

# Advanced Cryptography

## 1st Semester 2007-2008

### Tools

**Pascal Lafourcade**

*Université Joseph Fourier, Verimag*

Master: November 7th 2007

## Last Time (I)

### Active Intruder

- Bounded Number of Sessions: Constraints Resolution
- NP-Hardness
- Unbounded Number of Sessions: Horn CLauses

Remarks, questions, comments ?

## Last Time (II)

### Exercises done

- Constraints
- Horn Clauses

## Outline of Today:

- 1 Summerize
- 2 Hermes
- 3 AVISPA
- 4 Scyther
- 5 Proverif
- 6 Comparaison
- 7 Algebraic Properties
- 8 Conclusion

# Outline

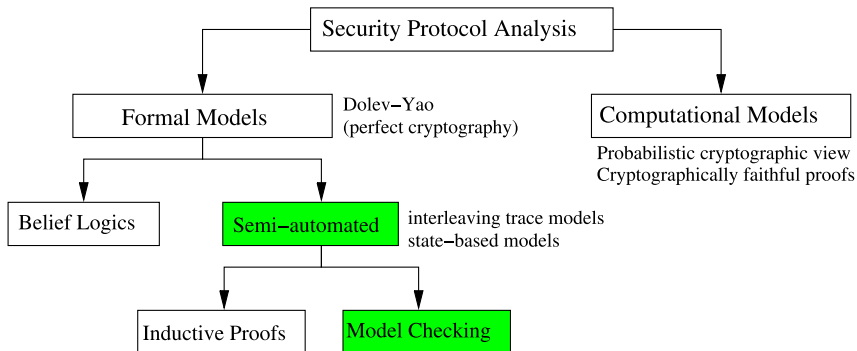
- 1 Summarize
- 2 Hermes
- 3 AVISPA
- 4 Scyther
- 5 Proverif
- 6 Comparaison
- 7 Algebraic Properties
- 8 Conclusion

# Complexity

Complexity depends of intruder capabilities. In classical Dolev-Yao intruder model we (pair + encryption) we have the following results:

- **Passive Intruder**  
Problem is **polynomial**
- **Bounded Number of sessions**  
Problem is **NP-complete**  
Tools can verify 3-4 sessions: useful to **finds flaws** ! OFMC, CI-Atse, SATMC, FDR, Athena...
- **Unbounded Number of sessions**  
Problem is in general **undecidable**  
Tools are **corrects, but uncomplete** (can find false attacks, can not terminate) Proverif, TA4SP, Scyther.

# Formal Landscape and Our Focus



N.B. Challenging as general problem is **undecidable** due e.g. to the possibility of unbounded number of protocol sessions.

## Tools studied Today

- **Hermes** : Using approximations analyzes unbounded number of sessions.
- **Avispa** : Platform with 4 tools: **OFMC**, **CL-AtSe**, **SATMC**, and **TA4SP**.
- **Proverif**: Analyses unbounded number of session using over-approximation with Horn Clauses.
- **Scyther**: Verifies bounded and unbounded number of session with backwards search based on partially ordered patterns.

# Outline

- 1 Summerize
- 2 Hermes**
- 3 AVISPA
- 4 Scyther
- 5 Proverif
- 6 Comparaison
- 7 Algebraic Properties
- 8 Conclusion

# Hermes

Hermes does not proceed by estimating the knowledge of the intruder!

- It computes a set of safe messages and uses a symbolic representation based on patterns to approximate the infinite set of safe messages.
- Hermes checks protocols for bounded and unbounded number of sessions.
  - TACAS'03 - **Pattern-based Abstraction for verifying Secrecy in Protocols**. L. Bozga, Y. Lakhnech and M. Perin
  - CAV'03 - **Hermes, a tool verifying secrecy properties of unbounded security protocols**. L. Bozga, Y. Lakhnech and M. Perin

<http://www-verimag.imag.fr/~async/hermes/prouve/hermes.php>  
developped at Verimag by Y. Bouzouzou, L. Bozga, C. Ene,  
R. Janvier, Y. Lakhnech, L. Mazaré and M. Périn.

