# Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning*

Luca Compagna

joint work with Alessandro Armando

MRG-Lab – DIST, University of Genova

FLoC 2002 – FCS and VERIFY, Copenhagen, July 25 2002

*This work has been supported by the FET Open Assessment Project AVISS IST-2000-26410.

# Introduction

- **Context:** Dramatic speed-up of SAT solvers in the last decade: problems with thousands of variables are now solved routinely in milliseconds.

  This has led to breakthroughs in planning and hardware verification.

- **Approach:** In this work we have investigated if similar results can be obtained by applying SAT-based model-checking to security protocols.

# Roadmap

- Example.
- The Model and Protocol Insecurity Problems.
- Encoding Protocol Insecurity Problems into SAT.
- Implementation and Experimental Results.
- Conclusions and Perspectives.

# An example

Consider the following simple protocol:

$$
\begin{array}{ll}
1. & A \rightarrow B : \quad \{N_A\}_{K_{AB}} \\
2. & B \rightarrow A : \quad \{f(N_A)\}_{K_{AB}}
\end{array}
$$

# An example

Consider the following simple protocol:

$$\begin{array}{lll} 1. & A \rightarrow B: & \{N_A\}_{K_{AB}} \\ 2. & B \rightarrow A: & \{f(N_A)\}_{K_{AB}} \end{array}$$

This protocol is flawed. In fact, by executing

$$\begin{array}{lll} 1.1. & A \rightarrow I(B): & \{N_A\}_{K_{AB}} \\ 2.1 & I(B) \rightarrow A: & \{N_A\}_{K_{AB}} \\ 2.2 & A \rightarrow I(B): & \{f(N_A)\}_{K_{AB}} \\ 1.2 & I(B) \rightarrow A: & \{f(N_A)\}_{K_{AB}} \end{array}$$

$A$ believes to speak with $B$ in the first session, but $A$ is really speaking with $I$.

# Modeling

- Perfect cryptography: an encrypted message can be neither altered nor read without the appropriate key.

- The Dolev-Yao attacker:
  - 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 Specification Language

We use an expressive formalism based on first-order multiset rewriting, called IF, (see Jacquemard, Rusinowitch, and Vigneron, "Compiling and Verifying Security Protocols" in LPAR 2000) that

- supports the specification of different protocol models and properties,
- has a clear, well-defined semantics, and
- results from automatic compilation of high-level protocol specifications in the "Alice&Bob"-style notation.

# The Intermediate Format

States represented by multisets of terms of the form:

- $m(j, s, r, t)$ means that sender $s$ has (supposedly) sent message $t$ to principal $r$ at protocol step $j$.

# The Intermediate Format

States represented by multisets of terms of the form:

- $m(j, s, r, t)$ means that sender $s$ has (supposedly) sent message $t$ to principal $r$ at protocol step $j$.
- $w(j, s, r, ak, ik, c)$ represents the state of principal $r$ at step $j$ of session $c$; it means that $r$
  - knows the terms stored in the lists $ak$ (*acquired knowledge*) and $ik$ (*initial knowledge*), and
  - is waiting for a message from $s$ (if $j \neq 0$).

# The Intermediate Format

States represented by multisets of terms of the form:

- $m(j, s, r, t)$ means that sender $s$ has (supposedly) sent message $t$ to principal $r$ at protocol step $j$.
- $w(j, s, r, ak, ik, c)$ represents the state of principal $r$ at step $j$ of session $c$; it means that $r$
  - knows the terms stored in the lists $ak$ (*acquired knowledge*) and $ik$ (*initial knowledge*), and
  - is waiting for a message from $s$ (if $j \neq 0$).
- $i(t)$ means that the intruder knows $t$.

# The Intermediate Format

States represented by multisets of terms of the form:

- $m(j, s, r, t)$ means that sender $s$ has (supposedly) sent message $t$ to principal $r$ at protocol step $j$.
- $w(j, s, r, ak, ik, c)$ represents the state of principal $r$ at step $j$ of session $c$; it means that $r$
  - knows the terms stored in the lists $ak$ (*acquired knowledge*) and $ik$ (*initial knowledge*), and
  - is waiting for a message from $s$ (if $j \neq 0$).
- $i(t)$ means that the intruder knows $t$.
- $fresh(n)$ means that $n$ has not been used yet.

