# On some Logic-Based Approaches to Security Protocol Verification

Bruno Blanchet

Google

On leave from Inria, Paris
bruno.blanchet@inria.fr

John Mitchell's Festschrift
May 2016

# 15 years ago

Proceedings

## $14^{th}$ IEEE Computer Security Foundations Workshop

**11-13 June 2001**

**Cape Breton, Novia Scotia, Canada**

**Sponsored by**

IEEE Computer Society Technical Committee on Security and Privacy

# Many interesting papers

## Table of Contents

### CSFW-14



### *Non-interference & Information Flow*



### *Access Control*



### *Protocols I*



### *Panel*



### *Information Flow & Multi-threading*



### *Protocols II*



### *Intrusion Tolerance & Detection*



### *Panel*



### *Logics for Protocol Verification*



### *Secrecy & Privacy*

# Panel

***Panel***



(Martín Abadi may have replaced Michael Waidner.)

# Model of protocols

- The symbolic model or "Dolev-Yao model":
  - The cryptographic primitives are blackboxes.
  - The messages are terms on these primitives.
  - The adversary is restricted to compute only using these primitives.
    $\Rightarrow$ perfect cryptography assumption

  This model facilitates automatic proofs.
- The computational model:
  - The messages are bitstrings.
  - The cryptographic primitives are functions on bitstrings.
  - The adversary is any probabilistic polynomial-time Turing machine.

  This model is more realistic than the symbolic model, but until recently proofs were only manual.

# Panel

**Panel**



- Lincoln, Mitchell, Mitchell, and Scedrov. A probabilistic poly-time framework for protocol analysis. CCS'98.
- Pfitzmann, Schunter and Waidner. Cryptographic security of reactive systems. Workshop on secure architectures and information flow, 1999.
- Abadi and Rogaway. Reconciling two views of cryptography (the computational soundness of encryption). IFIP TCS 2000.
- Canetti. A unified framework for analysing security of protocols. Cryptology ePrint Archive, report 2000/067.

# Two papers by John

## *Access Control*



## *Logics for Protocol Verification*

# First paper on protocol composition logic (PCL)

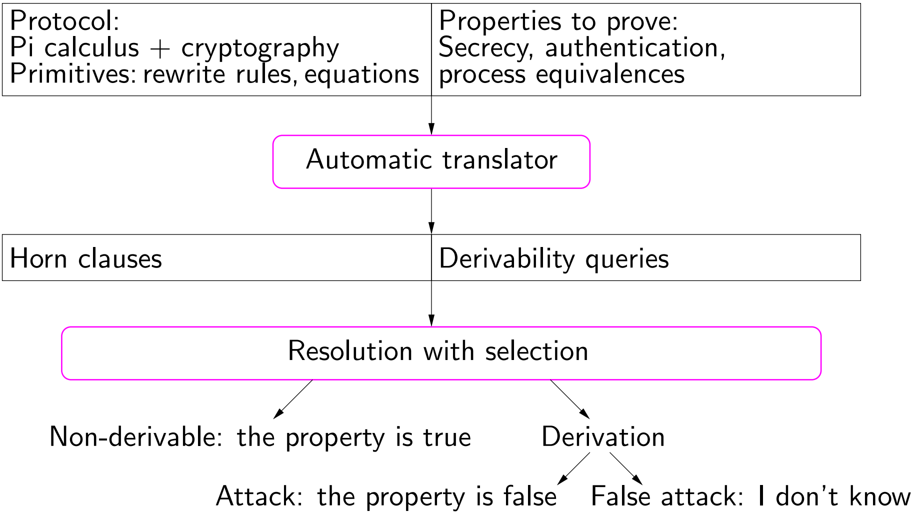
- A Hoare logic for security protocol verification.
- Based on a formal semantics of protocols.
- Initially: symbolic.
- Led to considerable developments (more than 20 papers):
  - Protocol derivation
  - Computational PCL
  - Proof assistant in Isabelle
- Applied to many protocols.
  - Found and fixed problems in standardized protocols (IEEE 802.11i, IETF GDOI)

# First paper on ProVerif

# ProVerif

- Symbolic security protocol verifier.
- Fully automatic.
- Unbounded number of sessions and message space.
- Cryptographic primitives: defined by rewrite rules or equations.
- Security properties: secrecy, authentication, some equivalences.
- Does not always terminate and is not complete. In practice:
  - Efficient: small examples verified in less than 0.1 s; complex ones from a few minutes to hours.
  - Precise: no false attack in 19 protocols of the literature tested for secrecy and authentication.

