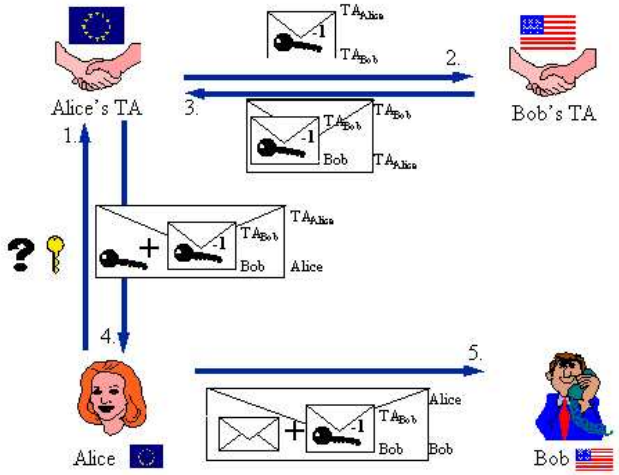
Protocoles cryptographiques,
analyse de code,
détection d'intrusions

Jean Goubault-Larrecq

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



Projets RNTL EVA, Prouvé ACI VERNAM, Psi-Robuste, Rossignol

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

La sécurité informatique, un thème vaste

Comment assurer la sécurité d'un système, d'un réseau ?

- Cryptographie, protocoles cryptographiques.
  Comment être sûr que les moyens mis en œuvre assurent une certaine sécurité ?
  → preuve + interprétation abstraite + autres.

- Sécurisation dynamique : détection d'intrusions.
  → une approche originale par model-checking.

Applications : commerce électronique, cartes à puce, réseaux d'entreprise, télécoms (tracking), etc.

Permanents:

Jean Goubault−Larrecq
(Prof. Univ.)

Hubert Comon−Lundh
(Prof. Univ.)

Stéphane Demri
(CR CNRS)

Florent Jacquemard
(CR INRIA)

Ralf Treinen
(M.de Conf.)

David Nowak
(CR CNRS)

Michel Bidoit
(DR CNRS)

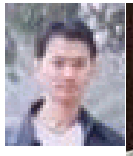
Doctorants et post−docs:

Kumar Neeraj Verma

Muriel Roger

Alexandre Boisseau

Véronique Cortier



Yu Zhang

Vincent Bernat

Stéphanie Delaune

Mathieu Baudet

Pascal Lafourcade

Fabrice Parrennes

Julien Olivain

Laboratoire Spécification et Vérification

CNRS ENS CACHAN INRIA Futurs

SECSI

I. Protocoles cryptographiques

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

Divers aspects :

secret : M est secret si aucun intrus ne peut émettre M;

authenticité : le seul processus qui peut fabriquer M est A;

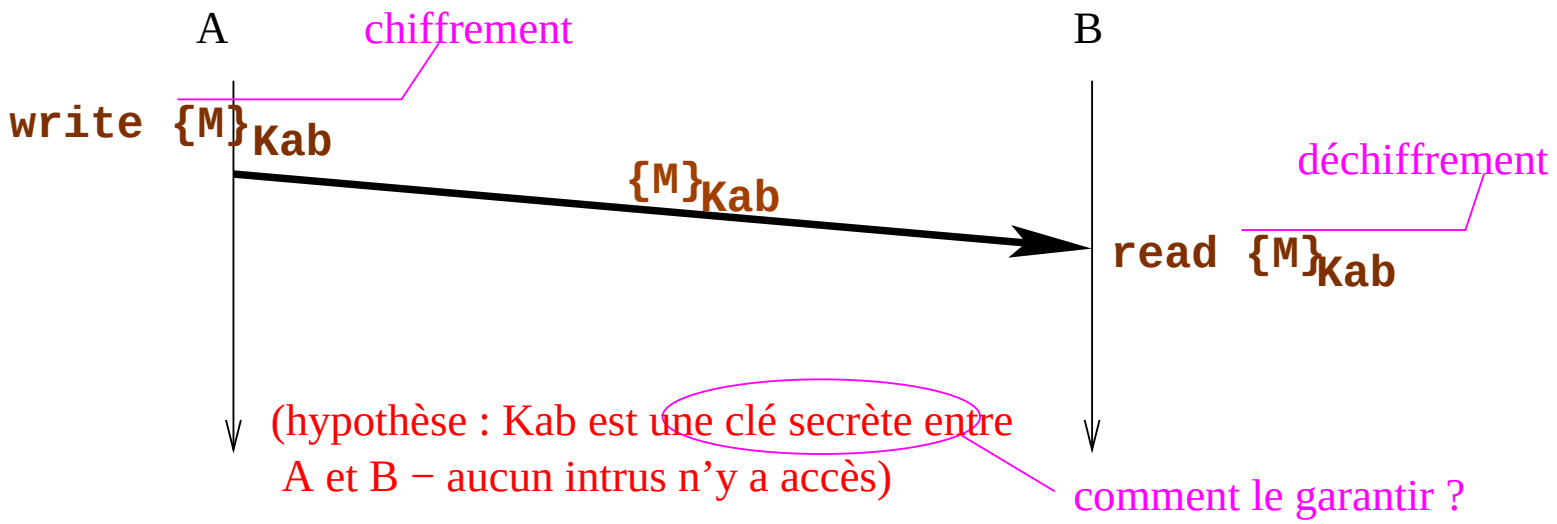
fraîcheur : le message M a été fabriqué récemment;

non-duplication : le message M ne peut être reçu qu'une fois (factures);

non-répudiation : A ne peut pas nier avoir émis M (commandes).

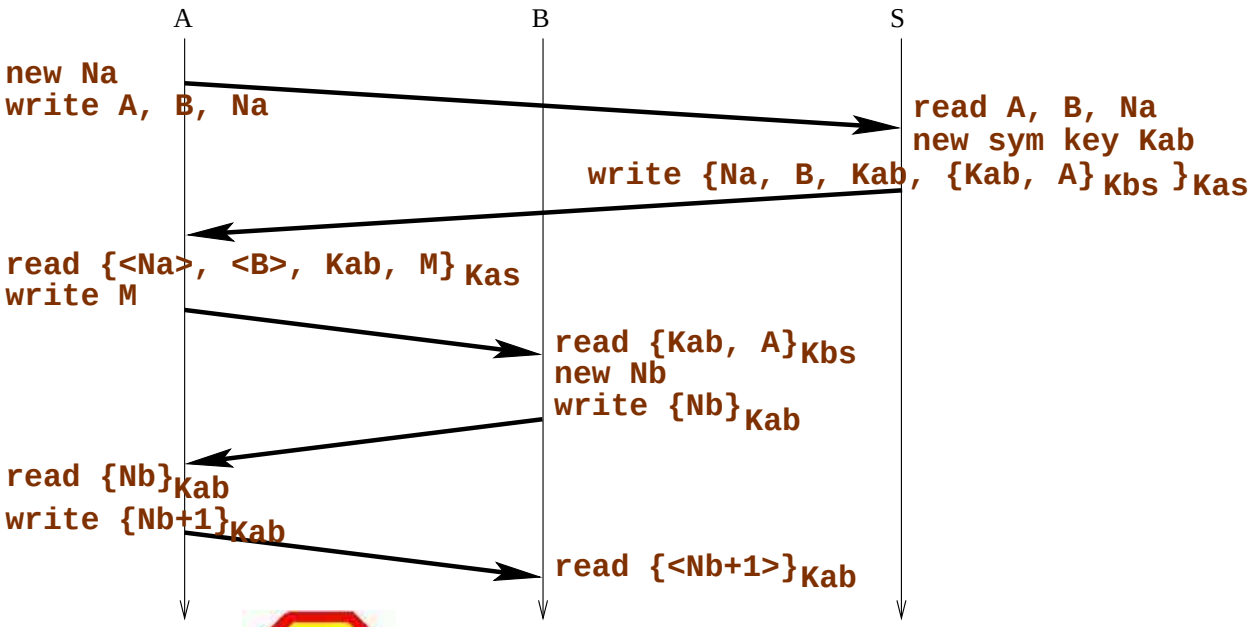
La cryptographie ne suffit pas !

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



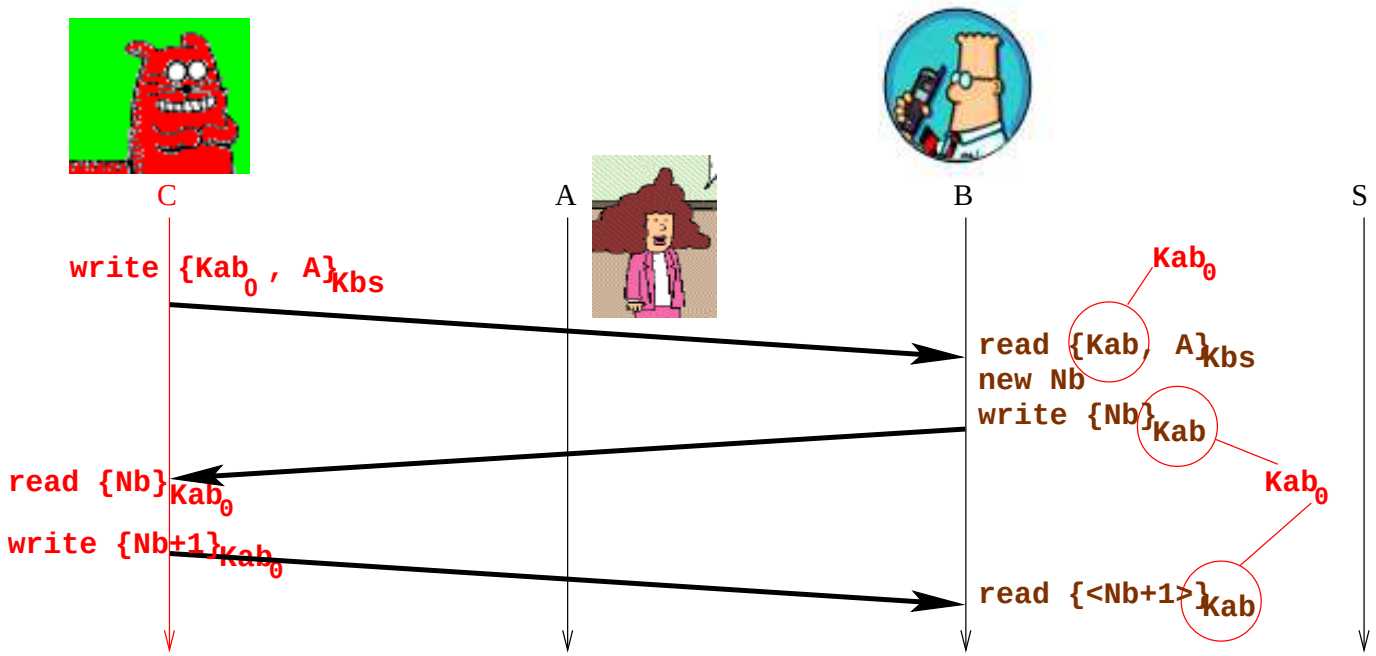
Ex : Needham-Schroeder à clés symétriques

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. $A \longrightarrow B : \{N_b + 1\}_{K_{ab}}$



Une attaque

C rejoue un vieux $\{\mathtt{Kab}_0, \mathtt{A} \mid \mathtt{Kbs}\}$ — suffisamment vieux pour avoir réussi à récupérer $\mathtt{Kab}_0$.



