
Automatic Compilation of Protocol Insecurity Problems into Logic Programming

Luca Compagna

joint work with Alessandro Armando and Yuliya Lierler



AI-Lab - DIST - University of Genova

JELIA, Lisbon, 27-30 Sept 2004



Automated Validation of Internet Security Protocols and Applications

Shared cost RTD (FET open) project IST-2001-39252

Motivations

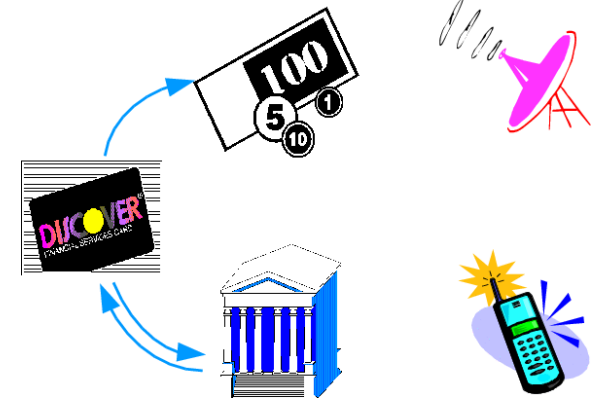
- The world is **distributed**.

E-commerce, wireless communication, ...

- **Security is critical**: breaches can be ruinous!
- Protocols are **cornerstone** of security.

Key distribution, authentication, ...

- Unfortunately, even simple protocols are **difficult** to get right by simple inspection. Therefore, protocol designers need help!
- **Idea**: use **Automated Reasoning techniques** for analyzing security protocols.



Introduction

In previous work,

- we proposed an automatic approach to **encode Security Protocols into SAT**; and
- we introduced a number of **optimizing transformations** that make the approach both **feasible** and **effective**.

Introduction

In previous work,

- we proposed an automatic approach to **encode Security Protocols into SAT**; and
- we introduced a number of **optimizing transformations** that make the approach both **feasible** and **effective**.

In this work,

- we **compile Security Protocols into Logic Programming (LP)** by re-using the optimizations transformations mentioned above; and
- we **compare** the two approaches against the Clark/Jacob library.

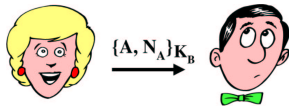
Roadmap

- Example and Protocol Analysis
- Protocol Insecurity Problems
- Encoding Protocol Insecurity Problems into Logic Programs
- Implementations and Results
- Conclusions and Perspectives

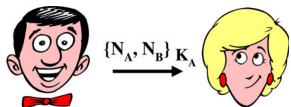
Example: Needham-Schroeder authentication protocol

1. $A \rightarrow B : \{A, N_A\}_{K_B}$ $\{\{M\}_{K_B}\}_{K_B^{-1}} = M$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A}$ $\{\{M\}_{K_B^{-1}}\}_{K_B} = M$
3. $A \rightarrow B : \{N_B\}_{K_B}$

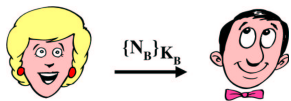
Translation:



“I am Alice and here is a Nonce N_A .”



“Here is your Nonce N_A and I also have one for you.”



“I got it! It is N_B .”

