Automated formal analysis of security protocols

Automated verification

- It is not easy and is error-prone itself to do formal analysis manually;
- Development of methods for automated or semi-automated (interactive) validation and verification is important area, especially in the context of security protocols;

Different directions

- **Model checking** (state exploration tools);
  - specific (NRL Protocol Analyser, etc)
  - general purpose tools (SMV, SPIN, Mocha, etc)
  - general purpose tools combined with specific translators (Casper/FDR, etc)
- **Theorem proving**
  - Automated (TAPS, etc)
  - Interactive (Isabell, PVS, etc)
- **Combinations of above techniques**
  - Athena, etc
- **Others**: decision procedures for specific theories, infinite state model checking, etc

General questions

- How to represent a protocol (system) to be analysed?
- How to express properties to be verified?
Model checking

- A protocol (system executing a protocol) is represented as a transition system $M$ with finitely many states;
- A property to be analysed is expressed by a formula of a logic (temporal, modal, etc) $f$;
- Then verification amounts to checking whether the formula $f$ is true in $M$;
- Model checking is done via efficient state exploration techniques;

Nice properties
- Fully automated procedures;
- Very efficient state exploration;

but
- Finite state abstraction is not always adequate, especially for protocols with unbounded number of participants or unbounded number of rounds.

Attack on Needham-Schroeder protocol

A particular success of model checking methods in security protocol verification was discovery of a flaw in NS protocol based on public key cryptography (Gavin Lowe, 1995-1996);

Original protocol

| Message 1. | $A \rightarrow B$: $A,B,[A,N_A]_{K_B}K_0(3)$ |
| Message 2. | $B \rightarrow A$: $B,A,[N_A,N_B]_{K_B}K_0(5)$ |
| Message 3. | $A \rightarrow B$: $A,B,[N_B]_{K_B}K_0(3)$ |

Attack

- $A \rightarrow I_0$: $A,I_0,[A,N_A]_{K_0(1)}$
- $I_1 \rightarrow B$: $A,B,[A,N_A]_{K_B}K_0(9)$
- $B \rightarrow I_4$: $B,A,[N_A,N_B]_{K_B}K_0(9)$
- $I_2 \rightarrow A$: $I_2,A,[N_A,N_B]_{K_B}K_0(9)$
- $A \rightarrow I_3$: $A,I_3,[N_B]_{K_B}K_0(9)$
- $I_4 \rightarrow B$: $A,B,[N_B]_{K_B}K_0(9)$

Theorem Proving

A protocol (a system) to be verified is described by a formula $F_s$ of a logic (classical first-order, higher-order, modal, temporal, etc);

- A property to be verified is expressed by a formula $P$ of the same logic;
- Then to establish the required property it is enough to prove the theorem $F_s \rightarrow P$;
Theorem proving

Potential benefits:
- the systems with *unbounded* (infinite) number states can be analysed;

But:
- The problems here are, in general, *undecidable*;
- Procedures are *incomplete* and of high complexity.

What to do?
- Apply automated procedures for fragments of first-order and higher-order logic
  - E. Cohen, TAPS system, Microsoft Research;
- Use interactive theorem proving
  - L. Paulson, Cambridge: using Isabelle, higher-order inductive theorem prover for the verification of security protocols;
  - J. Bryan, S. Schenider, using interactive theorem prover PVS;

Other interesting approaches
  - A protocol is presented as a set of Horn clauses (like a program in Prolog), defining capabilities of all participants;
  - Verification then amounts to checking whether a security breaching goal can be reached (derived) from the set of clauses;
  - If the system detects the goal is unreachable, then the protocol is correct;
  - Standard operational semantics of Prolog is not very useful here due to undesirable looping;
  - Novel operational semantics (search strategy) is defined;

ProVerif system

Denning-Sacco key distribution protocol

| Message 1 | $A ightarrow B: \{(k_{ab})_s\}_{sh}p$ |
|-----------|--------------------------------------------------|
| Message 2 | $B ightarrow A: \{\gamma\}$ |

Its representation in ProVerif system

- Denning-Sacco key distribution protocol
- Example of protocol representation in ProVerif system
- Syntax for representing protocols
- Operational semantics for verification
- Verification results

Protocol:
- First message: attacker(pk_{a}(\gamma)) -> attacker(pk_{b}(\gamma)) -> attacker(pk_{a}(\gamma)) -> attacker(pk_{b}(\gamma))
- Second message: attacker(pk_{a}(\gamma)) -> attacker(pk_{b}(\gamma)) -> attacker(pk_{a}(\gamma)) -> attacker(pk_{b}(\gamma))
Developments here at the Department

- Verification based on supercompilation (a program transformation technique);
- A system (protocol) is encoded as a functional program, then supercompilation is applied to get a simplified, but equivalent program for which correctness conditions may be easily checked;
- It has proved to be very efficient technique for verification of parameterised systems;
- **But**, it has not been tried yet for security protocols;
- Possible MSc (and PhD) projects. If interested, please contact A.Lisitsa.