# Welcome to Hermes!

## Online Examples

- [Needham Schroeder bounded scenario](#)
- [Needham Schroeder unbounded scenario](#)
- [Needham Schroeder iterative scenario](#)
- [Needham Schroeder Lowe bounded scenario](#)
- [Needham Schroeder Lowe unbounded scenario](#)
- [Needham Schroeder Lowe iterative scenario](#)
- [Electronic Purse symmetric keys bounded scenario](#)
- [Electronic Purse symmetric keys unbounded scenario](#)
- [Electronic Purse symmetric keys iterative scenario](#)
- [Electronic Vote bounded scenario](#)
- [Electronic Vote unbounded scenario](#)

# The Needham-Schroeder Protocole

```
# A,B :      Principal
# Na,Nb :    Nonce
# PKa,PKb,PKs,SKa,SKb,SKs :  Key
# PKa,SKa :  is a key pair
# PKb,SKb :  is a key pair
# PKs,SKs :  is a key pair
#
# 1.  A      ->  B      :      {Na, A}PKb
# 2.  B      ->  A      :      {Na, Nb}PKa
# 3.  A      ->  B      :      {Nb}PKb
```

signature

```
alice, bob, intruder : principal;
PK : principal -> pubkey;
```

end

role Alice (A: principal; B: principal; SKa: privkey)

```
declare
Na, v_Nb: message;
begin
new(Na);
send(crypt(asym,PK(B),[Na,A]));
rcv(crypt(asym,PK(A),[Na,v_Nb]));
send(crypt(asym,PK(B),v_Nb));
```

Execute:

hermes

Options: hi

not strong secret

all abstract executions

clear

public

```
alice, bob, intruder, PK
```

initial

```
x_M = [[ inv(PK(intruder)) ]]
```

## Example: Hermes I Needham-Schroeder (I)

Needham\_Schroeder\_cles\_publicques

alg : asym\_algo  
everybody knows alg

A, B: principal

Na, Nb : number

keypair<sup>alg</sup> PK, SK (principal)  
everybody knows PK

A knows A, B, SK(A)

B knows B, SK(B)

## Example: Hermes I Needham-Schroeder (I)

```
{
  1. A -> B : {A, Na}_ (PK(B))^alg
  2. B -> A : {Na, Nb}_ (PK(A))^alg
  3. A -> B : {Nb}_ (PK(B))^alg
}
```

```
s1. session *{A,B} A=A, B=B
```

```
assume secret (SK(A)@s1.A), secret (SK(B)@s1.B),
        secret (SK(B)@s1.A), secret (SK(A)@s1.B))
```

```
claim *A*G secret (SK(A)@s1.A),
        *A*G secret (SK(B)@s1.B),
        *A*G secret (Na@s1.A),
        *A*G secret (Nb@s1.B)
```

## Example: Hermes II Needham-Schroeder (I)

Signature

```
alice, bob, intruder : principal;  
PK : principal -> pubkey;
```

end

```
role Alice (A: principal; B: principal; SKa: privkey)
```

```
declare
```

```
Na, v_Nb: message;
```

```
begin
```

```
    new(Na);
```

```
    send(crypt(asm,PK(B), [Na,A]));
```

```
    rcv(crypt(asm,PK(A), [Na,v_Nb]));
```

```
    send(crypt(asm,PK(B), v_Nb)) ;
```

```
end
```

## Example: Hermes II Needham-Schroeder (II)

```
role Bob (B: principal; SKb: privkey)
  declare
    Nb, v_Na : message;
    v_A: principal;
  begin
    recv(crypt(asymp,PK(B), [v_Na,v_A]));
    new(Nb);
    send(crypt(asymp,PK(v_A), [v_Na,Nb]));
    recv(crypt(asymp,PK(B),Nb));
  end
end
```

## Example: Hermes II Needham-Schroeder (III)

```
scenario
```

```
    parallel
```

```
        Bob (bob,inv(PK(bob)))
```

```
    | Bob (bob,inv(PK(bob)))
```

```
    | Alice(alice,bob,inv(PK(alice)))
```

```
    | Alice(alice,intruder,inv(PK(alice)))
```

