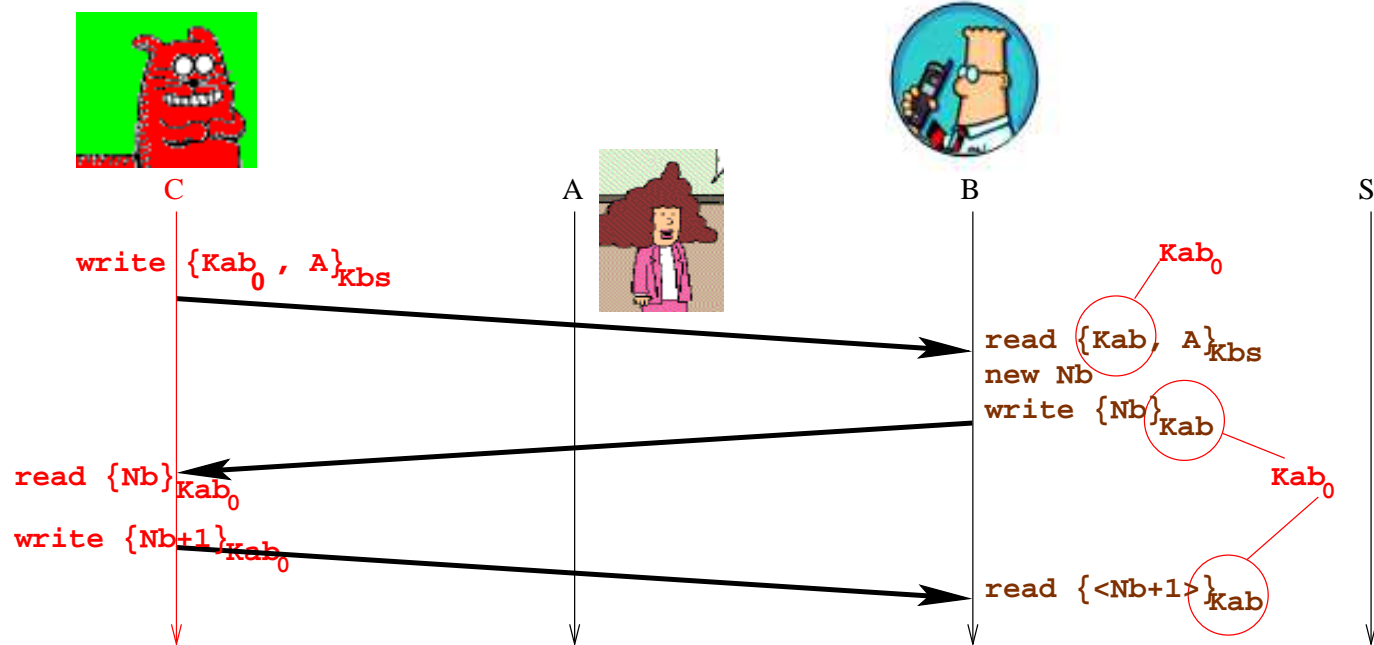


## Qui sont Alice et Bob?



## Une attaque

$C$  rejoue un vieux  $\{Kab_0, A \mid Kbs\}$  — suffisamment vieux pour avoir réussi à récupérer  $Kab_0$ .



1. La question abordée ici.
2. Les protocoles cryptographiques.
- 3. Modélisation en clauses de Horn.**
4. Qu'est-ce qu'une preuve de sécurité?
5. Comment trouver une preuve de sécurité?
6. Qu'est-ce qu'une preuve de sécurité, constructivement?
7. Comment vérifier une preuve de sécurité (constructive)?
8. Conclusion.

# Modélisation en clauses de Horn (Prolog pur)

## 1. Les capacités de l'intrus.

---

$\text{knows}(\{M\}_K) \Leftarrow \text{knows}(M), \text{knows}(K)$  (l'intrus sait chiffrer

$\text{knows}(M) \Leftarrow \text{knows}(\{M\}_{k(\text{sym}, X)}),$   
 $\text{knows}(k(\text{sym}, X))$  ...et déchiffrer [cas symétrique])

$\text{knows}([])$  (l'intrus sait fabriquer

$\text{knows}(M_1 :: M_2) \Leftarrow \text{knows}(M_1), \text{knows}(M_2)$  toutes les listes de messages connus)

$\text{knows}(M_1) \Leftarrow \text{knows}(M_1 :: M_2)$  (l'intrus peut lire les premières

$\text{knows}(M_2) \Leftarrow \text{knows}(M_1 :: M_2)$  et secondes composantes de chaque paire)

$\text{knows}(\text{suc}(M)) \Leftarrow \text{knows}(M)$  (l'intrus peut ajouter

$\text{knows}(M) \Leftarrow \text{knows}(\text{suc}(M))$  et retrancher un)

## 2. Les clauses du protocole—sessions courantes (à la Blanchet)

1.  $A \longrightarrow S : A, B, N_a \text{ knows}([a, b, \text{na}([a, b])])$

---

1.  $A \longrightarrow S : A, B, N_a$   
 2.  $S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$        $\text{knows} \left( \begin{array}{l} \{[N_a, B, k_{ab}, \\ \{[k_{ab}, A]\}_{k(\text{sym}, [B, s])}] \}_{k(\text{sym}, [A, s])} \end{array} \right) \Leftarrow \text{knows}([A, B, N_a])$   
( $k_{ab} \equiv k(\text{sym}, \text{cur}(A, B, N_a))$ )