Sécurité des *protocoles* cryptographiques

Il faut empêcher :

le rejeu de vieux messages par C

$\rightarrow$ utilisation de nonces ($\sim$ aléas), ou d'estampilles.

les impostures menées par C

$\rightarrow$ utilisation de certificats, de clés publiques.

etc.

Mais comment s'assurer qu'aucune attaque (même une non prévue) ne peut être perpétuée : preuve + approximations (interprétation abstraite).

(cf. par exemple Jean Goubault-Larrecq, FMPPTA'2000; JFLA'2004)

Modélisation en clauses de Horn (Prolog pur)

1. Les capacités de l'intrus.

$$
\begin{array}{rcll}
\texttt{knows}(\{M\}_K) & \Leftarrow & \texttt{knows}(M), \texttt{knows}(K) & \text{(l'intrus sait chiffrer} \\
\texttt{knows}(M) & \Leftarrow & \texttt{knows}(\{M\}_{\texttt{k}(\texttt{sym},X)}), & \\
 & & \texttt{knows}(\texttt{k}(\texttt{sym},X)) & \ldots\text{et déchiffrer [cas symétrique])} \\
\texttt{knows}([]) & & & \text{(l'intrus sait fabriquer} \\
\texttt{knows}(M_1 :: M_2) & \Leftarrow & \texttt{knows}(M_1), \texttt{knows}(M_2) & \text{toutes les listes de messages connus)} \\
\texttt{knows}(M_1) & \Leftarrow & \texttt{knows}(M_1 :: M_2) & \text{(l'intrus peut lire les premières} \\
\texttt{knows}(M_2) & \Leftarrow & \texttt{knows}(M_1 :: M_2) & \text{et secondes composantes de chaque paire)} \\
\texttt{knows}(\texttt{suc}(M)) & \Leftarrow & \texttt{knows}(M) & \text{(l'intrus peut ajouter} \\
\texttt{knows}(M) & \Leftarrow & \texttt{knows}(\texttt{suc}(M)) & \text{et retrancher un)}
\end{array}
$$

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

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

$1.\ A \rightarrow S : A, B, N_a$

$2.\ S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

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

$$(k_{ab} \equiv \texttt{k(sym,cur}(A,B,N_a)))$$

$2.\ S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

$3.\ A \longrightarrow B : \{K_{ab}, A\}_{K_{bs}}$

$$\texttt{knows}(M) \Leftarrow \texttt{knows}(\{[\texttt{na}([\texttt{a}, \texttt{b}]), \texttt{b}, K_{ab}, M]\}_{\texttt{k(sym,}[\texttt{a},\texttt{s}]\texttt{)}})$$

$$\texttt{a\_key}(K_{ab}) \Leftarrow \texttt{knows}(\{[\texttt{na}([\texttt{a}, \texttt{b}]), \texttt{b}, K_{ab}, M]\}_{\texttt{k(sym,}[\texttt{a},\texttt{s}]\texttt{)}})$$

$3.\ A \longrightarrow B : \{K_{ab}, A\}_{K_{bs}}$

$4.\ B \longrightarrow A : \{N_b\}_{K_{ab}}$

$$\texttt{knows}(\{\texttt{nb}(K_{ab}, A, B)\}_{K_{ab}}) \Leftarrow \texttt{knows}(\{[K_{ab}, A]\}_{\texttt{k(sym,}[B,\texttt{s}]\texttt{)}})$$

4. $B \longrightarrow A : \{N_b\}_{K_{ab}}$

5. $A \longrightarrow B : \{N_b + 1\}_{K_{ab}}$

$\texttt{knows}(\{\texttt{suc}(N_b)\}_{K_{ab}}) \Leftarrow \texttt{knows}(\{N_b\}_{K_{ab}})$

3. Les clauses du protocole—les vieilles sessions

1. $A \rightarrow S : A, B, N_a$
2. $S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

$$\texttt{knows}\Big(\{[N_a, B, k_{ab}, \{[k_{ab}, A]\}_{\texttt{k(sym,[}B\texttt{,s])}}]\}_{\texttt{k(sym,[}A\texttt{,s])}}\Big) \Leftarrow \texttt{knows}([A, B, N_a])$$

$$(k_{ab} \equiv \texttt{k(sym,prev}(A, B, N_a)\texttt{)})$$

4. Connaissances initiales de l'intrus

$$\begin{aligned}
\texttt{agent}(\texttt{a}) \qquad & \texttt{agent}(\texttt{b}) \\
\texttt{agent}(\texttt{s}) \qquad & \texttt{agent}(\texttt{i}) \\
\texttt{knows}(X) \quad \Leftarrow \quad & \texttt{agent}(X) \\
\texttt{knows}(\texttt{k}(\texttt{pub}, X)) & \\
\texttt{knows}(\texttt{k}(\texttt{prv}, \texttt{i})) & \\
\texttt{knows}(\texttt{k}(\texttt{sym}, \texttt{prev}(A, B, N_a))) \qquad & \text{(les clés des sessions} \\
& \text{précédentes sont compromises)}
\end{aligned}$$

5. Requêtes de sécurité

$\bot \Leftarrow \texttt{knows}(\texttt{k}(\texttt{sym}, \texttt{cur}(\texttt{a}, \texttt{b}, N_a)))$

l'intrus peut-il fabriquer K_{ab}

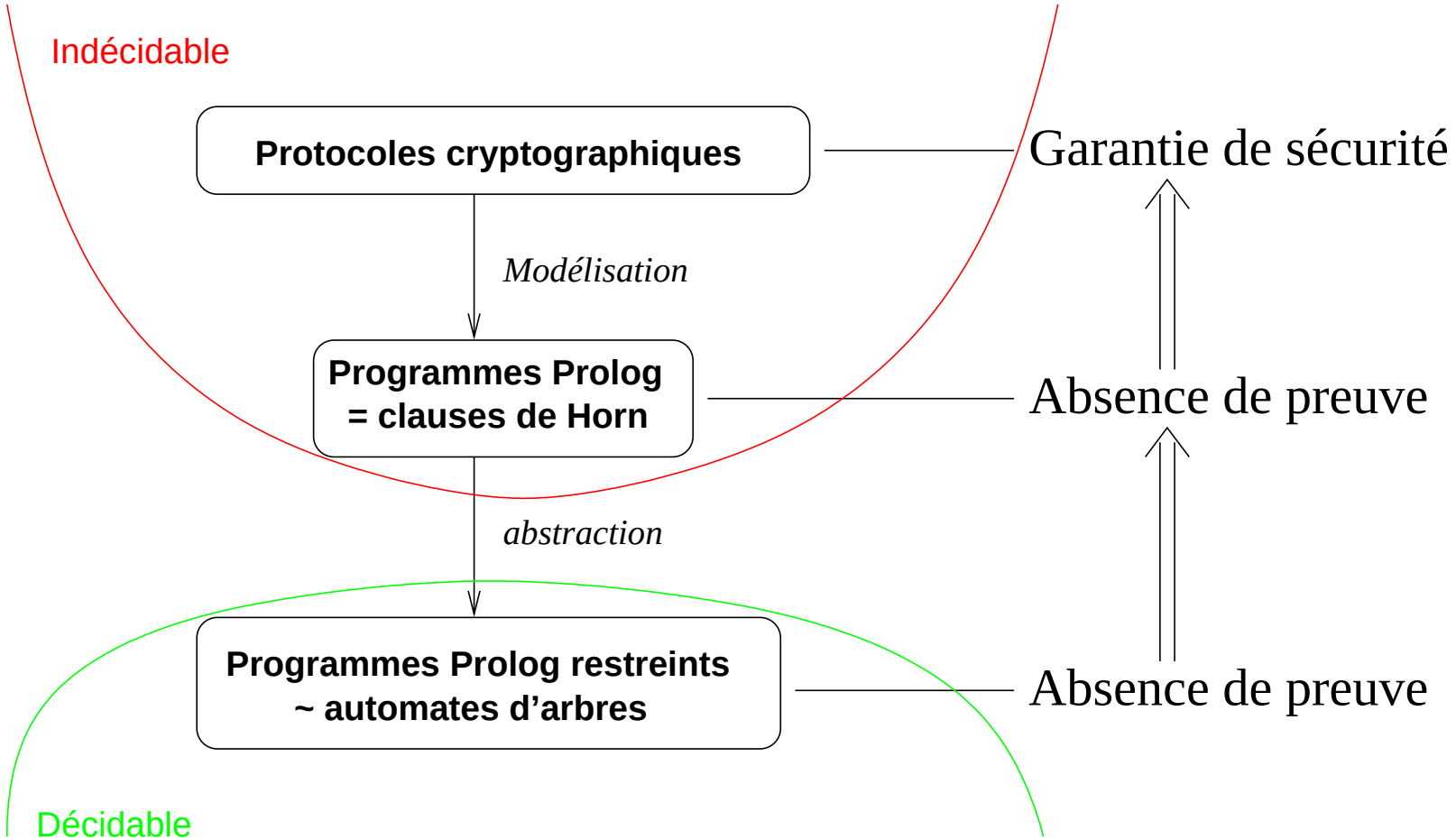
telle que fabriquée par S?

