The PROUVÉ Project : Tools for the Automatic Verification of Cryptographic Protocols

Ralf Treinen

Laboratoire Spécification et Vérification (LSV)

École Normale Supérieure de Cachan – CNRS – INRIA Futurs

France

The PROUVÉ Project

- PROUVÉ =
  Protocoles cryptographiques : outils de vérification automatique
- Three years, starting from November 2003 (in reality : January 2004)
- 3 academic partners : LORIA, LSV, VERIMAG
- 2 industrial partners : CRIL Technology, France Télécom R&D
- Financed by the French Ministry for Research

A Classical Example of a Vulnerable Protocol

The Needham-Schroeder Public Key Protocol

$$
\begin{aligned}
A \rightarrow B &\quad : \quad \{N_A, A\}_{\mathrm{pub(B)}} \\
B \rightarrow A &\quad : \quad \{N_A, N_B\}_{\mathrm{pub(A)}} \\
A \rightarrow B &\quad : \quad \{N_B\}_{\mathrm{pub(B)}}
\end{aligned}
$$

An attack (man in the middle attack) was found by G. Lowe 17 years after its publication.

Goal : automatic verification of logical correctness of protocols

Attack Against the NSPK Protocol

Two parallel sessions :

- Between A and I
- Between I(pretending to be A) and B

$$
\begin{aligned}
A \rightarrow I &: \{N_A, A\}_{\mathrm{pub(I)}} \\
I(A) \rightarrow B &: \{N_A, A\}_{\mathrm{pub(B)}} \\
B \rightarrow I(A) &: \{N_A, N_B\}_{\mathrm{pub(A)}} \\
I \rightarrow A &: \{N_A, N_B\}_{\mathrm{pub(A)}} \\
A \rightarrow I &: \{N_B\}_{\mathrm{pub(I)}} \\
I(A) \rightarrow B &: \{N_B\}_{\mathrm{pub(B)}}
\end{aligned}
$$

Why Yet Another Project on Verification of Protocols ?

- specification language of protocols : should be unambiguous and expressive, and independent of the properties to prove
- language of properties should be generic, and allow to express different properties of protocols
- different verification tools based on a common input language with a common semantics
- Weakening of the *Perfect Cryptography Assumption*

Another Example of a Vulnerable Protocol

Key exchange protocol [Tatebayashi, Matsuzuki, Newmann '89]

– two participants A and B want to agree on a session key
– they use a trusted server S
– participants A, B have limited computing resources :

1. generation of random numbers
2. encryption by public key cryptography : $m \mapsto m^3 \bmod n$
3. bitwise xor

The TMN Protocol

Public key encryption : n is public, but only S can compute cubic roots modulo n.

$$
\begin{array}{lll}
(1) & A \rightarrow S & \text{“I want to talk to B”}, r_A^3 \bmod n \\
(2) & S \rightarrow B & \text{“A wants to talk to you”} \\
(3) & B \rightarrow S & {r_B}^3 \bmod n \\
(4) & S \rightarrow A & r_A \oplus r_B
\end{array}
$$

Now, participant A can compute $r_B = r_A \oplus (r_A \oplus r_B)$.

Attack against a Naive Implementation of the TMN Protocol

- Attacker C has seen $r_B^3 \bmod n$ on the network.
- The value r_D is known to both attackers C and D.

$$
\begin{array}{lll}
(1) & C \rightarrow S & \text{“I want to talk to D”}, r_B^3 \bmod n \\
(2) & S \rightarrow D & \text{“C wants to talk to you”} \\
(3) & D \rightarrow S & r_D^3 \bmod n \\
(4) & S \rightarrow C & r_D \oplus r_B
\end{array}
$$

Attacker C can now calculate

- $r_B = r_D \oplus (r_D \oplus r_B)$

Attack against the TMN Protocol [Simmons '94]

– Attacker C has seen $r_B^3 \bmod n$ on the network.
– The value r_D is known to both attackers C and D.

$$
\begin{array}{lll}
(1) & C \rightarrow S & \text{“D”}, (r_C^3 * (r_B^3 \bmod n)) \bmod n = (r_C * r_B)^3 \bmod n \\
(2) & S \rightarrow D & \text{“C wants to talk to you”} \\
(3) & D \rightarrow S & r_D^3 \bmod n \\
(4) & S \rightarrow C & r_D \oplus (r_C * r_B)
\end{array}
$$

Attacker C can now calculate

– $r_C * r_B = r_D \oplus (r_D \oplus (r_C * r_B))$
– $r_B = r_C^{-1} * (r_C * r_B)$