---

2.  $S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$        $\text{knows}(M) \Leftarrow \text{knows}(\{[\text{na}([a, b]), b, K_{ab}, M]\}_{k(\text{sym}, [a, s])})$   
 $\text{a\_key}(K_{ab}) \Leftarrow \text{knows}(\{[\text{na}([a, b]), b, K_{ab}, M]\}_{k(\text{sym}, [a, s])})$   
 3.  $A \longrightarrow B : \{K_{ab}, A\}_{K_{bs}}$

---

3.  $A \longrightarrow B : \{K_{ab}, A\}_{K_{bs}}$        $\text{knows}(\{\text{nb}(K_{ab}, A, B)\}_{K_{ab}}) \Leftarrow \text{knows}(\{[K_{ab}, A]\}_{k(\text{sym}, [B, s])})$   
 4.  $B \longrightarrow A : \{N_b\}_{K_{ab}}$



- 
4.  $B \longrightarrow A : \{N_b\}_{K_{ab}}$
  5.  $A \longrightarrow B : \{N_b + 1\}_{K_{ab}}$
- $$\text{knows}(\{\text{suc}(N_b)\}_{K_{ab}}) \Leftarrow \text{knows}(\{N_b\}_{K_{ab}})$$

### 3. Les clauses du protocole—les vieilles sessions

---

1.  $A \rightarrow S : A, B, N_a$
  2.  $S \rightarrow A : \{N_a, B, K_{ab},$   
 $\{K_{ab}, A\}_{K_{bs}}$   
 $\}_{K_{as}}$
- $$\text{knows} \left( \begin{array}{l} \{[N_a, B, k_{ab}, \\ [k_{ab}, A]\}_{k(\text{sym}, [B, s])}, \\ ]\}_{k(\text{sym}, [A, s])} \end{array} \right) \Leftarrow \text{knows}([A, B, N_a])$$
- $$(k_{ab} \equiv k(\text{sym}, \text{prev}(A, B, N_a)))$$

## 4. Connaissances initiales de l'intrus

---

$\text{agent}(a)$                        $\text{agent}(b)$

$\text{agent}(s)$                        $\text{agent}(i)$

$\text{knows}(X) \Leftarrow \text{agent}(X)$

$\text{knows}(k(\text{pub}, X))$

$\text{knows}(k(\text{prv}, i))$

$\text{knows}(k(\text{sym}, \text{prev}(A, B, N_a)))$

(les clés des sessions  
précédentes sont compromises)

## 5. Requêtes de sécurité

---

$\perp \Leftarrow \text{knows}(\text{k}(\text{sym}, \text{cur}(\text{a}, \text{b}, N_a)))$

l'intrus peut-il fabriquer  $K_{ab}$

telle que fabriquée par  $S$ ?

$\perp \Leftarrow \text{knows}(K_{ab}), \text{a\_key}(K_{ab})$

... telle que reçue par  $A$ ?

$\perp \Leftarrow \text{knows}(\{\text{suc}(\text{nb}(K_{ab}, A, B))\}_{K_{ab}}), \text{knows}(K_{ab})$

... telle que reçue par  $B$ ?

1. La question abordée ici.
2. Les protocoles cryptographiques.
3. Modélisation en clauses de Horn.
- 4. Qu'est-ce qu'une preuve de sécurité?**
5. Comment trouver une preuve de sécurité?
6. Qu'est-ce qu'une preuve de sécurité, constructivement?
7. Comment vérifier une preuve de sécurité (constructive)?
8. Conclusion.

## Preuves de sécurité = absence de preuve

---

Une **preuve du faux**, c'est une **attaque**.

... c'est-à-dire une façon d'exécuter les clauses 1.–5.  
qui aboutit à ce que l'intrus connaisse une donnée sensible, ici.

**Thèse de Selinger:** Preuve de sécurité  $\equiv$  **absence** de preuve du faux.

# Démo 1



Si vous voyez ce transparent,  
demandez donc à l'orateur  
de faire plutôt tourner l'outil h1  
pour trouver les attaques sur  
Needham-Schroeder à clés symétriques!

Si l'orateur oublie:  
noter que ç a trouve une attaque sur  $B$ ,  
mais surtout, moins évident... qu'il n'y a pas d'attaque ni sur  $A$  ni sur  $S$ .

1. La question abordée ici.
2. Les protocoles cryptographiques.
3. Modélisation en clauses de Horn.
4. Qu'est-ce qu'une preuve de sécurité?
- 5. Comment trouver une preuve de sécurité?**
6. Qu'est-ce qu'une preuve de sécurité, constructivement?
7. Comment vérifier une preuve de sécurité (constructive)?
8. Conclusion.

## Démonstration automatique

---

⇒ Méthode de preuve:

Lancer un démonstrateur automatique de théorèmes (SPASS, Otter, Vampire, Waldmeister, Bliksem, ...) sur les clauses définissant le protocole.

Si contradiction trouvée, **attaque possible**.

Si le démonstrateur termine sans trouver de contradiction, **pas d'attaque**.