# The IF: an example

- Initial State:

  $w(0, alice, alice, [], [alice, bob, kab], 1) \cdot w(1, alice, bob, [], [bob, alice, kab], 1) \cdot \ldots \cdot fresh(nc(na, 1)) \cdot i(alice) \cdot i(bob)$

# The IF: an example

- Initial State:

$$w(0, alice, alice, [], [alice, bob, kab], 1) . w(1, alice, bob, [], [bob, alice, kab], 1) . \ldots . fresh(nc(na, 1)) . i(alice) . i(bob)$$

- Protocol Rules: $w(0, xA, xA, [], [xA, xB, xK], xC) . fresh(nc(na, xC)) \Rightarrow m(1, xA, xB, \{nc(na, xC)\}_{xK}) . w(2, xB, xA, [nc(na, xC)], [xA, xB, xK], xC)$

# The IF: an example

- Initial State:

$$w(0, alice, alice, [], [alice, bob, kab], 1) . w(1, alice, bob, [], [bob, alice, kab], 1) . \ldots . fresh(nc(na, 1)) . i(alice) . i(bob)$$

- Protocol Rules:

$$w(0, xA, xA, [], [xA, xB, xK], xC) . fresh(nc(na, xC)) \Rightarrow m(1, xA, xB, \{nc(na, xC)\}_{xK}) . w(2, xB, xA, [nc(na, xC)], [xA, xB, xK], xC)$$

- Intruder Rules:

$$m(xj, xs, xr, xmsg) \Rightarrow i(xs) . i(xr) . i(xmsg)$$
$$i(\{xmsg\}_{xK}) . i(xK) \Rightarrow i(xmsg) . i(\{xmsg\}_{xK}) . i(xK)$$

# The IF: an example

- Initial State:

$$w(0, alice, alice, [], [alice, bob, kab], 1) . w(1, alice, bob, [], [bob, alice, kab], 1) . \ldots . fresh(nc(na, 1)) . i(alice) . i(bob)$$

- Protocol Rules: $w(0, xA, xA, [], [xA, xB, xK], xC) . fresh(nc(na, xC)) \Rightarrow m(1, xA, xB, \{nc(na, xC)\}_{xK}) . w(2, xB, xA, [nc(na, xC)], [xA, xB, xK], xC)$

- Intruder Rules:

$$m(xj, xs, xr, xmsg) \Rightarrow i(xs) . i(xr) . i(xmsg)$$
$$i(\{xmsg\}_{xK}) . i(xK) \Rightarrow i(xmsg) . i(\{xmsg\}_{xK}) . i(xK)$$

- Bad States: $w(0, alice, alice, [], [alice, bob, kab], s(1)) . w(1, alice, bob, [], [bob, alice, kab], 1)$

# Protocol Insecurity Problems

A Protocol Insecurity Problem is a tuple $\langle \mathcal{S}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{B} \rangle$ where:

- $\mathcal{S}$ is a set of atomic formulae of a sorted first-order language called *facts*;
- $\mathcal{L}$ is a set of function symbols called *rule labels*;
- $\mathcal{R}$ is a set of (deterministic) labelled rewrite rules of the form $L \xrightarrow{l} R$, where $L, R \subseteq \mathcal{S}$, and $l \in \mathcal{L}$;
- $\mathcal{I}$ and $\mathcal{B}$ are the initial state and a set of bad states.

A solution to a Protocol Insecurity Problem is a sequence $S_1 \xrightarrow{l_1} S_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{n-1}} S_n$, where $l_i \in \mathcal{L}$, $\mathcal{I} \equiv S_1$, and there exists $S_{\mathcal{B}} \in \mathcal{B}$ such that $S_{\mathcal{B}} \subseteq S_n$.

# Encoding Protocol Insecurity Problems into SAT

The reduction of Protocol Insecurity Problems to SAT is carried out in two phases:

1. the Protocol Insecurity Problem is translated into Planning Problem;

2. the Planning Problem is then encoded into SAT, using standard encoding techniques, with iterative deepening on the number of steps.

# Planning Problems

A planning problem is a tuple $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, G \rangle$, where:

- $\mathcal{F}$ and $\mathcal{A}$ are sets of ground atomic formulas called fluents and actions respectively.
- $Ops$ is a set of expressions of the form

$$Pre \xrightarrow{Act} Add; Del,$$

  where $Act \in \mathcal{A}$, and $Pre$, $Add$, and $Del$ are finite sets of fluents such that $Add \cap Del = \emptyset$.

