

Protocols

Pascal Lafourcade

Université Joseph Fourier, Verimag

October 6th 2008

Last Time

- Historic and Motivation

General Schedule

- 1 Lundi 15 septembre **Historique de la cryptographie**
- 2 Lundi 22 septembre **Chiffrements Symétriques**
- 3 Lundi 29 septembre
Chiffrements asymétriques et applications
- 4 Lundi 6 octobre
Cryptanalyse des chiffrements asymétriques I
- 5 Lundi 13 octobre **Protocoles**
- 6 Lundi 20 octobre
Cryptanalyse des chiffrements asymétriques II
- 7 Lundi 10 novembre **Fonctions de Hachage**
- 8 Lundi 17 novembre **Trouver des failles avec un ordinateur**
B110
- 9 Lundi 24 novembre
Applications : E-voting, Secret Sharing ...
- 10 Lundi 1 decembre **Ouverture: courbes elliptiques, codes ...**

First Draft of the Project's List

- Elgamal
- RSA
- DES
- AES
- MD5
- SHA1
- IDEA
- Cryptanalyse
- Merkel-Hellman
- E-voting protocols
- E-auction Protocols
- Secret Sharing
- Weakness of RC4

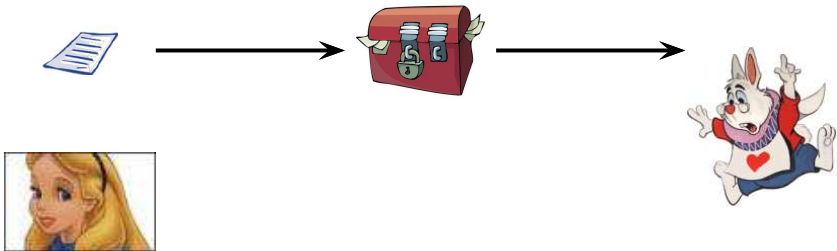
Outline

- 1 Logical Attacks
- 2 Intruder
- 3 Needham Schroeder
- 4 Undecidability for unbounded number of sessions
- 5 Verification: introduction to tools
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison
- 7 Conclusion

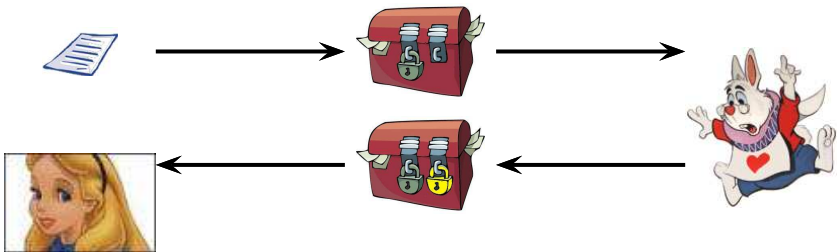
Outline

- 1 Logical Attacks
- 2 Intruder
- 3 Needham Schroeder
- 4 Undecidability for unbounded number of sessions
- 5 Verification: introduction to tools
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison
- 7 Conclusion

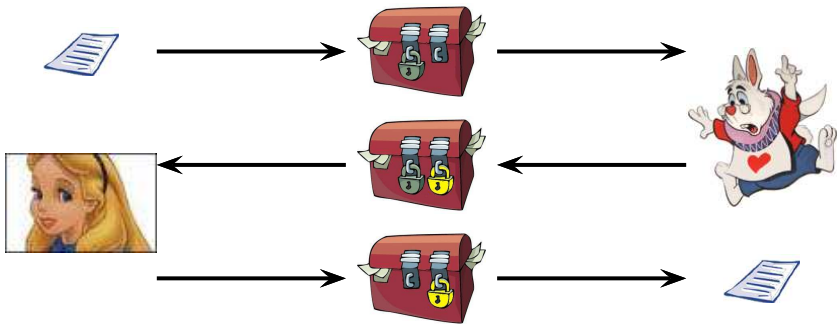
Example :



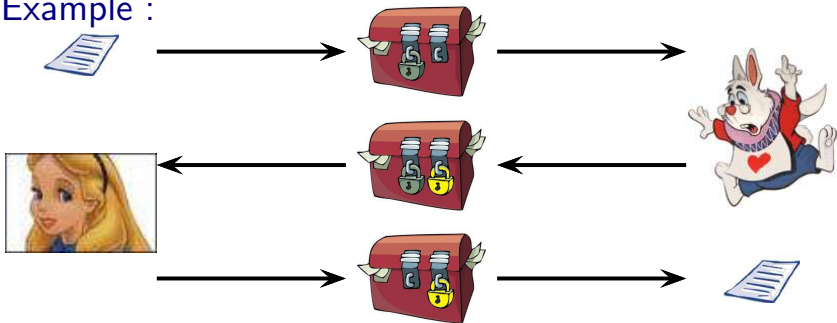
Example :



Example :