(Yes!)

Si le démonstrateur ne termine pas, ben, heu...

...c'est quand même ce qui se passe assez souvent...

Note: Blanchet utilise une stratégie de résolution ad hoc en deux étapes, qui termine souvent.



## Abstraction

---

Idée de base: transformer l'ensemble de clauses  $S$  de départ en un ensemble  $S'$  tel que:

- $S'$  est dans une sous-classe **décidable**.

...j'aime bien  $\mathcal{H}_1$  [NielsonNielsonSeidl02] personnellement.

- $S'$  implique  $S$  logiquement.

...ainsi, si  $S'$  n'est pas contradictoire,  $S$  non plus.

Magnifique, ça existe!

Le précurseur est [FrühwirthShapiroVardiYardeni91].

C'est indépendant de tout domaine d'application...

## Classe $\mathcal{H}_1$ et abstraction canonique

---

Clauses de  $\mathcal{H}_1$ :

$$P(X) \Leftarrow body \quad \text{ou} \quad P(f(X_1, \dots, X_n)) \Leftarrow body$$

Décidable

DEXPTIME-complet.

... par techniques ad hoc [NielsonNielsonSeidl02]

... par résolution ordonnée avec sélection [Goubault-Larrecq03]

Définit exactement les **langages rationnels d'arbres**.

... dans un langage beaucoup plus expressif que les automates d'arbres,

même les automates alternants,

et bidirectionnels,

... correspond aux contraintes ensemblistes définies

avec compréhensions,

même non linéaires.

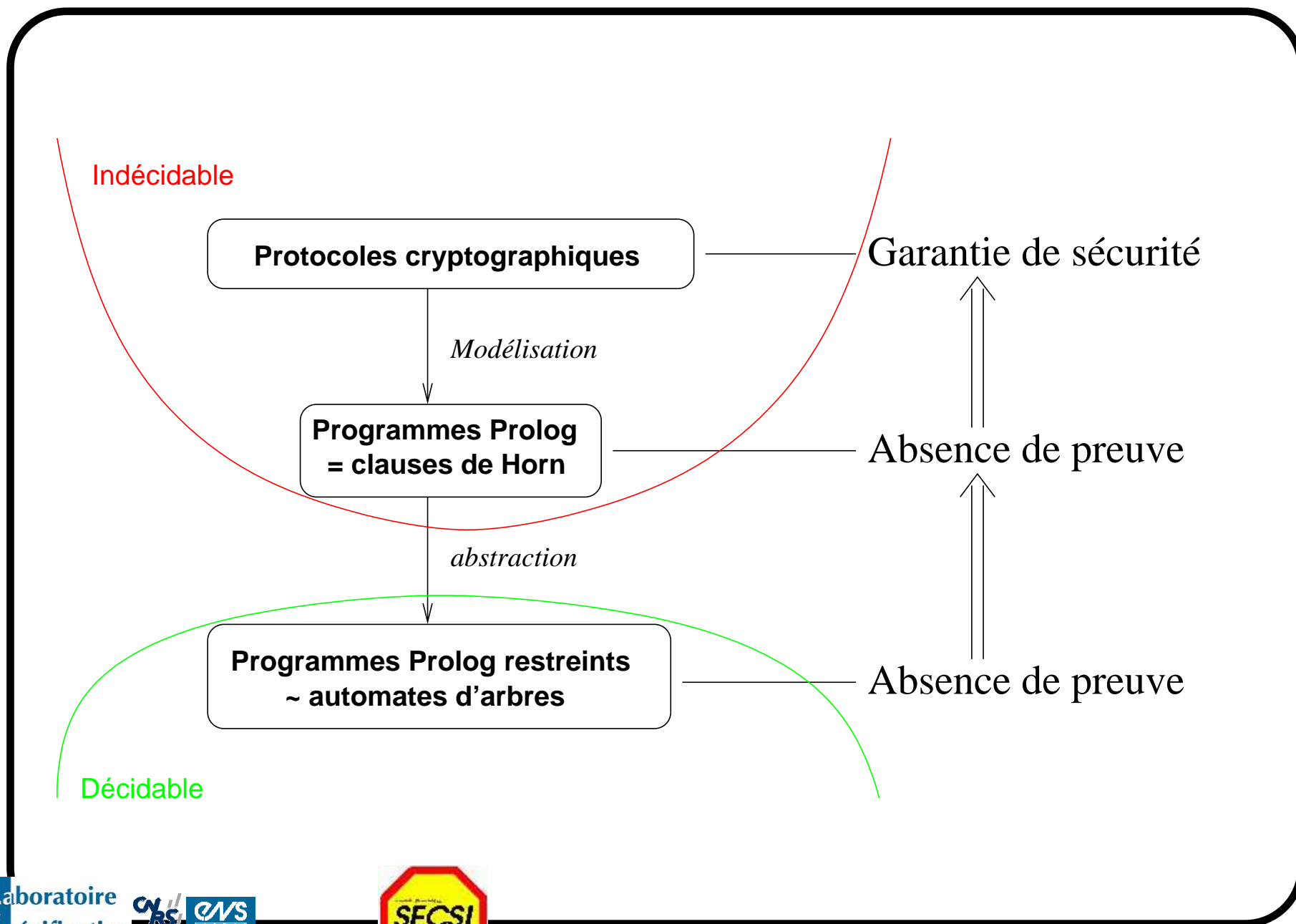
Et ..... toutes les clauses 1. (intrus) sont dans  $\mathcal{H}_1$ .