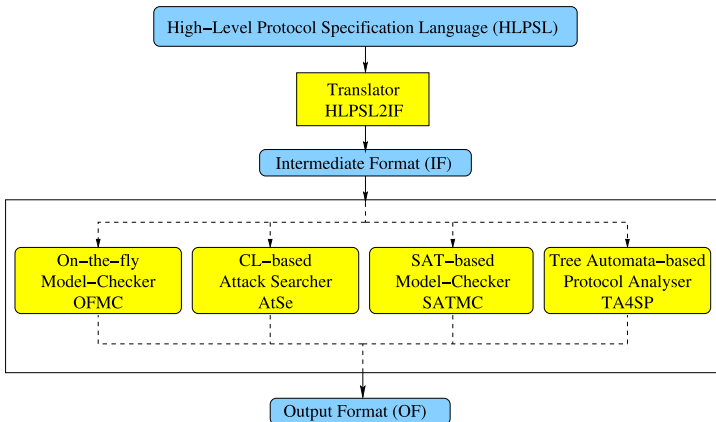
```
    end
```

```
end
```

# Outline

- 1 Summerize
- 2 Hermes
- 3 AVISPA**
- 4 Scyther
- 5 Proverif
- 6 Comparaison
- 7 Algebraic Properties
- 8 Conclusion

# The AVISPA Tool: Architecture



## The AVISPA Tool: the Back-Ends

- On-the-fly Model-Checker** (OFMC) employs several symbolic techniques to explore the state space in a demand-driven way.
- CL-AtSe** (Constraint-Logic-based Attack Searcher) applies constraint solving with simplification heuristics and redundancy elimination techniques.
- The SAT-based Model-Checker** (SATMC) builds a propositional formula encoding all the possible traces (of bounded length) on the protocol and uses a SAT solver.
- TA4SP** (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols) approximates the intruder knowledge by using regular tree languages and rewriting to produce under and over approximations.



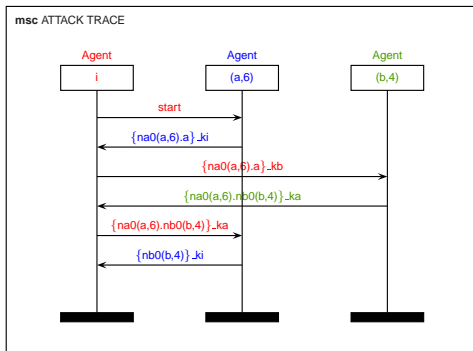
## Results in AVISPA

The screenshot displays the AVISPA web interface. At the top, the AVISPA logo is shown in red, with the text "Automated Validation of Internet Security Protocols and Applications" below it. To the right, a "Mode" selector has two buttons: "Basic" (highlighted in red) and "Expert". Below this is an "Output" section containing a text area with the following content:

```
AVISPA Tool Summary
OFMC : UNSAFE
CL-AtSe : UNSAFE
SATMC : UNSAFE
TA4SP : INCONCLUSIVE
Refer to individual tools output for details
```

Below the text area are two buttons: "HLPSL" and "IF". At the bottom of the interface, there are four tool-specific panels: "OFMC", "CL-AtSe", "SATMC", and "TA4SP". Each panel contains three buttons: "Result", "MSC", and "Postscript". The "Result" buttons are highlighted in red. In the center of the bottom section, there is a text prompt "View detailed output or Return to file selection" and a "File Selection" button.

## MSC Description of the Attack in AVISPA



## Needham-Schroeder : Alice

```
role alice (A, B      : agent,  
           Ka, Kb    : public_key,  
           SND, RCV: channel (dy))  
played_by A def=  
  local State : nat,  
        Na, Nb      : text  
init State := 0  
transition  
  0.  State = 0 /\ RCV(start) =|>  
      State' := 2 /\ Na' := new() /\ SND({Na'.A}_Kb)  
          /\ secret(Na',na,{A,B})  
  
  2.  State = 2 /\ RCV({Na.Nb'}_Ka) =|>  
      State' := 4 /\ SND({Nb'}_Kb)  
end role
```

## Needham-Schroeder: Bob

```
role bob(A, B      : agent,
         Ka, Kb    : public_key,
         SND, RCV  : channel (dy))
played_by B def=
  local State : nat,
        Na, Nb : text
init State := 1

transition
  1. State = 1 /\ RCV({Na'.A}_Kb) =|>
     State' := 3 /\ Nb' := new() /\ SND({Na'.Nb'}_Ka)
        /\ secret(Nb',nb,{A,B})

  3. State = 3 /\ RCV({Nb}_Kb) =|>
end role
```