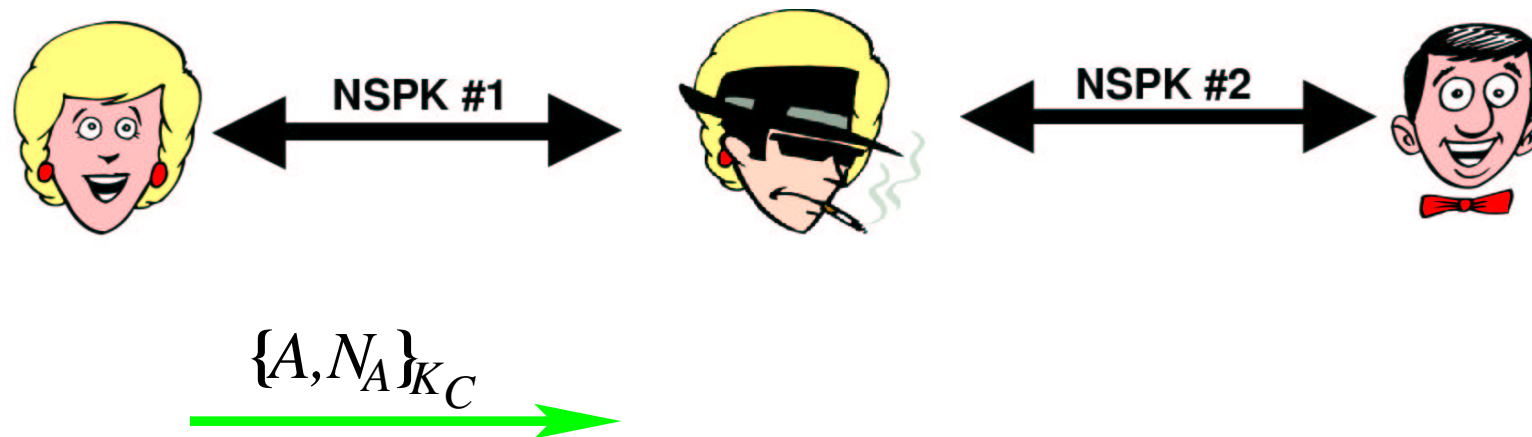
Protocols are typically small and convincing ...

... and also often wrong!

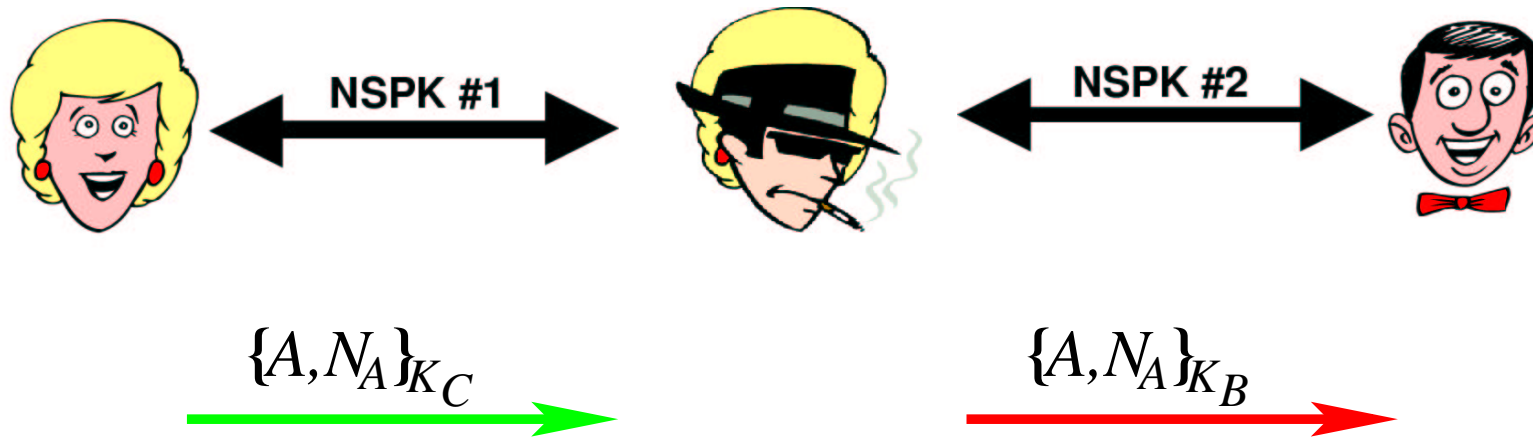
Attack on Needham-Schroeder: 2 concurrent sessions