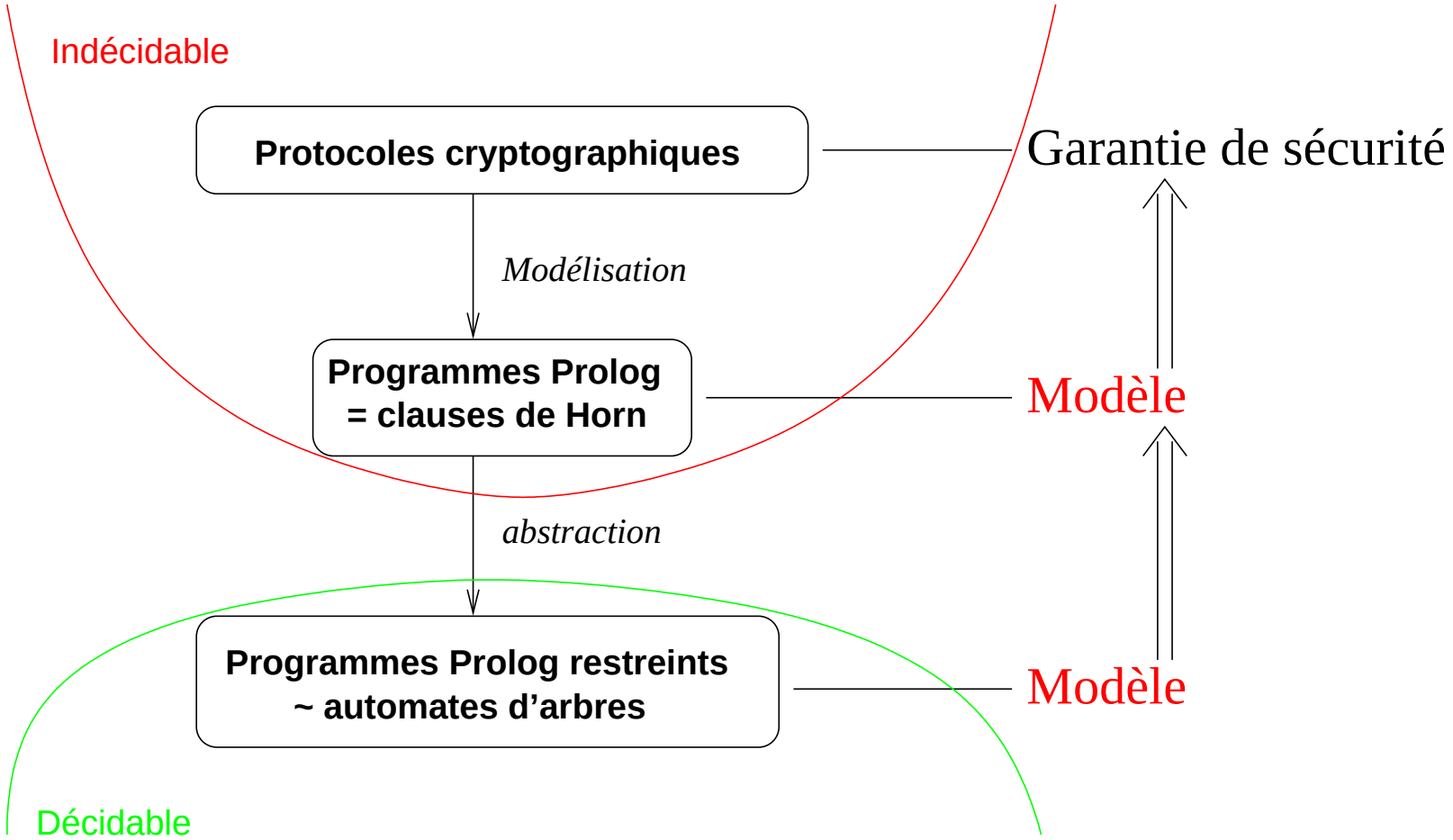
$\bot \Leftarrow \texttt{knows}(K_{ab}), \texttt{a\_key}(K_{ab})$

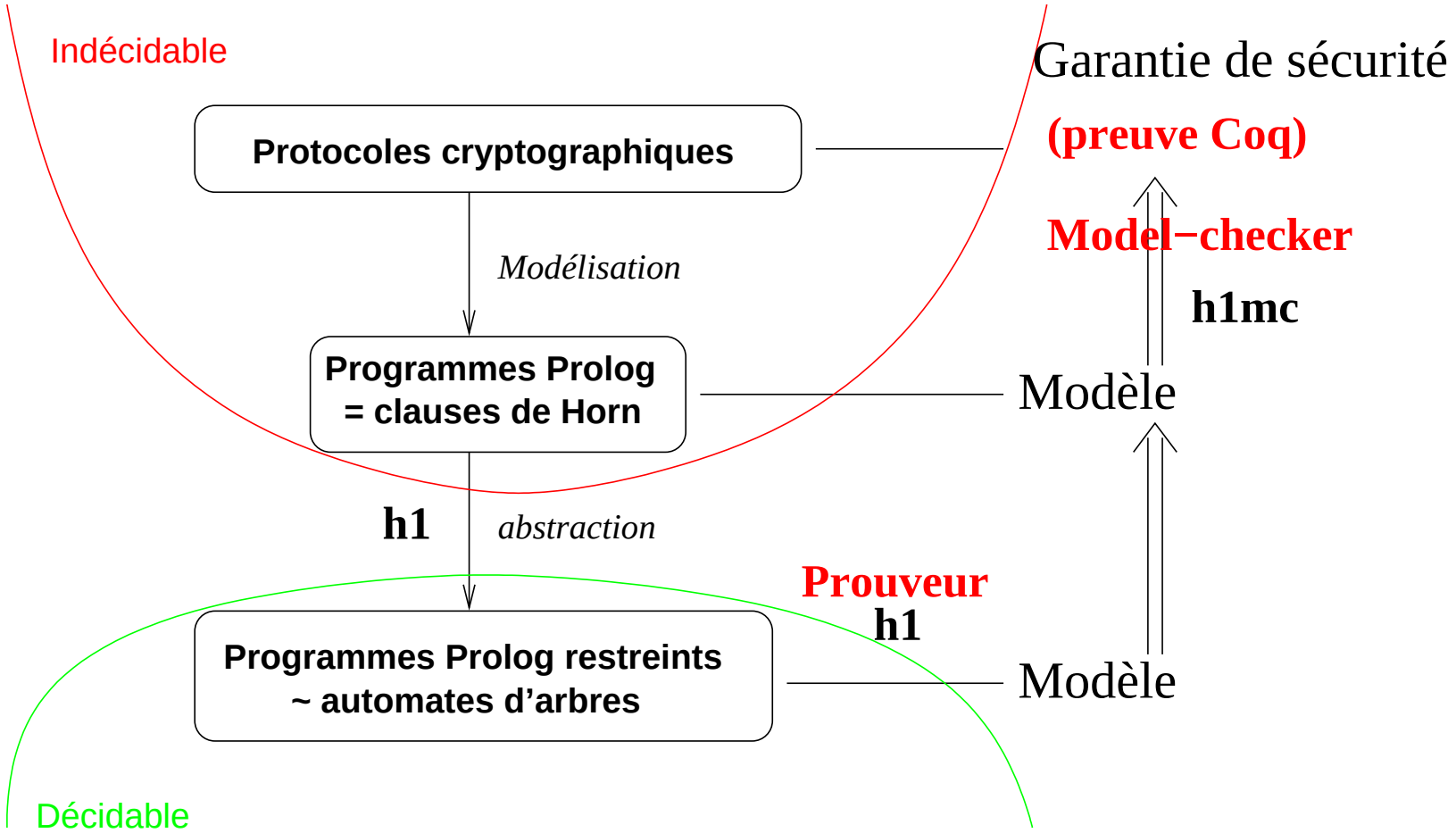
... telle que reçue par A?

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

... telle que reçue par B?


Indécidable
Protocoles cryptographiques
Garantie de sécurité
Modélisation
Programmes Prolog
= clauses de Horn
Absence de preuve
abstraction
Programmes Prolog restreints
~ automates d'arbres
Absence de preuve
Décidable


Indécidable
Protocoles cryptographiques
Garantie de sécurité
Modélisation
Programmes Prolog
= clauses de Horn
Modèle
abstraction
Programmes Prolog restreints
~ automates d'arbres
Modèle
Décidable


Indécidable
Protocoles cryptographiques
Modélisation
Programmes Prolog
= clauses de Horn
h1
abstraction
Programmes Prolog restreints
~ automates d'arbres
Décidable
Garantie de sécurité
(preuve Coq)
Model−checker
h1mc
Modèle
Prouveur
h1
Modèle

Quelques résultats récents

Théoriques:

- Primitives de Diffie-Hellman, ou exclusif (KV, MR, JGL, RT, HCL)
- Analyse de code “points-to” et clauses de Horn (FP, MB, JGL)
- La vérification dans un modèle temporisé probabiliste (Las Vegas) se réduit à l’accessibilité dans les graphes

... soumis pour publication (MB)

Pratiques:

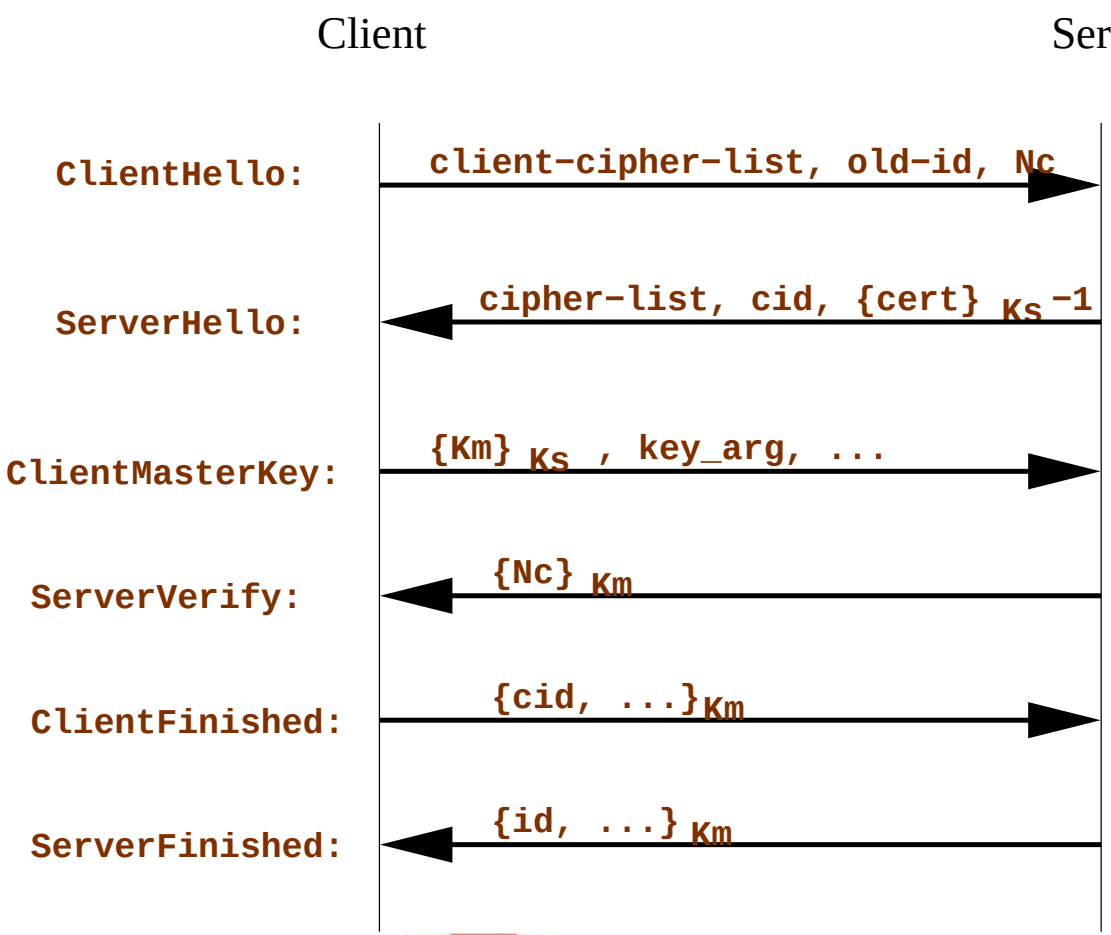
- L’outil `MoP` et la vérification automatique du protocole de distribution de clé en groupe IKA.1;

... publication fi n 2003 (MR, JGL)

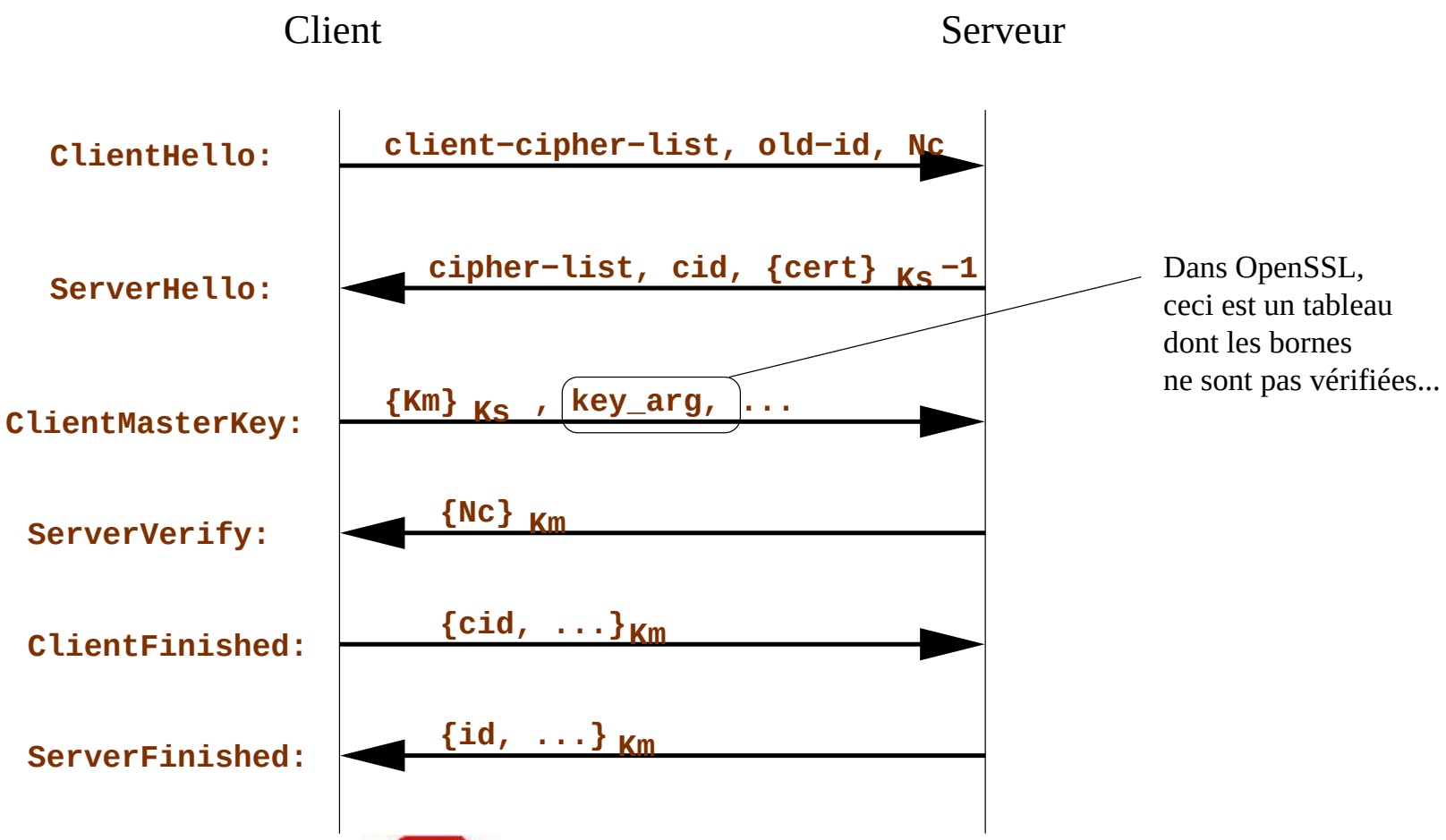
- L’outil `h1` (JGL)

