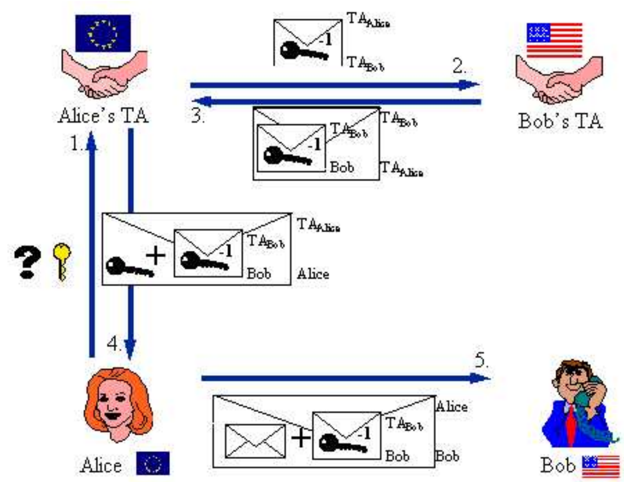
On cryptographic protocols, regular tree languages, and automated deduction

Jean Goubault-Larrecq

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



Projet RNTL EVA, RNTL Prouvé ACI VERNAM, Rossignol

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

1. **Cryptographic protocols.**
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. Conclusion.

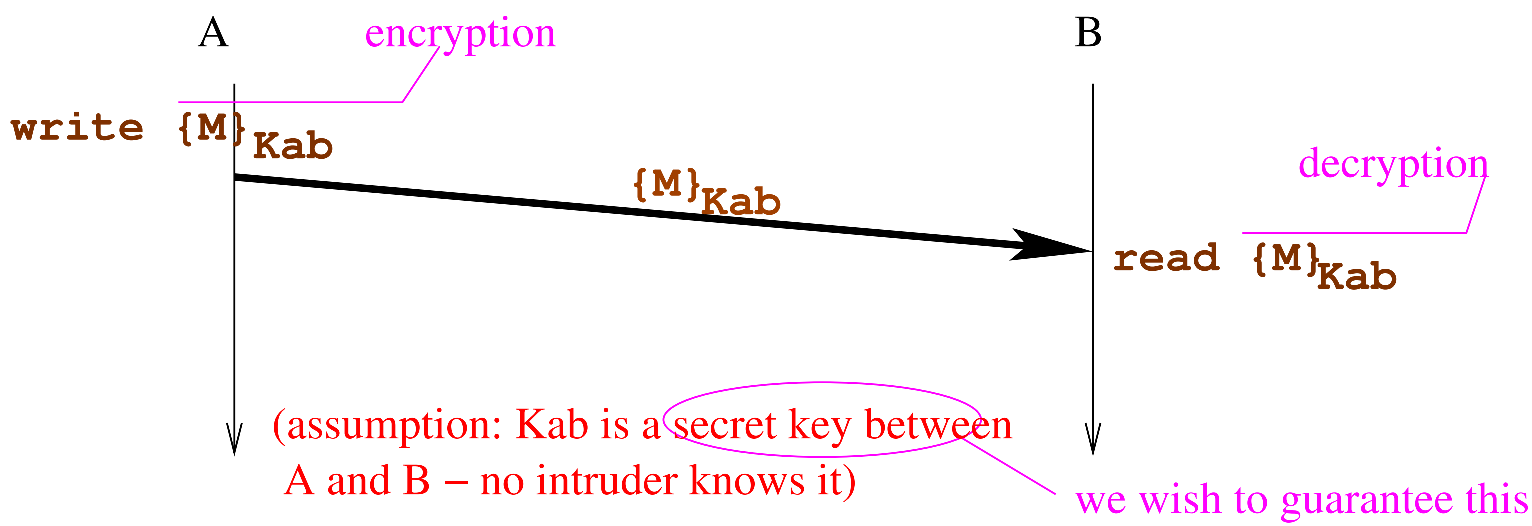
Cryptographic protocols

Increasing need for strong security: smartcards, e-banking, e-commerce, secure networks, etc.

Secrecy: M is secret if no intruder can emit M;
Authenticity: the only process that can emit M is A;
Freshness: M was built recently;
Non-duplication: M can only be received once (invoices);
Non-repudiation: A cannot deny having emitted M (orders).

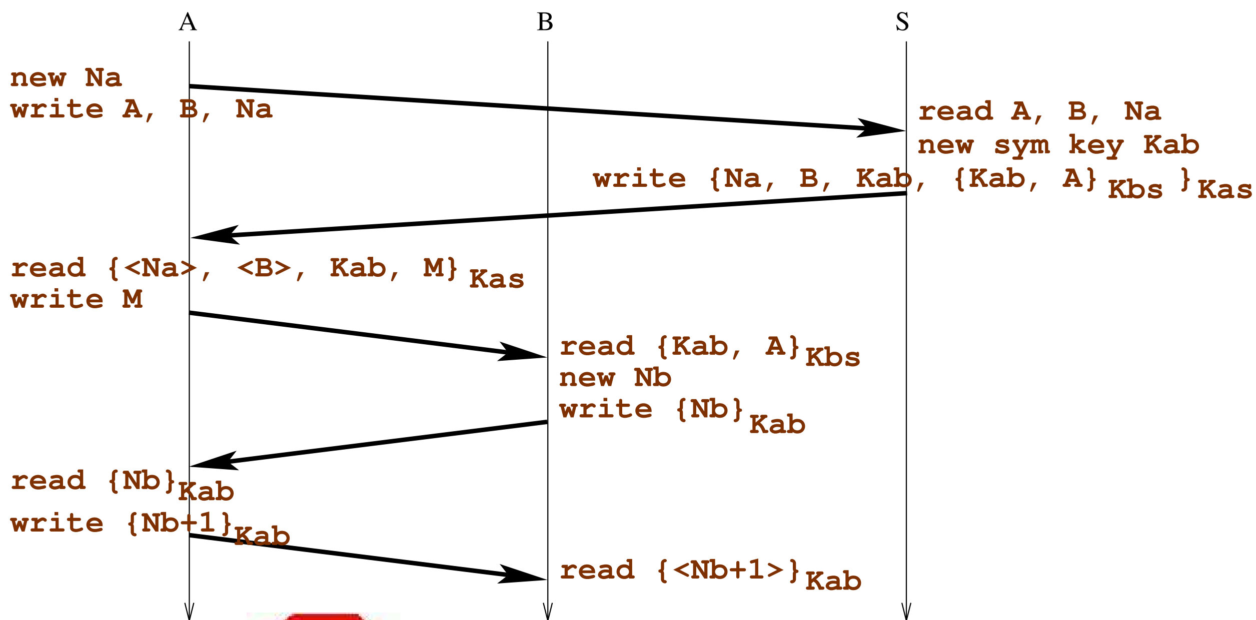
Cryptography is not enough

Even if you use perfect (unbreakable) encryption algorithms, it is not easy to preserve secrecy or authenticity:

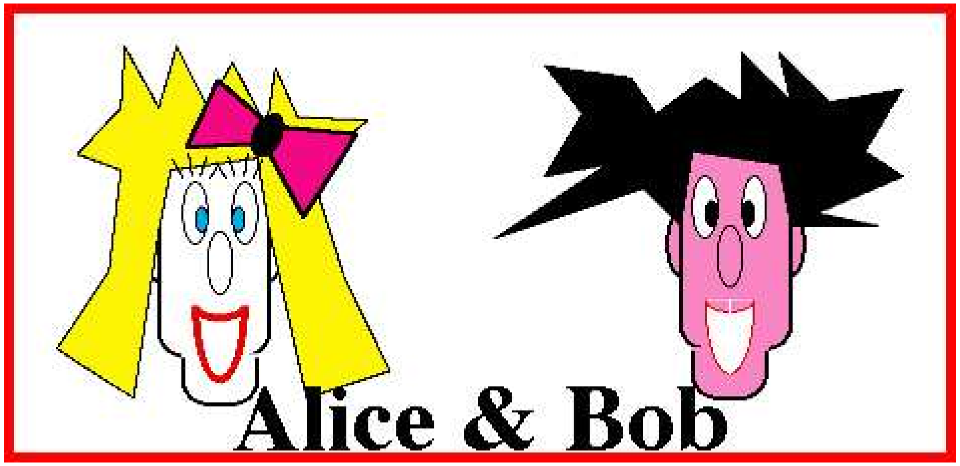


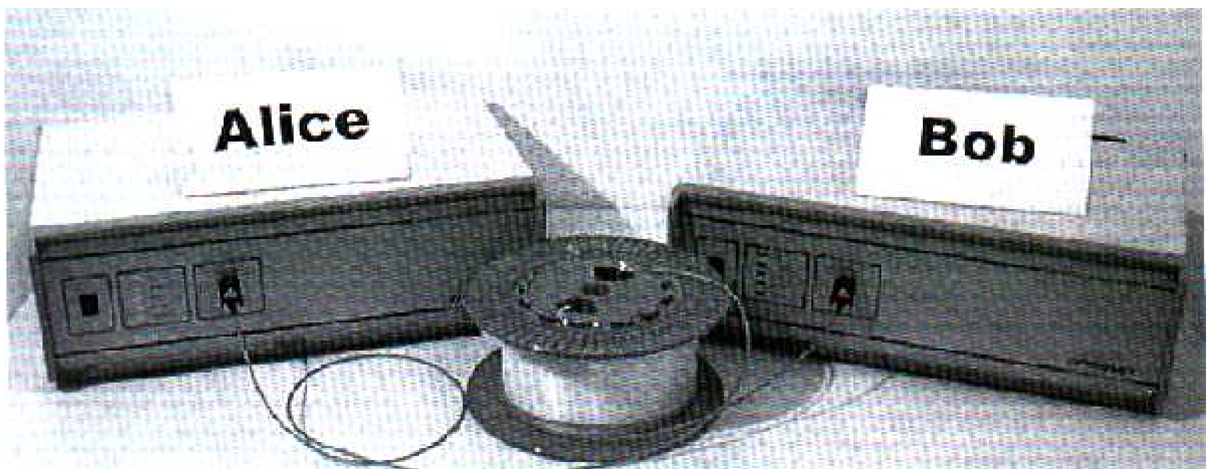
Ex.: symmetric key Needham-Schroeder

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



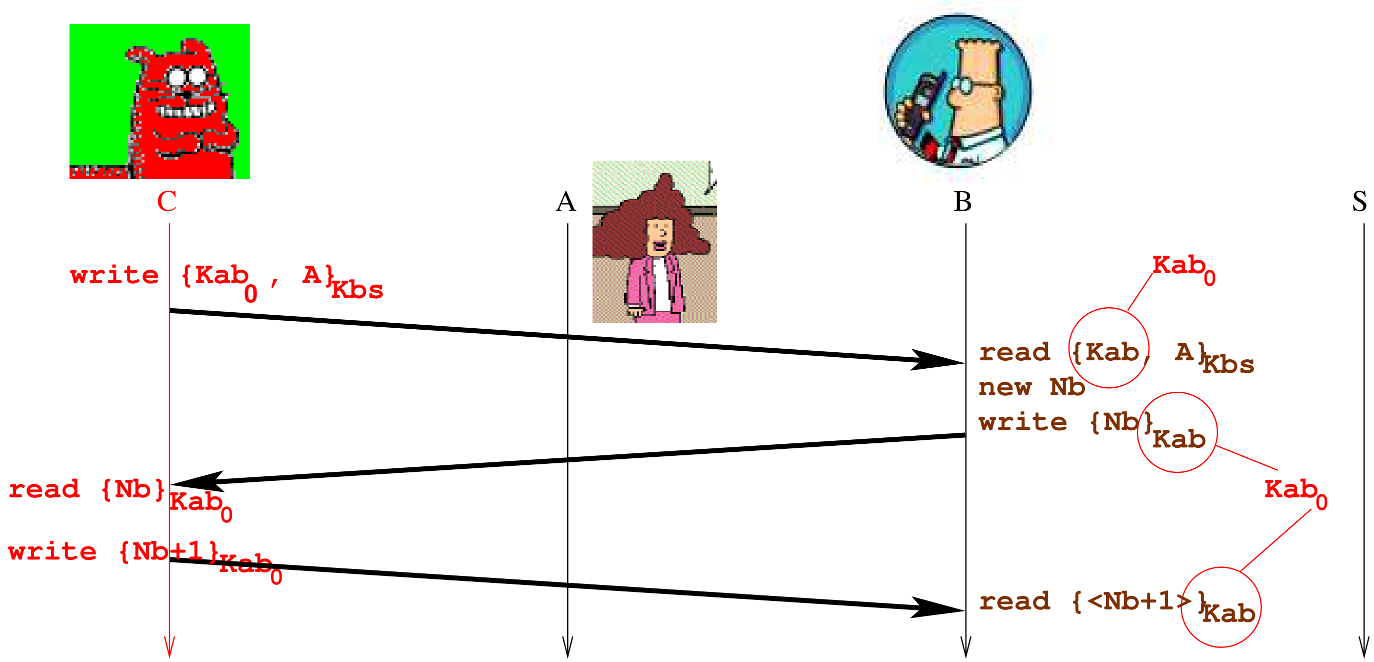
Who are Alice and Bob anyway?





An Attack

C replays an old $\{\texttt{Kab}_0\texttt{, A | Kbs}\}$ —old enough that C managed to get hold of $\texttt{Kab}_0$.



1. Cryptographic protocols.
2. **Modeling cryptographic protocols using Horn clauses.**
3. What is a security proof?
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. Conclusion.

A Horn clause (pure Prolog) model
1. Intruder abilities.

$$
\begin{array}{rcll}
\texttt{knows}(\{M\}_K) & \Leftarrow & \texttt{knows}(M), \texttt{knows}(K) & (C \text{ can encrypt}) \\
\texttt{knows}(M) & \Leftarrow & \texttt{knows}(\{M\}_{\texttt{k}(\texttt{sym},X)}), & \\
 & & \texttt{knows}(\texttt{k}(\texttt{sym}, X)) & \ldots\text{and decrypt [symmetric keys])} \\
\texttt{knows}([]) & & & (C \text{ can build} \\
\texttt{knows}(M_1 :: M_2) & \Leftarrow & \texttt{knows}(M_1), \texttt{knows}(M_2) & \text{any list of known messages)} \\
\texttt{knows}(M_1) & \Leftarrow & \texttt{knows}(M_1 :: M_2) & (C \text{ can read heads)} \\
\texttt{knows}(M_2) & \Leftarrow & \texttt{knows}(M_1 :: M_2) & (C \text{ can read tails)} \\
\texttt{knows}(\texttt{suc}(M)) & \Leftarrow & \texttt{knows}(M) & (C \text{ can add} \\
\texttt{knows}(M) & \Leftarrow & \texttt{knows}(\texttt{suc}(M)) & \text{and subtract one)}
\end{array}
$$

2. Protocol clauses—current sessions (à la Blanchet/Nielson²-Seidl)

1. $A \longrightarrow S : A, B, N_a$	$\mathtt{knows}([\mathsf{a}, \mathsf{b}, \mathsf{na}([\mathsf{a}, \mathsf{b}])])$
1. $A \rightarrow S : A, B, N_a$ 2. $S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}} \}_{K_{as}}$	$\mathtt{knows}\begin{pmatrix} \{[N_a, B, k_{ab}, \\ \{[k_{ab}, A]\}_{\mathtt{k}(\mathtt{sym},[B,\mathtt{s}])} \\ ]\}_{\mathtt{k}(\mathtt{sym},[A,\mathtt{s}])}) \end{pmatrix} \Leftarrow \mathtt{knows}([A, B, N_a])$ $(k_{ab} \equiv \mathtt{k}(\mathtt{sym}, \mathtt{cur}(A, B, N_a)))$
2. $S \longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}} \}_{K_{as}}$ 3. $A \longrightarrow B : \{K_{ab}, A\}_{K_{bs}}$	$\mathtt{knows}(M) \Leftarrow \mathtt{knows}(\{[\mathsf{na}([\mathsf{a}, \mathsf{b}]), \mathsf{b}, K_{ab}, M]\}_{\mathtt{k}(\mathtt{sym},[\mathtt{a},\mathtt{s}])})$ $\mathtt{a\_key}(K_{ab}) \Leftarrow \mathtt{knows}(\{[\mathsf{na}([\mathsf{a}, \mathsf{b}]), \mathsf{b}, K_{ab}, M]\}_{\mathtt{k}(\mathtt{sym},[\mathtt{a},\mathtt{s}])})$
3. $A \longrightarrow B : \{K_{ab}, A\}_{K_{bs}}$ 4. $B \longrightarrow A : \{N_b\}_{K_{ab}}$	$\mathtt{knows}(\{\mathsf{nb}(K_{ab}, A, B)\}_{K_{ab}}) \Leftarrow \mathtt{knows}(\{[K_{ab}, A]\}_{\mathtt{k}(\mathtt{sym},[B,\mathtt{s}])})$

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. Protocol clauses—old sessions

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

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

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

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

4. Initial intruder knowledge

$$
\begin{array}{rcl}
\texttt{agent}(\texttt{a}) & & \texttt{agent}(\texttt{b}) \\
\texttt{agent}(\texttt{s}) & & \texttt{agent}(\texttt{i}) \\
\texttt{knows}(X) & \Leftarrow & \texttt{agent}(X) \\
\texttt{knows}(\texttt{k}(\texttt{pub}, X)) & & \\
\texttt{knows}(\texttt{k}(\texttt{prv}, \texttt{i})) & & \\
\texttt{knows}(\texttt{k}(\texttt{sym}, \texttt{prev}(A, B, N_a))) & & \text{(old session keys} \\
 & & \text{are compromised)}
\end{array}
$$

5. Security queries

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

can C build K_{ab}

as created by S?

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

... as received by A?

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

... as received by B?

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. **What is a security proof?**
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. Conclusion.

Security proof = no proof

A proof of $\perp$ (false) is an attack.

… i.e., a way of running clauses 1.–5.

which enables C to eventually know some sensitive data, here.