- $I$ and $G$ are boolean combinations of fluents representing the initial state and the final states respectively.

A solution to a planning problem is a sequence of actions whose execution leads from the initial state to a final state and the preconditions of each action hold in the state to which it applies.

# Protocol Insecurity Problems as Planning Problems

Map IF rewrite rules into "actions":

| IF rules | STRIPS operators |
|---|---|
| $B \stackrel{P}{\Longrightarrow} A$ | $B \stackrel{P}{\longrightarrow} A; \neg B$ |
| $A, B \stackrel{Q}{\Longrightarrow}$ | $A, B \stackrel{Q}{\longrightarrow} ; \neg A, \neg B$ |
| $A, B \stackrel{R}{\Longrightarrow} A$ | $A, B \stackrel{R}{\longrightarrow} ; \neg B$ |

# Encoding Planning Problems into SAT (1)

**Fact:** Given,

- a planning problem $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, G \rangle$, and

- a positive integer $n$,

then it is possible to build a propositional formula $\Phi_\Pi^n$ such that any model of $\Phi_\Pi^n$ corresponds to a partial-order plan of $\Pi$.

**Intuition:** add an additional time-index parameter to each action or fluent, to indicate the state at which the action begins or the fluent holds.

# Encoding Planning Problems into SAT (2)

The formula $\Phi_\Pi^n$ is given by the conjunction of the following axioms:

**Universal Axioms:** for each action $\alpha \in \mathcal{A}$ s.t. $(Pre \xrightarrow{\alpha} Add; Del) \in Ops$ and for each $i = 0, \ldots, n-1$

$$
\begin{array}{lcl}
\alpha_i & \supset & \bigwedge\{p_i \mid p \in Pre\} \\
\alpha_i & \supset & \bigwedge\{p_{i+1} \mid p \in Add\} \\
\alpha_i & \supset & \bigwedge\{\neg p_{i+1} \mid p \in Del\}
\end{array}
$$

**Cardinality:** $O(n|\mathcal{A}||\mathcal{F}|)$, but usually $O(n|\mathcal{A}|r)$, where $r$ is the maximal number of fluents mentioned in an operator (usually a small number).

# Encoding Planning Problems into SAT (3)

**Explanatory Frame Axioms:** for all fluents $p \in F$ and for each $i = 1, \ldots, n-1$

$$(p_i \wedge \neg p_{i+1}) \supset \bigvee \left\{ \alpha_i \mid \alpha \in \mathcal{A}, Pre \xrightarrow{\alpha} Add; Del \in Ops, p \in Del \right\}$$

$$(\neg p_i \wedge p_{i+1}) \supset \bigvee \left\{ \alpha_i \mid \alpha \in \mathcal{A}, Pre \xrightarrow{\alpha} Add; Del \in Ops, p \in Add \right\}$$

**Cardinality:** $O(n|\mathcal{F}||\mathcal{A}|)$, but usually $O(n|\mathcal{F}|k)$, where $k$ is a small number.

# Encoding Planning Problems into SAT (4)

**Conflict Exclusion Axioms:** for each $i = 0, \ldots, n-1$

$$\neg(\alpha_i \wedge \alpha'_i)$$

for all $\alpha, \alpha' \in \mathcal{A}$ s.t. $\alpha \neq \alpha'$, $Pre \xrightarrow{\alpha} Add; Del \in Ops$, $Pre' \xrightarrow{\alpha'} Add'; Del' \in Ops$, and $Pre \cap Del' \neq \emptyset$ or $Pre' \cap Del \neq \emptyset$.
**Cardinality:** $O(n|\mathcal{A}|^2)$

**Initial State Axioms:** $I_0$, i.e. the formula $I$ in which each occurrence of a fluent is replaced by a fluent time-indexed with $0$.
**Cardinality:** $O(|\mathcal{F}|)$.

**Goal State Axioms:** $G_n$, i.e. the formula $G$ in which each occurrence of a fluent is replaced by a fluent time-indexed with $n$.
**Cardinality:** depending from the structure of $G$, typically small.

# Is a direct application of this encoding feasible for our task?

# Is a direct application of this encoding feasible for our task?

# No!

# Is a direct application of this encoding feasible for our task?

# No!

A number of optimizing transformations need to be done in order to get encodings of manageable size.