Démo: `cd ~/H1.1/; ./bigh1 +all +p +trace +m nspriv.p`

Le protocole du handshake SSL v2



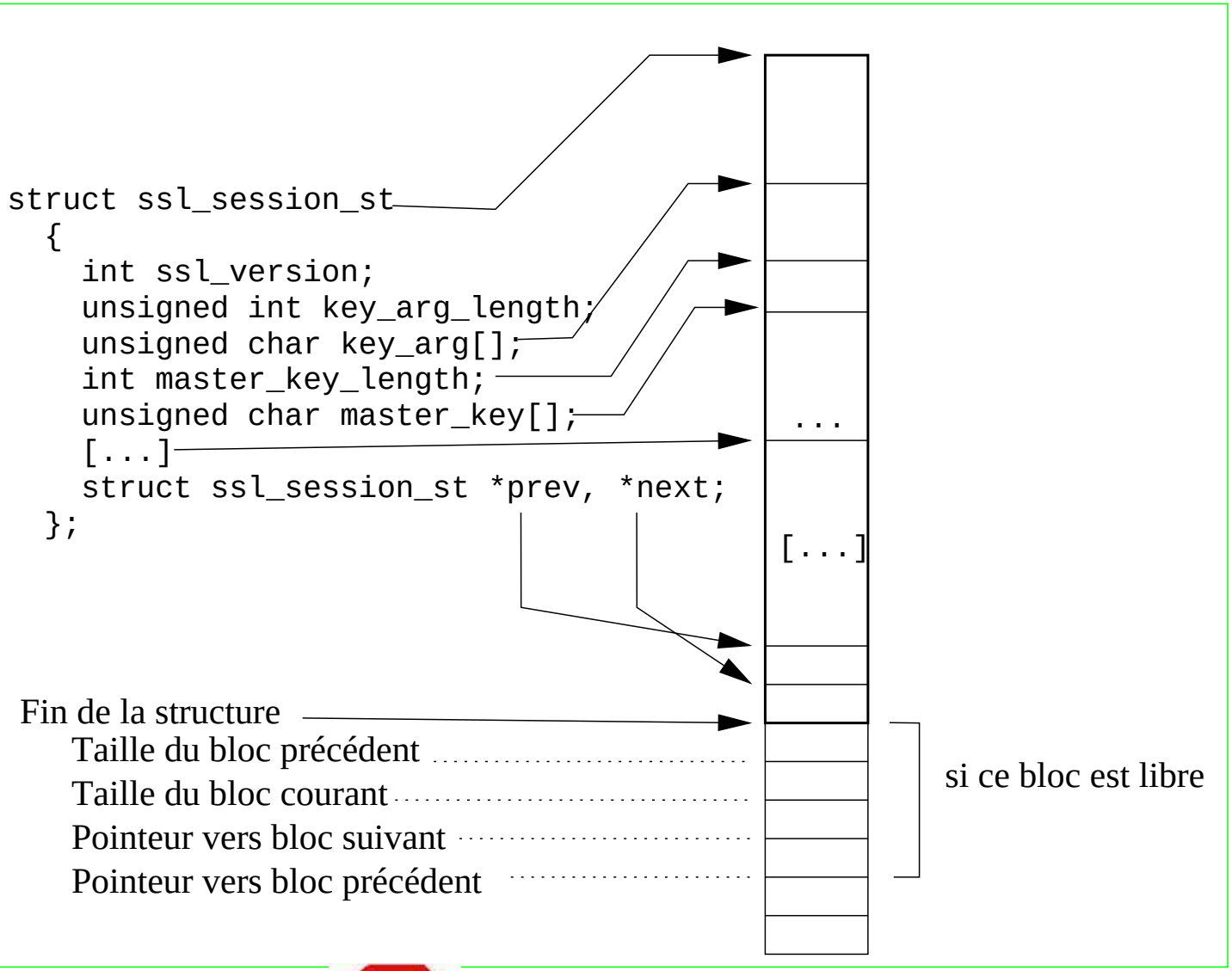
SSL v2, un détail d'implémentation

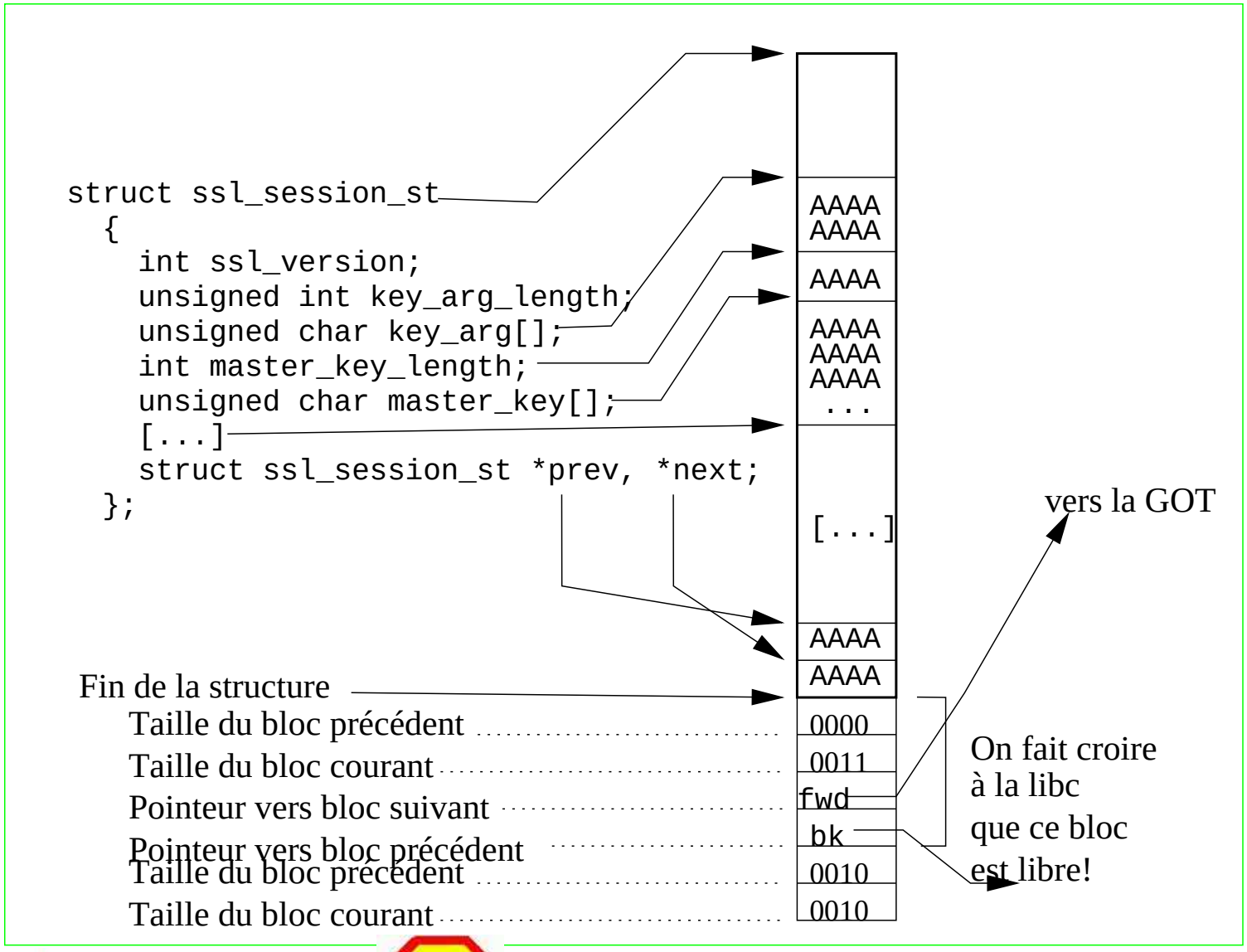


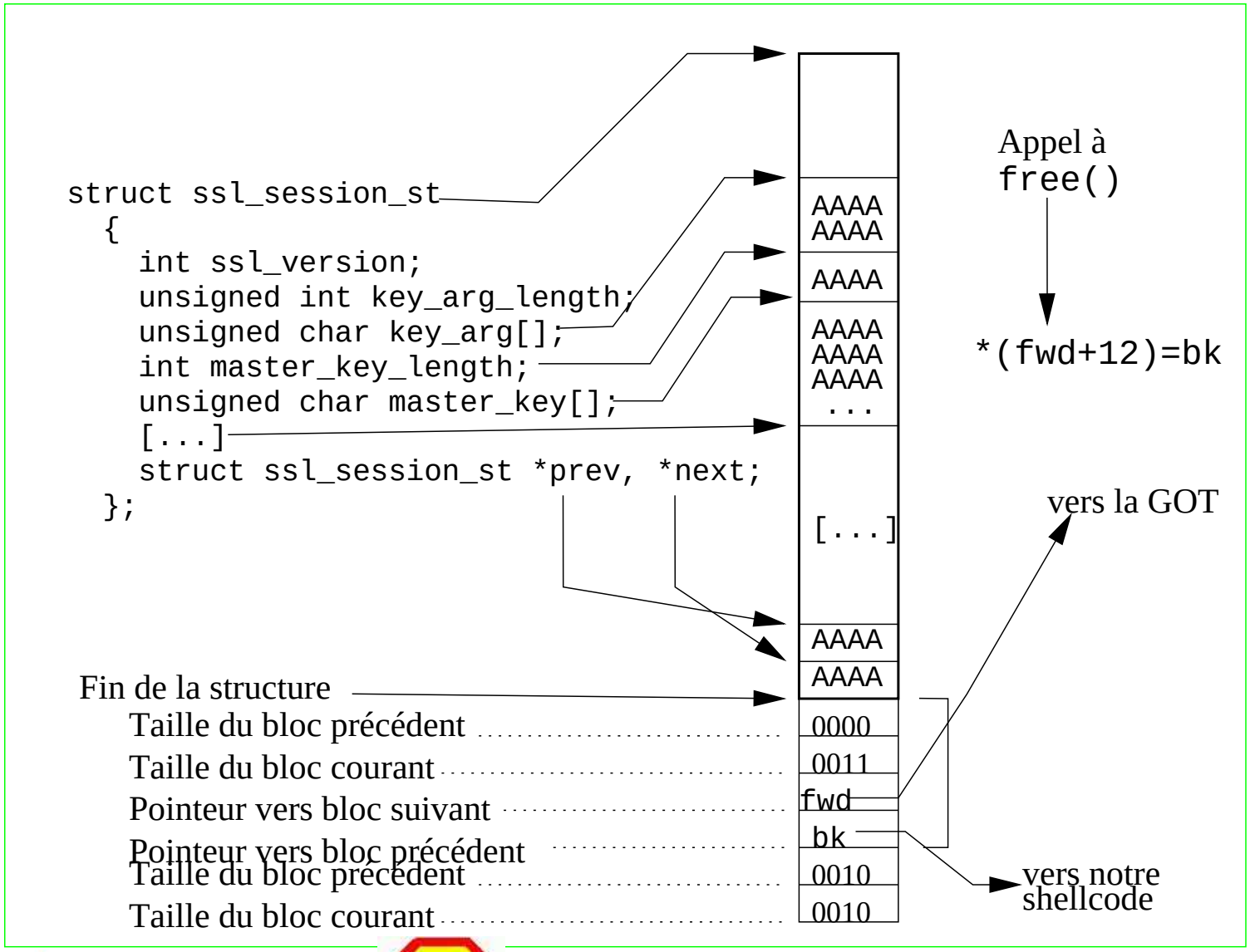
Une attaque sur OpenSSL

Lorsque le serveur reçoit `ClientMasterKey`, il le copie dans:

```
typedef struct ssl_session_st
    {
    int ssl_version;
    unsigned int key_arg_length;
    unsigned char key_arg[SSL_MAX_KEY_ARG_LENGTH];
      // key_arg: on va l'exploser!
    int master_key_length;
    unsigned char master_key[SSL_MAX_MASTER_KEY_LENGTH];
    [...]
    struct ssl_session_st *prev,*next;
    } SSL_SESSION;
```


struct ssl_session_st
{
int ssl_version;
unsigned int key_arg_length;
unsigned char key_arg[];
int master_key_length;
unsigned char master_key[];
[...]
struct ssl_session_st *prev, *next;
};
...
[...]
Fin de la structure
Taille du bloc précédent
Taille du bloc courant
Pointeur vers bloc suivant
Pointeur vers bloc précédent
si ce bloc est libre


struct ssl_session_st
{
int ssl_version;
unsigned int key_arg_length;
unsigned char key_arg[];
int master_key_length;
unsigned char master_key[];
[...]
struct ssl_session_st *prev, *next;
};
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
...
[...]
AAAA
AAAA
vers la GOT
Fin de la structure
Taille du bloc précédent
Taille du bloc courant
Pointeur vers bloc suivant
Pointeur vers bloc précédent
Taille du bloc précédent
Taille du bloc courant
0000
0011
fwd
bk
0010
0010
On fait croire
à la libc
que ce bloc
est libre!


struct ssl_session_st
{
int ssl_version;
unsigned int key_arg_length;
unsigned char key_arg[];
int master_key_length;
unsigned char master_key[];
[...]
struct ssl_session_st *prev, *next;
};
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
...
[...]
AAAA
AAAA
Appel à
free()
*(fwd+12)=bk
vers la GOT
Fin de la structure
Taille du bloc précédent
Taille du bloc courant
Pointeur vers bloc suivant
Pointeur vers bloc précédent
Taille du bloc précédent
Taille du bloc courant
0000
0011
fwd
bk
0010
0010
vers notre shellcode

Fin de l'attaque apache/OpenSSL

- Pour retrouver l'adresse du shellcode, on se fait retransmettre les infos en débordant le champ `session_id`, et en laissant le serveur nous l'envoyer (en chiffré, s'il-vous-plaît!) dans le message `ServerFinished`.
- Tout ceci nous dit à quelles adresses le serveur travaille.
- On rejoue maintenant une attaque similaire dans une seconde connection OpenSSL pour faire vraiment exécuter le shellcode (i.e., on se débrouille pour que `fwd=%esp`).
- Le serveur nous sert alors un shell `apache/nobody` via HTTP.
- Comment peut-on se prémunir contre ce genre d'attaques?