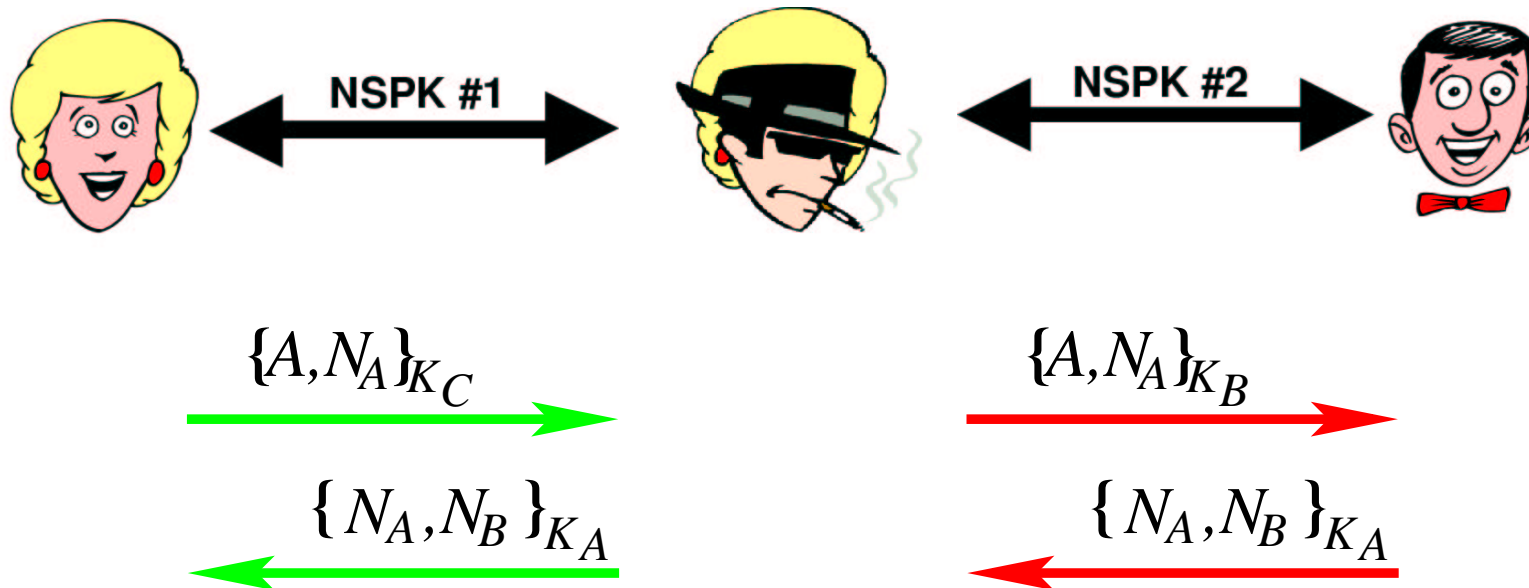
Attack on Needham-Schroeder: 2 concurrent sessions