## Needham-Schroeder: Session, Environment & Goal

```
role session(A, B: agent, Ka, Kb: public_key) def=  
  local SA, RA, SB, RB: channel (dy)  
  composition  
    alice(A,B,Ka,Kb,SA,RA) /\ bob (A,B,Ka,Kb,SB,RB)  
end role  
  
role environment() def=  
  const a, b      : agent,  
        ka, kb, ki : public_key,  
        na, nb,   : protocol_id  
  intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}  
  composition  
    session(a,b,ka,kb) /\ session(a,i,ka,ki)  
                       /\ session(i,b,ki,kb)  
end role  
goal  secrecy_of na, nb  
end goal  
environment()
```

- Agent: names of principles
- public key: asymmetric keys
- symmetric key: symmetric keys
- nat: natural numbers
- function: to model hash functions etc
- bool: Boolean values for modeling flags

Kinds of variables:

- State variables: Those that are within the scope of a role.
- Declared at the top of a role
- Unprimed versions indicate current state
- Primed versions indicate next state

## Role Definition

- 1 Role declaration: its name and the list of formal arguments, along with (in the case of basic roles) a player declaration;
- 2 Declaration of local variables and ownership rules, if any;
- 3 Initialization of variables, if required;
- 4 Declaration of accepting states, if any;
- 5 Knowledge declarations, if applicable;
- 6 Either (optionally) : a transition section (for basic roles) or a composition section (for composed roles).

# Outline

- 1 Summerize
- 2 Hermes
- 3 AVISPA
- 4 Scyther**
- 5 Proverif
- 6 Comparaison
- 7 Algebraic Properties
- 8 Conclusion

# Scyther

- Alternative: backwards search based on patterns
  - Security properties represented by claim events in the protocol.
  - Supports symmetric and asymmetric keys, cryptographic hash functions, key-tables, multiple protocols in parallel, composed keys, etc (but no user-definable algebraic functions)
  - Can perform unbounded verification of protocols
  - Provides *complete characterization* of protocol roles:  
Answer to: “after execution of a protocol role, what events must also have occurred?”
- Also state-of-art. Freely available for download for Windows, Linux and Mac OS X.
- Will be used in the exercise sessions.

## Input

```

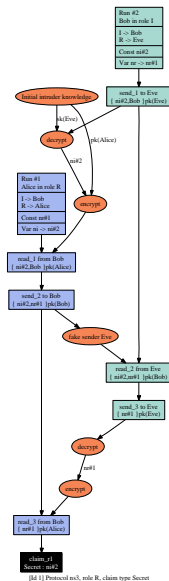
protocol ns3(I,R) {
  role I {
    const ni: Nonce;
    var nr: Nonce;
    send_1(I,R, {ni,I}pk(R) );
    read_2(R,I, {ni,nr}pk(I) );
    send_3(I,R, {nr}pk(R) );

    claim_i1(I,Secret,ni);
    claim_i2(I,Nisynch);
  }
  role R {
    var ni: Nonce;
    const nr: Nonce;
    read_1(I,R, {ni,I}pk(R) );
    send_2(R,I, {ni,nr}pk(I) );
    read_3(I,R, {nr}pk(R) );

    claim_r1(R,Secret,ni);
    claim_r2(R,Nisynch);
  }
}

```

## Output (&lt;0.02 seconds)



# Outline

- 1 Summerize
- 2 Hermes
- 3 AVISPA
- 4 Scyther
- 5 Proverif**
- 6 Comparaison
- 7 Algebraic Properties
- 8 Conclusion

# Proverif

Proverif uses spi-calculus or Horn Clauses

analyser -in horn toto.pv OR analyser -in pi toto.pv

## Proverif: Horn Clauses

```
(* Needham Shroeder Lowe *)
pred c/1 elimVar,decompData.
nounif c:x.
fun pk/1.
fun encrypt/2.

query c:secret [].
reduc

(* Initialization *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);

c:x & c:encrypt(m,pk(x)) -> c:m;
c:x -> c:pk(x);
c:x & c:y -> c:encrypt(x,y);
```

