

---

# SAT-based Model-Checking of Security Protocols using Planning Graph Analysis

Pierre Ganty\*

joint work with **Alessandro Armando** and **Luca Compagna**

DIST – University of Genoa

FM'03, Pisa, 12 September 2003



*Automated Validation of Internet Security Protocols and Applications*

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

---

\*For this work, Pierre Ganty was funded by the IHP-RTN EC project CALCULEMUS (HPRN-CT-2000-00102)

---

## Outline

- Our contribution
- Context: Security-protocols
- Protocol Insecurity Problems
- Protocol Insecurity Problems as SAT problems
- Experimental Results : Linear encoding *vs.* Graphplan-based encoding
- Conclusions and future works

## Our Contribution

We **define**, **implement** and **evaluate** the benefits of **SAT-based AI planning techniques**<sup>†</sup> for the **Bounded Model-Checking** area and especially for Bounded MC of Security Protocols.

<sup>†</sup>A. Blum and M. Furst. *Fast Planning through Planning Graph Analysis (IJCAI'95)*  
H. Kautz and B. Selman. *Unifying SAT-based and Graph-based Planning (IJCAI'99)*.

## Context: Security-protocols

- **Basically**: Communication protocols using cryptographic primitives to achieve security requirements.
- **Salient features**:
  - A protocol is defined by a finite set of **roles** (initiator, responder, . . . )
  - Roles instantiations form **session instances**
  - Several session instances can be run in **parallel**
  - An **agent** can play different roles in different sessions
  - Agents can generate **fresh constants** during a session

## Context: Security-protocols (2)

- **Moreover we make the standard assumptions :**
  - **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 (i.e. of the type expected by agents).
  - **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.

## The model: underlying ideas

- Model specified by a **rule-based language** suitable for Security-protocols.
- Concurrent execution of multiple session instances (with the DY intruder) as a **transition system**.
- **States** represented by **sets of ground facts** (viz. variables-free atomic formula) and **transitions** by means of **rewrite rules** over sets of facts.
- Intuitively,
  - states** = state of agents involved in the protocol + state of the intruder;
  - rules** = rules of the protocol + rules of the intruder.

## The problem

- Security requirements such as **authentication** and **secrecy** are reduced to **reachability problems** on this model.
- On this model, the reachability problem is **undecidable**.
- This stems from **infiniteness** of sessions.
- We thus 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.

## The formal model: 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 formula 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) labeled *rewrite rules* of the form  $L \xrightarrow{\lambda} R$ , where  $L, R \subseteq \mathcal{S}$ , and  $\lambda \in \mathcal{L}$ ;
- $\mathcal{I}$  and  $\mathcal{B}$  are respectively the *initial state* and a set of *bad states*.

A **solution to a Protocol Insecurity Problem** is a sequence  $S_1 \xrightarrow{\lambda_1} S_2 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_{n-1}} S_n$ , where  $\lambda_i \in \mathcal{L}$ ,  $\mathcal{I} \equiv S_1$ , and there exists  $S_{\mathcal{B}} \in \mathcal{B}$  such that  $S_{\mathcal{B}} \subseteq S_n$ .

## Example

- Initial State:  $state_1(a, \langle b, k(b) \rangle).state_2(b, \langle a, k(a) \rangle)$
- A Protocol Rule:

$$state_1(a, \langle b, k(b) \rangle) \xrightarrow{step_1} state_3(a, \langle b, k(b), na \rangle).net(\{na\}_{k(b)}).secret(na)$$

- Some Intruder Rules:

$$net(X) \xrightarrow{overhear} net(X).iknows(X)$$

$$net(\{X\}_{k(b)}).iknows(k^{-1}(b)) \xrightarrow{decrypt} iknows(X).net(\{X\}_{k(b)}).iknows(k^{-1}(b))$$

- Bad States:  $iknows(X).secret(X)$

## Prot. Insec. Problems as Satisfiability Problems

Given,  $\Pi = \langle \mathcal{S}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{B} \rangle$  and a positive integer  $n$ , we build a propositional formula  $\Phi_{\Pi}^n$  such that any **model of  $\Phi_{\Pi}^n$**  corresponds to **solutions of  $\Pi$** .