Issues of the TMN Protocol

- algebraic properties of some operations
- the server has to store a value obtained in one protocol step, and reuse it in a later step
- specify that the server responds to all requests (in a sequential loop)
- specify how the server compares requests to keys seen before

Task 1 : Protocols and Properties

– define a language for the specification of protocols with its semantics
– define a language of properties with its semantics
– define simplifications and reductions in the language

Candidate Language : Alice-Bob Notation (1)

for instance *Common Syntax* of [Burrows&Abadi&Needham]

Problems with the Alice-Bob notation : Ambiguity

Alice-Bob notation : describes a correct execution of one protocol trace, but does not specify

- how sessions are initiated
- how the participants do process received messages
- who and when keys and nonces are created

Confusion between roles and principals

Candidate Language : Alice-Bob Notation (2)

Problems with the Alice-Bob notation : Lack of expressivity

- Variables that can be instantiated once
- Variables that represent state
- Conditionals, Loops
- A language for the composition of role instances
- Persistent variables
- Shared Variables
- Data types like lists or arrays
- ...

Candidate Languages : Process Calculi

for instance

– *Spi-calculus* [Abadi&Gordon]

– *Applied* π*-calculus* [Abadi&Fournet]

Features :

– very expressive

– minimal

– security properties based on observational equivalence and very hard to verify automatically

Design Objectives : Protocols

A specification should contain

– a specification of the signature : symbols and their type

– a list of equational axioms which are satisfied by the functions

– a definition of roles

– a specification of how the roles are instantiated

Similar in spirit to MuCAPSL [Millen&Denker], CASRUL [Rusinowitch&Vigneron, ...]

Design Objectives : Roles

- distinguish roles and principals
- employ constructs from programming languages to describe roles :
  - parameterized definition of roles
  - basic instructions : receive, send, assignments
  - variables may be logical or mutable
  - branching instructions : conditionals, switch
  - dynamic creation of key, nonces, fresh values

Restrictions by design :

- no looping constructs
- no internal non-determinism (but can probably not be avoided)

Design Objectives : Instantiation of Roles

- convenience of common programming language constructs like assignments, conditional, loop construct, sequential composition,
- persistent variables (variables living outside of role instances)
- existential and universal quantification, spawn an instance of a process for every value of some domain
- *variables with explicit skolemization (will be explained later)*
- *universal quantification over a tuple-space with restriction to some relation (will be explained later)*

The PROUVÉ Specification Language

$$
\begin{array}{rcl}
\langle\text{specification}\rangle & \rightarrow & \langle\text{signature}\rangle\ ?\ \ \langle\text{axioms}\rangle\ ?\ \ \langle\text{role}\rangle\ *\\
 & & \langle\text{globals}\rangle\ ?\ \ \langle\text{scenario}\rangle\\
\langle\text{signature}\rangle & \rightarrow & \underline{\text{signature}}\ \ (\ \langle\text{ne-identlist}\rangle\ :\ \langle\text{typeexp}\rangle\ ;\ )*\ \underline{\text{end}}\\
\langle\text{axioms}\rangle & \rightarrow & \underline{\text{axioms}}\ (\ \underline{\text{declare}}\ \ \langle\text{declarations}\rangle\ \ \underline{\text{begin}}\ )?\\
 & & (\ \langle\text{expr}\rangle\ =\ \langle\text{expr}\rangle\ ;\ )*\ \underline{\text{end}}
\end{array}
$$

Types (1)

Basic types (pairwise disjoint) :

Name	Comment
`message`	the type of all (non-functional) values
`int`	the type of integers
`bool`	the type of boolean values
`nonce`	the type of nonces
`algo`	the enumeration type of encryption algorithms
`pubkey`	the type of public encryption keys
`privkey`	the type of private decryption keys (or private signing keys)
`principal`	the type of principals

Types (2)

Universal type `message` is the union of all basic types.

Polymorphic type constructors :

Name	Arity	Comment
`list`	1	polymorphic lists
`table`	2	polymorphic association lists

Variable Declaration

$$
\begin{array}{lcl}
\langle\text{declaration}\rangle & \rightarrow & \langle\text{ne-identlist}\rangle \\
 & \mid & \langle\text{ne-identlist}\rangle \ : \ \langle\text{typeexp}\rangle \\
 & \mid & \langle\text{ne-identlist}\rangle \ : \ \underline{\text{mutable}} \ \ \langle\text{typeexp}\rangle \ ?
\end{array}
$$

– logical variables (default) can be assigned only once

– mutable variables can be assigned multiple times

– the default type is `message`

Role Definitions

$\langle \text{role} \rangle \quad \rightarrow \quad \underline{\text{role}} \quad \langle \text{ident} \rangle \ (\ \langle \text{nt-decllist} \rangle \ ?) \ \langle \text{block} \rangle$