# Optimizations

- ☑ Language Specialization.
- ☑ Fluent Splitting.
- ☑ Exploiting Static Fluents.
- ☑ Invariant-based Simplification.
- ☑ Reducing the Number of Conflict Exclusion Axioms.
- ☑ Step-Compression (`step_compression`).
- ☐ Building Encryption Properties into the Encoding.
- ☐ Exploiting Mutual Exclusion of $w$-terms.
- ☐ Unit Propagation.

# Optimizations: Fluent Splitting

- Since $\langle j, s, r, c\rangle$ is a key for $w(j, s, r, ak, ik, c)$, replace $w(j, s, r, ak, ik, c)$ with the conjunction of (new predicates) $wk(j, s, r, ak, c)$ and $inknw(j, s, r, ik, c)$.

- Similar considerations allow us to simplify $inknw(j, s, r, ik, c)$ to $inknw(r, ik, c)$.

- Replace $wk(j, s, r, [ak_1, \ldots, ak_l], c)$ with:

$$wk(j, s, r, ak_1, 1, c), \ldots, wk(j, s, r, ak_l, l, c)$$

The number of $wk$ terms reduces from $O(|\texttt{text}|^l)$ to $O(|l| * |\texttt{text}|)$.

# Optimizations: Reducing the Number of Conflict Exclusion Axioms

**Problem:** The number of Conflict Exclusion Axioms grows quadratically in the number of actions.

**Solution:** exploit the monotonicity of the intruder knowledge. Since a monotonic fluent never appears in the delete list of some action, then it cannot be a cause of a conflict.

**Example:** IF rewrite rules of the form:

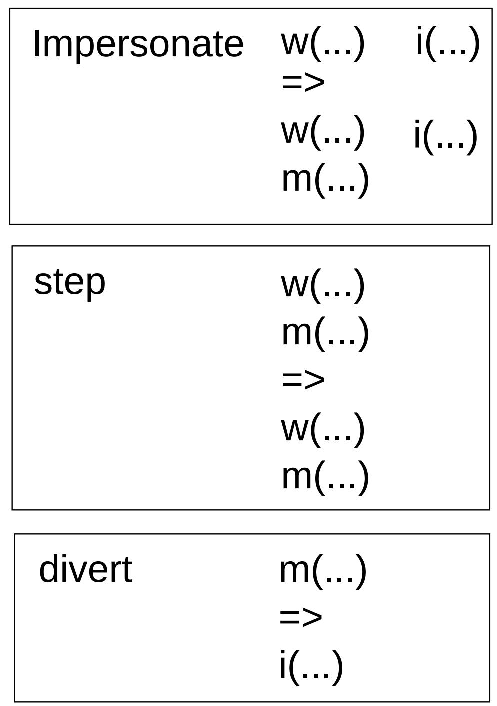
$$i(\langle xM1, xM2 \rangle) \Rightarrow i(xM1) . i(xM2)$$

are replaced by:

$$i(\langle xM1, xM2 \rangle) \Rightarrow i(xM1) . i(xM2) . i(\langle xM1, xM2 \rangle)$$

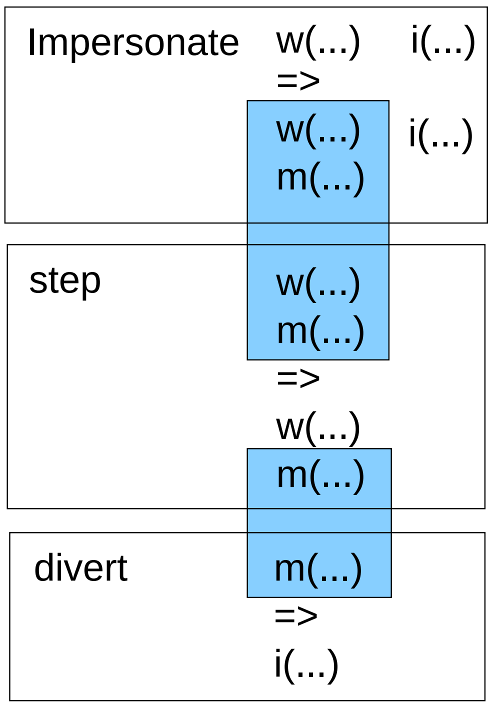
# Optimizations: Step-Compression

Simple form of partial-order reduction: significant savings by compressing these rule triples.



# Optimizations: Step-Compression

Simple form of partial-order reduction: significant savings by compressing these rule triples.