**To do so, we:**

1. use a propositional variable for each **fact**, *e.g.*  $p$ , and **rule**, *e.g.*  $\lambda$
2. add an additional **time-index** parameter to each  $\lambda$  or  $p$ , to indicate the state at which time the rule begins or the fact holds.
3. build  $\Phi_{\Pi}^n$  by **unfolding  $n$  times the transition relation**:

$$\Phi_{\Pi}^n = I(p_0) \wedge \bigwedge_{i=0}^{n-1} T_i(p_i, \lambda_i, p_{i+1}) \wedge F(p_n)$$

where  $I(p_0)$  is a formula defining **the initial state**,  $T$  the **transition relation** and  $F(p_n)$  the **final states**.

## Linear Encoding

The formula  $T_i(p_i, \lambda_i, p_{i+1})$  for  $i = 0, \dots, n - 1$  is given by the conjunction of the following axioms:

**Universal Axioms:** for each rewrite rule  $\lambda \in \mathcal{L}$  s.t.  $(L \xrightarrow{\lambda} R) \in \mathcal{R}$

$$\lambda_i \supset \bigwedge \{p_i \mid p \in L\}$$

$$\lambda_i \supset \bigwedge \{p_{i+1} \mid p \in R \setminus L\}$$

$$\lambda_i \supset \bigwedge \{\neg p_{i+1} \mid p \in L \setminus R\}$$

**Cardinality:**  $O(n|\mathcal{L}|r)$ , where  $r$  is the maximal number of facts mentioned in a rule (usually a small number).

## Linear Encoding (2)

**Explanatory Frame Axioms:** for all facts  $p \in \mathcal{S}$

$$\begin{aligned} (p_i \wedge \neg p_{i+1}) &\supset \bigvee \left\{ \lambda_i \mid \lambda \in \mathcal{L}, L \xrightarrow{\lambda} R \in \mathcal{R}, p \in L \setminus R \right\} \\ (\neg p_i \wedge p_{i+1}) &\supset \bigvee \left\{ \lambda_i \mid \lambda \in \mathcal{L}, L \xrightarrow{\lambda} R \in \mathcal{R}, p \in R \setminus L \right\} \end{aligned}$$

**Cardinality:**  $O(n|\mathcal{S}|)$ .

**Conflict Exclusion Axioms:** for all distinct rule  $\lambda^1, \lambda^2$  such that  $(L^1 \xrightarrow{\lambda^1} R^1) \in \mathcal{R}, (L^2 \xrightarrow{\lambda^2} R^2) \in \mathcal{R}$  with  $L^1 \cap (L^2 \setminus R^2) \neq \emptyset$

$$\neg(\lambda_i^1 \wedge \lambda_i^2)$$

**Cardinality:**  $O(n|\mathcal{L}|^2)$ .

## Linear Encoding (3)

- In practice, the **linear encoding** is very efficient for small security protocols **but it does not scale up**.

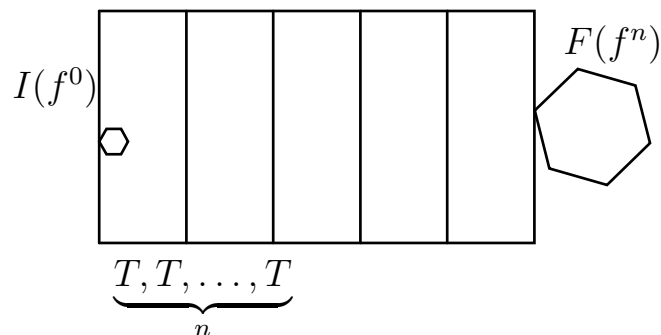
Protocol	Atoms	Clauses
NSPK	6612	33326
Kaochow-3	995323	5736662

- **Because** the **linear encoding** can contain **useless information**; e.g. if  $\mathcal{I} = \emptyset$ , we know that the truth-value of the variables in  $p_0$  is false and thus they can all be simplified away.
- This stems from the **independence of the unfolding**, viz.  $T_0 = T_1 = \dots = T_n$  in our definition of a  $n$ -bounded unfolding.

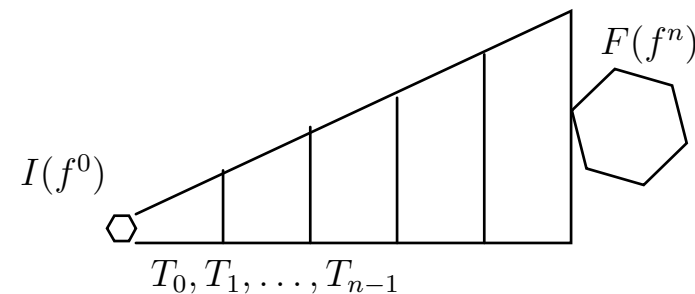
## Graphplan-based Encoding

- **Idea:** Use knowledge about  $\mathcal{I}$  to simplify the  $T_i$ 's.
- In order to do so, **working on the Protocol Insecurity Problem**,
  - we **want** to propagate information provided by the initial state  $\mathcal{I}$ ;
  - we **do not want** to build the forward search tree (otherwise the SAT search would be useless), we want an **over-approximation**.