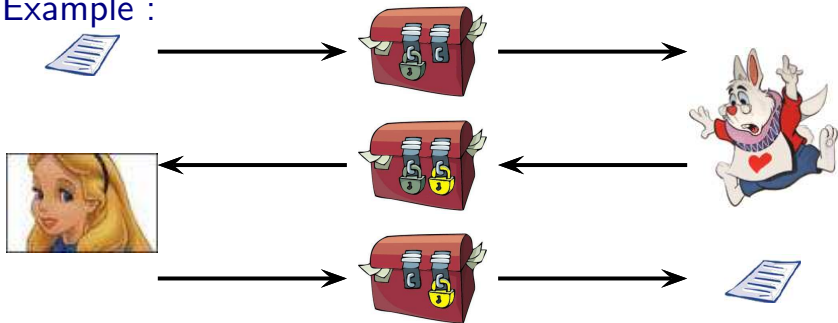
Example :



Shamir 3-Pass Protocol

$$1 \quad A \rightarrow B : \{m\}_{K_A}$$

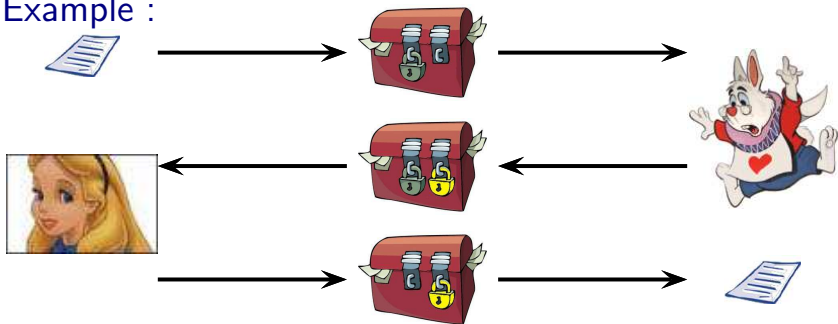
Example :



Shamir 3-Pass Protocol

- 1 $A \rightarrow B : \{m\}_{K_A}$
- 2 $B \rightarrow A : \{\{m\}_{K_A}\}_{K_B}$

Example :



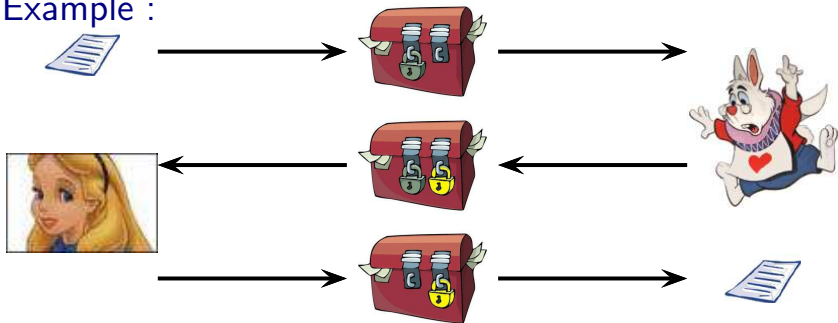
Shamir 3-Pass Protocol

$$1 \quad A \rightarrow B : \{m\}_{K_A}$$

$$2 \quad B \rightarrow A : \{\{m\}_{K_A}\}_{K_B} = \{\{m\}_{K_B}\}_{K_A}$$

Commutative
Encryption

Example :



Shamir 3-Pass Protocol

- 1 $A \rightarrow B : \{m\}_{K_A}$
- 2 $B \rightarrow A : \{\{m\}_{K_A}\}_{K_B} = \{\{m\}_{K_B}\}_{K_A}$
- 3 $A \rightarrow B : \{m\}_{K_B}$

Commutative
Encryption

Attacks

Computational Model Cryptanalysis



Attacks

Computational Model Cryptanalysis



Attacks

Computational Model
Cryptanalysis



Symbolic Model
Logical Attack

Perfect Encryption hypothesis

Needham-Schroeder Public Key Protocol (1978)

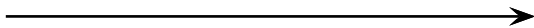
“Man in the middle attack” [Lowe'96]