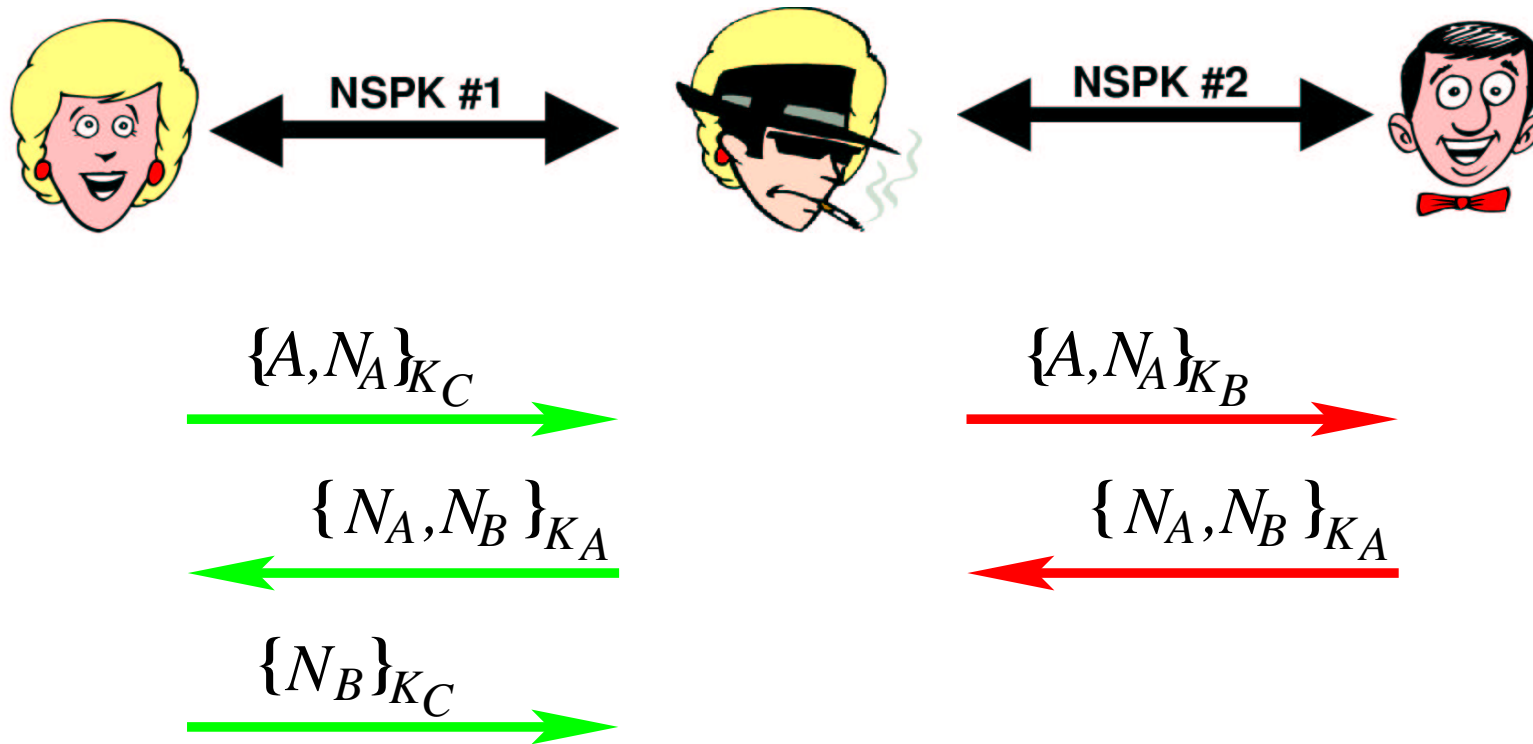
Attack on Needham-Schroeder: 2 concurrent sessions