# Optimizations: Step-Compression

Simple form of partial-order reduction: significant savings by compressing these rule triples.

| | |
|---|---|
| Impersonate | w(...) |
| | i(...) |
| step | => |
| | w(...) |
| divert | i(...) |

# Introduction of Bounds and Constraints

- Bounding the Number of Session Runs (`session_repetitions`).
- Constraining Variable Instantiation (`constrain_rule_variables`).

**Note:** These may introduce incompleteness, which can be overcome by searching (using iterative deepening) the parameter space.

# Constraining Variable Instantiation

Let us consider part of the Kao-Chow protocol:

$$2.\quad S \rightarrow B : \{A, B, Na, Kab\}Kas, \{A, B, Na, Kab\}Kbs$$

$$3.\quad B \rightarrow A : \{A, B, Na, Kab\}Kas, \{Na\}Kab, Nb$$

$B$ cannot check that the occurrence of $A$ in the first component is equal to that inside the second. As a matter of fact, we might have different terms at those positions.

This constraint imposes that the occurrences of $A$ (as well as of $B$, $Na$, and $Kab$) in the first and in the second part of the message must coincide.

For instance, messages of the form

$$m(2, a, b, c(\{a, b, nc(na, s(1))\}kas, \{a, b, nc(nb, s(1))\}kbs)))$$

would be ruled out by the constraint.

# Implementation

- IF2SATE: a translator from the IF to SATE (a STRIPS-like language).

  6,200 lines of Prolog code.

- SATE: a compiler from SATE to SAT (DIMACS format).

  2,800 lines of Prolog code.

**NOTE:** state-of-the-art tools carrying out the SAT encoding of planning problems (Medic, BlackBox) are unable to handle STRIPS languages with complex term structure (only individual constants allowed).

# Experiments: Results

| Protocol | Atoms | Clauses | EncTime | SolvTime |
|---|---|---|---|---|
| *ISO symmetric key 1-pass unilateral authentication* | 679 | 2,073 | 0.18 | 0.00 |
| *ISO symmetric key 2-pass mutual authentication* | 1,970 | 7,382 | 0.43 | 0.01 |
| *Andrew Secure RPC Protocol* | 161,615 | 2,506,889 | 80.57 | 2.65 |
| *ISO CCF 1-pass unilateral authentication* | 649 | 2,033 | 0.17 | 0.00 |
| *ISO CCF 2-pass mutual authentication* | 2,211 | 10,595 | 0.46 | 0.00 |
| *Needham-Schroeder Conventional Key* | 126,505 | 370,449 | 29.25 | 0.39 |
| *Woo-Lam* $\Pi$ | 7,988 | 56,744 | 3.31 | 0.04 |
| *Woo-Lam Mutual Authentication* | 771,934 | 4,133,390 | 1,024.00 | 7.95 |
| *Needham-Schroeder Signature protocol* | 17,867 | 59,911 | 3.77 | 0.05 |
| *Neuman Stubblebine repeated part* | 39,579 | 312,107 | 15.17 | 0.21 |
| *Kao Chow Repeated Authentication, 1* | 50,703 | 185,317 | 16.34 | 0.17 |
| *Kao Chow Repeated Authentication, 2* | 586,033 | 1,999,959 | 339.70 | 2.11 |
| *Kao Chow Repeated Authentication, 3* | 1,100,428 | 6,367,574 | 1,288.00 | MO |
| *ISO public key 1-pass unilateral authentication* | 1,161 | 3,835 | 0.32 | 0.00 |
| *ISO public key 2-pass mutual authentication* | 4,165 | 23,883 | 1.18 | 0.01 |
| *Needham-Schroeder Public Key* | 9,318 | 47,474 | 1.77 | 0.05 |
| *Needham-Schroeder Public Key with key server* | 11,339 | 67,056 | 4.29 | 0.04 |
| *SPLICE/AS Authentication Protocol* | 15,622 | 69,226 | 5.48 | 0.05 |
| *Encrypted Key Exchange* | 121,868 | 1,500,317 | 75.39 | 1.78 |
| *Davis Swick Private Key Certificates, protocol 1* | 8,036 | 25,372 | 1.37 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 2* | 12,123 | 47,149 | 2.68 | 0.03 |
| *Davis Swick Private Key Certificates, protocol 3* | 10,606 | 27,680 | 1.50 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 4* | 27,757 | 96,482 | 8.18 | 0.13 |