## Proverif: Horn Clauses

```
(* The protocol *)
```

```
(* A *)
```

```
c:pk(x) -> c:encrypt((Na[pk(x)], pk(sA[])), pk(x));
```

```
c:pk(x) & c:encrypt((Na[pk(x)], y), pk(sA[]))
-> c:encrypt((y,k[pk(x)]), pk(x));
```

```
(* B *)
```

```
c:encrypt((x,y), pk(sB[]))
-> c:encrypt((x, Nb[x,y], pk(sB[])), y);
```

```
c:encrypt((x,pk(sA[])))
& c:encrypt((Nb[x, pk(sA[])], z), pk(sB[]))
-> c:encrypt(secret[], pk(z)).
```

Proverif

goal reachable: c:secret[]

```

rule 7 c:secret[]
  any c:x_182
  rule 1 c:encrypt(secret[],pk(x_182))
    rule 5 c:encrypt((Na[pk(x_168)],pk(sA[])),pk(sB[]))
      2-tuple c:(Na[pk(x_168)],pk(sA[]))
      0-th c:Na[pk(x_168)]
      rule 7 c:(Na[pk(x_168)],pk(sA[]))
        any c:x_168
        rule 4 c:encrypt((Na[pk(x_168)],pk(sA[])),pk(x_168))
          rule 6 c:pk(x_168)
            any c:x_168
          rule 9 c:pk(sA[])
            rule 8 c:pk(sB[])
            rule 5 c:encrypt((Nb[Na[pk(x_168)],pk(sA[])],x_182),pk(sB[]))
              2-tuple c:(Nb[Na[pk(x_168)],pk(sA[])],x_182)
              0-th c:Nb[Na[pk(x_168)],pk(sA[])]
              rule 7 c:(Nb[Na[pk(x_168)],pk(sA[])],k[pk(x_168)])
                any c:x_168
                rule 3 c:encrypt((Nb[Na[pk(x_168)],pk(sA[])],k[pk(x_168)]),pk(x_168))
                  rule 6 c:pk(x_168)
                    any c:x_168
                  rule 2 c:encrypt((Na[pk(x_168)],Nb[Na[pk(x_168)],pk(sA[])]),pk(sA[]))
                    rule 5 c:encrypt((Na[pk(x_168)],pk(sA[])),pk(sB[]))
                      2-tuple c:(Na[pk(x_168)],pk(sA[]))
                      0-th c:Na[pk(x_168)]
                      rule 7 c:(Na[pk(x_168)],pk(sA[]))
                        any c:x_168
                        rule 4 c:encrypt((Na[pk(x_168)],pk(sA[])),pk(x_168))
                          rule 6 c:pk(x_168)
                            any c:x_168
                          rule 9 c:pk(sA[])
                            rule 8 c:pk(sB[])
                        any c:x_182
                      rule 8 c:pk(sB[])
                    any c:x_182
                  rule 8 c:pk(sB[])
                any c:x_182
              rule 8 c:pk(sB[])
            any c:x_182
          rule 8 c:pk(sB[])
        any c:x_182
      rule 8 c:pk(sB[])
    any c:x_182
  rule 8 c:pk(sB[])

```

RESULT goal reachable: c:secret[]

## What is the spi-calculus?

The **spi-calculus** is an extension of the pi-calculus designed to represent cryptographic protocols.

The **pi-calculus** is a process calculus:

- processes **communicate**: they can send and receive messages on channels several processes can **execute in parallel**.
- In the pi-calculus, messages and channels are **names**, that is, atomic values  $a, b, c, \dots$

## What is the spi-calculus ? (continued)

Example:

$$\bar{c} \langle a \rangle \mid c(x). \bar{d} \langle x \rangle$$

The first process sends  $a$  on channel  $c$ , the second one inputs this message, puts it in variable  $x$  and sends  $x$  on channel  $d$ .

The link with cryptographic protocols is clear:

- Each participant of the protocol is represented by a process
- The messages exchanged by processes are the messages of the protocol.

However, in protocols, messages are not necessarily atomic values. The names of the pi calculus are replaced by terms in the spi

calculus.