# ProVerif


Protocol:
Pi calculus + cryptography
Primitives: rewrite rules, equations
Properties to prove:
Secrecy, authentication,
process equivalences
Automatic translator
Horn clauses
Derivability queries
Resolution with selection
Non-derivable: the property is true
Derivation
Attack: the property is false
False attack: I don't know

# Definition of cryptographic primitives

Two kinds of operations:

- Constructors $f$ are used to build terms $f(M_1, \ldots, M_n)$

  **fun** $f(T_1, \ldots, T_n) : T.$

**Example: shared-key encryption**

**fun** encrypt(bitstring, key) : bitstring.

- Destructors $g$ manipulate terms
  Destructors are defined by rewrite rules $g(M_1, \ldots, M_n) \rightarrow M.$

  **reduc forall** $x_1 : T_1, \ldots, x_k : T_k; g(M_1, \ldots, M_n) = M.$

**Example: shared-key decryption decrypt(encrypt($m, k$), $k$) $\rightarrow m$**

**reduc forall** $x$ : bitstring, $y$ : key; decrypt(encrypt($x, y$), $y$) $= x$.

# Syntax of the process calculus

A dialect of the applied pi calculus (Abadi, Fournet, POPL'01):
Pi calculus + cryptographic primitives

| | |
|---|---|
| $M, N ::=$ | terms |
| $\quad x, y, z$ | variable |
| $\quad a, b, c, k, s$ | name |
| $\quad f(M_1, \ldots, M_n)$ | constructor application |
| | |
| $P, Q ::=$ | processes |
| $\quad \mathbf{out}(M, N); P$ | output |
| $\quad \mathbf{in}(M, x : T); P$ | input |
| $\quad \mathbf{let}\ x = g(M_1, \ldots, M_n)\ \mathbf{in}\ P\ \mathbf{else}\ Q$ | destructor application |
| $\quad \mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q$ | conditional |
| $\quad \mathbf{new}\ a : T; P$ | restriction |
| $\quad 0 \quad P \mid Q \quad !P$ | |

# Example: The Denning-Sacco protocol

$$\begin{array}{llll} \text{Message 1.} & A \to B : & \{\{k\}_{sk_A}\}_{pk_B} & k \text{ fresh} \\ \text{Message 2.} & B \to A : & \{s\}_k & \end{array}$$

$$\begin{array}{lll}
& & \textbf{new } sk_A : \text{sskey}; \textbf{let } pk_A = \text{spk}(sk_A) \textbf{ in} \\
& & \textbf{new } sk_B : \text{skey}; \textbf{let } pk_B = \text{pk}(sk_B) \textbf{ in} \\
& & \textbf{out}(c, pk_A); \textbf{out}(c, pk_B); \\
(A) & & \quad ! \, \textbf{in}(c, x_pk_B : \text{pkey}); \\
& & \quad \textbf{new } k : \text{key}; \textbf{out}(c, \text{pencrypt}(\text{sign}(\text{k2b}(k), sk_A), x_pk_B)); \\
& & \quad \textbf{in}(c, x : \text{bitstring}); \textbf{let } s = \text{decrypt}(x, k) \textbf{ in } 0 \\
(B) & | & \quad ! \, \textbf{in}(c, y : \text{bitstring}); \textbf{let } y' = \text{pdecrypt}(y, sk_B) \textbf{ in} \\
& & \quad \textbf{let } \text{k2b}(k) = \text{checksign}(y', pk_A) \textbf{ in } \textbf{out}(c, \text{encrypt}(\text{s}, k))
\end{array}$$

# Security properties

- Secrecy: the adversary cannot obtain the secret s.

  **query** attacker(s).

- Correspondence assertions: (authentication)
  If an event has been executed, then some other events must have been executed.
- Process equivalences: the adversary cannot distinguish between two processes.
  - Strong secrecy: the adversary cannot see when the value of the secret changes.
  - Equivalences between processes that differ only by terms they contain (joint work with Martín Abadi and Cédric Fournet)

    In particular, proof of protocols relying on weak secrets.

# The Horn clause representation

- The main predicate used by the Horn clause representation of protocols is attacker:

$$\text{attacker}(M) \quad \textit{means} \quad \textit{"the adversary may have } M\textit{"}.$$

- We can model actions of the adversary and of the protocol participants thanks to this predicate.
- The term $M$ is secret when it cannot be built by an adversary.

**Theorem (Secrecy)**

*If* attacker($M$) *cannot be derived from the clauses, then* $M$ *is secret.*

- The resolution algorithm determines whether a fact is derivable from the clauses.

# Applications

1. Case studies:
   - 19 protocols of the literature
   - Certified email (with Martín Abadi)
   - JFK (with Martín Abadi and Cédric Fournet)
   - Plutus (with Avik Chaudhuri)
   - Avionic protocols (ARINC 823)

   Case studies by others:
   - E-voting protocols (Delaune, Kremer, and Ryan; Backes et al)
   - Zero-knowledge protocols, DAA (Backes et al)
   - Shared authorisation data in TCG TPM (Chen and Ryan)
   - Electronic cash (Luo et al)
   - ...
2. Extensions
3. ProVerif as back-end

# Applications

1. Case studies
2. Extensions:
   - Extensions to XOR and Diffie-Hellman (Küsters and Truderung), to bilinear pairings (Pankova and Laud)
   - StatVerif: extension to mutable state (Arapinis et al)
   - Set-Pi: extension to sets with revocation (Bruni et al)
3. ProVerif as back-end:
   - TulaFale: Web service verifier (Bhargavan et al)
   - FS2PV: F# to ProVerif, applied to TLS and TPM (Bhargavan et al)
   - JavaSpi: Java to ProVerif (Avalle et al)
   - Web-spi: web security mechanisms (Bansal et al)

# Conclusion

- Power of logic for modeling and reasoning on protocols.
- Used in very different ways.
- Any relation between these approaches?