Selinger's Thesis: Security proof $\equiv$ no proof of $\perp$.

Demo 1

If you see this slide,

please ask the speaker

to run `h1`

to find the attacks on

symmetric-key Needham-Schroeder.

In case the speaker forgets:
this finds an attack on B,
mostly and less obvious... there is no attack on either A or S.

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. **Finding security proofs.**
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. Conclusion.

Automated deduction

$\Longrightarrow$ Roadmap:

Launch some automated prover (SPASS, Otter, Vampire, Waldmeister, Bliksem, ...) on the given set of clauses 1.–5.

If $\perp$ was derived, there is a possible attack.

If the prover terminates without deriving $\perp$, no attack.

(Yes!)

If the prover does not terminate, well, er...

...this actually happens fairly often...

Note: Blanchet uses an ad hoc two-step resolution strategy

that terminates often (always on so-called tagged protocols).

You can also use finite model finders, e.g., Paradox [CS03] (very promising).

Abstraction

Basic Idea: turn the initial clause set S into a clause set S' such that:

- S' falls into a decidable subclass.

  ...I tend to like $\mathcal{H}_1$ [Nielson&Nielson&Seidl02] personally.

- S' implies S.

  ...so if S' is not contradictory, neither is S.

Great, this exists!

Forerunner is [Frühwirth&Shapiro&Vardi&Yardeni91].

This is independent of every application domain...

The $\mathcal{H}_1$ class, and the canonical abstraction

Clauses of $\mathcal{H}_1$:

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

Decidable DEXPTIME-complete.

…by ad hoc techniques [Nielson&Nielson&Seidl02]

…by ordered resolution with selection [Goubault-Larrecq03]

Defines exactly the *regular tree languages*.

…using a clause language that is much more expressive than ordinary tree automata,

even alternating tree automata,

even two-way,

…matches exactly the definite set constraints

with unrestricted (even non-linear) comprehensions.

And ………………………… all clauses 1. (intruder) are in $\mathcal{H}_1$ already.

Canonical abstraction: name subterms

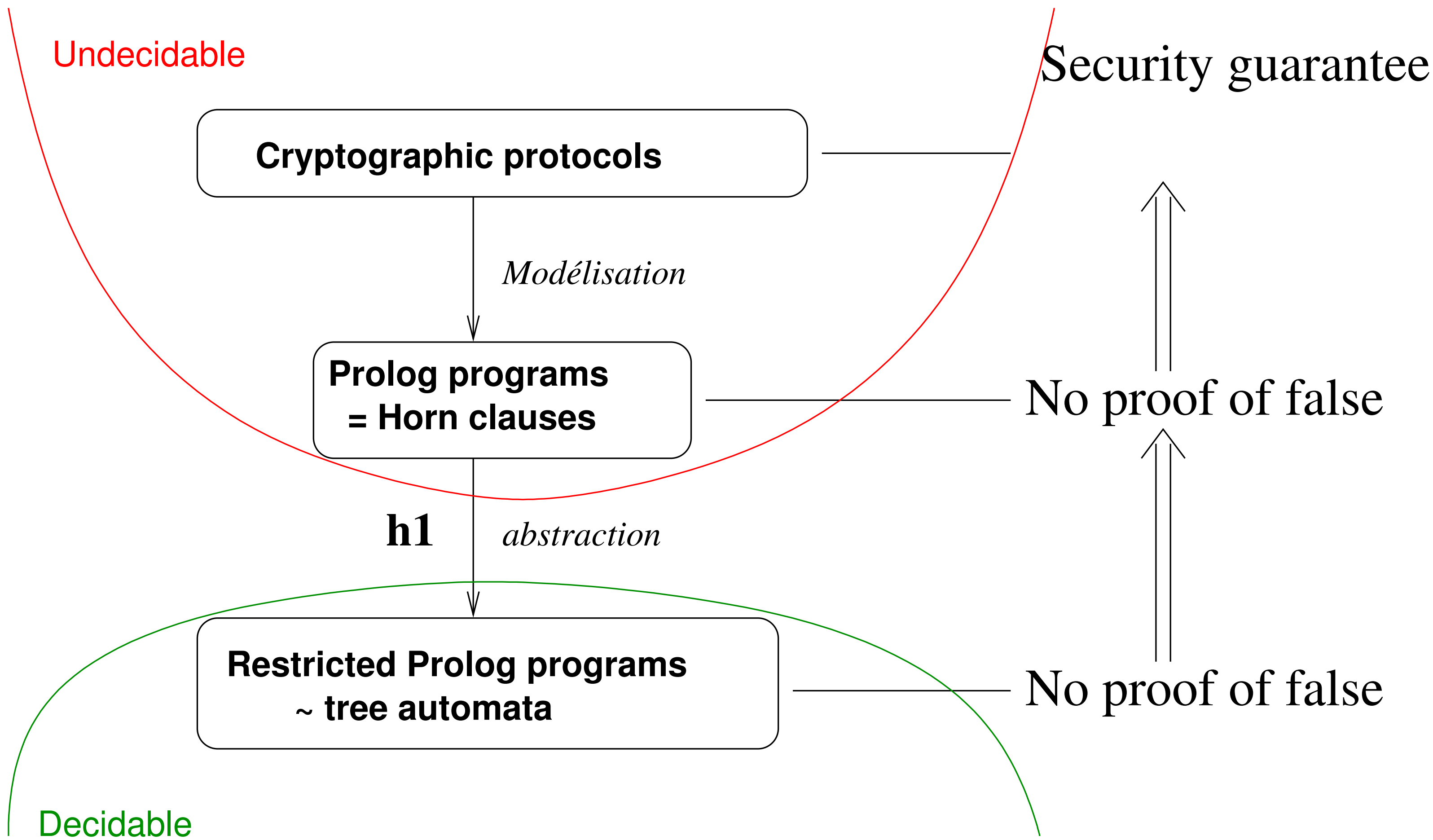
$$\mathtt{knows}\left(\begin{array}{ll}\{ & [N_a, B, \mathtt{k}(\mathtt{sym}, \mathtt{cur}(A, B, N_a)), \\ & \{[\mathtt{k}(\mathtt{sym}, \mathtt{cur}(A, B, N_a)), A]\}_{\mathtt{k}(\mathtt{sym},[B,\mathtt{s}])} \\ &]\}_{\mathtt{k}(\mathtt{sym},[A,\mathtt{s}])})\end{array}\right) \Leftarrow \mathtt{knows}([A,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

$$\longrightarrow$$

$$q_{15}(g(A, B, N_a)) \Leftarrow \mathtt{knows}([A, B, N_a])$$

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

$$\mathtt{knows}(\{X_1\}_{X_2}) \Leftarrow q_{16}(X_1), q_{17}(X_2)$$


Undecidable
Security guarantee
Cryptographic protocols
Modélisation
Prolog programs
= Horn clauses
No proof of false
h1
abstraction
Restricted Prolog programs
~ tree automata
No proof of false
Decidable

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. Finding security proofs.
5. **Deciding $\mathcal{H}_1$ using resolution.**
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. Conclusion.

Er, would you mind if I skipped this part and the next one?

Deciding $\mathcal{H}_1$ using resolution

Idea: using some specific refinement of resolution, show that only finitely many clauses can be inferred.

dates back to [Joyner76], even to [Maslov64,Mints80]

We use a pretty general refinement: ordered resolution

$$\frac{\overbrace{C \vee +A_1 \vee \ldots \vee +A_n}^{\text{main premise}} \quad \overbrace{C' \vee -A'}^{\text{side premise}}}{C\sigma \vee C'\sigma}$$

(i) $n \geq 1$;

(ii) $\sigma = \text{mgu}\,(A_1 \doteq A', \ldots, A_n \doteq A')$;

(iii) $A_1, \ldots, A_n$ are $\succ$-maximal in main;

(iv) A' is $\succ$-maximal in side.

Deciding $\mathcal{H}_1$ using resolution

Idea: using some specific refinement of resolution, show that only finitely many clauses can be inferred.

dates back to [Joyner76], even to [Maslov64,Mints80]

We use a pretty general refinement: ordered resolution with selection.

$$\frac{\overbrace{C \vee +A_1 \vee \ldots \vee +A_n}^{\text{main premise}} \quad \overbrace{C' \vee -A'}^{\text{side premise}}}{C\sigma \vee C'\sigma}$$

(i) $n \geq 1$;

(ii) $\sigma = \text{mgu}\,(A_1 \doteq A', \ldots, A_n \doteq A')$;

(iii) sel $(C \vee +A_1 \vee \ldots \vee +A_n) = \emptyset$ and $A_1, \ldots, A_n$ are $\succ$-maximal in main;

(iv) $-A' \in$ sel $(C' \vee -A')$, or sel $(C' \vee -A') = \emptyset$ and A' is $\succ$-maximal in side.

Specializing ordered resolution with selection

To decide $\mathcal{H}_1$, define:

- $P(t) \succ Q(t')$ iff t strict super-term of t';
- sel (C) is set of all literals $-P(t)$ of depth $\geq$ depth of head.