# Proverif

## Pi calculus + cryptographic primitives

$M, N ::=$	terms
$x, y, z$	variable
$a, b, c, k,$	name
$f(M_1, \dots, M_n)$	constructor application
$P, Q ::=$	processes
$\overline{M} < N > .P$	output
$M(x).P$	input
let $x = g(M_1, \dots, M_n)$ in $P$ else $Q$	destructor application
if $M = N$ then $P$ else $Q$	conditional
$0$	nil process
$P Q$	parallel composition
$!P$	replication
$(\nu a)P$	restriction

## Example: Denning Saco

Message 1.  $A \rightarrow B : \{\{k\}_{sk_A}\}_{pk_B}$

Message 2.  $B \rightarrow A \{s\}_k, k \text{ fresh}$

$(\nu sk_A)(\nu sk_B)$  let  $pk_A = pk(sk_A)$  in let  $pk_B = pk(sk_B)$  in  
 $\bar{c} < pk_A > \bar{c} < pk_B > .$

(A)  $!c(x_{pk_B}).(\nu k)\bar{c} < pencrypt(sign(k, sk_A), x_{pk_B}) >$   
 $.c(x).lets = sdecrypt(x, k)$  in 0

(B)  $!c(y).lety' = pdecrypt(y, sk_B)$  in let  $k = checksign(y', pk_A)$  in  
 $\bar{c} < sencrypt(s, k) >$

## Proverif: Pi Calculus

```
free c.
```

```
(* Public key cryptography *)
```

```
fun pk/1.
```

```
private fun sk/1.
```

```
(* just encryption, no signing *)
```

```
fun encrypt/2.
```

```
reduc decrypt(encrypt(x,pk(y)),sk(y)) = x.
```

```
(* Symmetric key cryptography *)
```

```
fun symcrypt/2.
```

```
reduc symdecrypt(symcrypt(z,j),j) = z.
```

```
(* Effectively the claim signals *)
```

```
private free secretANa, secretANb, secretBNa, secretBNb, secretAtoB, secretBtoA
```

```
(* Security claims to verify *)
```

```
query attacker:secretANa;
```

```
attacker:secretANb;
```

```
attacker:secretAtoB;
```

```
attacker:secretBNa;
```

```
attacker:secretBNb;
```

```
attacker:secretBtoA.
```

## Proverif: Pi Calculus

```
let processA =
  (* Choose the other host *)
  in(c, X);
  new Na;
  out(c, encrypt((Na,X),pk(X)));
  in(c,m2);
  let (=Na, nb) = decrypt(m2, sk(A)) in
  out(c, encrypt(nb,pk(X)));
  if X = A then
    out(c, symcrypt(secretANa, Na));
    out(c, symcrypt(secretANb, nb))
  else
    if X = B then
      out(c, symcrypt(secretAtoB, Na));
      out(c, symcrypt(secretAtoB, nb));
      out(c, symcrypt(secretANa, Na));
      out(c, symcrypt(secretANb, nb)). 41 / 57
```

## Proverif: Pi Calculus

```
let processB =
  in(c,m1);
  let (na,Y) = decrypt(m1, sk(B)) in
  new Nb;
  out(c, encrypt((na, Nb), pk(Y)));
  in(c,m3);
  let (=Nb) = decrypt(m3, sk(B)) in
  if Y = A then
    out(c, symcrypt(secretBtoA, na));
    out(c, symcrypt(secretBtoA, Nb));
    out(c, symcrypt(secretBNa, na));
    out(c, symcrypt(secretBNb, Nb))
  else
    if Y = B then
      out(c, symcrypt(secretBNa, na));
      out(c, symcrypt(secretBNb, Nb)).
```

## Proverif: Pi Calculus

```
let processBbyA =
  in(c,m1);
  let (na,Y) = decrypt(m1, sk(A)) in
  new Nb;
  out(c, encrypt((na, Nb), pk(Y)));
  in(c,m3);
  let (=Nb) = decrypt(m3, sk(A)) in
  if Y = A then
    out(c, symcrypt(secretBtoA, na));
    out(c, symcrypt(secretBtoA, Nb));
    out(c, symcrypt(secretBNa, na));
    out(c, symcrypt(secretBNb, Nb))
  else
    if Y = B then
      out(c, symcrypt(secretBNa, na));
      out(c, symcrypt(secretBNb, Nb)).
```