# Abstraction canonique: nommer les sous-termes

$$\text{knows} \left( \begin{array}{l} \{ [N_a, B, k(\text{sym}, \text{cur}(A, B, N_a)), \\ \{ [k(\text{sym}, \text{cur}(A, B, N_a)), A] \}_{k(\text{sym}, [B, s])} \\ ] \}_{k(\text{sym}, [A, s])} \end{array} \right) \Leftarrow \text{knows}([A, B, N_a])$$

→

$$\begin{aligned} q_{15}(g(A, B, N_a)) &\Leftarrow \text{knows}([A, B, N_a]) \\ q_{18}(N_a) &\Leftarrow q_{15}(g(A, B, N_a)) & q_{20}(B) &\Leftarrow q_{15}(g(A, B, N_a)) \\ q_{31}(A) &\Leftarrow q_{15}(g(A, B, N_a)) & q_{24}(\text{sym}) &\Leftarrow q_{15}(g(A, B, N_a)) \\ q_{27}([]) &\Leftarrow q_{15}(g(A, B, N_a)) & q_{34}(s) &\Leftarrow q_{15}(g(A, B, N_a)) \\ q_{25}(\text{cur}(A, B, N_a)) &\Leftarrow q_{15}(g(A, B, N_a)) & q_{22}(k(X_1, X_2)) &\Leftarrow q_{24}(X_1), q_{25}(X_2) \\ q_{30}(A :: X_2) &\Leftarrow q_{31}(A), q_{27}(X_2) & q_{28}(X_1 :: X_2) &\Leftarrow q_{22}(X_1), q_{30}(X_2) \\ q_{33}(X_1 :: X_2) &\Leftarrow q_{34}(X_1), q_{27}(X_2) & q_{32}(B :: X_2) &\Leftarrow q_{20}(B), q_{33}(X_2) \\ q_{29}(k(X_1, X_2)) &\Leftarrow q_{24}(X_1), q_{32}(X_2) & q_{26}(\{X_1\}X_2) &\Leftarrow q_{28}(X_1), q_{29}(X_2) \\ q_{23}(X_1 :: X_2) &\Leftarrow q_{26}(X_1), q_{27}(X_2) & q_{21}(X_1 :: X_2) &\Leftarrow q_{22}(X_1), q_{23}(X_2) \\ q_{19}(B :: X_2) &\Leftarrow q_{20}(B), q_{21}(X_2) & q_{16}(N_a :: X_2) &\Leftarrow q_{18}(N_a), q_{19}(X_2) \\ q_{35}(A :: X_2) &\Leftarrow q_{31}(A), q_{33}(X_2) & q_{17}(k(X_1, X_2)) &\Leftarrow q_{24}(X_1), q_{35}(X_2) \\ \text{knows}(\{X_1\}X_2) &\Leftarrow q_{16}(X_1), q_{17}(X_2) \end{aligned}$$



1. La question abordée ici.
2. Les protocoles cryptographiques.
3. Modélisation en clauses de Horn.
4. Qu'est-ce qu'une preuve de sécurité?
5. Comment trouver une preuve de sécurité?
- 6. Qu'est-ce qu'une preuve de sécurité, constructivement?**
7. Comment vérifier une preuve de sécurité (constructive)?
8. Conclusion.

## Preuves de sécurité = absence de preuve

---

Une **preuve du faux**, c'est une **attaque**.

... c'est-à-dire une façon d'exécuter les clauses 1.–5.  
qui aboutit à ce que l'intrus connaisse une donnée sensible, ici.

**Thèse de Selinger:** Preuve de sécurité  $\equiv$  **absence** de preuve du faux.

## Preuves de sécurité (constructivement) = modèles

---

**Thèse de Selinger:** Preuve de sécurité  $\equiv$  absence de preuve du faux.

[Selinger'01], *Models for an Adversary-Centric Protocol Logic*

1st LACPV, JGL, ed., 2001.

Constructivement, une absence de preuve, c'est un modèle.

C'est le théorème de complétude de la logique du premier ordre [Gödel1930].

## Modèles (finis)

**Exemple [Selinger01]:** preuve de Needham-Schroeder-Lowe par:

$k$	$W$	$K$	$U$	$N$	$S$
$\{W\}_k$	$K$	$K$	$U$	$U$	$U$
$\{K\}_k$	$K$	$K$	$U$	$U$	$U$
$\{U\}_k$	$U$	$K$	$U$	$U$	$U$
$\{N\}_k$	$U$	$U$	$U$	$U$	$U$
$\{S\}_k$	$U$	$U$	$U$	$U$	$U$

$K = \text{connu}$

$U = \text{inconnu}$

$W = \text{connu},$   
d'inverse connu

etc.

Le modèle est un **invariant** sur toute exécution du protocole, qui valide toutes les clauses, y compris les requêtes de sécurité.

...ex:  $\{U\}_K = K$ : chiffrer une donnée inconnue avec une clé connue  
produit un message (possiblement) connu.