$\Rightarrow$ Main premises are:

- $P(f(X_1, \ldots, X_n)) \Leftarrow B_1(X_1), \ldots, B_n(X_n),$

$$B_{n+1}(X_{n+1}), \ldots, B_m(X_m)$$

where $B(X)$ denotes some conjunction $P_1(X), \ldots, P_k(X)$

$\ldots$ these are (almost) alternating tree automata clauses

- $P(X)$

universal clauses

Deciding $\mathcal{H}_1$ using resolution (cont'd)

E.g.,

$$\frac{\underline{P(f(X_1, X_2))} \Leftarrow Q(X_1), R(X_1), T(X_3) \quad U(X) \Leftarrow \underline{P(f(g(X,X), g(X,Y)))}, V(X))}{U(X) \Leftarrow Q(g(X,X)), R(g(X,X)), V(X), T(X_3)}$$

Conclusion is smaller than side premise (in some multiset ordering).

Deciding $\mathcal{H}_1$ using resolution (cont'd)

This may loop:

$$\frac{\underline{P(f(X_1, X_2))} \Leftarrow Q(X_1), R(X_2) \quad S(X) \Leftarrow \underline{P(X)}, T(X)}{S(f(X_1, X_2)) \Leftarrow T(f(X_1, X_2)), Q(X_1), R(X_2)}$$

Conclusion is not smaller than premisses, but at least it is not too large.

If only this happened, then we would still generate only finitely many clauses.

The need for splitting

$$\dfrac{\dfrac{Q(f(X,Y)) \Leftarrow Q'(X) \qquad \dfrac{\begin{array}{r}P(\{M\}_K) \Leftarrow Q(M), R(K)\\ S(M) \Leftarrow P(\{M\}_K), U(K)\end{array}}{S(M) \Leftarrow Q(M), R(K), U(K)}}{S(f(X,Y)) \Leftarrow Q'(X), R(K), U(K)} \qquad \begin{array}{r}\vdots\\ S'(X) \Leftarrow S(f(X,Y)),\\ R'(Y), U'(Y)\end{array}}{S'(X) \Leftarrow Q'(X), R(K), U(K), R'(Y), U'(Y)}$$

$\Rightarrow$ larger and larger clauses (no bound).

Splitting variants

- Condensing [Joyner76];
- Splitting [tableaux community]: if $C \vee C'$ holds (where $\mathrm{fv}(C) \cap \mathrm{fv}(C') = \emptyset$), then C or C' must hold.

  $\Rightarrow$ replace $C \vee C'$ non-deterministically by C or C'

  This would decide $\mathcal{H}_1 \ldots$ in NEXPTIME.
- Splittingless splitting [Voronkov&Riazanov01]: $C \vee C'$ is equivalent to $\exists q \cdot (C \vee q) \wedge (C' \vee \neg q)$.

  e.g., replace $S(M) \Leftarrow Q(M), R(K), U(K)$

  by $S(M) \Leftarrow Q(M), q$ and $q \Leftarrow P(K), U(K)$

  with $q = ne(P \cap U)$

  This decides $\mathcal{H}_1 \ldots$ in DEXPTIME (optimal).

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. **Deciding other classes using resolution.**
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. Conclusion.

Solving decidable classes using resolution: a long history

- Maslov [64] designs the inverse method, shows several classes decidable.
  Mints [80] shows that the inverse method is essentially positive hyperresolution (i.e., sel $(C) = \{\text{all negative literals of } C\}$) on a definitional clausal form [Tseitin58].

- Joyner [76] shows that ordered resolution (i.e., sel $(C) = \emptyset$) decides the monadic, Ackermann, Gödel, extended Skolem and Maslov classes.
  Note: still no resolution method decides the Bernays-Schönfinkel class!

- de Nivelle [98] introduces the guarded fragment, shows it decidable using ordered resolution.

- See chapter of HAR by Fermüller, Leitsch, Hustadt, Tammet for more info.

Positive set constraints are clause sets

Set constraint	Automatic clause
$\xi \subseteq \eta$	$-\xi(X) \vee +\eta(X)$
$\xi \subseteq \eta \cup \zeta$	$-\xi(X) \vee +\eta(X) \vee +\zeta(X)$
$\xi \cap \eta \subseteq \zeta$	$-\xi(X) \vee -\eta(X) \vee +\zeta(X)$
$\xi \subseteq \complement\eta$	$-\xi(X) \vee -\eta(X)$
$\complement\xi \subseteq \eta$	$+\xi(X) \vee +\eta(X)$
$\xi \subseteq f(\xi_1, \ldots, \xi_n)$	$\left\{\begin{array}{l} -\xi(f(X_1, \ldots, X_n)) \vee +\xi_1(X_1) \\ \ldots \\ -\xi(f(X_1, \ldots, X_n)) \vee +\xi_n(X_n) \\ -\xi(g(X_1, \ldots, X_m)) \quad \text{(for all } g \neq f) \end{array}\right.$
$f(\xi_1, \ldots, \xi_n) \subseteq \xi$	$\bigvee_{i=1}^{n} -\xi_i(X_i) \vee +\xi(f(X_1, \ldots, X_n))$
$f_i^{-1}(\xi) \subseteq \eta$	$-\xi(f(X_1, \ldots, X_n)) \vee +\eta(X_i)$

Solving first-order automatic clauses by ordered resolution

Looking at the previous slide, we have two kinds of clauses:

- *Blocks* $B(X) = \pm P_1(X) \vee \ldots \vee \pm P_m(X)$;
- *Complex clauses* $\bigvee_i \pm P_i(f_i(X_1, \ldots, X_n)) \vee B_1(X_1) \vee \ldots \vee B_n(X_n)$

Ordered resolution (with splitting) generates only finitely many such clauses.

$\Rightarrow$ terminates in NEXPTIME.

– this is optimal: the problem is NEXPTIME-complete.

– in fact this is $\sim$ a way of deciding the monadic class [Bachmair&Ganzinger&Waldmann93].

– when restricted to Horn clauses, defines languages recognized by tree automata with equality tests between brothers.

A nice extension [Limet&Salzer04]: tree tuple languages

Tree tuple languages:

$$e ::= X|\{()\}|e \times e|\Box \circ e|e/\Box$$

where $\Box$ denotes *template tuples* (e.g., $g(1,2)$).

Constraints: $X \supseteq e$.

Several subclasses shown decidable (in particular pseudo-regular TTLs) using variants of resolution + definition introduction.

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. **Equational theories, xor, Diffie-Hellman, etc.**
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. Conclusion.

The need for equational theories

See e.g., NRL analyzer (C. Meadows): handled through rewrite rules.

- E.g., the RSA rule (see this morning's talk):

$$\begin{aligned} \{\{M\}_K\}_{K^{-1}} &\rightarrow M \\ K^{-1^{-1}} &\rightarrow K \end{aligned}$$

- E.g., explicit decryption (Meadows, Millen, Blanchet, Jacquemard and Delaune, etc.):

$$\texttt{decrypt}(\{M\}_K, K^{-1}) \rightarrow M$$

Some theories resists the rewrite rule approach (see next slides).

at least if we want terminating algorithms, which you may or may not care about.

The need for equational theories — Group Diffie-Hellman

Consider a group of N people, wishing to get some key K, such that:

1. No intruder outside the group knows the key;

The need for equational theories — Group Diffie-Hellman

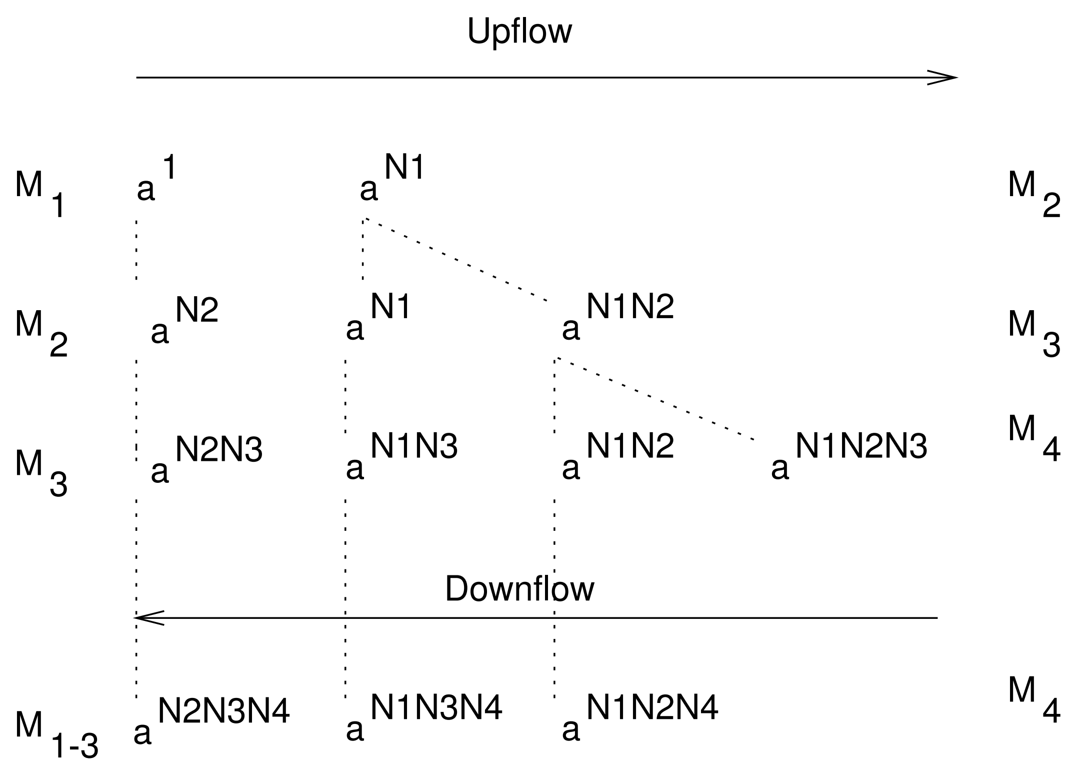
Consider a group of N people, wishing to get some key K, such that:

1. No intruder outside the group knows the key;

2. and no single person (or even no proper subgroup) can force a predicted value of K for the entire group.