In the parameter list of a role definition

– mutable parameters are call-by-reference

– logical variables are call-by-value

Instructions in Role Definitions

$$
\begin{array}{rcl}
\langle\text{block}\rangle & \rightarrow & (\ \underline{\text{declare}}\ \ \langle\text{nt-decllist}\rangle\ ;\ )?\ \underline{\text{begin}}\ \ \langle\text{instlist}\rangle\ \ \underline{\text{end}} \\
\langle\text{instruction}\rangle & \rightarrow & \langle\text{block}\rangle \\
 & \mid & \underline{\text{send}}\ (\ \langle\text{expression}\rangle\ )\ \mid\ \underline{\text{recv}}\ (\ \langle\text{patterns}\rangle\ ) \\
 & \mid & \underline{\text{fail}} \\
 & \mid & \langle\text{ident}\rangle\ :=\ \langle\text{expression}\rangle \\
 & \mid & \underline{\text{new}}\ (\ \langle\text{ident}\rangle\ ) \\
 & \mid & \underline{\text{match}}\ \ \langle\text{expression}\rangle\ \ \underline{\text{with}}\ \ \langle\text{patterns}\rangle\ \ \underline{\text{end}} \\
 & \mid & \underline{\text{if}}\ \ \langle\text{expression}\rangle\ \ \underline{\text{then}}\ \ \langle\text{instlist}\rangle\ (\ \underline{\text{else}}\ \ \langle\text{instlist}\rangle\ )?\ \underline{\text{fi}} \\
 & \mid & \underline{\text{case}}\ \ \langle\text{expression}\rangle\ \ \underline{\text{of}}\ (\ \langle\text{case}\rangle\ ;\ )*\ \langle\text{case}\rangle \\
 & & \qquad \underline{\text{else}}\ \ \langle\text{instlist}\rangle\ \ \underline{\text{esac}}
\end{array}
$$

Patterns and Case Distinctions

$$\begin{array}{rcl}
\langle \text{pattern} \rangle & \rightarrow & \langle \text{expression} \rangle \mid (\langle \text{expression} \rangle : \langle \text{typeexp} \rangle) \\
\langle \text{patterns} \rangle & \rightarrow & (\langle \text{pattern} \rangle \; ;)* \langle \text{pattern} \rangle \\
\langle \text{case} \rangle & \rightarrow & \langle \text{patterns} \rangle \; - > \; \langle \text{instlist} \rangle
\end{array}$$

The construction $(exp : type)$ serves to check the type of an expression (this is useful for checking the type of a value obtained in a <u>recv</u> instruction).

Matching of patterns is tested in sequential order.

Example TMN : The Server

```
role server(seen_keys: mutable list(nonce);
            my_secretkey: privkey)
  declare
    initiator_nonce, partner_nonce: nonce;
    initiator_name, partner_name: principal;
  begin
    recv([crypt(asym,inv(my_secretkey),initiator_nonce),
    initiator_name,partner_name]);
    if element(initiator_nonce,seen_keys)
    then fail;
    else
      seen_keys ← initiator_nonce :: seen_keys;
      send([initiator_name,partner_name]);
      .....
```

Instructions in Scenarios

$$
\begin{array}{lcl}
\langle\text{sceneinst}\rangle & \rightarrow & \langle\text{ident}\rangle\ (\ \langle\text{exprlist}\rangle\ ) \\
 & | & \langle\text{l-expression}\rangle\ :=\ \langle\text{expression}\rangle \\
 & | & \underline{\text{new}}\ (\ \langle\text{l-expression}\rangle\ ) \\
 & | & \underline{\text{if}}\ \langle\text{expression}\rangle\ \underline{\text{then}}\ \langle\text{scenelist}\rangle\ (\ \underline{\text{else}}\ \langle\text{scenelist}\rangle\ )?\ \underline{\text{fi}} \\
 & | & \underline{\text{forever}}\ \langle\text{scenelist}\rangle\ \underline{\text{end}} \\
 & | & \underline{\text{parallel}}\ (\ \langle\text{sceneinst}\rangle\ |)*\ \langle\text{sceneinst}\rangle\ \underline{\text{end}} \\
 & | & \underline{\text{begin}}\ \langle\text{scenelist}\rangle\ \underline{\text{end}} \\
 & | & \underline{\text{exists}}\ \langle\text{valuedec}\rangle\ .\ \langle\text{sceneinst}\rangle \\
 & | & \underline{\text{forall}}\ \langle\text{valuedec}\rangle\ (\ \langle\text{relation-constraint}\rangle\ )?.\ \langle\text{sceneinst}\rangle \\
 & | & \underline{\text{declare}}\ \langle\text{nt-decllist}\rangle\ ;\ \underline{\text{begin}}\ \langle\text{scenelist}\rangle\ \underline{\text{end}}
\end{array}
$$