Simple Example



$\{12h10\}_{K_B}$



Simple Example

 $\{12h10\}_{K_B}$  $\{12h10\}_{K_B}$ 

Simple Example



Day After



Simple Example



Day After

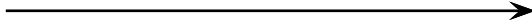


This kind of attack is valid for all encryptions

Another Simple Example using RSA



$\{12345\}_{K_A}$



Another Simple Example using RSA

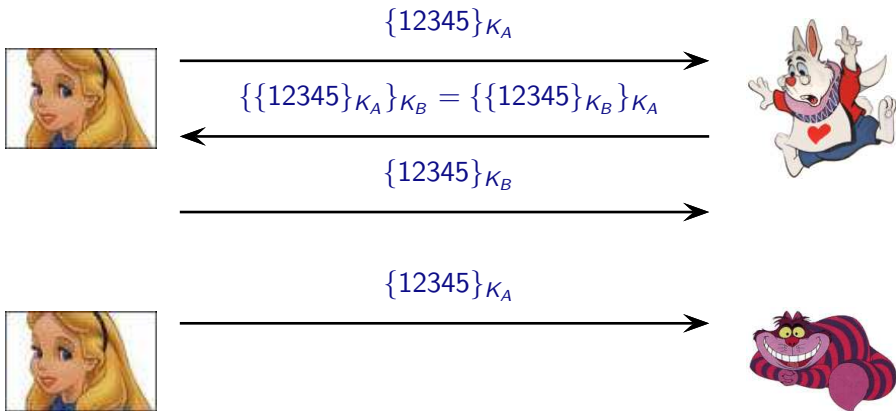

$$\{12345\}_{K_A}$$

$$\{\{12345\}_{K_A}\}_{K_B} = \{\{12345\}_{K_B}\}_{K_A}$$


Another Simple Example using RSA

 $\{12345\}_{K_A}$  $\{\{12345\}_{K_A}\}_{K_B} = \{\{12345\}_{K_B}\}_{K_A}$  $\{12345\}_{K_B}$ 

Another Simple Example using RSA



Another Simple Example using RSA



$$\{12345\}_{K_A}$$


$$\{\{12345\}_{K_A}\}_{K_B} = \{\{12345\}_{K_B}\}_{K_A}$$


$$\{12345\}_{K_B}$$


$$\{12345\}_{K_A}$$

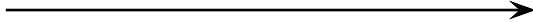

$$\{\{12345\}_{K_A}\}_{K_I} = \{\{12345\}_{K_I}\}_{K_A}$$


Another Simple Example using RSA



$$\{12345\}_{K_A}$$


$$\{\{12345\}_{K_A}\}_{K_B} = \{\{12345\}_{K_B}\}_{K_A}$$


$$\{12345\}_{K_B}$$


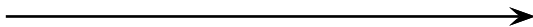
$$\{12345\}_{K_A}$$

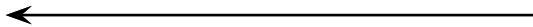

$$\{\{12345\}_{K_A}\}_{K_I} = \{\{12345\}_{K_I}\}_{K_A}$$


$$\{12345\}_{K_I}$$

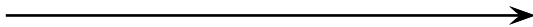

Another Simple Example using RSA

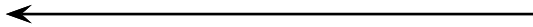


$$\{12345\}_{K_A}$$


$$\{\{12345\}_{K_A}\}_{K_B} = \{\{12345\}_{K_B}\}_{K_A}$$


$$\{12345\}_{K_B}$$


$$\{12345\}_{K_A}$$


$$\{\{12345\}_{K_A}\}_{K_I} = \{\{12345\}_{K_I}\}_{K_A}$$


$$\{12345\}_{K_I}$$


Problem of Authentication

Examples of kinds of attack

- **Man-in-the-middle (or parallel sessions) attack**: pass messages through to another session $A \leftrightarrow I \leftrightarrow B$.
- **Replay (or freshness) attack**: record and later re-introduce a message or part.
- **Reflection attack**: send transmitted information back to originator.
- **Oracle attack**: take advantage of normal protocol responses as encryption and decryption “services”.
- **Type flaw (confusion) attack**: substitute a different type of message field (e.g. a key vs. a name).

Outline

- 1 Logical Attacks
- 2 Intruder**
- 3 Needham Schroeder
- 4 Undecidability for unbounded number of sessions
- 5 Verification: introduction to tools
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison
- 7 Conclusion

The Intruder is the Network (Worst Case)



The Intruder is the Network (Worst Case)



Listen

Passive: Intruder deduction problem

The Intruder is the Network (Worst Case)



Listen

Intercept message

(Re)play message

Delete message

Passive: Intruder deduction problem

Active: Security problem

The Intruder is the Network (Worst Case)



Listen

Intercept message

(Re)play message

Delete message

Passive: Intruder deduction problem

Active: Security problem

Intruder Capabilities (Dolev-Yao Model 80's)

- Encryption, Decryption with a key
- Pairing, Projection.

Dolev-Yao 1982

- Intruder controls the network and can:
 - intercept messages
 - modify messages
 - block messages
 - generate new messages
 - insert new messages
- Perfect cryptography:
 - Abstraction with terms algebra
 - Decryption only if inverse key is known
- Protocol has
 - Arbitrary number of principals
 - Arbitrary number of parallel sessions
 - Messages with arbitrary size

Outline

- 1 Logical Attacks
- 2 Intruder
- 3 Needham Schroeder**
- 4 Undecidability for unbounded number of sessions
- 5 Verification: introduction to tools
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison
- 7 Conclusion

Messages Abstraction

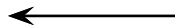
- Names: A , B or Alice, Bob, ...
- Nonces: N_A . Fresh data.
- Keys: K and **inverse keys** K^{-1}
- Asymmetric Encryption: $\{M\}_{K_A}$
- Symmetric Encryption: $\{M\}_{K_{AB}}$.
- Message concatenation: $\langle M_1, M_2 \rangle$.

Example: $\{\langle A \oplus N_B, K_{AB} \rangle\}_{K_B}$.

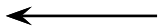
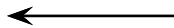
Example: Needham-Schroeder Protocol 1978

 $\{A, N_A\}_{K_B}$ 

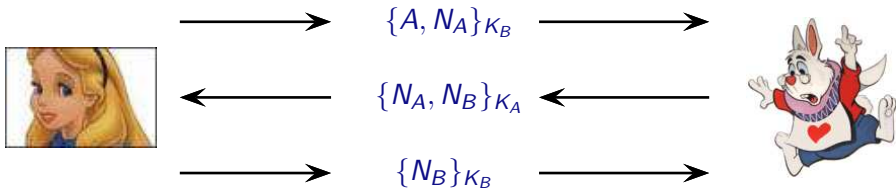
Example: Needham-Schroeder Protocol 1978

 $\{A, N_A\}_{K_B}$  $\{N_A, N_B\}_{K_A}$ 

Example: Needham-Schroeder Protocol 1978

 $\{A, N_A\}_{K_B}$  $\{N_A, N_B\}_{K_A}$  $\{N_B\}_{K_B}$ 

Example: Needham-Schroeder Protocol 1978



Question

- Is N_B a shared secret between A et B ?