# Experiments: Results

| Protocol | Atoms | Clauses | EncTime | SolvTime |
|---|---|---|---|---|
| *ISO symmetric key 1-pass unilateral authentication* | 679 | 2,073 | 0.18 | 0.00 |
| *ISO symmetric key 2-pass mutual authentication* | 1,970 | 7,382 | 0.43 | 0.01 |
| *Andrew Secure RPC Protocol* | 161,615 | 2,506,889 | 80.57 | 2.65 |
| *ISO CCF 1-pass unilateral authentication* | 649 | 2,033 | 0.17 | 0.00 |
| *ISO CCF 2-pass mutual authentication* | 2,211 | 10,595 | 0.46 | 0.00 |
| *Needham-Schroeder Conventional Key* | 126,505 | 370,449 | 29.25 | 0.39 |
| *Woo-Lam* $\Pi$ | 7,988 | 56,744 | 3.31 | 0.04 |
| *Woo-Lam Mutual Authentication* | 771,934 | 4,133,390 | 1,024.00 | 7.95 |
| *Needham-Schroeder Signature protocol* | 17,867 | 59,911 | 3.77 | 0.05 |
| *Neuman Stubblebine repeated part* | 39,579 | 312,107 | 15.17 | 0.21 |
| ***Kao Chow Repeated Authentication, 1*** | **50,703** | **185,317** | **16.34** | **0.17** |
| *Kao Chow Repeated Authentication, 2* | 586,033 | 1,999,959 | 339.70 | 2.11 |
| *Kao Chow Repeated Authentication, 3* | 1,100,428 | 6,367,574 | 1,288.00 | MO |
| *ISO public key 1-pass unilateral authentication* | 1,161 | 3,835 | 0.32 | 0.00 |
| *ISO public key 2-pass mutual authentication* | 4,165 | 23,883 | 1.18 | 0.01 |
| *Needham-Schroeder Public Key* | 9,318 | 47,474 | 1.77 | 0.05 |
| *Needham-Schroeder Public Key with key server* | 11,339 | 67,056 | 4.29 | 0.04 |
| *SPLICE/AS Authentication Protocol* | 15,622 | 69,226 | 5.48 | 0.05 |
| *Encrypted Key Exchange* | 121,868 | 1,500,317 | 75.39 | 1.78 |
| *Davis Swick Private Key Certificates, protocol 1* | 8,036 | 25,372 | 1.37 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 2* | 12,123 | 47,149 | 2.68 | 0.03 |
| *Davis Swick Private Key Certificates, protocol 3* | 10,606 | 27,680 | 1.50 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 4* | 27,757 | 96,482 | 8.18 | 0.13 |

# Experiments: Results

| Protocol | Atoms | Clauses | EncTime | SolvTime |
|---|---|---|---|---|
| *ISO symmetric key 1-pass unilateral authentication* | 679 | 2,073 | 0.18 | 0.00 |
| *ISO symmetric key 2-pass mutual authentication* | 1,970 | 7,382 | 0.43 | 0.01 |
| *Andrew Secure RPC Protocol* | 161,615 | 2,506,889 | 80.57 | 2.65 |
| *ISO CCF 1-pass unilateral authentication* | 649 | 2,033 | 0.17 | 0.00 |
| *ISO CCF 2-pass mutual authentication* | 2,211 | 10,595 | 0.46 | 0.00 |
| *Needham-Schroeder Conventional Key* | 126,505 | 370,449 | 29.25 | 0.39 |
| *Woo-Lam* $\Pi$ | 7,988 | 56,744 | 3.31 | 0.04 |
| *Woo-Lam Mutual Authentication* | 771,934 | 4,133,390 | 1,024.00 | 7.95 |
| *Needham-Schroeder Signature protocol* | 17,867 | 59,911 | 3.77 | 0.05 |
| *Neuman Stubblebine repeated part* | 39,579 | 312,107 | 15.17 | 0.21 |
| *Kao Chow Repeated Authentication, 1* | 50,703 | 185,317 | 16.34 | 0.17 |
| ***Kao Chow Repeated Authentication, 2*** | **586,033** | **1,999,959** | **339.70** | **2.11** |
| *Kao Chow Repeated Authentication, 3* | 1,100,428 | 6,367,574 | 1,288.00 | MO |
| *ISO public key 1-pass unilateral authentication* | 1,161 | 3,835 | 0.32 | 0.00 |
| *ISO public key 2-pass mutual authentication* | 4,165 | 23,883 | 1.18 | 0.01 |
| *Needham-Schroeder Public Key* | 9,318 | 47,474 | 1.77 | 0.05 |
| *Needham-Schroeder Public Key with key server* | 11,339 | 67,056 | 4.29 | 0.04 |
| *SPLICE/AS Authentication Protocol* | 15,622 | 69,226 | 5.48 | 0.05 |
| *Encrypted Key Exchange* | 121,868 | 1,500,317 | 75.39 | 1.78 |
| *Davis Swick Private Key Certificates, protocol 1* | 8,036 | 25,372 | 1.37 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 2* | 12,123 | 47,149 | 2.68 | 0.03 |
| *Davis Swick Private Key Certificates, protocol 3* | 10,606 | 27,680 | 1.50 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 4* | 27,757 | 96,482 | 8.18 | 0.13 |