II. Détection d'intrusions par model-checking

(Commencé avec Muriel Roger, Computer Security Foundations Workshop XIV, 2001. Version longue en `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/rr-lsv-2001-3.rr.ps`.)

Idées de base :

- Détecter des attaques dans un fichier de log, en temps réel.
- Fichier de log = suite d'événements = modèle de Kripke linéaire.
- Signature d'attaque = spécification = formule de logique temporelle.
- Trouver les suites d'événements / signature = model-checking.

Sécurité des systèmes d'information, en vrai

- Virus, vers, chevaux de troie, buffer overflows, etc.

(attaques système)

- Dénis de service, IP/ARP spoofing, sniffing, etc.

(attaques réseau)

- Attaques sur formulaires http (perl, pgp), insertion SQL, virus [vers] Internet Explorer/Word, etc.

(attaques applicatives)

Évolution actuelle

- Systèmes plus vastes.

Sécurité plus diffi cileà assurer.

- Enjeux plus grands.

Bases de données en ligne [banque, santé, impôts, ...],

commerce électronique, etc.

- Attaques plus complexes (automatisées et distribuées).

Packages tout prêts [via Google]

Attaques nécessitant de nombreuses étapes,

… toutes bénignes individuellement

- Nouveaux besoins de détection d'intrusions

Tracking de confi gs utilisateur

Détection de fraudes

Cartes à puce

Le présent... et le futur


Julien Olivain

- L'outil ORCHIDS, développé par Julien Olivain;

  développé dans le cadre du projet RNTL DICO.

- Détecte des corrélations d'événements

  cf. transparents précédents;

  ... très effi cacement.

- Va intégrer bientôt les événements complexes (agrégés), une logique à intervalles, et l'intégration de l'imprécision et des dérives des horloges;

  ... publication à venir, S. Demri et J. Olivain, 2004.

Démo: `cd ~/Hack/orchids-demo/`

Projet Orchids

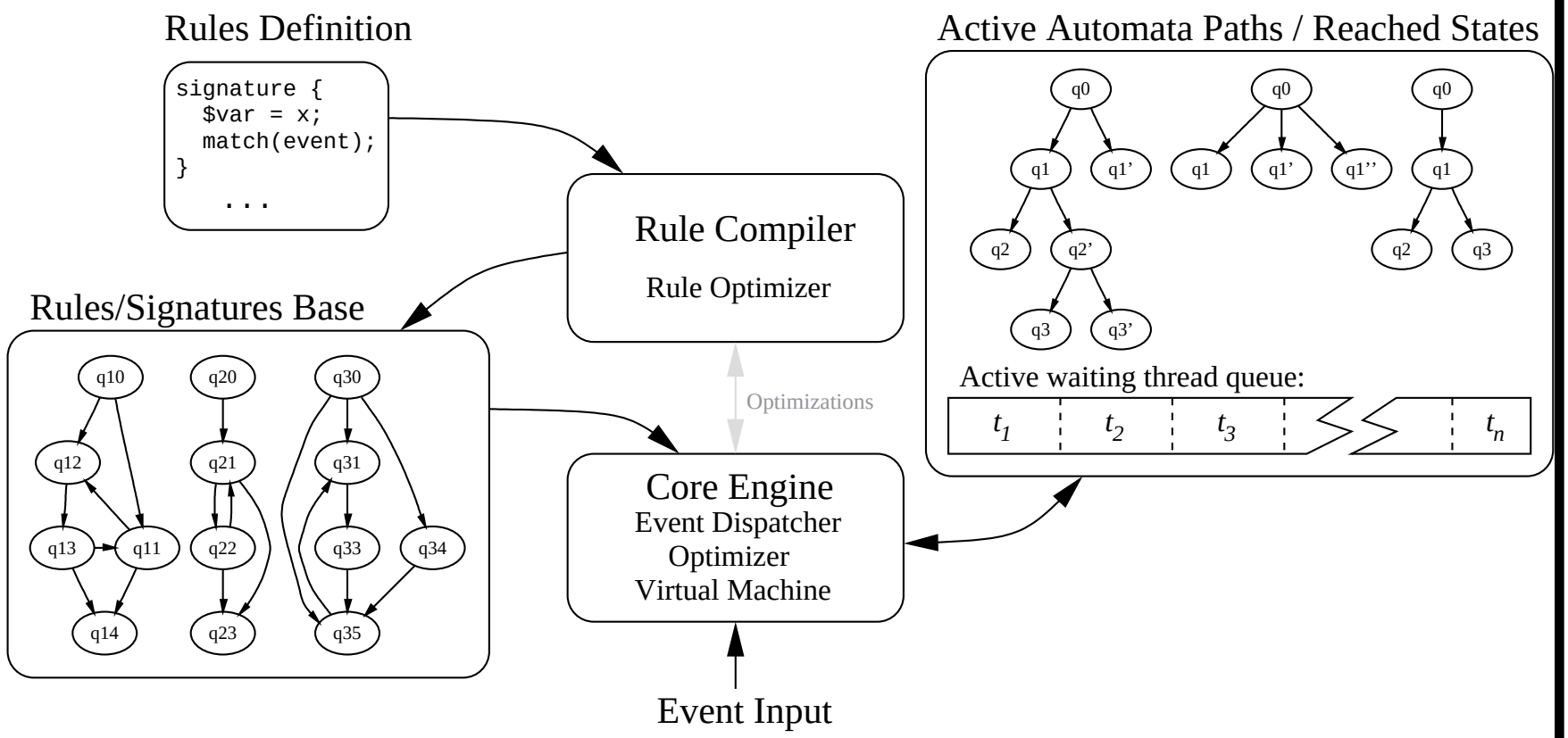
Débuté en 2003 dans le projet RNTL DICO.

Orchids: Détection générique

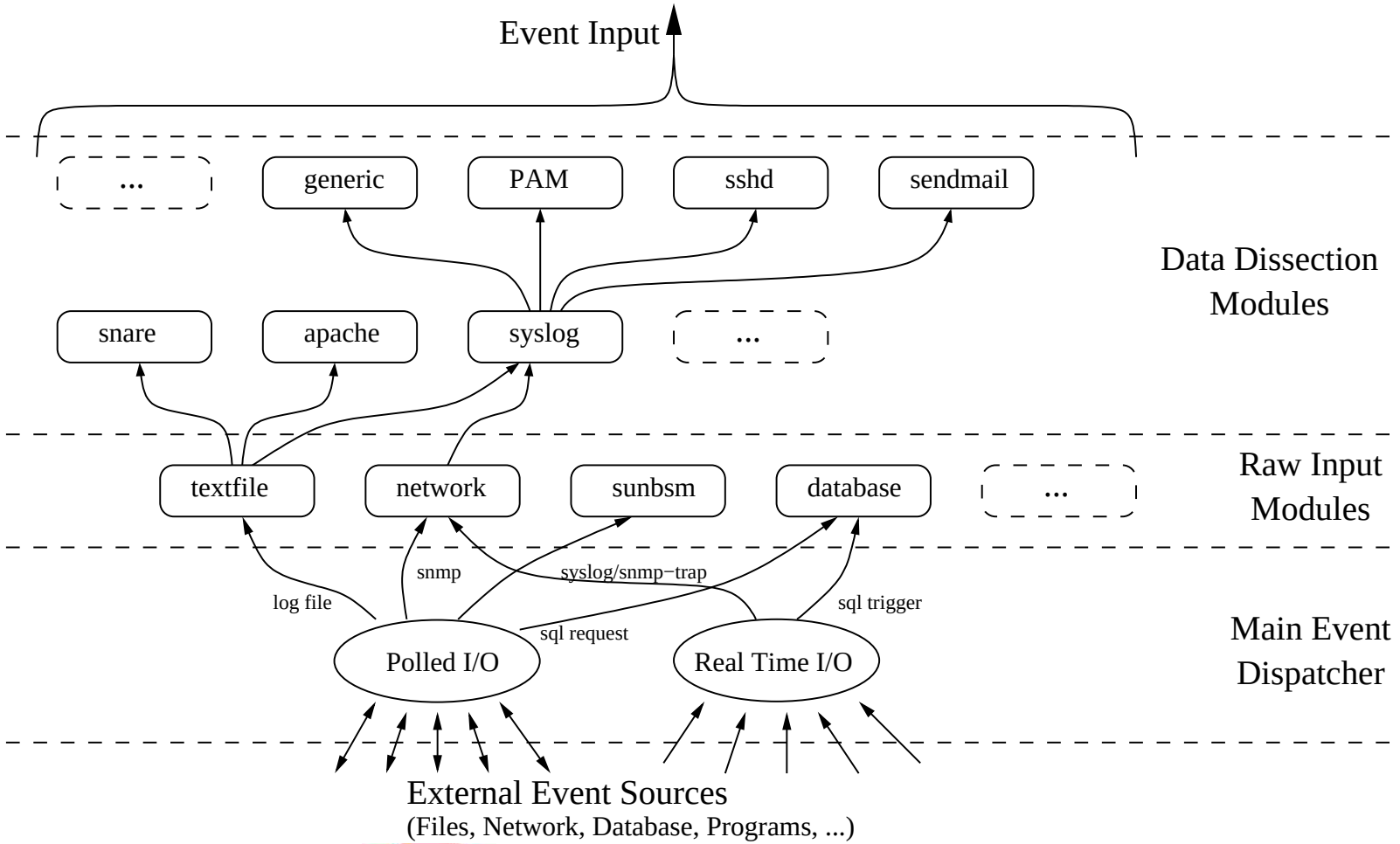
	Hôte	Réseau	Application
Recherche explicite (“misuse”)	•	•	•
Vérification (“anomaly”)	•	•	•

Temps réel (“online”)	Différé (“offline”)
•	•

Architecture de la plate-forme Orchids



Hiérarchie de modules d'entrée



Sources d'entrée d'Orchids

Multi-équipements:

- Audit d'appel systèmes (*Raw Snare*).
- Évènements/journaux *Cisco*.
- Journaux système *Unix* (*Syslog*).
- Journaux système *MS Windows* (*MS EVT*).
- Informations de supervision d'équipements (*SNMP*).
- Informations réseau (*Linux NetFilter Firewall*).
- Autres...