**Problème** laissé ouvert par Selinger: comment trouver le modèle?



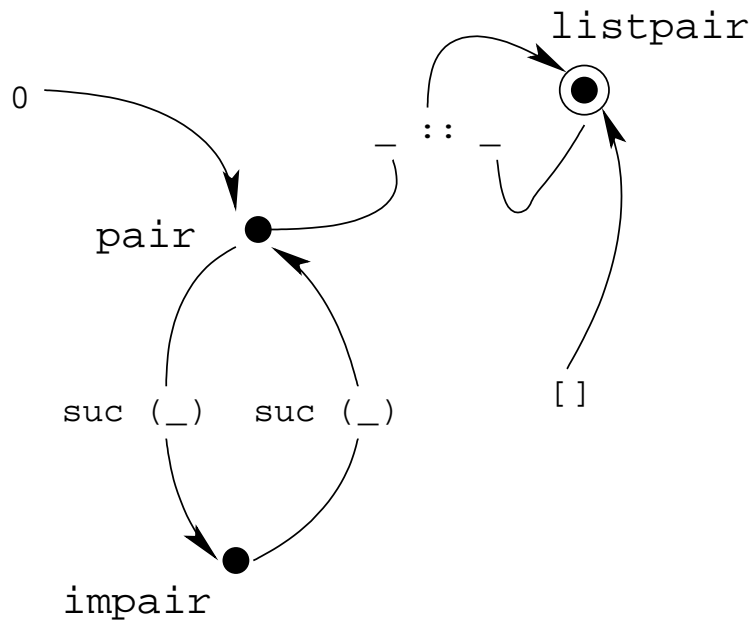
## Construction d'un modèle

---

Partant de clauses de  $\mathcal{H}_1$ :

- Saturer par résolution ordonnée avec sélection et splitting sans splitting.  
(cf. examen du cours de vérification de protocoles crypto,  
DEA Programmation; JGL, avril 2003)
- Les clauses productives résultantes sont un **automate d'arbres alternants**  
(cf. poly du cours de vérification de protocoles crypto,  
DEA Programmation; JGL, mars 2003)

## Automates d'arbres et ensembles de clauses de Horn



`pair(0).`  
`impair(suc(X))  $\Leftarrow$  pair(X).`  
`pair(suc(X))  $\Leftarrow$  impair(X).`  
`listpair(X :: Y)  $\Leftarrow$  pair(X), listpair(Y).`  
`listpair([]).`

**Non-vacuité**  $\Leftrightarrow$  **Contradiction**

(de listpair) (avec  $\perp \Leftarrow \text{listpair}(X).$ )

## Automates déterministes

L'automate de la page précédente est même **déterministe**.

Ceci définit directement un modèle  $\{\text{pair}, \text{impair}, \text{listpair}, \perp\}$ .

		suc		::	pair	impair	listpair	$\perp$
0	pair	pair	impair	pair	$\perp$	$\perp$	listpair	$\perp$
		impair	pair	impair	$\perp$	$\perp$	$\perp$	$\perp$
□	listpair	listpair	$\perp$	listpair	$\perp$	$\perp$	$\perp$	$\perp$
		$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

## Automates non déterministes et alternants

---

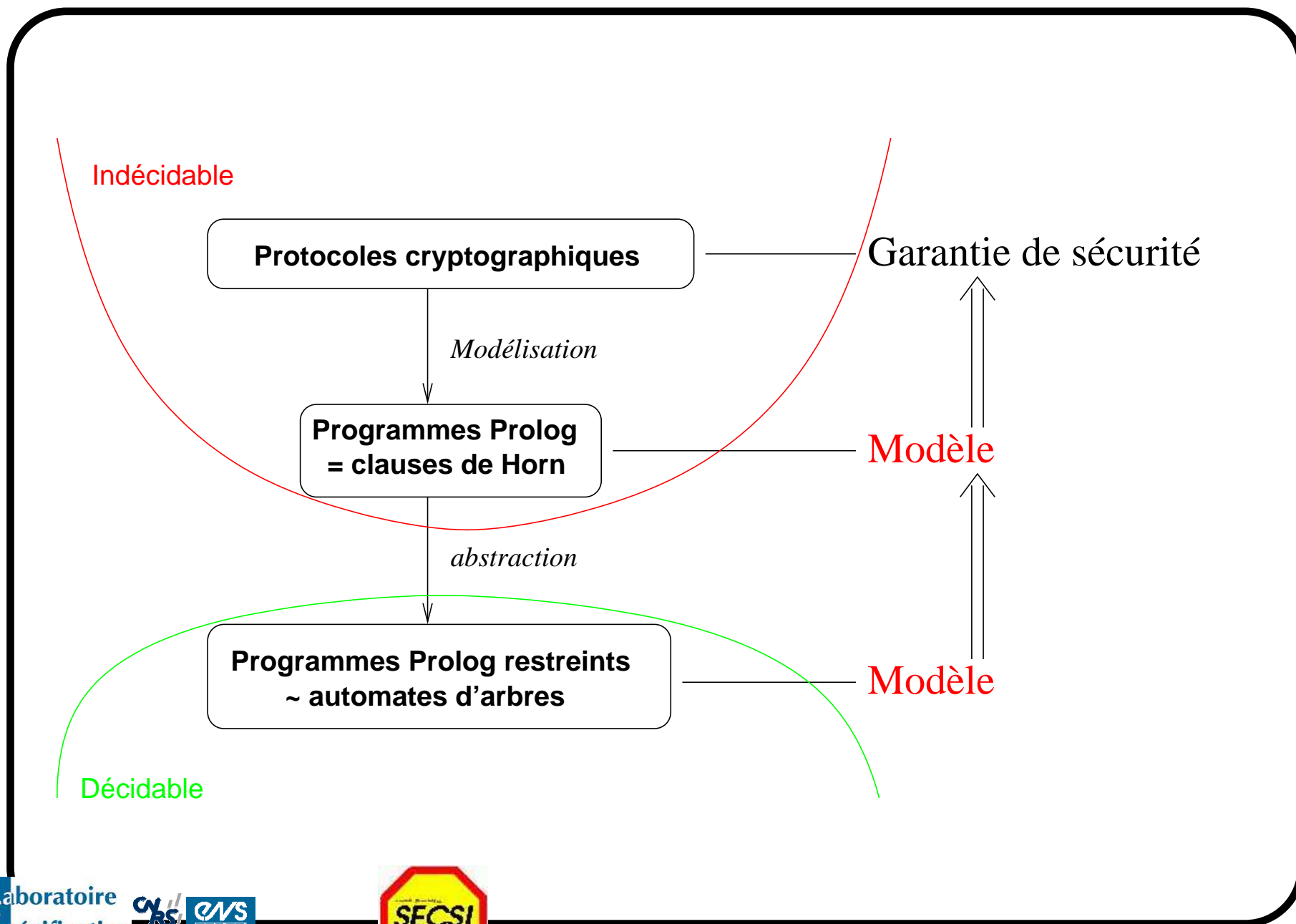
Non-déterminisme:

$$\begin{aligned}\text{knows}(\{X_1\}_{X_2}) &\Leftarrow \text{--aux\_36}(X_1), \text{--aux\_17}(X_2). \\ \text{--aux\_20}(\{X_1\}_{X_2}) &\Leftarrow \text{--aux\_36}(X_1), \text{--aux\_17}(X_2). \\ \text{knows}(\{X_1\}_{X_2}) &\Leftarrow \text{knows}(X_1), \text{knows}(X_2).\end{aligned}$$

Alternance:

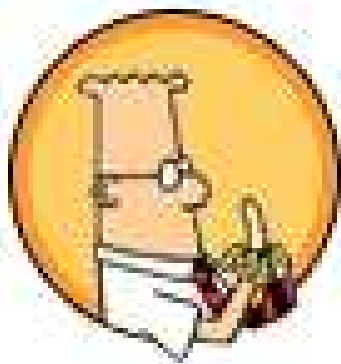
$$\begin{aligned}P(X) &\Leftarrow Q(X), R(X) \\ P(f(X, Y)) &\Leftarrow Q(X), R(X), S(Y)\end{aligned}$$

**Note:** on peut toujours convertir un automate alternant en automate déterministe  
(en temps exponentiel).



## Démo 2

---



Ici, l'orateur devrait vous montrer  
la tête du modèle trouvé pour  
Needham-Schroeder à clés symétriques.

Si l'orateur oublie:  
déterminer l'automate est sans espoir...

1. La question abordée ici.
2. Les protocoles cryptographiques.
3. Modélisation en clauses de Horn.
4. Qu'est-ce qu'une preuve de sécurité?
5. Comment trouver une preuve de sécurité?
6. Qu'est-ce qu'une preuve de sécurité, constructivement?
- 7. Comment vérifier une preuve de sécurité (constructive)?**
8. Conclusion.

## Comment vérifier [en Coq] une preuve de sécurité?

---

Le but du jeu: écrire une preuve Coq de  $\mathcal{M} \models S$ , où  $\mathcal{M}$  est décrit par un automate d'arbres alternant  $\mathcal{A}$ .

**Première approche:** Déterminiser  $\mathcal{A}$

$\Rightarrow$  un automate déterministe complet  $\equiv$  **un modèle fini  $\mathcal{M}$** .

On produit une preuve de  $\mathcal{M} \models S$  par énumération de tous les éléments de  $\mathcal{M}$  (comme dans l'approche de Selinger).

Problème 1: déterminer prend un temps exponentiel (et pas qu'en théorie!)

Problème 2:  $\mathcal{C}$  a se traduit en Coq de façon assez technique! (page suivante)



# $\mathcal{M} \models S$ en Coq — $\mathcal{M}$ donné explicitement

**Section** def.

**Variable**  $\mathbb{N} : \text{Set}, 0 : \mathbb{N}, \text{suc} : \mathbb{N} \rightarrow \mathbb{N}.$

**Inductive**  $\text{pair} : \mathbb{N} \rightarrow \text{Prop} :=$   
 $\text{pair\_0} : \text{pair}(0)$   
 $| \text{pair\_S} : \forall N : \mathbb{N}. \text{impair}(N) \rightarrow \text{pair}(\text{suc}(N))$

**with**  $\text{impair} : \mathbb{N} \rightarrow \text{Prop} :=$   
 $\text{impair\_S} : \forall N : \mathbb{N}. \text{pair}(N) \rightarrow \text{impair}(\text{suc}(N))$

**End** def.

Clauses: appliquer à  $\mathbb{N} = \text{term}$  **Inductive**  $\text{term} : \text{Set} := 0 : \text{term} | S : \text{term} \rightarrow \text{term}.$

Modèle: appliquer à  $\mathbb{N} = D$  défini par des tables, à la Selinger.