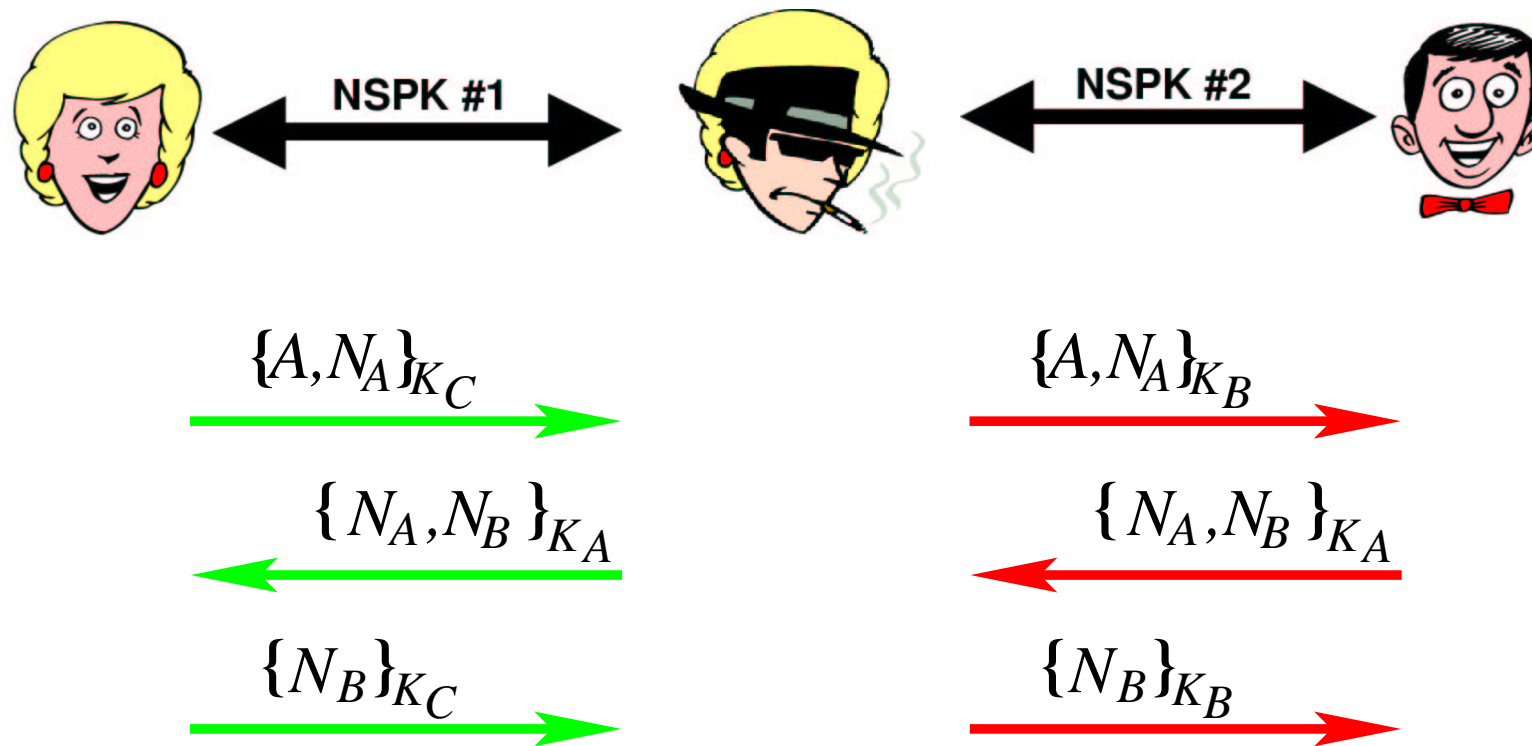
Attack on Needham-Schroeder: 2 concurrent sessions



Attack on Needham-Schroeder: 2 concurrent sessions



Attack on Needham-Schroeder: 2 concurrent sessions



In session #2, *B* believes he is speaking with *A*!

Protocol Analysis: Modeling

- Protocol as a **state transition system** in which states correspond to information possessed by participating agents.
- **Perfect cryptography**: an encrypted message can be neither altered nor read without the appropriate key.
- **Strong Typing**: all the messages considered in the analysis are assumed to be well-typed (type confusion is not allowed).
- **The Dolev-Yao intruder**:
 - controls all the traffic in the network;
 - can compose and send fraudulent messages from the knowledge he can glean from the observed traffic and his own initial knowledge.

Protocol Analysis: Security Problems

- Specified by means of the **IF rule-based language** suitable for security protocols:
 - **state**: set of facts;
 - **transition relation**: labeled rewrite rules.
- Security requirements such as **authentication** and **secrecy** are reduced to **reachability problems** on this model.
- The general verification problem is **undecidable**: we focus on **reachability problem with finite number of sessions**.
- This is adequate in practice as **attacks** on well-known protocols often exploit a **small number of sessions**.

Protocol Insecurity Problems

A **Protocol Insecurity Problem (PIP)** is a tuple $\Xi = \langle \mathcal{F}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{G} \rangle$ where:

- \mathcal{F} and \mathcal{L} are sets of atomic formula of sorted 1st-order languages called *facts* and *rule labels*, respectively;
- \mathcal{R} is a set of labeled **rewrite rules** of the form $L \xrightarrow{\ell} R$, where $L, R \subseteq \mathcal{F}$, and $\ell \in \mathcal{L}$;
- \mathcal{I} and \mathcal{G} are respectively the **initial state** and a DNF **boolean formula** representing the **bad states**.