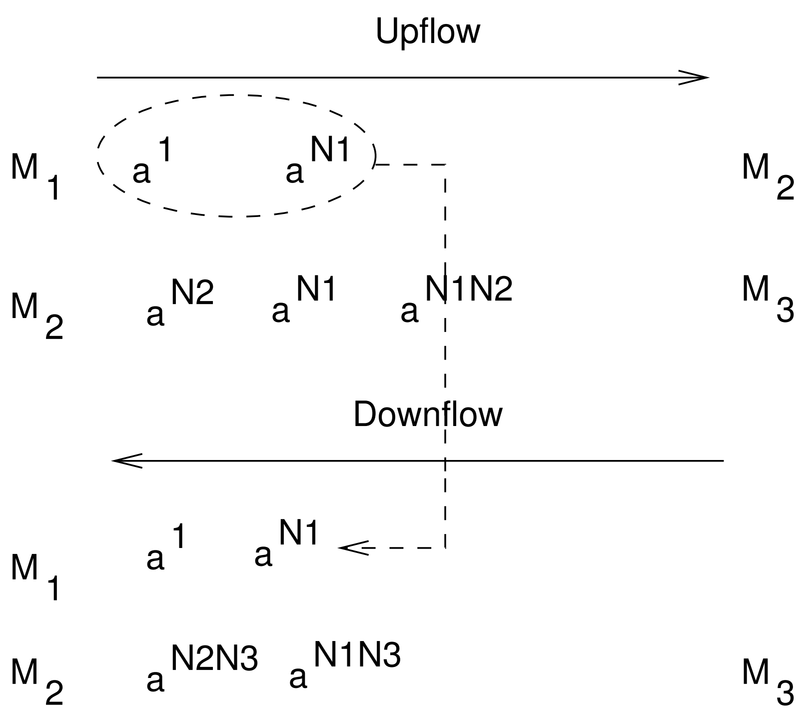
Group Diffie-Hellman: the IKA.1 protocol

(taken from [Millen&Denker02]



Group key: $a^{N1N2N3N4}$

An attack on IKA.1



M_2, M_3: $K_g = a^{N1N2N3}$

M_1: $K_g = a^{N1}$

Modular exponentiation

The IKA.1 protocol rests on Abelian group laws for exponents:

$$(a^M)^N = a^{MN} \quad M(NP) = (MN)P \quad MN = NM$$
$$1M = M1 = M \qquad MM^{-1} = 1$$

This is not handled in the free term model.

Modeling IKA.1

Encode a^M as $e(M)$, exponent multiplication as an associative-commutative (AC) symbol $\oplus$.

... possibly with unit (ACU), possibly an inverse (AbGrp).

(Main) new intruder rule:

$$\texttt{knows}(e(X \oplus Y)) \quad \Leftarrow \quad \texttt{knows}(e(X)), \texttt{knows}(Y)$$

Drawback: We still miss some specific equations, e.g. $a^M b^M = (ab)^M$.

... but see [Chevalier&Küste&Rusinowitch&Turuani03],

[Kapur&Narendran&Wang03]

Nice point: This models variants in other groups, e.g., using elliptic curve cryptography ($e(M)$ is M times some fixed point on the curve).

... close to Stern and Pointcheval's Generic Group Model [SP94].

Tree automata modulo an equational theory $\mathcal{E}$

- In case $\mathcal{E}$ is AC, ACU, or AbGrp, we recently used resolution techniques to design a complete (but unsound) approximation procedure [JGL,Roger,Verma04];

first automated verification of the IKA.1 group key establishment protocol

in the pure eavesdropper model

this approximation implemented in the MOP platform [Roger03]

- Various decidability/undecidability results known mod AC, ACU, ACI, ACUX, AbGrp, etc.;

The expert on $\mathcal{E}$-tree automata: K.N. Verma (now at TUM) 

The author of the MOP tool: M. Roger (now at CEA) 

The need for equational theories — exclusive-or (xor)

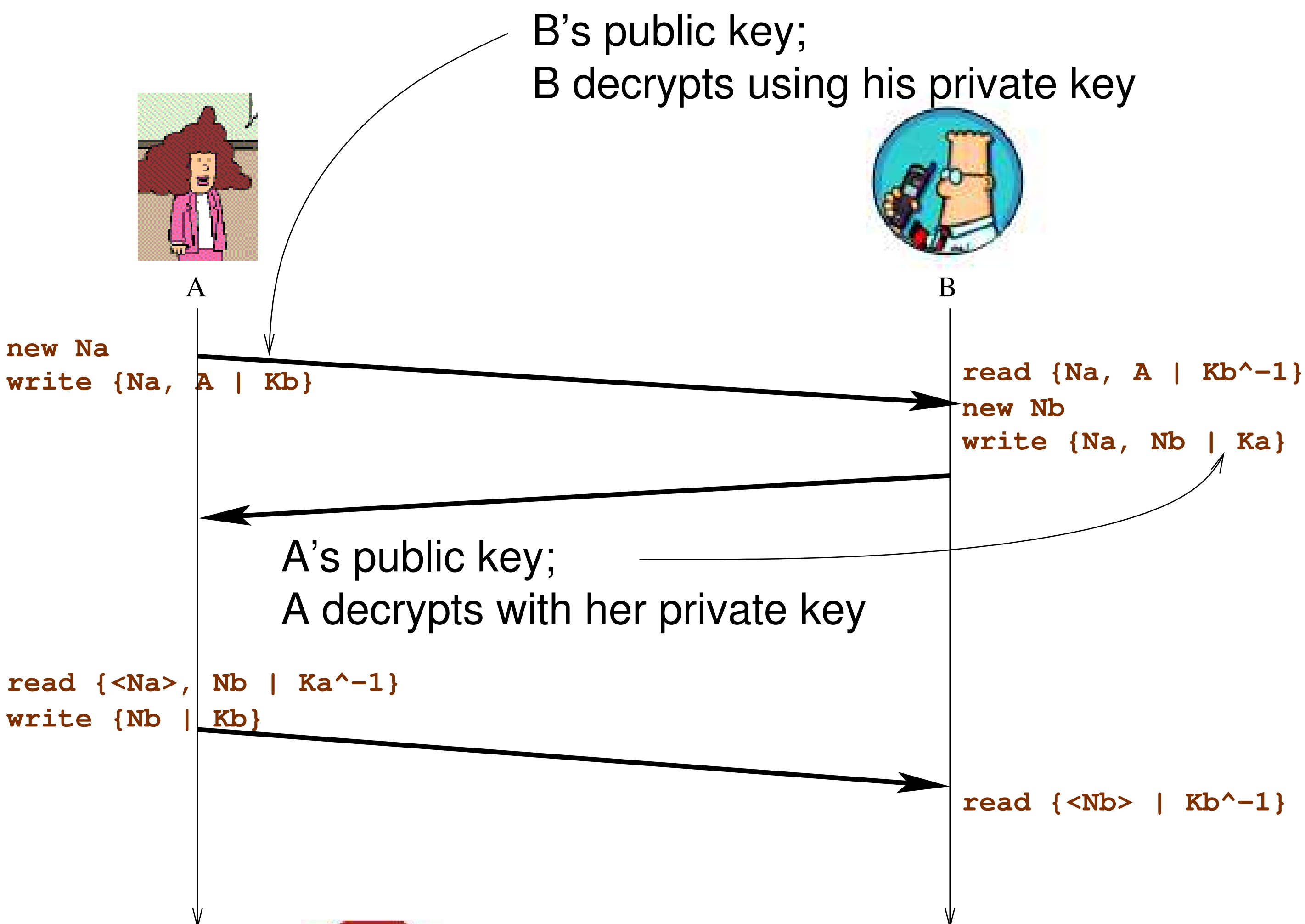
Used for various duties:

- mutual secret exchange ($A_i \to S : \{M_{A_i}\}_{K_{A_i}}$ $(i = 1, 2)$, $S \to A_i : M_1 \oplus M_2$);
- encryption (one-time pad, ElGamal encryption): encrypt M by computing $M \oplus K$.