Théorème:  $C \in S \quad \forall \vec{v} : D^k. \llbracket C \rrbracket [\vec{x} := \vec{v}]$   $\llbracket - \rrbracket$  défini par **Fixpoint**.

Démonstration: par énumération de  $D^k$  temps  $O(2^k |S|).$

## Comment vérifier [en Coq] une preuve de sécurité?

---

Le but du jeu: écrire une preuve Coq de  $\mathcal{M} \models S$ .

**Deuxième approche:** conserver  $\mathcal{M}$  sous forme d'automate d'arbres alternant.

... forme exponentiellement plus succincte du modèle fini  $\mathcal{M}$

– Vérifier que  $\mathcal{M} \models S$  par une procédure de **model-checking** de clauses du premier ordre sur des automates d'arbres alternants.

DEXPTIME-complet, mais ... efficace en pratique.

– Garder une **trace** du model-checking sous forme de preuve Coq.

# Model-checking de clauses sur un automate alternant

$$\frac{\begin{array}{c} h; C' \vee -P(t) \\ P \text{ universel} \\ \text{dans } \pi_1 \end{array}}{h; C'} \text{ (Univ-)}$$

**Apply**

$$\frac{h \cup \{C\}; C}{\text{ (Loop)}}$$

**Exact** (application d'hyp. réc.)

$$\frac{\begin{array}{c} h; C' \vee +P(t) \\ P \text{ universel} \\ \text{dans } \pi_1 \end{array}}{\text{ (Univ+)}}$$

**Exact**

$$\frac{\begin{array}{c} h; C_1 \vee \dots \vee C_n \quad (n \geq 2) \\ \text{les } C_i \text{ étant non vides et ne partageant aucune variable libre} \\ 1 \leq i \leq n \end{array}}{h; C_i} \text{ (Split)}$$

**Cut, Tauto**

## Inversion, Elim, Tauto

$$\begin{array}{c}
 h; C' \vee -P(f(\vec{t})) \quad P \text{ non universel dans } \pi_1 \\
 \{P(f(\vec{X})) \Leftarrow D_i(\vec{X}) \\
 | 1 \leq i \leq m\} \\
 = \text{clauses de } \pi_1 \text{ de tête } P(f(\vec{X})) \\
 \hline
 C' \Leftarrow D_1(\vec{t}) \quad \dots \quad C' \Leftarrow D_m(\vec{t})
 \end{array}
 \quad (\mathbf{Elim} - / \mathbf{Fun})$$

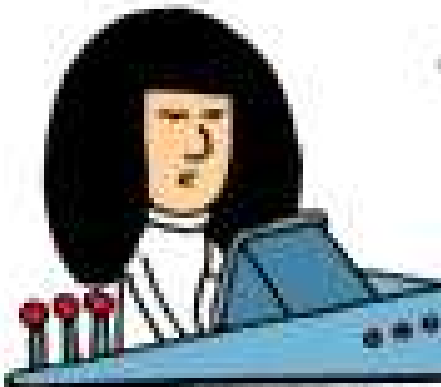
## Fix, Case, Inversion (récurrence)

$$\begin{array}{c}
 h; -P(X) \vee \bigvee_{j=1}^k \pm_j P_j(X) \\
 P, P_i \text{ non universels dans } \pi_1, 1 \leq i \leq k \\
 \{P(f_i(\vec{X})) \Leftarrow D_i(\vec{X}) \\
 | 1 \leq i \leq m\} \\
 = \text{clauses de } \pi_1 \text{ ayant } P \text{ en tête} \\
 h' = h \cup \{-P(X) \vee \bigvee_{j=1}^k \pm_j P_j(X)\} \\
 C_i = \bigvee_{j=1}^k \pm_j P_j(f_i(\vec{X})) \\
 \hline
 h'; C_1 \Leftarrow D_1(\vec{X}) \quad \dots \quad h'; C_m \Leftarrow D_m(\vec{X})
 \end{array}
 \quad (\mathbf{Elim} - / \mathbf{Var})$$

## Cut, Tauto (lourd)

$$\begin{array}{c}
 h; C' \vee +P(f(\vec{t})) \quad P \text{ non universel dans } \pi_1 \\
 \{P(f(\vec{X})) \Leftarrow \bigwedge_j B_{ij}(X_j) \\
 | 1 \leq i \leq m\} \\
 = \text{clauses de } \pi_1 \text{ de tête } P(f(\vec{X})) \\
 \text{et } C_1 \wedge \dots \wedge C_k \text{ est une forme normale conjonctive} \\
 \text{de } C' \vee \bigvee_{i=1}^m \bigwedge_j B_{ij}(t_j) \\
 \hline
 h'; C_1 \quad \dots \quad h'; C_k
 \end{array}
 \quad (\mathbf{Elim}+)$$