Exercise

Answer

- In 1995, G.Lowe find an attack **17 years** after its publication!

Exercise: Try to Find an attack on this famous protocol.

If you already know this attack try to correct the protocol to avoid this attack.

If you already know the Lowe correction too try to find a flaw on this corrected protocol (Very Difficult)

Lowé Attack on the Needham-Schroeder

so-called “Man in the middle attack”



Agent A



Intruder I



Agent B

$$\begin{aligned}
 A &\longrightarrow B : \{A, N_a\}_{K_B} \\
 B &\longrightarrow A : \{N_a, N_b\}_{K_A} \\
 A &\longrightarrow B : \{N_b\}_{K_B}
 \end{aligned}$$

Lowé Attack on the Needham-Schroeder

so-called “Man in the middle attack”



Agent A

$\{A, N_a\}_{K_I}$



Intruder I

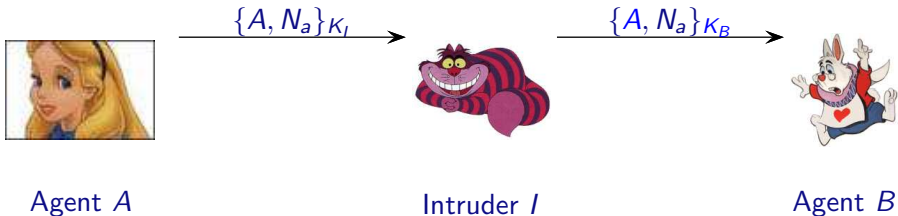


Agent B

- $A \longrightarrow B : \{A, N_a\}_{K_B}$
 $B \longrightarrow A : \{N_a, N_b\}_{K_A}$
 $A \longrightarrow B : \{N_b\}_{K_B}$

Low Attack on the Needham-Schroeder

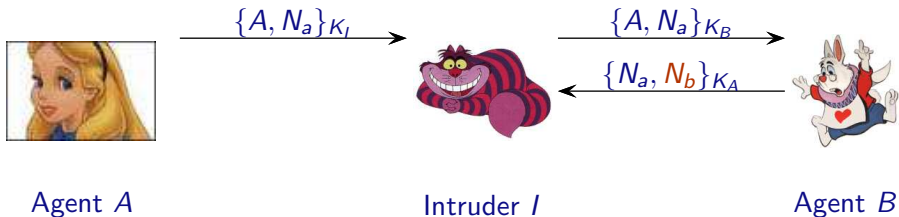
so-called “Man in the middle attack”



- $A \longrightarrow B : \{A, N_a\}_{K_B}$
 $B \longrightarrow A : \{N_a, N_b\}_{K_A}$
 $A \longrightarrow B : \{N_b\}_{K_B}$

Low Attack on the Needham-Schroeder

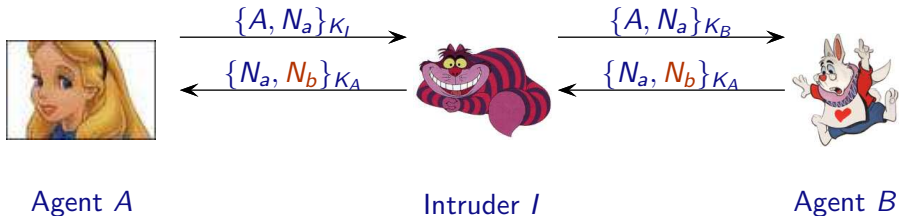
so-called “Man in the middle attack”



- $A \longrightarrow B : \{A, N_a\}_{K_B}$
 $B \longrightarrow A : \{N_a, N_b\}_{K_A}$
 $A \longrightarrow B : \{N_b\}_{K_B}$

Low Attack on the Needham-Schroeder

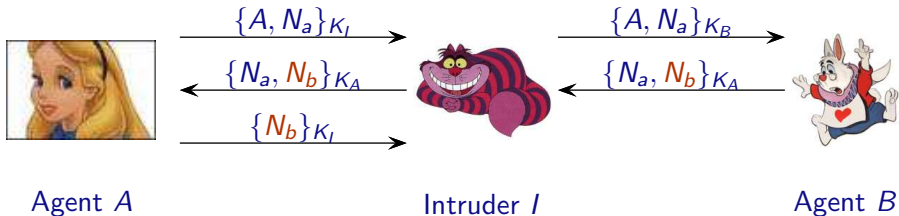
so-called “Man in the middle attack”



- $A \longrightarrow B : \{A, N_a\}_{K_B}$
 $B \longrightarrow A : \{N_a, N_b\}_{K_A}$
 $A \longrightarrow B : \{N_b\}_{K_B}$

