Security protocols and their analysis.

Security protocols

- A **security protocol** is a set of rules, adhered to by the communication parties in order to ensure achieving various security or privacy goals, such as establishing a common cryptographic key, a achieving authentication, etc.

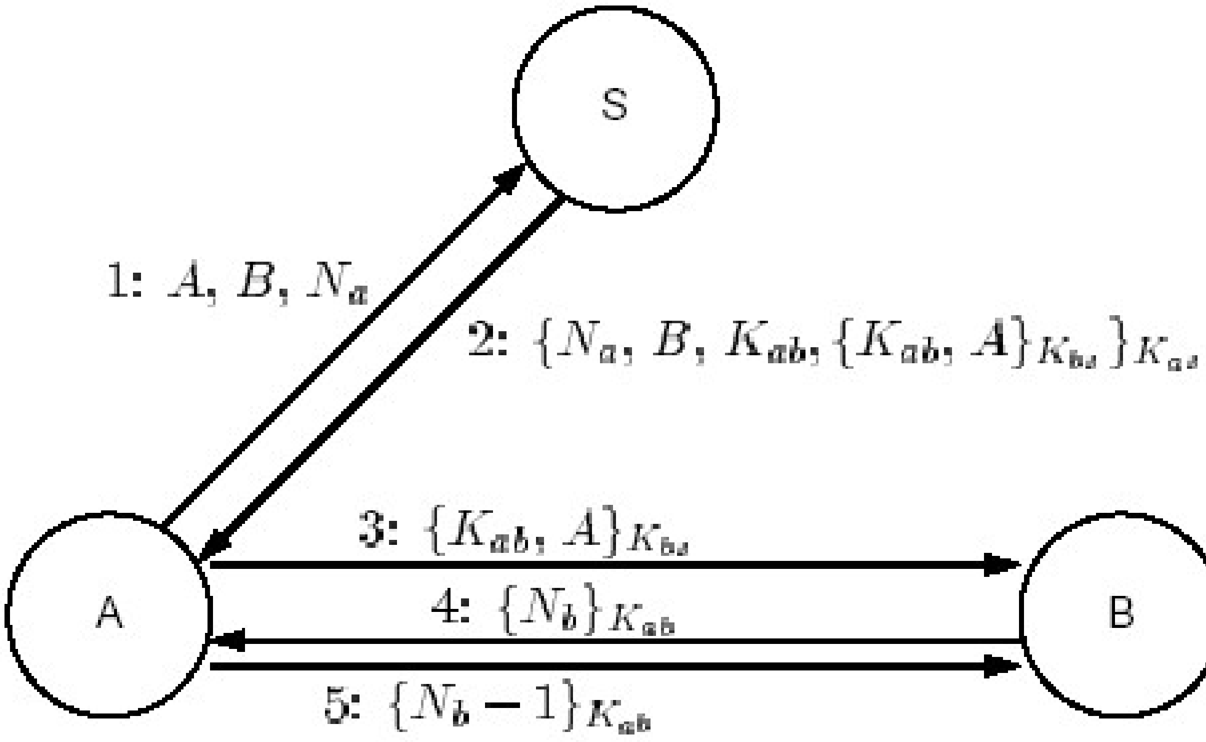
- We have discussed already a few protocols, e.g. Diffie-Hellman protocol for key exchange.

Example: Needham-Schroeder protocol

- The goal of the protocol is to establish mutual authentication between two parties A and B in the presence of adversary, who can
  - Intercept messages;
  - Delay messages;
  - Read and copy messages;
  - Generate messages,

  But who does not know
  - secret keys of principals, which they share with the authentication server S.
- A and B obtain a secret shared key though authentication server S.
- The protocol uses shared keys encryption/decryption

Needham-Schroeder protocol



The Needham-Schroeder Protocol (with shared keys)

Needham-Schroeder protocol

- Message 1 $A \rightarrow S: A, B, N_A$
- Message 2 $S \rightarrow A: \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$
- Message 3 $A \rightarrow B: \{K_{AB}, A\}_{K_B}$
- Message 4 $B \rightarrow A: \{N_B\}_{K_{AB}}$
- Message 5 $A \rightarrow B: \{N_B - 1\}_{K_{AB}}$

- Here K_A and K_B are keys of *A* and *B* shared with S, resp.
- N_A and N_B are nonces, introduced by *A* and *B*, resp.
- K_{AB} is a secret session key for *A* and *B* provided by *S*

How it works

- A makes contact with the authentication server S, sending identities A and B and *nonce* N_A;
- S responds with a message encrypted with the key of A.
  The message contains session key K_{AB} (to be used by A and B) and certificate encrypted with B's key conveying the session key and A's identity;
- A sends the certificate to B;
- B decrypts the certificates and sends his own nonce encrypted by the session key to A; (*nonce handshake*);
- A decrypts the last message and sends *modified nonce* back to B.

By the end of the message exchange both A and B share the secret key and both are assured in the presence of each other.