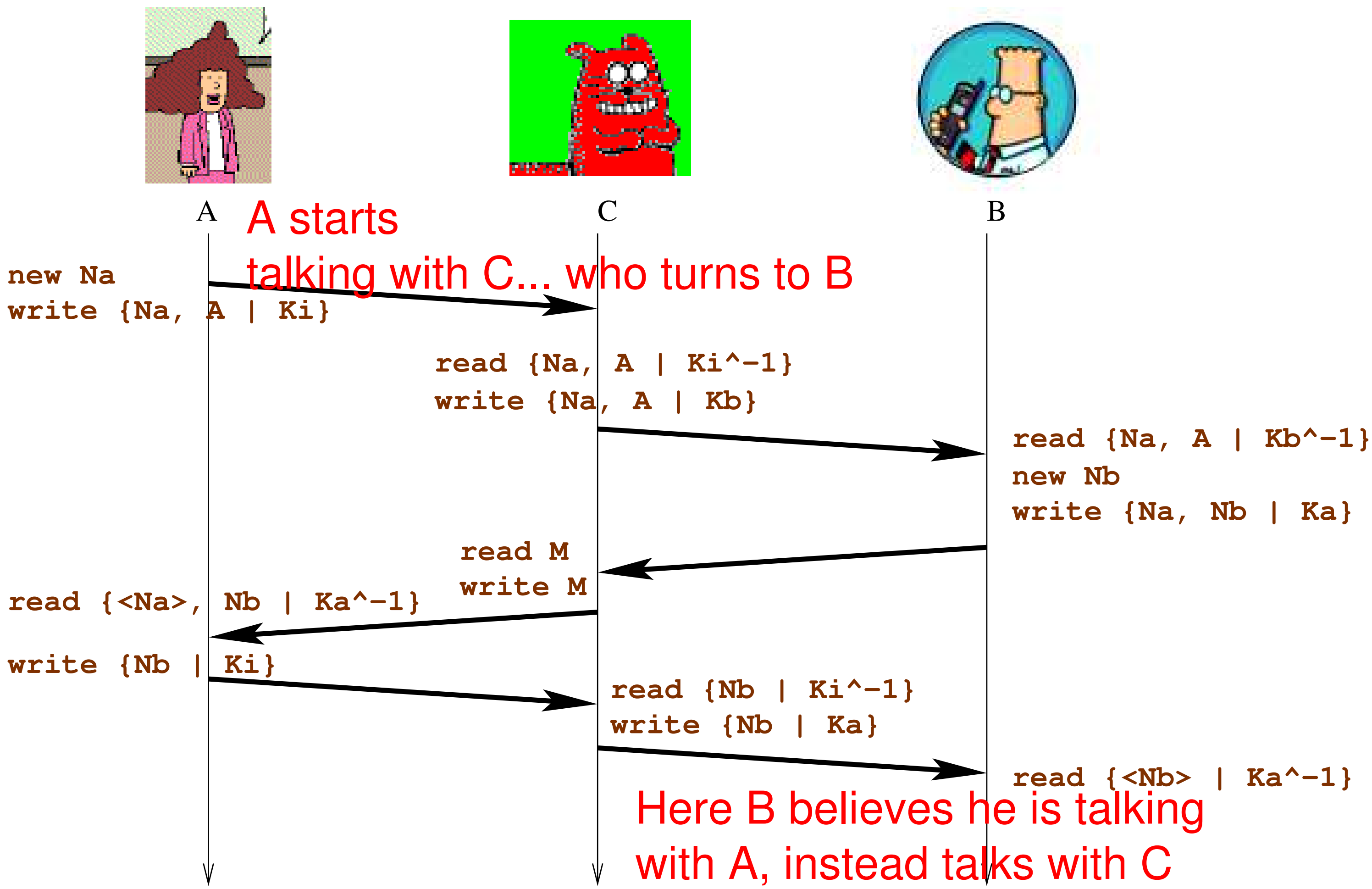
Theory of xor = ACU plus $M \oplus M = 0$.

see works by Comon and Cortier, by Rusinowitch and Turuani, by Verma.

The Needham-Schroeder public key protocol (1978)

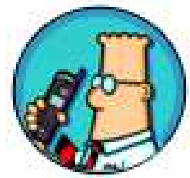


Lowe's Attack (1995)



The corrected Needham-Schroeder-Lowe protocol

A



B

A:

```
new Na
write {Na, A | Kb}
```

B:

```
read {Na, A | Kb^-1}
new Nb
write {Na, Nb, B | Ka}
```

A now checks B's identity.

A:

```
read {<Na>, Nb, <B> | Ka^-1}
write {Nb | Kb}
```

B:

```
read {<Nb> | Kb^-1}
```

The Joux attack

(I learnt it from Antoine Joux (DCSSI), sep. 2002)

- Encrypt using ElGamal encryption. Interesting point:

$$\{M\}_K \quad = \quad M \oplus K$$

modulo the theory of xor, plus the theory of homomorphism:

$$\{M_1, \ldots, M_n\}_K \quad = \quad \{M_1\}_K, \ldots, \{M_n\}_K$$

- Intruder xors second message from B with $0, 0, (B \oplus I)$ to substitute his own identity I for B. . . . this defeats Lowe's fix.

Note that ElGamal encryption is very secure, though.

- Paradox: attack works even with $\{M\}_K$ as one-time pad.

. . . the only provably secure encryption scheme!

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. **Security proofs, constructively.**
9. Formally verifying security proofs.
10. Conclusion.

Security proof = no proof (revised)

A proof of $\perp$ (false) is an attack.

… i.e., a way of running clauses 1.–5.

which enables C to eventually know some sensitive data, here.

Selinger's Thesis: Security proof $\equiv$ no proof of $\perp$.

[Selinger01], *Models for an Adversary-Centric Protocol Logic*

1st LACPV, JGL, ed., 2001.

Constructively, the non-existence of a proof will be witnessed by a model.

This is by completeness of first-order logic [Gödel1930].

(Finite models)

Example [Selinger01]: proof of Needham-Schroeder-Lowe using:

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 = known

U = unknown

W = known key,
with known inverse

etc.

The model is an invariant of every run of the protocol; it satisfies all the clauses, including the security queries.

…e.g., $\{U\}_K = K$: encrypting known data with a known key yields a (possibly) known message.

Problem left open by Selinger: find the model.

Getting models from failed proofs

Let us return to $\mathcal{H}_1$.

In case SPASS, `h1`, $\ldots$, tells you there is no proof of $\perp$, what do you do?

Idea [Tammet and others]:

- the saturated clause set must be a description of some model;
- more precisely, extracting the productive clauses (i.e., C such that sel $(C) = \emptyset$) describes a model [folklore, Bachmair&Ganzinger].

In the $\mathcal{H}_1$ case, provided you use ordered resolution with selection + splittingless splitting, the productive clauses are:

- $P(f(X_1, \ldots, X_n)) \Leftarrow B_1(X_1), \ldots, B_n(X_n)$,

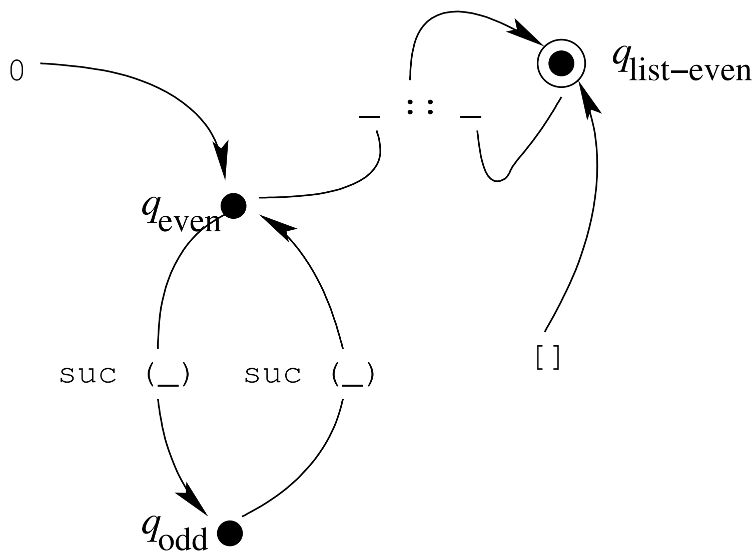
where $B(X)$ denotes some conjunction $P_1(X), \ldots, P_k(X)$

$\ldots$ these are alternating tree automata clauses

- $P(X)$

universal clauses

Tree automata and sets of Horn clauses



$\mathtt{even}(0).$
$\mathtt{odd}(\mathtt{suc}(X)) \Leftarrow \mathtt{even}(X).$
$\mathtt{even}(\mathtt{suc}(X)) \Leftarrow \mathtt{odd}(X).$
$\mathtt{listeven}(X :: Y) \Leftarrow \mathtt{even}(X), \mathtt{listeven}(Y)$
$\mathtt{listeven}([\,]).$

Non-emptiness $\Leftrightarrow$ Contradiction

(of `listeven`) (with $\perp \Leftarrow \mathtt{listeven}(X)$.)