Linear Encoding

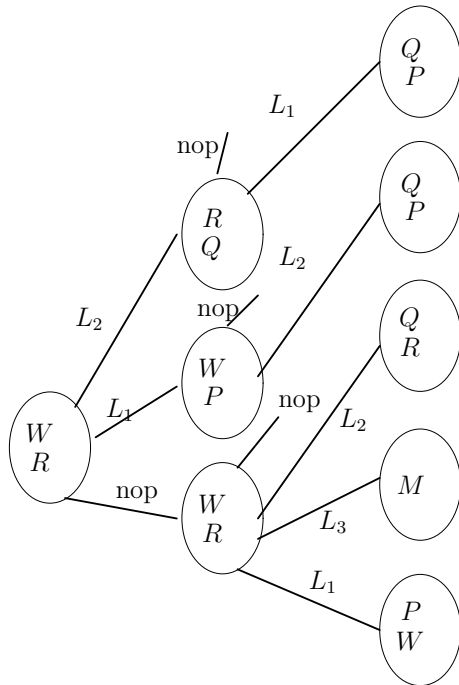


Graphplan-based encoding

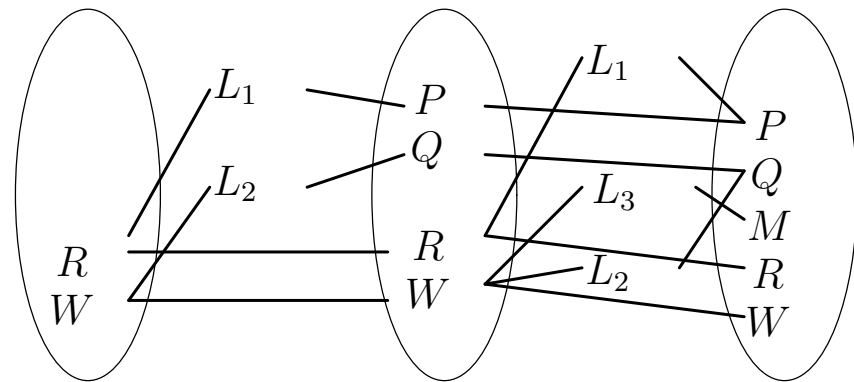


## Graphplan-based Encoding (2)

Consider the following search tree:



An approximation would be to abstract each ply of the search tree into a unique set of facts with mutex constraints over them. We call the data structure a planning graph.



Starting from  $\mathcal{I}$ , we can iteratively build the the planning graph by adpating the forward search algorithm.

## Graphplan-based Encoding (3)

Constraints are **mutual exclusion constraints** defined between rules or between facts. In a ply of a planning graph, we **add a constraint** between 2 distinct rules or facts if one of the following conditions holds:

- one rule  $\lambda$  **deletes a fact** contained in the **LHS** of an other rule  $\lambda'$
- one rule  $\lambda$  **deletes a fact** that an another rule  $\lambda'$  adds
- two rules  $\lambda$  and  $\lambda'$  contain in their **LHS** two distinct **constrained facts**
- two facts  $p$  and  $p'$  are added in the graph by rules that are **all pairwise constrained**

## Graphplan-based Encoding (4)

Given a Planning Graph  $G = \langle V, E \rangle$  of  $n$  plies, the formula  $T_i(p_i, \lambda_i, p_{i+1})$  for  $i = 0, \dots, n - 1$  is the conjunction of

- **Action Preconditions Add-effects Axioms:** for each node of  $\lambda \in V_i$  s.t.  $\lambda \in \mathcal{L}$  and  $(L \xrightarrow{\lambda} R) \in \mathcal{R}$

$$\lambda_i \supset \bigwedge \{p_i \mid \langle p, \lambda \rangle \in E_i\}$$

$$\lambda_i \supset \bigwedge \{p_{i+1} \mid \langle \lambda, p \rangle \in E_i\}$$