Low Attack on the Needham-Schroeder

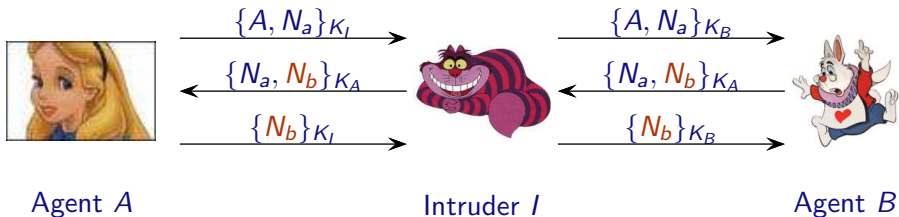
so-called “Man in the middle attack”



- $A \longrightarrow B : \{A, N_a\}_{K_B}$
- $B \longrightarrow A : \{N_a, N_b\}_{K_A}$
- $A \longrightarrow B : \{N_b\}_{K_B}$

Low Attack on the Needham-Schroeder

so-called “Man in the middle attack”

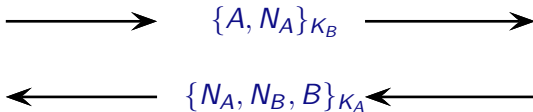


- $A \longrightarrow B : \{A, N_a\}_{K_B}$
- $B \longrightarrow A : \{N_a, N_b\}_{K_A}$
- $A \longrightarrow B : \{N_b\}_{K_B}$

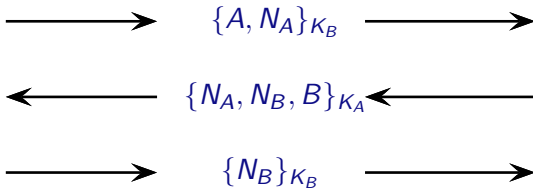
Needham-Schroeder corrected by Lowe 1995


$$\{A, N_A\}_{K_B}$$

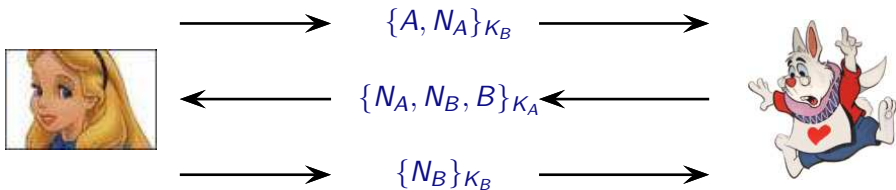

Needham-Schroeder corrected by Lowe 1995



Needham-Schroeder corrected by Lowe 1995



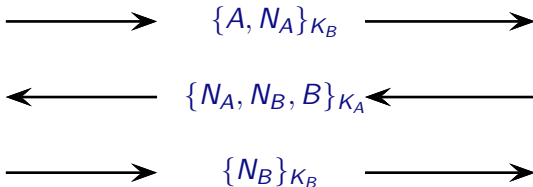
Needham-Schroeder corrected by Lowe 1995



Question

- This time the protocol is secure?

Needham-Schroeder corrected by Lowe 1995



Question

- This time the protocol is secure?

Answer

- There exists a type flaw attack.

Type Flaw Attack on the Needham-Schroeder-Lowe



Agent A



Intruder I



Agent B

$$\begin{aligned} A &\longrightarrow B : \{A, N_a\}_{K_B} \\ B &\longrightarrow A : \{N_a, N_b, B\}_{K_A} \\ A &\longrightarrow B : \{N_b\}_{K_B} \end{aligned}$$

Type Flaw Attack on the Needham-Schroeder-Lowe



Agent A



Intruder I

$$\xrightarrow{\{A, I\}_{K_B}}$$


Agent B

- $$A \longrightarrow B : \{A, N_a\}_{K_B}$$

$$B \longrightarrow A : \{N_a, N_b, B\}_{K_A}$$

$$A \longrightarrow B : \{N_b\}_{K_B}$$

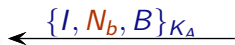
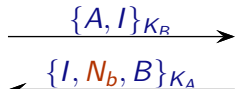
Type Flaw Attack on the Needham-Schroeder-Lowe



Agent A



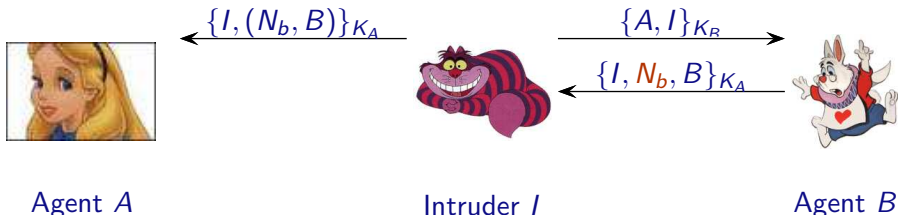
Intruder I



Agent B

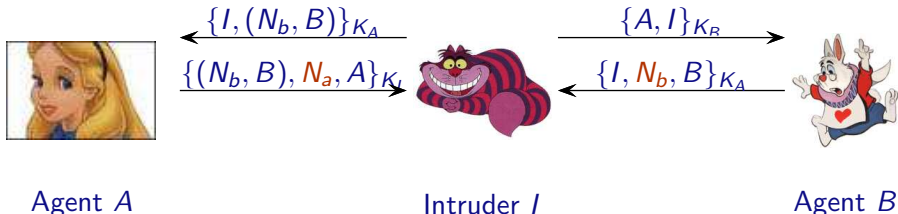
- $A \longrightarrow B : \{A, N_a\}_{K_B}$
 $B \longrightarrow A : \{N_a, N_b, B\}_{K_A}$
 $A \longrightarrow B : \{N_b\}_{K_B}$