Deterministic automata

The automaton on the previous slide is even deterministic.

Important: such automata define models.

Here the domain is $\{\texttt{even}, \texttt{odd}, \texttt{listeven}, \perp\}$.

0	even
[]	listeven

suc	
even	odd
odd	even
listeven	$\perp$
$\perp$	$\perp$

::	even	odd	listeven	$\perp$
even	$\perp$	$\perp$	listeven	$\perp$
odd	$\perp$	$\perp$	$\perp$	$\perp$
listeven	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Non-determinism, alternation

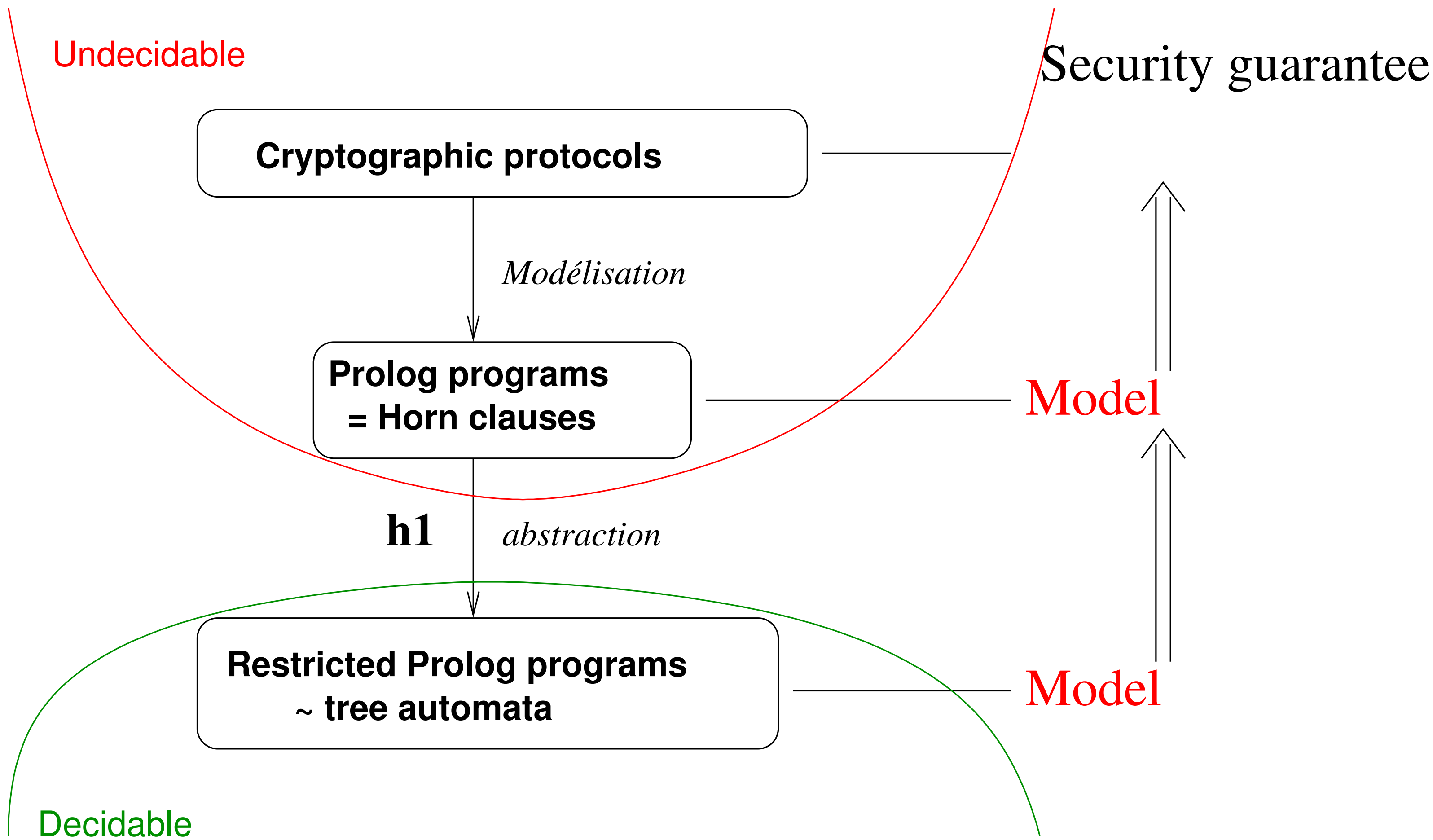
Non-determinism:

$$
\begin{array}{rcl}
\texttt{knows}(\{X_1\}_{X_2}) & \Leftarrow & \texttt{\_\_aux\_36}(X_1), \texttt{\_\_aux\_17}(X_2). \\
\texttt{\_\_aux\_20}(\{X_1\}_{X_2}) & \Leftarrow & \texttt{\_\_aux\_36}(X_1), \texttt{\_\_aux\_17}(X_2). \\
\texttt{knows}(\{X_1\}_{X_2}) & \Leftarrow & \texttt{knows}(X_1), \texttt{knows}(X_2).
\end{array}
$$

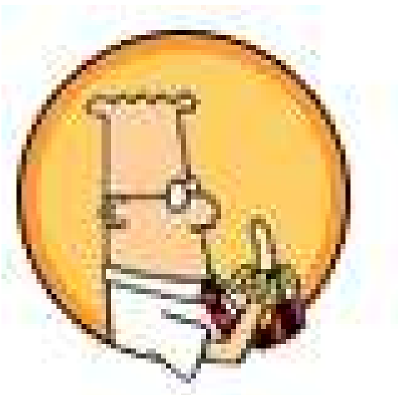
Alternation:

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

Note: alternating automata can be converted to deterministic automata (in exponential time).


Undecidable
Security guarantee
Cryptographic protocols
Modélisation
Prolog programs
= Horn clauses
Model
h1
abstraction
Restricted Prolog programs
~ tree automata
Model
Decidable

Demo 2



Here the speaker should show you the model `h1` found on symmetric-key Needham-Schroeder.

If the speaker forgets:
it is hopeless to determinize it. . .

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. **Formally verifying security proofs.**
10. Conclusion.

Checking security proofs formally [in Coq here]

Name of the game: write a Coq proof of $\mathcal{M} \models S$, where $\mathcal{M}$ is described by an alternating tree automaton $\mathcal{A}$.

First approach: Determinize $\mathcal{A}$

$\Rightarrow$ a complete deterministic tree automaton $\equiv$ a finite model $\mathcal{M}$.

Produce a proof of $\mathcal{M} \models S$ by enumerating all elements of $\mathcal{M}$ (as in Selinger's approach).

Problem 1: determinizing takes exponential time (in practice too!)

Problem 2: translating it to Coq requires some skills!

$\mathcal{M} \models S$ in Coq — $\mathcal{M}$ given explicitly

$$
\begin{array}{rl}
\textbf{Section} & \texttt{def}. \\
\textbf{Variable} & \mathbb{N} : \texttt{Set},\ 0 : \mathbb{N},\ \texttt{suc} : \mathbb{N} \to \mathbb{N}. \\
\textbf{Inductive} & \texttt{pair} : \mathbb{N} \to \texttt{Prop} := \\
 & \quad \texttt{pair\_0} : \texttt{pair}(0) \\
 & \mid\ \texttt{pair\_S} : \forall N : \mathbb{N} \cdot \texttt{impair}(N) \to \texttt{pair}(\texttt{suc}(N)) \\
\textbf{with} & \texttt{impair} : \mathbb{N} \to \texttt{Prop} := \\
 & \quad \texttt{impair\_S} : \forall N : \mathbb{N} \cdot \texttt{pair}(N) \to \texttt{impair}(\texttt{suc}(N)) \\
\textbf{End} & \texttt{def}.
\end{array}
$$

Clauses: apply to $\mathbb{N} \widehat{=} \texttt{term}$ — **Inductive** $\texttt{term} : \texttt{Set} := \texttt{0} : \texttt{term} | \texttt{S} : \texttt{term} \to \texttt{term}$.

Model: apply to $\mathbb{N} \widehat{=} D$ — defined using tables, à la Selinger.

Theorem: $\bigwedge_{C \in S} \forall \vec{v} : D^k \cdot [\![C]\!]\,[\vec{x} := \vec{v}]$ — $[\![\_]\!]$ defined using **Fixpoint**.

Proof: enumerate D^k — time $O(2^{k|S|})$.

Checking security proofs formally [in Coq here]

Name of the game: write a Coq proof of $\mathcal{M} \models S$.

Second approach: keep $\mathcal{M}$ as an alternating tree automaton.

…exponentially more succinct than finite model $\mathcal{M}$

– Check $\mathcal{M} \models S$ by model-checking first-order clauses against alternating tree automata.

DEXPTIME-complete, but … efficient in practice.

– Keep a trace of model-checking as a Coq proof.