Correctness of protocols

- Are they correct at all?
- How do we establish correctness?
- We have used semi-formal arguments, like
- *If a message is encrypted with the public key of Alice, then only a participant who knows private key of Alice (presumably Alice herself only) can decrypt it.*
- Typically we have considered possible attacks and argued using the reasoning as above, that attacks are impossible (under some reasonable assumptions).
- Is that enough? Are we sure that we have considered all possible situations of use?

Correctness of protocols. II

- Security protocols are designed to succeed even in the presence of a malicious agent, often called *intruder (adversary)*;
- Intruder may have complete or partial control over the communication network and may have different computational capabilities;
- The correctness of the protocols depends on the *assumptions* on capabilities of possible intruder;
- Assumptions are often left implicit;
- Typically in security we have to deal with numerous non-trivial assumptions.

The power of formal methods

- What should we do about establishing correctness of security protocols?
- Apply formal methods!
  - Make ***explicit*** all the assumptions involved in a protocol;
  - Make a formal model of the protocol (and its execution);
  - Apply formal reasoning, which would establish the correctness of the protocol.
- Two important aspects:
  - The correctness is established only for a particular formal model of the protocol;
  - and under explicit assumptions (about capabilities of participants, etc);

Logical representation

- Formal aspects of reasoning is an important part of logic;
- Logical representation and analysis of the security protocols is a particular successful approach for the protocols verification;
- Non-classical modal epistemic logics dealing with such notions as “*belief*” and “*knowledge*” , are more suitable here than classical logics dealing primarily with “*truth*”.

Automated verification/analysis

- 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)
  - Unbounded model checking for crypto protocols (ProVerif, Tamarin, 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

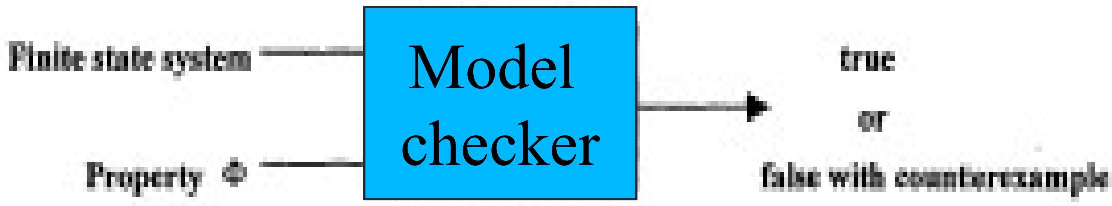
General questions

- 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;

Model checking



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

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** **Attack**

Message 1.	$A \rightarrow B$:	$A.B.\{A, N_A\}_{PK(B)}$
Message 2.	$B \rightarrow A$:	$B.A.\{N_A, N_B\}_{PK(A)}$
Message 3.	$A \rightarrow B$:	$A.B.\{N_B\}_{PK(B)}$.

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

Corrupt participant I impersonates A

Theorem Proving

- A protocol (a system) to be verified is described by a formula **Fs** 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 **Fs → 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.

Theorem proving

- 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;

Specialized 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 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;
- Recent versions use pi-calculus as a language for front-end

ProVerif system

Denning-Sacco key distribution protocol

Message 1. $A \rightarrow B : \{\{k\}_{sk_A}\}_{pk_B}$
Message 2. $B \rightarrow A : \{s\}_k$

Its representation in ProVerif system (old syntax)

Computation abilities of the attacker:

pencrypt	$\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \rightarrow \mathsf{attacker}(\mathsf{pencrypt}(m, pk))$
pk	$\mathsf{attacker}(sk) \rightarrow \mathsf{attacker}(\mathsf{pk}(sk))$
pdecrypt	$\mathsf{attacker}(\mathsf{pencrypt}(m, \mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \rightarrow \mathsf{attacker}(m)$
sign	$\mathsf{attacker}(m) \wedge \mathsf{attacker}(sk) \rightarrow \mathsf{attacker}(\mathsf{sign}(m, sk))$
getmess	$\mathsf{attacker}(\mathsf{sign}(m, sk)) \rightarrow \mathsf{attacker}(m)$
checksign	removed since implied by **getmess**
sencrypt	$\mathsf{attacker}(m) \wedge \mathsf{attacker}(k) \rightarrow \mathsf{attacker}(\mathsf{sencrypt}(m, k))$
sdecrypt	$\mathsf{attacker}(\mathsf{sencrypt}(m, k)) \wedge \mathsf{attacker}(k) \rightarrow \mathsf{attacker}(m)$

Initial knowledge of the attacker:

$\mathsf{attacker}(\mathsf{pk}(sk_A[])), \quad \mathsf{attacker}(\mathsf{pk}(sk_B[])), \quad \mathsf{attacker}(a[])$

Protocol:

First message: $\mathsf{attacker}(\mathsf{pk}(x)) \rightarrow \mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(k[\mathsf{pk}(x)], sk_A[]), \mathsf{pk}(x)))$

Second message: $\mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(k', sk_A[]), \mathsf{pk}(sk_B[]))) \rightarrow \mathsf{attacker}(\mathsf{sencrypt}(s[], k'))$