Type Flaw Attack on the Needham-Schroeder-Lowe



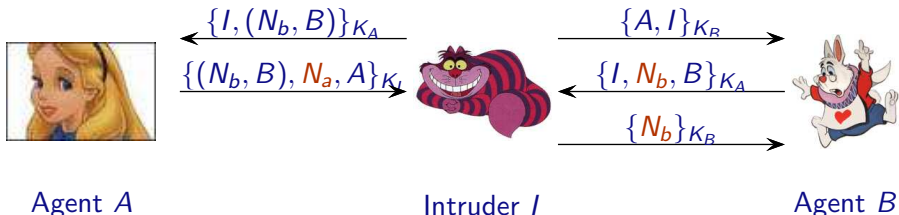
- $A \longrightarrow B : \{A, N_a\}_{K_B}$
 $B \longrightarrow A : \{N_a, N_b, B\}_{K_A}$
 $A \longrightarrow B : \{N_b\}_{K_B}$

Type Flaw Attack on the Needham-Schroeder-Lowe



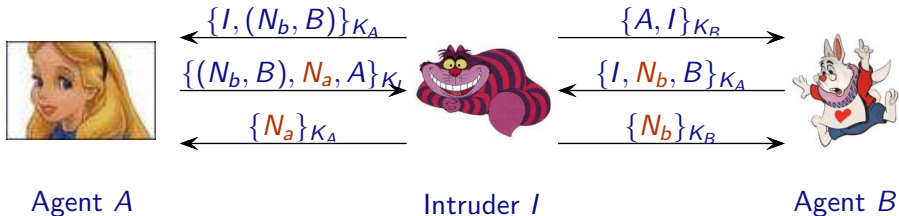
- $A \longrightarrow B : \{A, N_a\}_{K_B}$
 $B \longrightarrow A : \{N_a, N_b, B\}_{K_A}$
 $A \longrightarrow B : \{N_b\}_{K_B}$

Type Flaw Attack on the Needham-Schroeder-Lowe



- $A \longrightarrow B : \{A, N_a\}_{K_B}$
- $B \longrightarrow A : \{N_a, N_b, B\}_{K_A}$
- $A \longrightarrow B : \{N_b\}_{K_B}$

Type Flaw Attack on the Needham-Schroeder-Lowe



- $A \longrightarrow B : \{A, N_a\}_{K_B}$
- $B \longrightarrow A : \{N_a, N_b, B\}_{K_A}$
- $A \longrightarrow B : \{N_b\}_{K_B}$

Questions?

How can we find such attacks?

- Models for Protocols
- Models for Properties
- Theories
- Dedicated Techniques
- Tools
 - Automatic
 - Semi-automatic

Why is it difficult to verify such protocols?

- Messages: Size not bounded
- Nonces: Arbitrary number
- Channel: Unsecure
- Intruder: Unlimited capabilities
- Instances: Unbounded numbers of principals
- Interleaving: Unlimited applications of the protocol.

Outline

- 1 Logical Attacks
- 2 Intruder
- 3 Needham Schroeder
- 4 Undecidability for unbounded number of sessions**
- 5 Verification: introduction to tools
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison
- 7 Conclusion

Main Results

In general security problem **undecidable** [DLMS'99, AC'01]

Bounded number of session \Rightarrow **Decidability** [AL'00, RT'01]

Undecidability

Definition (Post Correspondance Problem (PCP))

Let Σ be a finite alphabet.

Input : Sequence of pairs $\langle u_i, v_i \rangle_{1 \leq i \leq n}$ $u_i, v_i \in \Sigma^*$, $n \in \mathbb{N}$

Question : Existence of $k, i_1, \dots, i_k \in \mathbb{N}$ such that

$$u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}?$$

Undecidability

Definition (Post Correspondance Problem (PCP))

Let Σ be a finite alphabet.

Input : Sequence of pairs $\langle u_i, v_i \rangle_{1 \leq i \leq n}$ $u_i, v_i \in \Sigma^*$, $n \in \mathbb{N}$

Question : Existence of $k, i_1, \dots, i_k \in \mathbb{N}$ such that

$$u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}?$$

Example

u_1	u_2	u_3	u_4	v_1	v_2	v_3	v_4
<i>aba</i>	<i>bbb</i>	<i>aab</i>	<i>bb</i>	<i>a</i>	<i>aaa</i>	<i>abab</i>	<i>babba</i>

Solution: **1431**

$$u_1 \cdot u_4 \cdot u_3 \cdot u_1 = aba \cdot bb \cdot aab \cdot aba = a \cdot babba \cdot abab \cdot a = v_1 \cdot v_4 \cdot v_3 \cdot v_1$$

But no solution for $\langle \mathbf{u}_1, \mathbf{v}_1 \rangle, \langle \mathbf{u}_2, \mathbf{v}_2 \rangle, \langle \mathbf{u}_3, \mathbf{v}_3 \rangle$

Undecidability

Definition (Post Correspondance Problem (PCP))

Let Σ be a finite alphabet.