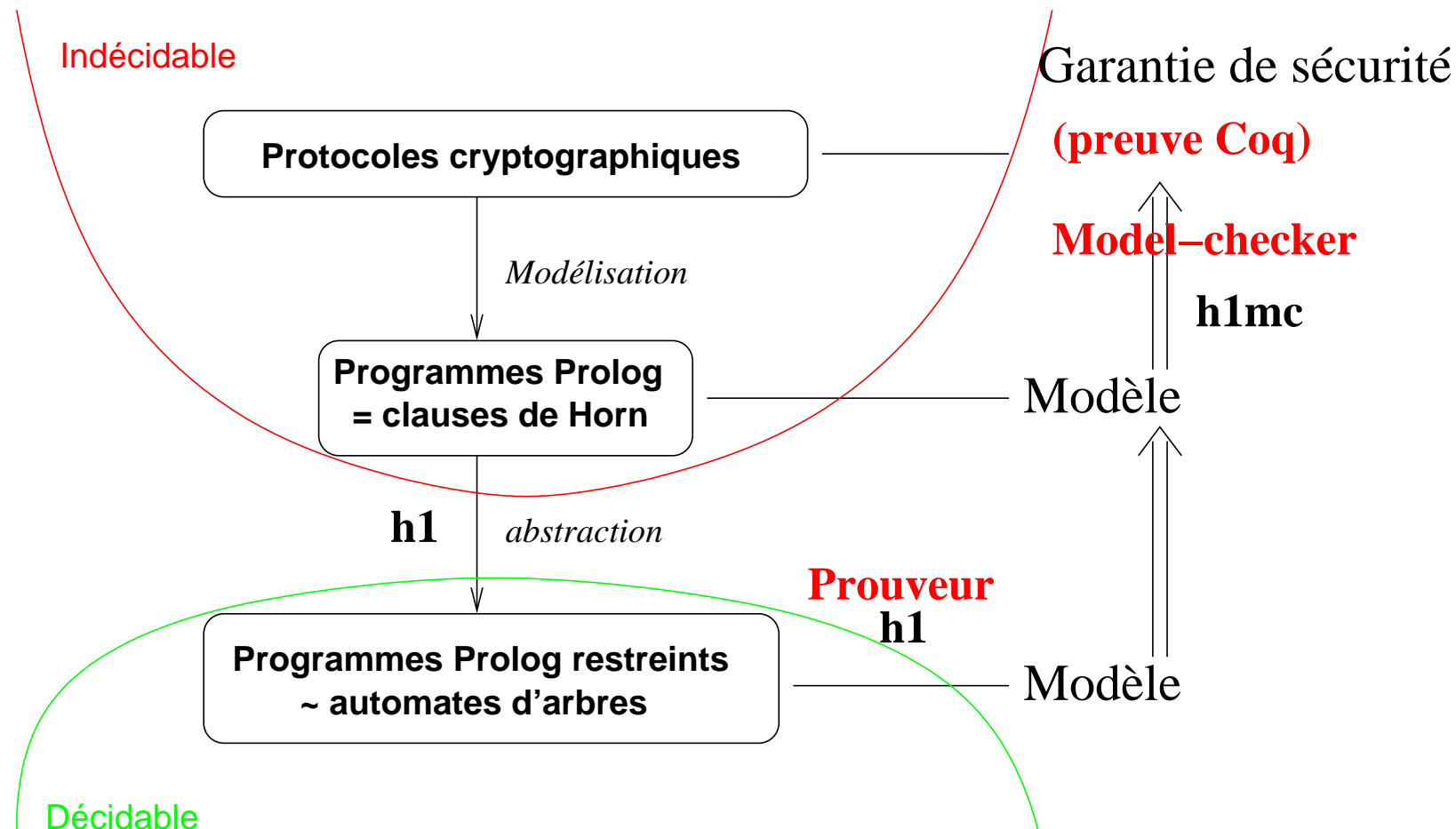
## Démo 3









L'orateur vous a-t-il montré le  
model-checker h1mc en action?  
Et la preuve Coq qui en sort?  
Vous a-t-il fait vérifier cette preuve par Coq?

1. La question abordée ici.
2. Les protocoles cryptographiques.
3. Modélisation en clauses de Horn.
4. Qu'est-ce qu'une preuve de sécurité?
5. Comment trouver une preuve de sécurité?
6. Qu'est-ce qu'une preuve de sécurité, constructivement?
7. Comment vérifier une preuve de sécurité (constructive)?
- 8. Conclusion.**

# Résumé



## Synthèse et perspectives

- **Outils mathématiques:** une belle intégration:  
démonstration automatique/automates/model-checking/preuve assistée
- **Outil logiciel:** la suite h1:  
une bibliothèque d'automates d'arbres;                      un démonstrateur automatique;  
un model-checker de logique du premier ordre;  
un générateur de preuves Coq par récurrence;
- **Autres résultats:**
  - primitives de Diffie-Hellman, ou exclusif, automates mod.  $\mathcal{E}$       (H. Comon-Lundh , R. Treinen , V. Cortier , M. Roger  [outil MOP],  
K.N. Verma  ).
  - analyse de code “points-to” [pointeurs]      (F. Parrennes )



## Conclusion

“Logic wins!”

(Roy Dyckhoff, mai 1996,  
communication privée,  
— hors contexte.)