# Proverif: Pi Calculus

```
process
```

```
  new A;
```

```
  new B;
```

```
  new I;
```

```
  out(c,A);
```

```
  out(c,B);
```

```
  out(c,I);
```

```
  out(c,sk(I));
```

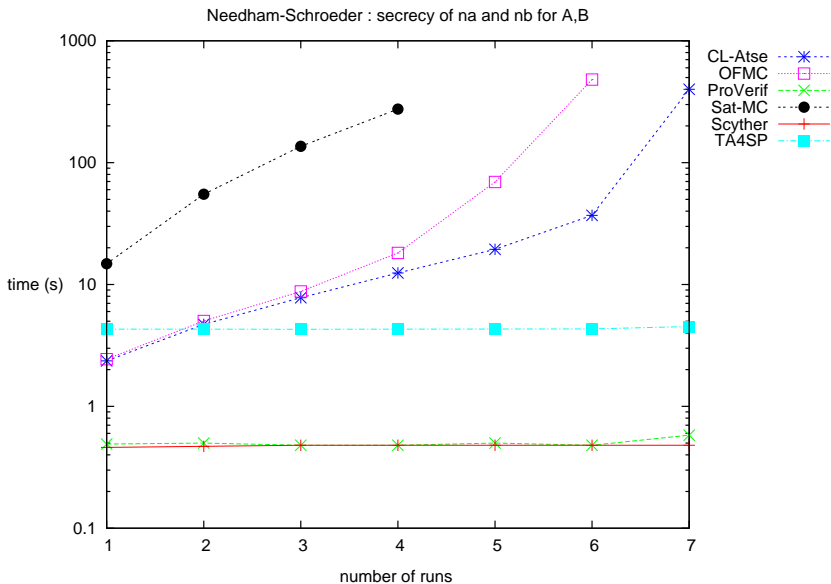
```
((!processA) | (!processB) | (!processBbyA))
```

## Proverif: Pi Calculus

```
RESULT not attacker:secretANa[] is true.  
RESULT not attacker:secretANb[] is false.  
RESULT not attacker:secretAtoB[] is true.  
RESULT not attacker:secretBNa[] is false.  
RESULT not attacker:secretBNb[] is false.  
RESULT not attacker:secretBtoA[] is false.
```

# Outline

- 1 Summerize
- 2 Hermes
- 3 AVISPA
- 4 Scyther
- 5 Proverif
- 6 Comparaison**
- 7 Algebraic Properties
- 8 Conclusion



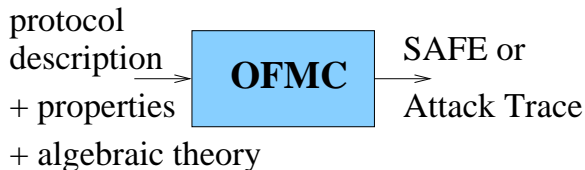
## Bibliography

- **Time performance comparison of AVISPA Tools**  
L. Vigano “Automated Security Protocol Analysis With the AVISPA Tool” ENTCS 2006.
- **Usability comparison between AVISPA and HERMES**  
M. Hussain and D. Seret “A Comparative study of Security Protocols Validation Tools: HERMES vs. AVISPA”.In the 8th International Conference Advanced Communication Technology, ICACT’06.
- Cas Cremers and Pascal Lafourcade. **Comparing State Spaces in Automatic Security Protocol Verification**. In Michael Goldsmith and Bill Roscoe (eds.), Proceedings of the 7th International Workshop on Automated Verification of Critical Systems (AVoCS’07), Oxford, UK, September 2007, Electronic Notes in Theoretical Computer Science, pages 49-63. Elsevier Science Publishers.

# Outline

- 1 Summerize
- 2 Hermes
- 3 AVISPA
- 4 Scyther
- 5 Proverif
- 6 Comparaison
- 7 Algebraic Properties**
- 8 Conclusion

## Model Checking with OFMC



### Input:

- Transition system (initial state + transition relation).
- Goal : insecure states (i.e., attacks).
- Algebraic properties.

### Output:

- **SAFE** indicating security for bounded sessions **or**
- An attack trace.