- **Backward Chaining Axioms:** for each each node of  $p \in V_{i+1}$  s.t.  $\exists \lambda \in \mathcal{L}$  with  $(L \xrightarrow{\lambda} R) \in \mathcal{R}$  and  $p \in R$

$$p_{i+1} \supset \bigvee \{\lambda_i \mid \langle \lambda, p \rangle \in E_i\}$$



## Graphplan-based Encoding (5)

- **Mutex Axioms:** for each 2 nodes  $\lambda, \lambda' \in V_i$  and each 2 nodes  $p, p' \in V_{i+1}$  that are constrained :

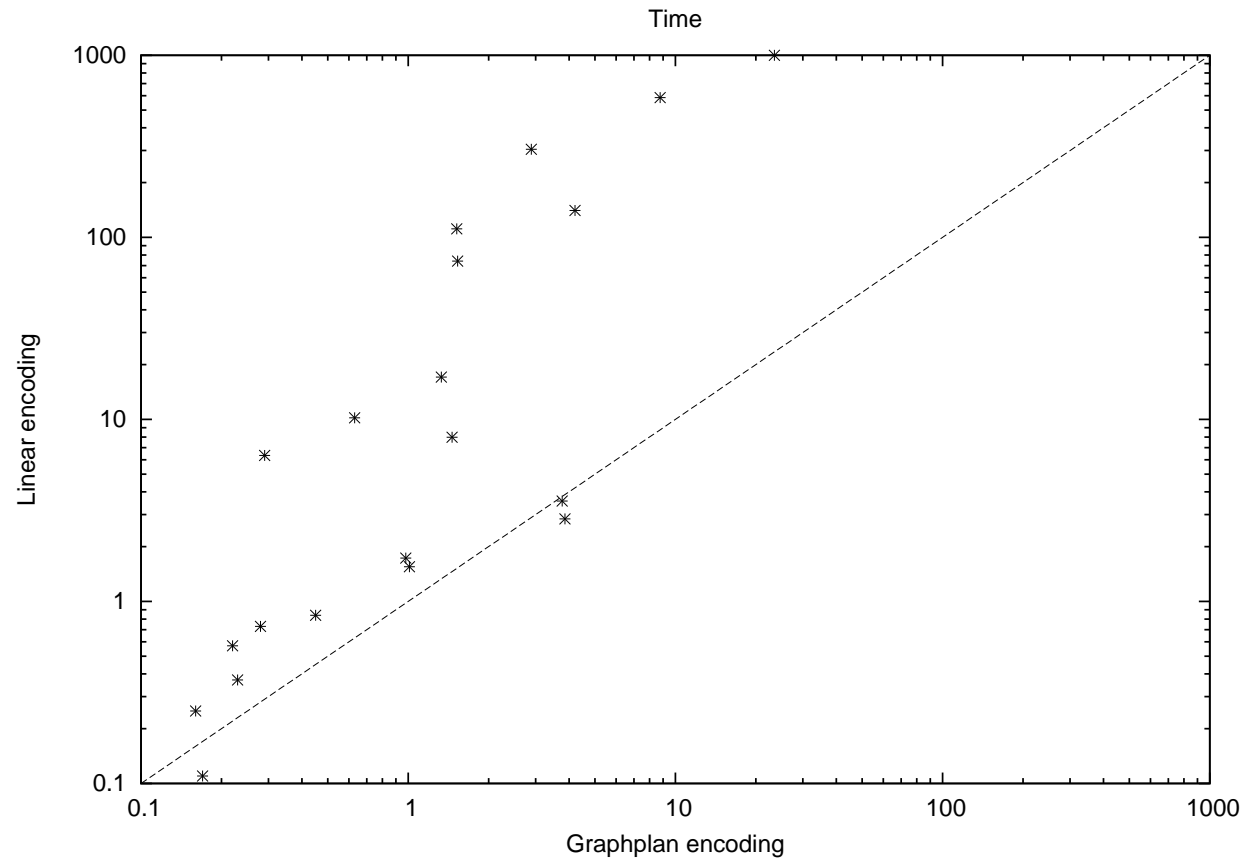
$$\neg(\lambda_i \wedge \lambda'_i) \wedge \neg(p_{i+1} \wedge p'_{i+1})$$