Protocol Insecurity Problems

A **Protocol Insecurity Problem (PIP)** is a tuple $\Xi = \langle \mathcal{F}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{G} \rangle$ where:

- \mathcal{F} and \mathcal{L} are sets of atomic formula of sorted 1st-order languages called *facts* and *rule labels*, respectively;
- \mathcal{R} is a set of labeled **rewrite rules** of the form $L \xrightarrow{\ell} R$, where $L, R \subseteq \mathcal{F}$, and $\ell \in \mathcal{L}$;
- \mathcal{I} and \mathcal{G} are respectively the **initial state** and a DNF **boolean formula** representing the **bad states**.

A **(linear) solution, or attack, to a PIP** is a sequence of rules ℓ_1, \dots, ℓ_k such that $S_i \xrightarrow{\ell_i} S_{i+1}$ for $i = 1, \dots, k$ with $S_1 = \mathcal{I}$ and $S_k \models \mathcal{G}$.

Protocol Insecurity Problems

A **Protocol Insecurity Problem (PIP)** is a tuple $\Xi = \langle \mathcal{F}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{G} \rangle$ where:

- \mathcal{F} and \mathcal{L} are sets of atomic formula of sorted 1st-order languages called *facts* and *rule labels*, respectively;
- \mathcal{R} is a set of labeled **rewrite rules** of the form $L \xrightarrow{\ell} R$, where $L, R \subseteq \mathcal{F}$, and $\ell \in \mathcal{L}$;
- \mathcal{I} and \mathcal{G} are respectively the **initial state** and a DNF **boolean formula** representing the **bad states**.

A **(linear) solution, or attack, to a PIP** is a sequence of rules ℓ_1, \dots, ℓ_k such that $S_i \xrightarrow{\ell_i} S_{i+1}$ for $i = 1, \dots, k$ with $S_1 = \mathcal{I}$ and $S_k \models \mathcal{G}$.

Attacks can be compactly represented through **partial-order attack (POA)**.

E.g. $\{\ell_1, \ell_2\}, \{\ell_3\}$ represents the linear attacks ℓ_1, ℓ_2, ℓ_3 and ℓ_2, ℓ_1, ℓ_3 .

Modeling: Needham-Schroeder authentication prot. (1)

Let us model the well known NSPK protocol:

1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$

Scenario: two concurrent sessions of the protocol

session 1: a wants to talk to the intruder i ;

session 2: a wants to talk to b .

Security Requirement: B authenticates A on N_A .

Modeling: Needham-Schroeder authentication prot. (2)

States are represented as sets of the following facts:

- $fresh(N)$ means that the nonce N has not been used yet.
- $ik(T)$ means that the intruder knows T .
- $m(J, S, R, T)$ means that sender S has (supposedly) sent message T to principal R at protocol step J .
- $w(J, S, R, [T_1, \dots, T_k], C)$ represents the state of principal R at step J of session C ; it means that R
 - knows the messages stored in the lists $[T_1, \dots, T_k]$, and
 - is waiting for a message from S (if $J \neq 0$).

Modeling: Needham-Schroeder authentication prot. (3)

- Initial State:

$w(0, a, a, [a, i, ka, ka^{-1}, ki], 1)$

$\cdot w(0, a, a, [a, b, ka, ka^{-1}, kb], 2) \cdot w(1, a, b, [b, a, kb, kb^{-1}, ka], 2)$

$\cdot fresh(nc(na, 1))$

$\cdot fresh(nc(na, 2))$

$\cdot fresh(nc(nb, 2))$

$\cdot ik(i) \cdot ik(a) \cdot ik(b) \cdot ik(ki) \cdot ik(ki^{-1}) \cdot ik(ka) \cdot ik(kb)$

- Goal formula (bad states):

$w(0, a, a, [a, b, ka, ka^{-1}, kb], 2) \wedge w(1, a, b, [b, a, kb, kb^{-1}, ka], s(2))$

Modeling: Needham-Schroeder authentication prot. (4)