Input : Sequence of pairs $\langle u_i, v_i \rangle_{1 \leq i \leq n}$ $u_i, v_i \in \Sigma^*$, $n \in \mathbb{N}$

Question : Existence of $k, i_1, \dots, i_k \in \mathbb{N}$ such that

$$u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}?$$

Example

u_1	u_2	u_3	u_4	v_1	v_2	v_3	v_4
<i>aba</i>	<i>bbb</i>	<i>aab</i>	<i>bb</i>	<i>a</i>	<i>aaa</i>	<i>abab</i>	<i>babba</i>

Solution: **1431**

$$u_1 \cdot u_4 \cdot u_3 \cdot u_1 = aba \cdot bb \cdot aab \cdot aba = a \cdot babba \cdot abab \cdot a = v_1 \cdot v_4 \cdot v_3 \cdot v_1$$

But no solution for $\langle \mathbf{u}_1, \mathbf{v}_1 \rangle, \langle \mathbf{u}_2, \mathbf{v}_2 \rangle, \langle \mathbf{u}_3, \mathbf{v}_3 \rangle$

PCP is undecidable

Undecidability for Protocols

We construct a protocol such that decidability of secret implies decidability of PCP.

$$A : \text{ send}(\{\langle u_i, v_i \rangle\}_{K_{ab}}) \quad (1 \leq i \leq n)$$

$$B : \text{ receive}(\{\langle x, y \rangle\}_{K_{ab}}) \\ \text{ send}(\{\langle \langle x \cdot u_i, y \cdot v_i \rangle\}_{K_{ab}}, \{s\}_{\{\langle x \cdot u_i, x \cdot u_i \rangle\}_{K_{ab}}}\}) \quad (1 \leq i \leq n)$$

We assume that \mathbf{K}_{AB} is a shared key between **A** and **B**.

Intruder can find \mathbf{s} iff he can solve PCP.

Outline

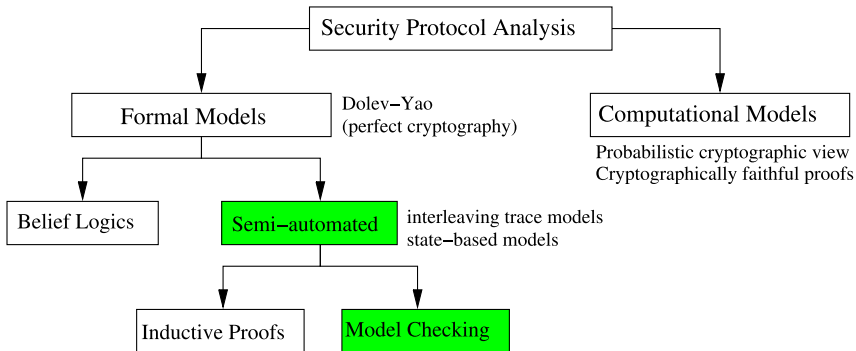
- 1 Logical Attacks
- 2 Intruder
- 3 Needham Schroeder
- 4 Undecidability for unbounded number of sessions
- 5 Verification: introduction to tools**
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison
- 7 Conclusion

Complexity

Complexity depends of intruder capabilities. In classical Dolev-Yao intruder model (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 **find flaws** ! OFMC, CI-Atse, SATMC, FDR, Athena...
- **Unbounded Number of sessions**
Problem is in general **undecidable**
Tools are **correct, 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

- **Avispa** : Plateform 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.

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:
Answers 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) );