## Implementation

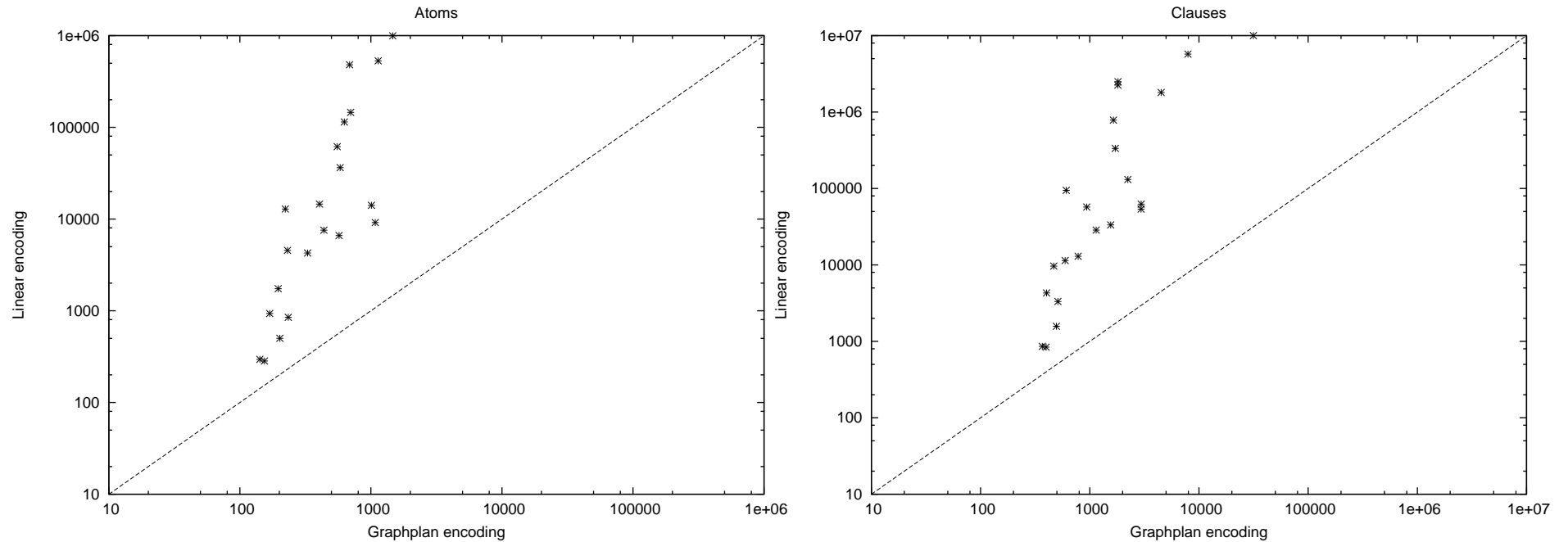
### SATMC:

- **high-level specification language** close to the Alice&Bob notation;
- linear encoding and graphplan-based encoding with **iterative deepening** on the number of steps;
- produces a **SAT formula** in the DIMACS format that can be feed in **any SAT solver**.

## Experimental Results (Time)



# Experimental Results (Memory)



## Conclusions and future works

- Proposed an **automatic** approach to **compile Protocol Insecurity Problems into SAT**.
- Graphplan-based techniques outperforms **up to 2 order of magnitude** linear techniques.
- **AVISPA** project: industrial-strength technology for the **A**utomated **V**erification of large-scale **I**nternet **S**ecurity **P**rotocols and **A**pplications.  
[www.avispa-project.org](http://www.avispa-project.org)
- **Graphplan-based techniques** could **outperform** the encoding techniques currently implemented in the state-of-the-art **Bounded Model-Checkers**.