Scenarios

$$\langle \text{scenario} \rangle \quad \color{red}{\longrightarrow} \quad \underline{\text{scenario}} \ \langle \text{sceneinst} \rangle \ \underline{\text{end}}$$

– <u>forever</u> indicates an infinite sequential loop
– <u>forall</u> is a generalisation of the Bang-operator of the π-calculus

Example TMN : The Scenario

```
scenario
  begin
    makekey(rsa,serverkey);
    seen_keys ← [[ ]];
    parallel
      alice(Alice, Bob, inv(serverkey))
    | bob(Bob, inv(serverkey))
    | forever server(seen_keys, serverkey); end
    end;
  end
end
```

Declaration of Global Variables (1)

Declare for every $x_1 \in D_1$ a variable $v_1(x_1)$, and for every $x_2 \in D_2$ a variable $v_2(x_2)$.

Example in the Electronic Purse Protocol :

– declare for every name of a client a variable holding the balance of the purse,
– declare for every name of a payment terminal a variable holding the balance of the cash register

With first-order quantifiers :

$$\forall x_1 \exists v_1 \forall x_2 \exists v_2 \ldots$$

the variable v_2 depends both on x_1 and x_2.

Declaration of Global Variables (2)

Solution (example Electronic Purse) :

variables

```
        ep_balance(principal) : mutable int;
        sam_balance(principal): mutable int;
        trace(principal) : mutable list(message);
        ...
```

end

Declaration of functions which assign memory cells to values : Global Arrays.

Access by roles to global memory cells only via call-by-reference parameters.

Restrictions in Universal Quantifications (1)

In the Electronic Purse protocol : at every moment, *client* processes are interacting with *terminal processes*.

No client can talk to more than one terminal at a time (and vice versa).

```
forever
   forall purse, module: principal [ 1..1, 1..1 ] .
      parallel
         purse(...)
      |  terminal (...)
      end; # parallel
end;  # forever
```

Restrictions in Universal Quantifications (2)

A quantified expression of the form

$$\underline{\text{forall}}\ x_1 : t_1, \ldots, x_n : t_n\ \mathit{constraint}\ .\ S$$

is understood as

$$\exists R :\ \mathit{constraint}\ \forall x_1 : t_1, \ldots, x_n : t_n . R(x_1, \ldots, x_n) \Rightarrow S$$

where an n-ary relation R satisfies a constraint $[l_1..u_1, \ldots, l_n..u_n]$ iff

$$\bigwedge_{i=1}^{n} \quad \forall x_1 : t_1, \ldots, x_{i-1} : t_{i-1}, x_{i+1} : t_{i+1}, \ldots, x_n : t_n .$$

$$l_i \leq \#\{(x_1, \ldots, x_n) \mid R(x_1, \ldots, x_n)\} \leq u_i$$

What Is Missing in the PROUVÉ Language?

– user-defined types

– loop constructs in the roles

– an explicit choice construct in roles

– multiple channels, and different security models for channels

– ...

Task 5 : Weakening of the Perfect Cryptography Assumption

The Perfect Cryptography Assumption : There is no way to obtain knowledge about the plain text pertaining to a cipher text without knowing the key.

However :

- Encryption methods may satisfy certain properties : exclusive or, homomorphism, commutativity, . . .
- Nonces may be replaced by timestamps or even counters
- Dictionary attacks

Task 4 : The Verification Toolbox

Integration of various theorem proving tools developed by the project partners :

1. CASRUL [Loria] based on rewriting
2. HERMES [Verimag] approximation of “safe” messages by patterns
3. H1 [LSV] decidable class of Horn clauses

Translation from the common specification language to the input languages of the tools. Characterise restrictions and approximations of the tools.

Task 2 : Presentation of Results

– Common graphical interface
– Presentation of proofs and attacks
– Symbolic execution of attacks

Task 3 : Case Studies

- Validation at hand of realistic case studies provided by France Télécom :
  - an electronic payment protocol
  - an electronic vote protocol
- Maintenance of a data base of protocols and attacks :

  SPORE : Security Protocols Open Repository

  `http://www.lsv.ens-cachan.fr/spore`

More about PROUVÉ

- `http://www.lsv.ens-cachan.fr/prouve/`
- Parser (Frontend) : will be made available soon on the web page (free licence, probably LGPL)
- Documents as of today :
  - description of the protocol specification language
  - a survey of algebraic properties in cryptographic protocols
  - an overview on results about weakening of the perfect cryptography assumption obtained in the context of the project
  - the case study *electronic purse*