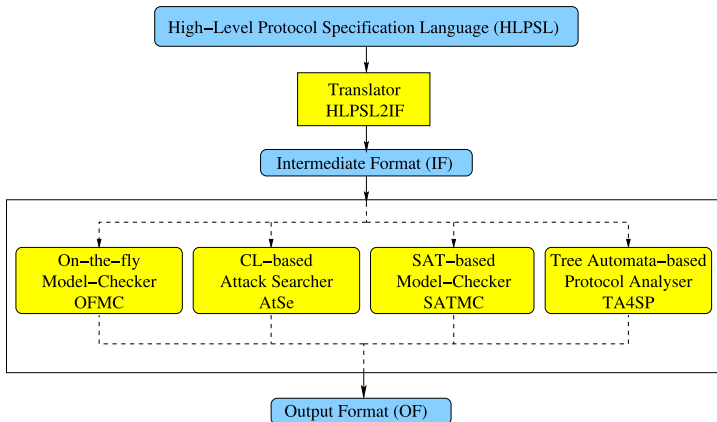
    claim_r1(R,Secret,ni);
    claim_r2(R,Nisynch);
  }
}

```

Output (<0.02 seconds)



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.

The Web Interface www.avispa-project.org

The screenshot displays the AVISPA Web Tool interface in a Mozilla browser window. The main page features the AVISPA logo and the text "Automated Validation of Internet Security Protocols and Applications". A "Protocol" section is visible, detailing the H.530 symmetric security procedures. A "Tools" sidebar on the left lists various analysis tools: HPSL, HPSLRF, F, CFMC, RTSE, SPTMC, and TR4SP.

An "Mode" window is open, showing the configuration for the "VisitedDateKeeper" role. The configuration includes variables for agent, channel, function, symmetric key, and nonce, along with a list of states and transitions.

An "msc ATTACK TRACE" window is also open, showing a sequence of messages between three agents: Agent i, Agent (a.3), and Agent (a.7). The messages are as follows:

- Agent i sends `start` to Agent (a.3).
- Agent (a.3) sends `a.b.ni.CH1(1).exp(g.X(1)).!(zz.a.auf.a.b.ni.CH1(1).exp(g.X(1)))` to Agent i.
- Agent (a.3) sends `b.a.ni.x99.g.x92` to Agent (a.7).
- Agent (a.7) sends `b.a.ni.x99.g.x92.a.g` to Agent (a.3).
- Agent (a.3) sends `a.b.ni.CH1(1).exp(g.X(1)).!(zz.a.auf.a.b.ni.CH1(1).exp(g.X(1)))` to Agent i.
- Agent (a.3) sends `a.b.x99.CH2(3).exp(g.Y(2)).CH1(1).exp(g.X(1)).ni!(exp(g.Y(2)).a.b.x99.CH2(3).exp(g.Y(2)).CH1(1).exp(g.X(1))).ni!` to Agent i.
- Agent (a.3) sends `b.a.CH2(3).x102!(exp(g.Y(2)).b.a.CH2(3).x102)` to Agent (a.7).
- Agent (a.7) sends `a.b.x102.CH4(4)!(exp(g.Y(2)).a.b.x102.CH4(4))` to Agent (a.3).

Results in AVISPA

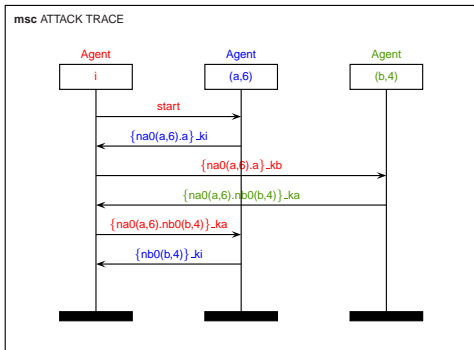
The screenshot displays the AVISPA web interface. At the top, the AVISPA logo is shown in red and white, with the text "Automated Validation of Internet Security Protocols and Applications" below it. To the right, a "Mode" selector has two buttons: "Basic" (highlighted in red) and "Expert". Below the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

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)).
```

Verification: introduction to tools

goal_reachable: c:secret[]

Proverif

```

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[])),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[])

```

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
 $\bar{c} \langle pk_A \rangle \bar{c} \langle pk_B \rangle .$

(A) $!c(x_{pk_B}).(\nu k)\bar{c} \langle pencrypt(sign(k, sk_A), x_{pk_B}) \rangle$

$.c(x).lets = sdecrypt(x, k)$ in 0

(B) $!c(y).lety' = pdecrypt(y, sk_B)$ in let $k = checksign(y', pk_A)$ in

$\bar{c} \langle sencrypt(s, k) \rangle$

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)).
```

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.

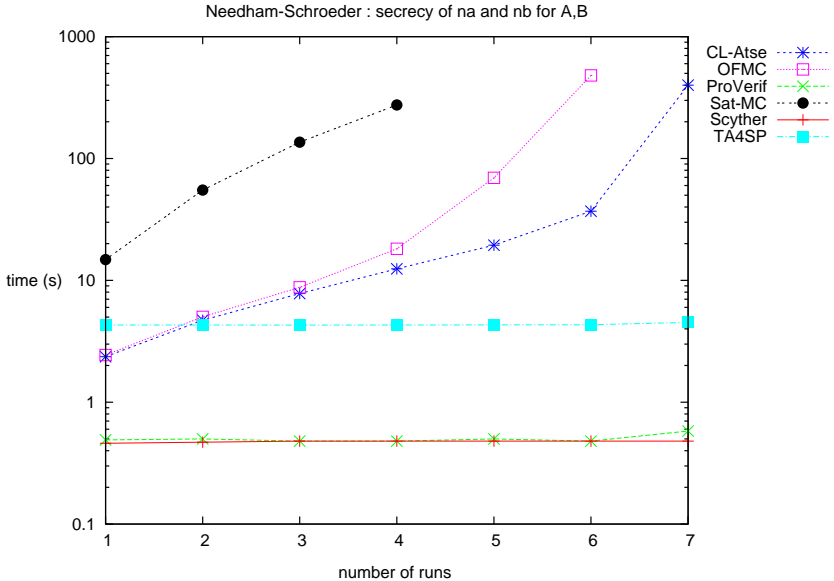
RESULT not attacker:secretBtoA[] is false.

Outline

- 1 Logical Attacks
- 2 Intruder
- 3 Needham Schroeder
- 4 Undecidability for unbounded number of sessions
- 5 Verification: introduction to tools
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison**
- 7 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 Logical Attacks
- 2 Intruder
- 3 Needham Schroeder
- 4 Undecidability for unbounded number of sessions
- 5 Verification: introduction to tools
 - Scyther
 - AVISPA
 - Proverif
- 6 Comparison
- 7 Conclusion**

Conclusion

- Not only cryptanalysis attack but LOGICAL
- Existing Tools for automatic verification
- Modeling protocols is trick
- Next week: Playing with tools on computer

Thank you for your attention



Questions ?