- Labeled Rewrite Rules:

- Behaviour of Honest Participants:

$fresh(nc(na, S)).$

$w(0, A, A, [A, B, Ka, Ka^{-1}, Kb], S) \xrightarrow{step_0(A, B, Ka, Kb, S)}$

$w(2, B, A, [nc(na, S), A, B, Ka, Ka^{-1}, Kb], S).$

$m(1, A, B, \{A, nc(na, S)\}_{Kb})$

- Behaviour of the Intruder:

$m(J, S, R, M) \xrightarrow{divert(J, M, R, S)} ik(S)$

$ik(\{M\}_K).ik(K^{-1}) \xrightarrow{decrypt(K, M)} ik(M).ik(\{M\}_K).ik(K^{-1})$

Logic Programs

A **Logic Program (LP)** is a finite set of **logic rules** (lp-rules) of the form

$$l_0 \leftarrow l_1, \dots, l_m, \text{not } l_{m+1}, \dots, \text{not } l_n$$

or

$$\{l_0\} \leftarrow l_1, \dots, l_m, \text{not } l_{m+1}, \dots, \text{not } l_n$$

where l_0 is either a literal or the symbol \perp , and l_1, \dots, l_n are literals.

Logic constraint: $\perp \leftarrow l_1, \dots, l_m, \text{not } l_{m+1}, \dots, \text{not } l_n$

Logic fact: $l_0 \leftarrow$

We interpret Logic Programs via the **answer set semantics**.

Encoding PIP into Logic Programs

Given a PIP Ξ and a positive integer k , we build a logic program P_{Ξ}^k such that any **answer set of P_{Ξ}^k** corresponds to a **partial-order attack on Ξ** .

Encoding PIP into Logic Programs

Given a PIP Ξ and a positive integer k , we build a logic program P_{Ξ}^k such that any **answer set of P_{Ξ}^k** corresponds to a **partial-order attack on Ξ** .

To do so, we:

1. add an additional **time-index** parameter to each **rule ℓ** or **fact p** , to indicate the time at which the rule executes or the fact holds.
2. build P_{Ξ}^k as **union of lp-rule schemes**:

$$P_{\Pi}^k = I_0 \cup \bigcup_{k=0}^{k-1} T_i^{i+1} \cup G_k$$

in which T_i^{i+1} is built on top of the **rewrite rules execution**, **inertia**, and **mutual exclusion** lp-rule schemes.

Implementation: SATMC

SATMC v2.0:

- input specification in **IF** rule-based language;
- set of **optimizing transformations** to get encodings of manageable size;
- **linear** and **graphplan-based encodings** with **iterative deepening** on the number of steps;
- **abstraction/refinement strategy** based on neglecting mutex relations;
- a translator from **IF to LPARSE** specifications which are fed into the logic program solvers **CMODELS** and **S MODELS**.

Download it at: <http://www.ai.dist.unige.it/satmc>

Experimental Results on Clark/Jacob library

Reduction to Logic programming (LP)

Reduction to SAT* (SAT)