# Experiments: Results

| Protocol | Atoms | Clauses | EncTime | SolvTime |
|---|---|---|---|---|
| *ISO symmetric key 1-pass unilateral authentication* | 679 | 2,073 | 0.18 | 0.00 |
| *ISO symmetric key 2-pass mutual authentication* | 1,970 | 7,382 | 0.43 | 0.01 |
| *Andrew Secure RPC Protocol* | 161,615 | 2,506,889 | 80.57 | 2.65 |
| *ISO CCF 1-pass unilateral authentication* | 649 | 2,033 | 0.17 | 0.00 |
| *ISO CCF 2-pass mutual authentication* | 2,211 | 10,595 | 0.46 | 0.00 |
| *Needham-Schroeder Conventional Key* | 126,505 | 370,449 | 29.25 | 0.39 |
| *Woo-Lam* $\Pi$ | 7,988 | 56,744 | 3.31 | 0.04 |
| *Woo-Lam Mutual Authentication* | 771,934 | 4,133,390 | 1,024.00 | 7.95 |
| *Needham-Schroeder Signature protocol* | 17,867 | 59,911 | 3.77 | 0.05 |
| *Neuman Stubblebine repeated part* | 39,579 | 312,107 | 15.17 | 0.21 |
| *Kao Chow Repeated Authentication, 1* | 50,703 | 185,317 | 16.34 | 0.17 |
| *Kao Chow Repeated Authentication, 2* | 586,033 | 1,999,959 | 339.70 | 2.11 |
| ***Kao Chow Repeated Authentication, 3*** | **1,100,428** | **6,367,574** | **1,288.00** | **MO** |
| *ISO public key 1-pass unilateral authentication* | 1,161 | 3,835 | 0.32 | 0.00 |
| *ISO public key 2-pass mutual authentication* | 4,165 | 23,883 | 1.18 | 0.01 |
| *Needham-Schroeder Public Key* | 9,318 | 47,474 | 1.77 | 0.05 |
| *Needham-Schroeder Public Key with key server* | 11,339 | 67,056 | 4.29 | 0.04 |
| *SPLICE/AS Authentication Protocol* | 15,622 | 69,226 | 5.48 | 0.05 |
| *Encrypted Key Exchange* | 121,868 | 1,500,317 | 75.39 | 1.78 |
| *Davis Swick Private Key Certificates, protocol 1* | 8,036 | 25,372 | 1.37 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 2* | 12,123 | 47,149 | 2.68 | 0.03 |
| *Davis Swick Private Key Certificates, protocol 3* | 10,606 | 27,680 | 1.50 | 0.02 |
| *Davis Swick Private Key Certificates, protocol 4* | 27,757 | 96,482 | 8.18 | 0.13 |

# Experiments: Analysis

Over the Clark/Jacob library:

- **Coverage:** unsuited to detect type-flaws, but effective on the others.
- **Effectiveness:** it finds most of the known attacks.
- **Performance:** Encoding Time largely dominates Solving Time.

# Conclusions & Perspectives

- SATMC performs well on the Clark/Jacob library.
- Optimizations bring a lot! Some remain to be implemented.

  We expect orders of magnitude reduction in compilation time by:

  - building encryption properties into the encoding,
  - exploiting more sophisticated encoding (e.g. graphplan encoding), and
  - applying unit propation during the encoding phase.
- Approach is declarative and incremental: we can easily modify goal or initial state without recompilation.
- Symbolic representation trivially allows sets of initial states.