Model-checking clauses against an alternating tree automaton

$$\frac{h; C' \vee -P(t) \quad P \text{ universal in } \pi_1}{h; C'} \; (\mathbf{Univ-})$$

Apply

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

Exact (using an ind. hyp.)

$$\frac{h; C' \vee +P(t) \quad P \text{ universal in } \pi_1}{} \; (\mathbf{Univ+})$$

Exact

$$\frac{h; C_1 \vee \ldots \vee C_n \quad (n \geq 2) \quad \text{the } C_i\text{'s being non-empty and sharing no free variable} \quad 1 \leq i \leq n}{h; C_i} \; (\mathbf{Split})$$

Cut, Tauto

$$\frac{\begin{array}{c} h; C' \vee -P(f(\vec{t})) \quad P \text{ not universal in } \pi_1 \\ \{P(f(\vec{X})) \Leftarrow D_i(\vec{X}) \\ \mid 1 \leq i \leq m\} \\ = \text{clauses in } \pi_1 \text{ with head } P(f(\vec{X})) \end{array}}{C' \Leftarrow D_1(\vec{t}) \quad \ldots \quad C' \Leftarrow D_m(\vec{t})} \; (\mathbf{Elim-/Fun})$$

Inversion, Elim, Tauto

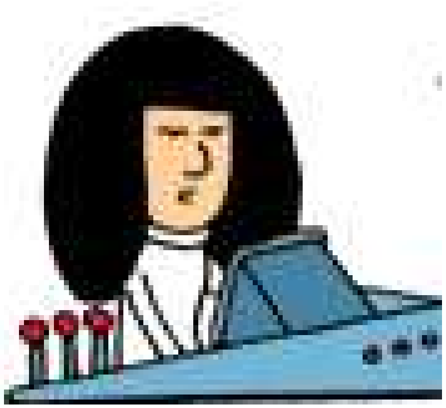
$$\frac{\begin{array}{c} h; -P(X) \vee \bigvee_{j=1}^{k} \pm_j P_j(X) \\ P, P_i \text{ not universal in } \pi_1, 1 \leq i \leq k \\ \{P(f_i(\vec{X})) \Leftarrow D_i(\vec{X}) \\ \mid 1 \leq i \leq m\} \\ = \text{clauses of } \pi_1 \text{ with head } P \\ h' = h \cup \{-P(X) \vee \bigvee_{j=1}^{k} \pm_j P_j(X)\} \\ C_i = \bigvee_{j=1}^{k} \pm_j P_j(f_i(\vec{X})) \end{array}}{h'; C_1 \Leftarrow D_1(\vec{X}) \quad \ldots \quad h'; C_m \Leftarrow D_m(\vec{X})} \; (\mathbf{Elim-/Var})$$

Fix, Case, Inversion (induction)

$$\frac{\begin{array}{c} h; C' \vee +P(f(\vec{t})) \quad P \text{ not universal in } \pi_1 \\ \{P(f(\vec{X})) \Leftarrow \bigwedge_j B_{ij}(X_j) \\ \mid 1 \leq i \leq m\} \\ = \text{clauses of } \pi_1 \text{ with head } P(f(\vec{X})) \\ \text{et } C_1 \wedge \ldots \wedge C_k \text{ is a CNF} \\ \text{of } C' \vee \bigvee_{i=1}^{m} \bigwedge_j B_{ij}(t_j) \end{array}}{h'; C_1 \quad \ldots \quad h'; C_k} \; (\mathbf{Elim+})$$

Cut, Tauto (heavy)

Demo 3



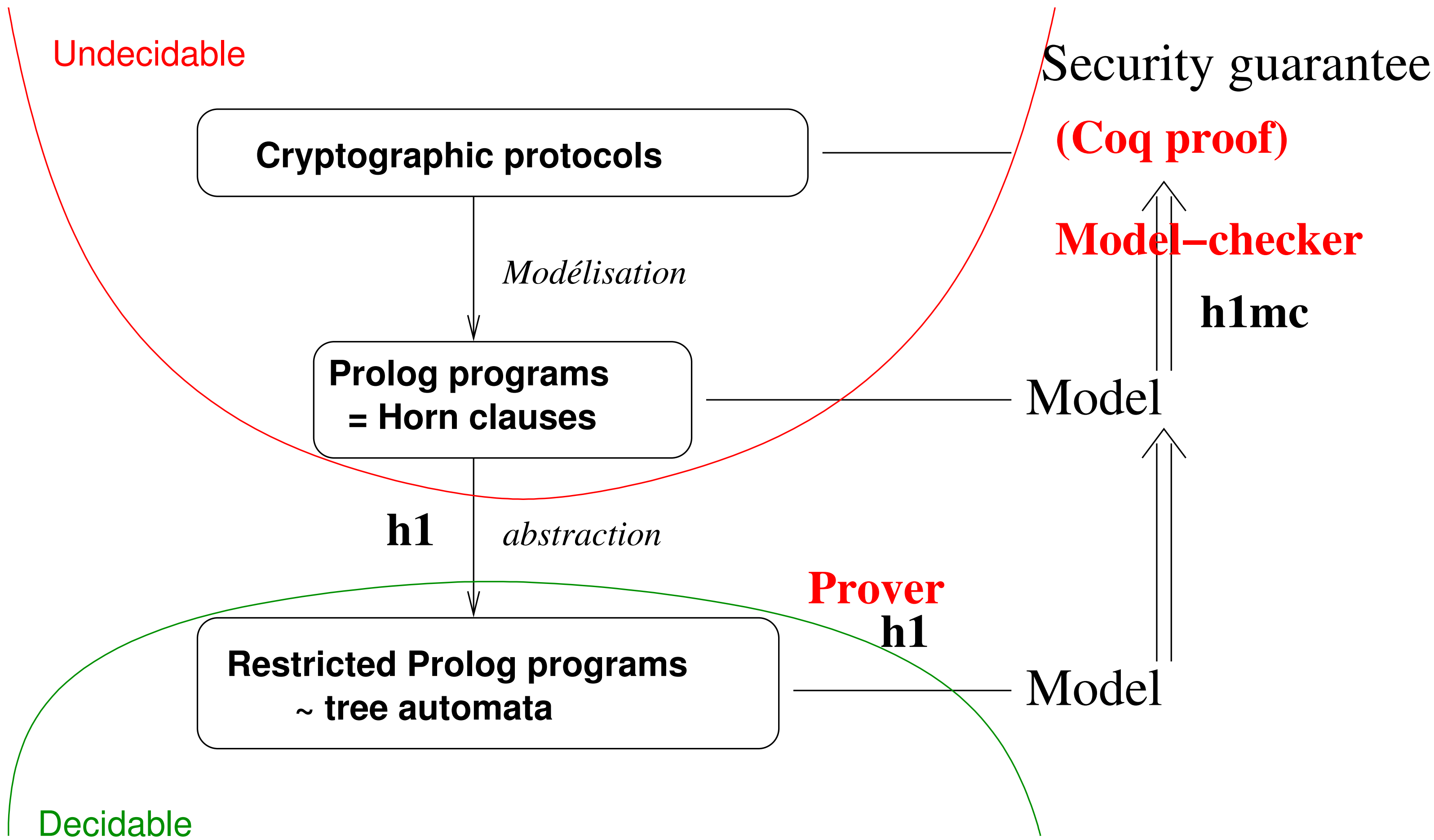
Did the speaker show you

the `h1mc` model-checker in action?

And the resulting Coq proof?

Did he showed you Coq check this proof?

To sum up


Undecidable
Cryptographic protocols
Security guarantee
(Coq proof)
Model–checker
h1mc
Modélisation
Prolog programs
= Horn clauses
Model
h1
abstraction
Prover
h1
Restricted Prolog programs
~ tree automata
Model
Decidable

1. Cryptographic protocols.
2. Modeling cryptographic protocols using Horn clauses.
3. What is a security proof?
4. Finding security proofs.
5. Deciding $\mathcal{H}_1$ using resolution.
6. Deciding other classes using resolution.
7. Equational theories, xor, Diffie-Hellman, etc.
8. Security proofs, constructively.
9. Formally verifying security proofs.
10. **Conclusion.**

Conclusion and perspectives

- **Verifying protocols is finding models:**

  How do model-finding tools fare (e.g., `Paradox` [CS03])?

  Preliminary experiments: (with Ankit Gupta, IIT Delhi)

  – works faster than `h1` for most *secure* protocols in Blanchet/Seidl style (loops on insecure protocols), produces *much* smaller (deterministic) models;

  – should adapt without problems to equational theories (under investigation);

  – clauses from *precise* models (from EVA, or from `Csur`, see next slide) *easier* for `h1` than for `Paradox`: why?

Conclusion and perspectives

- **Mathematical tools:** a nice integration:
  automated deduction/automata/model-checking/computer-aided proofs;
- **Relation between logic models and cryptographers' proofs:**
  mentioned by C. Meadows this morning, many references
  ... a simple and elegant theorem in a model with time and probabilities:
  see M. Baudet's talk (tomorrow).
- **Towards analyzing actual code:**
  most protocols exist as C/C++ code, not little diagrams!
  ...source of many attacks (buffer overflow, swapping attacks, plain bugs, ...)
  ...under investigation in the `Csur` project (with F. Parrennes, now at RATP)

Conclusion

"Logic wins!"

(Roy Dyckhoff, may 1996,

private communication,

— out of context.)