Protocol	N	$\oplus = \oplus_a \cup \oplus_f$				$\oplus = \hat{\oplus}$			
		A	CL	EncT	SoIT	A	CL	EncT	SoIT
<i>Andrew</i>	9	699	4,459	17.0	0.0	701	1,811	1.5	0.0
<i>EKE</i>	5	545	3,584	10.9	0.0	553	1,648	1.5	0.0
<i>ISO-CCF-1 U</i>	4	154	621	0.2	0.0	154	398	0.1	0.0
<i>ISO-CCF-2 M</i>	4	169	600	0.3	0.0	169	402	0.2	0.0
<i>ISO-PK-1 U</i>	4	202	794	0.5	0.0	202	494	0.2	0.0
<i>ISO-PK-2 M</i>	4	196	707	0.6	0.0	196	468	0.2	0.0
<i>ISO-SK-1 U</i>	4	142	561	0.2	0.0	142	367	0.1	0.0
<i>ISO-SK-2 M</i>	4	234	747	0.5	0.0	234	511	0.2	0.0
<i>KaoChow 1</i>	7	526	4,055	14.6	0.0	583	2,230	1.5	0.01
<i>KaoChow 2</i>	9	1,020	10,325	108.1	0.02	1,136	4,507	4.2	0.03
<i>KaoChow 3</i>	9	1,229	18,532	419.1	0.01	1,467	7,927	8.8	0.05
<i>KLS rep.</i>	7	2,018	62,836	3,557.4	0.03	2,092	31,510	23.5	0.07
<i>NSCK</i>	9	622	3,823	9.5	0.01	627	1,710	1.3	0.01
<i>NSPK</i>	7	559	3,129	6.7	0.0	572	1,555	1.0	0.0
<i>NSPK-server</i>	8	1,077	6,043	24.5	0.0	1,079	2,951	3.9	0.0
<i>SPLICE</i>	9	972	7,545	52.2	0.01	1,008	2,953	3.8	0.0
<i>Swick 1</i>	5	329	1,300	1.1	0.0	329	783	0.4	0.01
<i>Swick 2</i>	6	438	1,937	2.6	0.0	438	1,143	0.1	0.0
<i>Swick 3</i>	4	231	889	0.7	0.0	231	594	0.3	0.0
<i>Swick 4</i>	5	405	1,469	2.7	0.0	405	939	0.6	0.0
<i>Stubblebine rep</i>	3	220	791	0.7	0.0	222	608	0.3	0.0
<i>Woo-Lam M</i>	6	665	3,806	17.3	0.0	687	1,813	2.9	0.0

Protocol	N	A	CEA = on				CEA = off				
			CL	EncT	SolvingT		CL	EncT	SolvingT		
					Last	Tot			Last	Tot	#
<i>Andrew</i>	9	145	-	-	-	-	2,256	111.4	2.0	12.1	1
<i>EKE</i>	5	62	13,949	7,100	7.6	19.7	783	74.1	0.7	3.7	2
<i>ISO-CCF-1 U</i>	4	< 1	< 1	0.1	0.0	0.0	< 1	0.1	0.0	0.0	0
<i>ISO-CCF-2 M</i>	4	< 1	6	0.3	0.0	0.1	4	0.2	0.0	0.1	0
<i>ISO-PK-1 U</i>	4	< 1	2	0.1	0.0	0.0	2	0.1	0.0	0.0	0
<i>ISO-PK-2 M</i>	4	2	17	0.9	0.0	0.0	10	0.6	0.1	0.1	0
<i>ISO-SK-1 U</i>	4	< 1	< 1	0.1	0.0	0.0	< 1	0.1	0.0	0.0	1
<i>ISO-SK-2 M</i>	4	< 1	3	0.4	0.0	0.0	3	0.4	0.0	0.0	0
<i>KaoChow 1</i>	7	36	355	18.4	0.1	0.7	131	8.0	0.1	0.7	4
<i>KaoChow 2</i>	9	586	35,178	1,494	-	-	1,804	140.4	1.6	15.6	5
<i>KaoChow 3</i>	9	995	-	-	-	-	5,737	585.5	7.5	41.6	1
<i>KLS rep.</i>	-	-	-	-	-	-	-	-	-	-	-
<i>NSCK</i>	9	115	787	41.3	0.4	1.6	334	17.1	0.3	1.3	0
<i>NSPK</i>	7	7	51	2.3	0.1	0.1	33	1.5	0.1	0.1	0
<i>NSPK-server</i>	8	9	212	8.0	0.1	0.2	54	2.8	0.1	0.1	0
<i>SPLICE</i>	9	14	91	4.6	0.1	0.2	62	3.6	0.1	0.2	0
<i>Swick 1</i>	5	4	17	1.0	0.1	0.1	13	0.8	0.0	0.1	1
<i>Swick 2</i>	6	8	59	3.2	0.1	0.1	29	1.7	0.1	0.1	0
<i>Swick 3</i>	4	5	12	0.8	0.1	0.1	11	0.7	0.0	0.1	1
<i>Swick 4</i>	5	15	64	11.0	0.1	0.2	57	10.2	0.1	0.3	1
<i>Stubblebine rep</i>	3	13	2,048	82.9	0.3	0.6	95	6.3	0.1	0.1	1
<i>Woo-Lam M</i>	6	481	-	-	-	-	2,498	304.4	1.8	7.7	1