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 (Isabelle, 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;

---

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. $A \rightarrow B$: $A.B\{A,N_A\}_{FK(B)}$
Message 2. $B \rightarrow A$: $B.A\{N_A,N_B\}_{FK(A)}$
Message 3. $A \rightarrow B$: $A.B\{N_B\}_{FK(B)}$

**Attack**

Message 1a. $A \rightarrow I$: $A.I\{A,N_A\}_{FK(I)}$
Message 1b. $I \rightarrow B$: $A.B\{A,N_A\}_{FK(B)}$
Message 2b. $B \rightarrow I_C$: $B.A\{N_A,N_B\}_{FK(A)}$
Message 2a. $I \rightarrow A$: $I.A\{N_A,N_B\}_{FK(A)}$
Message 3a. $A \rightarrow I$: $A.I\{N_B\}_{FK(I)}$
Message 3b. $I \rightarrow B$: $A.B\{N_B\}_{FK(B)}$

Corrupt participant I impersonates A

---

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 Isabell, higher-order inductive theorem prover for the verification of security protocols;
  • J.Bryans, S. Schenider, using interactive theorem prover PVS;

Other interesting approaches
• Bruno Blanchet, INRIA: approach based on ideas from Logic Programming (ProVerif, available online at http://www.di.ens.fr/~blanchet/crypto-eng.html):
  • 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 unreacha, 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: \( A \rightarrow B : \{ \{ k_{A,B} \} \}_{k_B} \)

Message 2: \( B \rightarrow A : \{ k_{B,A} \}_{k_B} \)

Its representation in ProVerif system

Computation obligations of the attacker:

- \( p_{\text{crypt}} : (z_{\text{sender}}(a) \land z_{\text{sender}}(k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(a), k)) \)
- \( p_{\text{pk}} : z_{\text{sender}}(a) \land z_{\text{sender}}(k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(k), k) \)
- \( p_{\text{sym}} : z_{\text{sender}}(a) \land \neg z_{\text{attacker}}(z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \)
- \( p_{\text{sym}} : \neg z_{\text{sender}}(a) \land z_{\text{sender}}(k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \)

Initial knowledge of the attacker:

- \( p_{\text{sym}} : z_{\text{sender}}(a) \land z_{\text{sender}}(k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \)

Protocol:

First message: \( z_{\text{sender}}(a) \land z_{\text{sender}}(k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \)

Second message: \( z_{\text{sender}}(a) \land z_{\text{sender}}(k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(k), k) \land \neg z_{\text{attacker}}(\neg z_{\text{sender}}(a), k) \)
Possible Projects

- 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 been tried for security protocols, but further investigation is required
- Possible MSc (and PhD) projects. If interested, please contact A.Lisitsa.

---

Other topics on Verification in Security