Architecture modulaire...

En cours de développement: BSM.

Attaque de démonstration

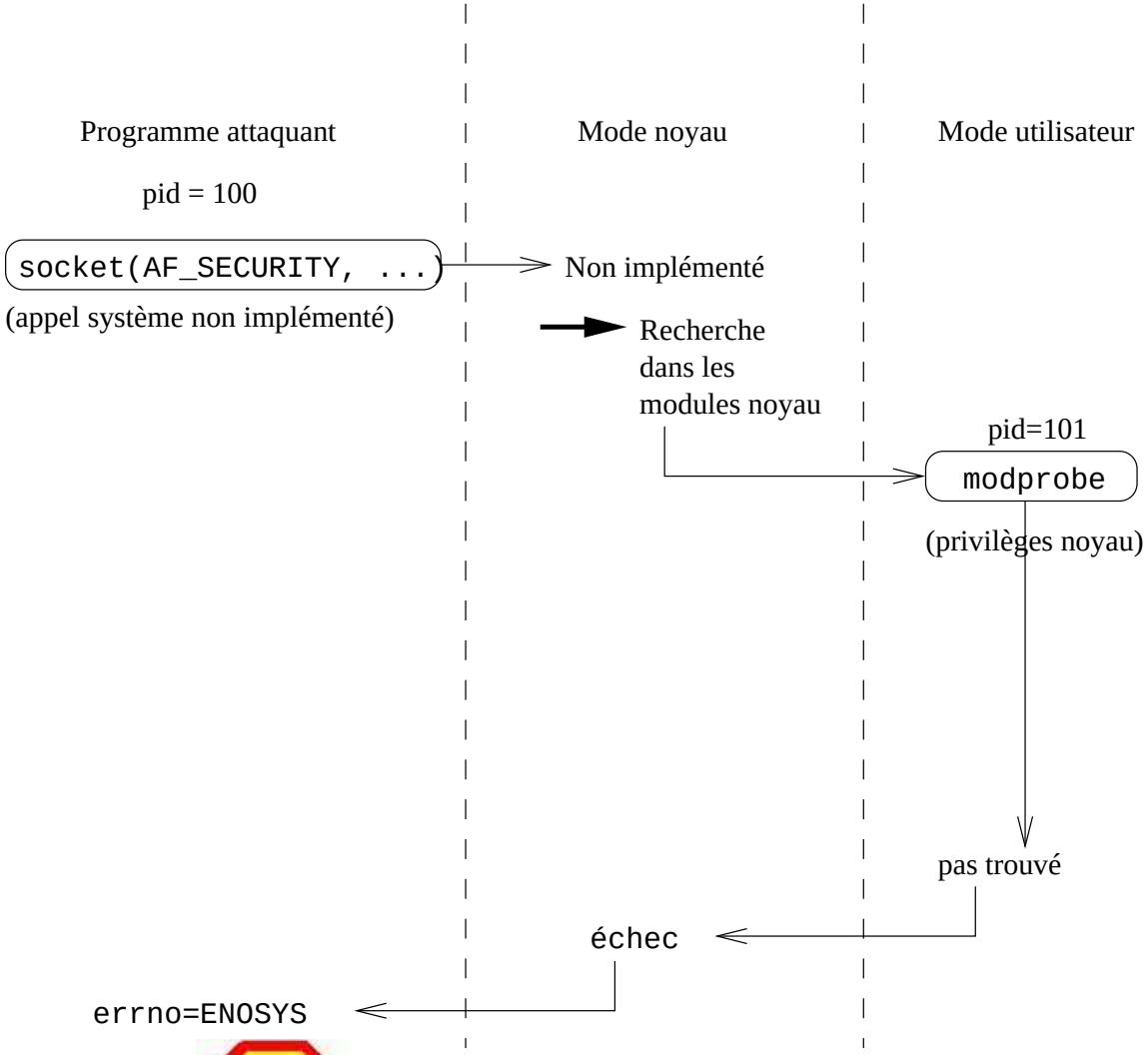
- Exploitation d'un problème d'héritage de permissions et de l'appel système `ptrace()`.

… l'appel système utilisé par tous les debuggers [bénin!]

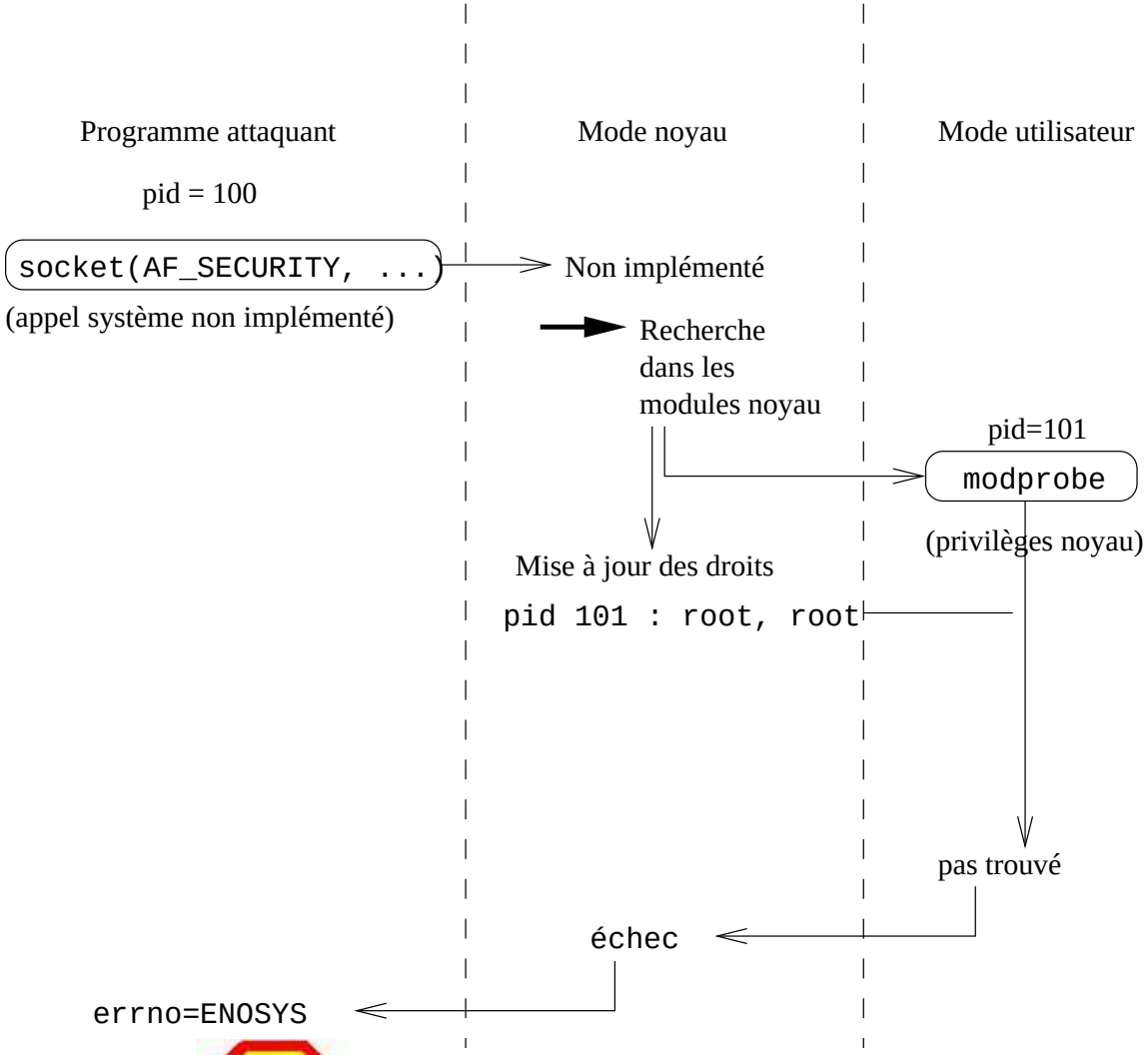
… même Linux n'est pas sûr (il n'y a pas que Windows)

… attaque subtile, fondée sur une race condition dans le noyau

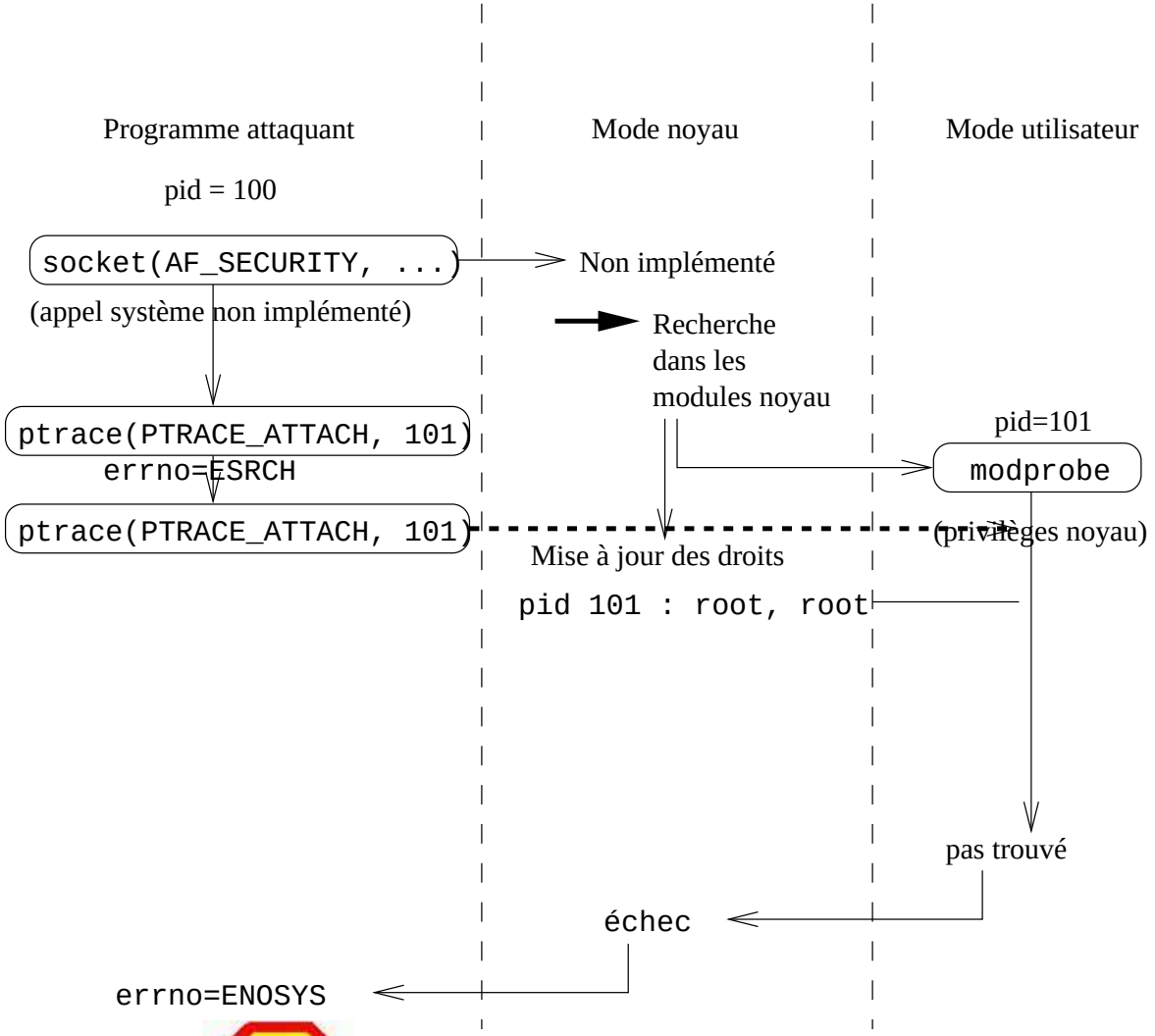
L'attaque `ptrace` (1)