## Supported Theories by OFMC

$$(x_1^{x_2})^{x_3} \approx (x_1^{x_3})^{x_2}$$

$$x_1 \oplus x_2 \approx x_2 \oplus x_1$$

$$x_1 \oplus (x_2 \oplus x_3) \approx (x_1 \oplus x_2) \oplus x_3$$

$$\text{dec}(x_1, \{x_1\}_{x_2}) \approx x_2$$

$$x_1 \oplus x_1 \approx 0$$

$$x_1 \oplus 0 \approx x_1$$

Finite Theories  $F$ :

The  $F$ -equivalence class of every term is finite.

Cancellation theories  $C$ :

One side of each equation is a variable of the other side, or a constant.

## Supported Theories by OFMC

$$(x_1^{x_2})^{x_3} \approx (x_1^{x_3})^{x_2}$$

$$x_1 \oplus x_2 \approx x_2 \oplus x_1$$

$$x_1 \oplus (x_2 \oplus x_3) \approx (x_1 \oplus x_2) \oplus x_3$$

$$\text{dec}(x_1, \{x_1\}_{x_2}) \approx x_2$$

$$x_1 \oplus x_1 \approx 0$$

$$x_1 \oplus 0 \approx x_1$$

Finite Theories  $F$ :

The  $F$ -equivalence class of every term is finite.

Rewriting with  $C$  modulo  $F$ , e.g.

$$a \oplus b \oplus a \rightarrow_{C/F} 0 \oplus b \rightarrow_{C/F} b.$$

We require:  $\rightarrow_{C/F}$  is convergent.

Cancellation theories  $C$ :

One side of each equation is a variable of the other side, or a constant.

## On-the-Fly Model-Checker (OFMC)

- Common language for specifying protocols and security properties.
- Supports symmetric and asymmetric keys, cryptographic hash functions, key-tables, user-definable algebraic functions, etc.

Input	Output (<1 second)
<pre> PROTOCOL Needham-Schroeder; Identifiers   A, B: user;   Na, Nb: nonce;   Ka, Kb: public_key; Messages   1. A -&gt; B: {A,Na}Kb   2. B -&gt; A: {Na,Nb}Ka   3. A -&gt; B: {Nb}Kb Intruder_knowledge Spy, b, ka, kb, kspy; Goal correspondence_between A B;           </pre>	<pre> A -&gt; Spy: {A,Na}Kspy Spy -&gt; B: {A,Na}Kb B -&gt; A: {Na,Nb}Ka A -&gt; Spy: {Nb}Kspy Spy -&gt; B {Nb}Kb           </pre>

## Secure or not ?

One protocol:  $K$  secret key between A and B ?

$$A \rightarrow S : A, B, \{A \oplus N_A\}K_S, \{N_A \oplus c\}K_S$$

$$S \rightarrow B : A, B, S$$

$$B \rightarrow S : B, A, \{B \oplus N_B\}K_S, \{N_B \oplus c\}K_S$$

$$S \rightarrow A : K \oplus \{N_A\}K_S$$

$$S \rightarrow B : K \oplus \{N_B\}K_S$$

OFMC answers : SAFE with exclusive-or  $\oplus$

## Secure or not ?

One protocol:  $K$  secret key between A and B ?

$$A \rightarrow S: A, B, \{A \oplus N_A\}_{K_S}, \{N_A \oplus c\}_{K_S}$$

$$S \rightarrow B: A, B, S$$

$$B \rightarrow S: B, A, \{B \oplus N_B\}_{K_S}, \{N_B \oplus c\}_{K_S}$$

$$S \rightarrow A: K \oplus \{N_A\}_{K_S}$$

$$S \rightarrow B: K \oplus \{N_B\}_{K_S}$$

OFMC answers : SAFE with exclusive-or  $\oplus$

$$\text{But with } \{x \oplus y\}_{K_S} = \{x\}_{K_S} \oplus \{y\}_{K_S}$$

There is an attack !

# Now

## Playing with Tools

- Hermes
- Scyther
- Avispa: OFMC, CI-Atse, SATMC, TA4SP
- Proverif

# Outline

- 1 Summerize
- 2 Hermes
- 3 AVISPA
- 4 Scyther
- 5 Proverif
- 6 Comparaison
- 7 Algebraic Properties
- 8 Conclusion**

# Summary

## Today

- Hermes
- Scyther
- Avispa
- Proverif

**Thank you for your attention.**

**Questions ?**