Pb	K	LP EncT (IF2LP / LPARSE)	LP ATs	LP Rules	C MODELS			EncT	ATs	CLs	SolT
					ATs	CLs	LP SolT				
<i>ISO-CCF-1 U</i>	4	0.10 / 0.28	< 1	2	< 1	1	0.02	0.04	< 1	< 1	0.00
<i>ISO-CCF-2 M</i>	4	0.18 / 1.12	2	7	2	3	0.05	0.32	< 1	6	0.01
<i>ISO-PK-1 U</i>	4	0.19 / 0.95	1	4	1	2	0.03	0.12	< 1	2	0.00
<i>ISO-PK-2 M</i>	4	0.50 / 3.87	4	17	4	4	0.14	0.89	2	17	0.00
<i>ISO-SK-1 U</i>	4	0.09 / 0.29	< 1	2	< 1	1	0.02	0.04	< 1	< 1	0.00
<i>ISO-SK-2 M</i>	4	0.25 / 1.65	4	11	4	4	0.11	0.38	< 1	3	0.00
<i>NSPK</i>	7	0.46 / 7.54	12	80	12	14	0.65	2.31	7	51	0.09
<i>NSPK-server</i>	8	2.74 / 26.54	17	198	17	19	1.47	8.03	9	212	0.22
<i>SPLICE</i>	9	1.79 / 72.44	27	333	27	32	2.47	4.63	14	91	0.21
<i>Swick 1</i>	5	0.97 / 14.81	11	36	11	11	0.32	1.03	4	17	0.02
<i>Swick 2</i>	6	1.63 / 51.81	16	148	16	18	1.11	3.18	8	59	0.08
<i>Swick 3</i>	4	0.38 / 24.36	6	12	6	7	0.13	0.82	5	12	0.02
<i>Swick 4</i>	5	1.85 / 137.98	20	62	20	21	0.59	11.05	15	64	0.18
<i>Stubblebine rep</i>	3	1.40 / 371.00	29	2,010	29	30	14.37	82.93	13	2,048	0.63

(k) (k) (k) (k)

(k) (k)

(k): in thousands;

(*): using the linear encoding technique

Experimental Results: Sum-up

Reduction to Logic programming (**LP**)Reduction to SAT (**SAT**)

Pb	K	LP EncT		LP	LP	CMODELS			EncT	ATs	CLs	SoIT
		(IF2LP / LPARSE)		ATs	Rules	ATs	CLs	LP SoIT				
<i>NSPK</i>	7	0.46	7.54	12	80	12	14	0.65	2.31	7	51	0.09
<i>SPLICE</i>	9	1.79	72.44	27	333	27	32	2.47	4.63	14	91	0.21
<i>Swick 1</i>	5	0.97	14.81	11	36	11	11	0.32	1.03	4	17	0.02
<i>Swick 2</i>	6	1.63	51.81	16	148	16	18	1.11	3.18	8	59	0.08
<i>Stubblebine rep</i>	3	1.40	371.00	29	2,010	29	30	14.37	82.93	13	2,048	0.63

(k) (k) (k) (k) (k) (k)

(k): in thousands;

(*): using the linear encoding technique

- **SAT** uses a smaller number of atoms, while the number of clauses generated is in favor of CMODELS (**LP**);
- **SAT** approach outperforms the **LP** approach for what concerns timings: the time spent by LPARSE largely dominates the others.

Exploiting the expressiveness of Logic Programs

Even if the **SAT** performs better, **LP** can readily take into account:

1. an **optimized intruder model** (knowledge extended **immediately**);
2. specific **algebraic properties** of cryptographic operators (e.g. $g^{x^y} = g^{y^x}$).

by means of **axioms** (i.e. formula that states a **relation between facts** and that **holds in all reachable states**).

Exploiting the expressiveness of Logic Programs

Even if the **SAT** performs better, **LP** can readily take into account:

1. an **optimized intruder model** (knowledge extended **immediately**);
2. specific **algebraic properties** of cryptographic operators (e.g. $g^{xy} = g^{yx}$).

by means of **axioms** (i.e. formula that states a **relation between facts** and that **holds in all reachable states**).

Results:

1. in both the approaches we find **attacks** up to **3 steps shorter** thereby **saving up to 44% atoms and clauses**;
2. able to generate a logic program whose answer set corresponds to an **attack on the Diffie-Hellman** protocol.

Conclusions and Perspectives

- Proposed an **automatic** approach to **compile Security Protocols into Logic Programming**.
- Assessed the **effectiveness** of this approach on the Clark & Jacob library.
- **SAT** vs **LP**: even if **SAT performs better**, **LP** can readily take into account **specific algebraic properties** of cryptographic operators.
- Use the **SATMC grounder** instead of LPARSE for grounding the resulting logic programs.
- **AVISPA** project: **industrial-strength** technology for the **A**utomated **V**erification of **large-scale I**nternet **S**ecurity **P**rotocols and **A**pplications.
<http://www.avispa-project.org>

Thanks for you attention