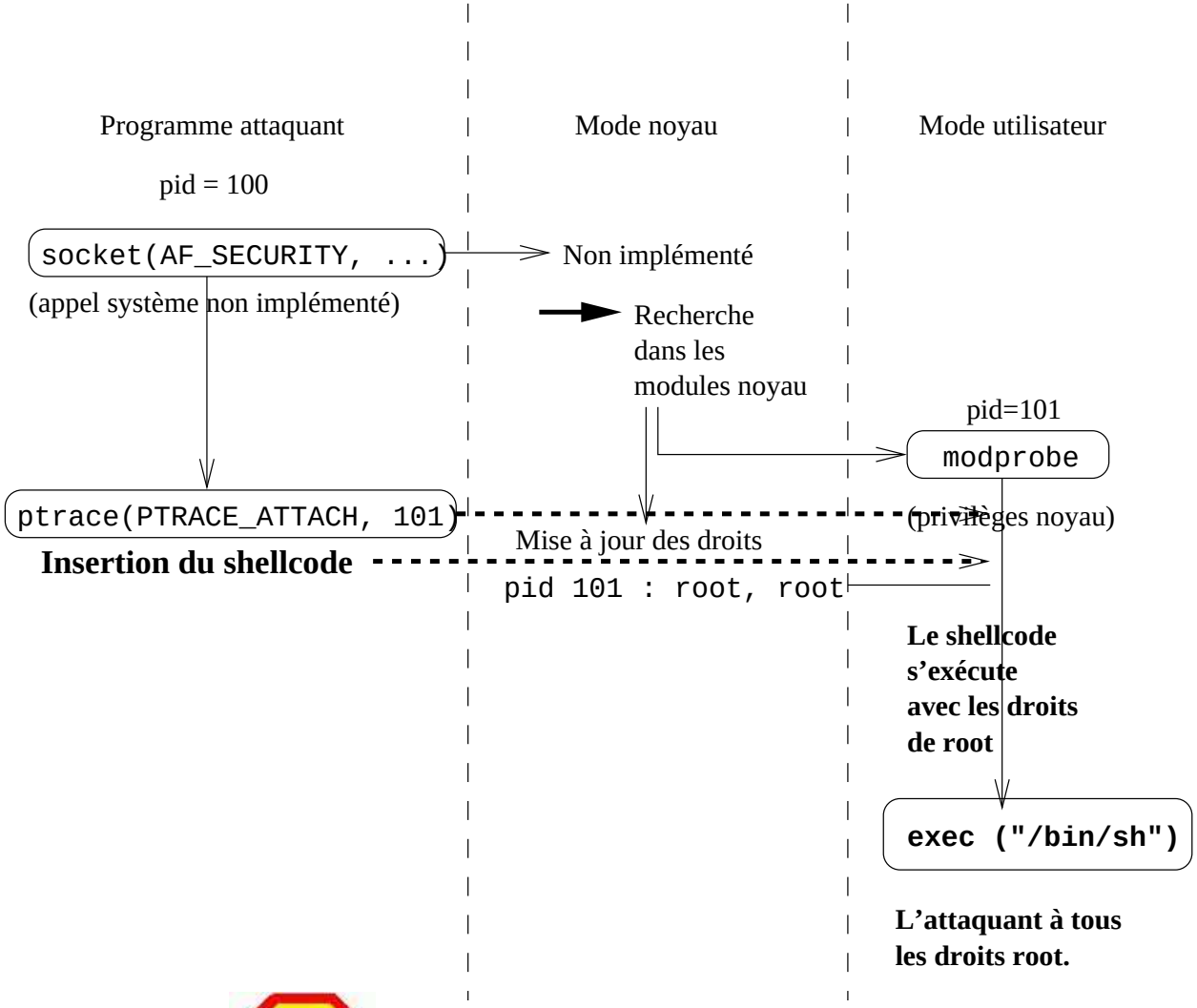
L'attaque `ptrace` (2)



L'attaque `ptrace` (3)

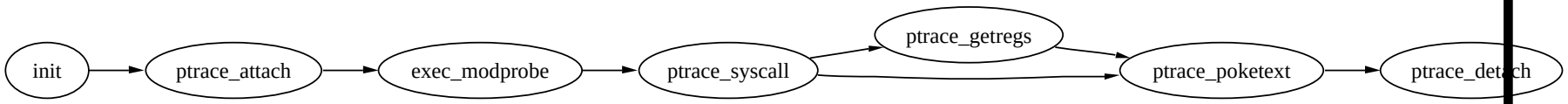


L'attaque `ptrace` (4)

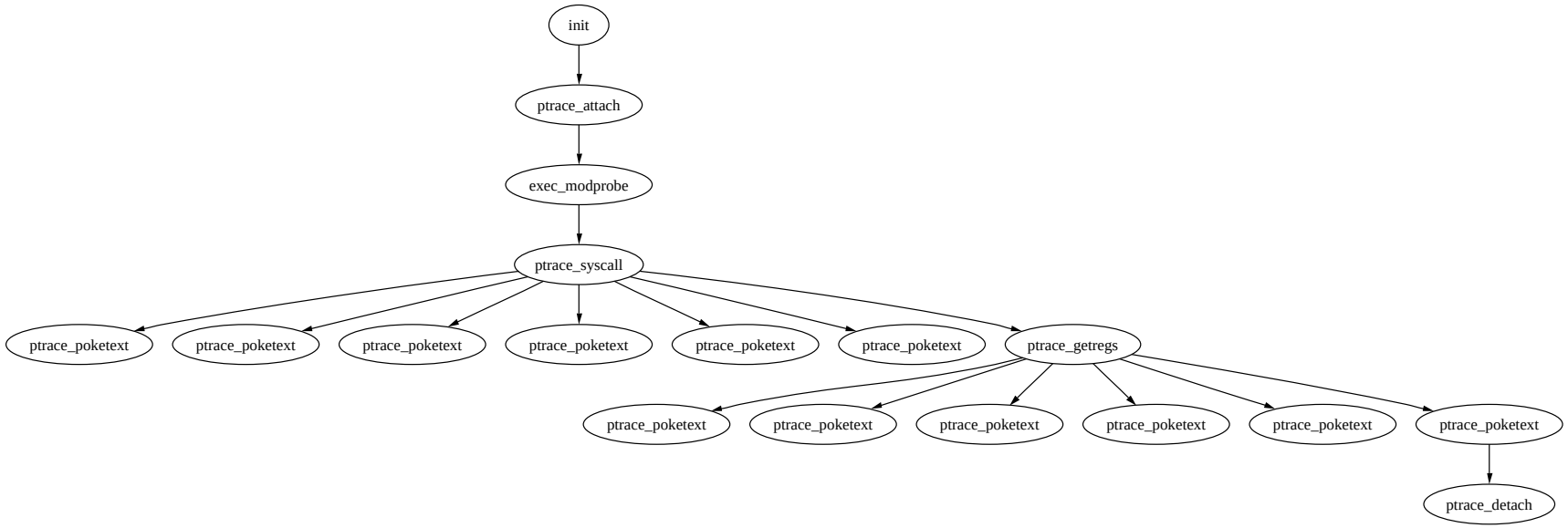


Résultats de la détection

Automate représentant le scénario:



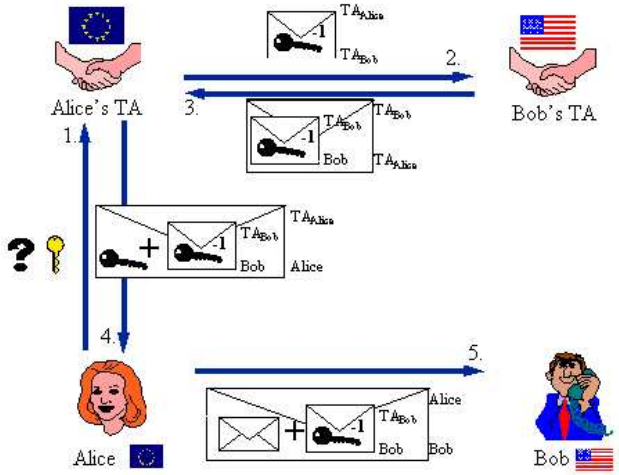
États atteints par l'instance de l'alerte:



Protocoles cryptographiques,
analyse de code,
détection d'intrusions

Jean Goubault-Larrecq

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



Projets RNTL EVA, Prouvé ACI VERNAM, Psi-Robuste, Rossignol

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