Foundations of security protocol analysis

R. Ramanujam

The Institute of Mathematical Sciences, Chennai, India

jam@imsc.res.in

Your email account

- Do you have an account on some public email facility ?
- How do you login to such an account ?
- You give user name and password. **Why ?**

Sending secrets

- How does A send a secret x to B on a public channel ?
- A locks x in a box, sends the box across to B.
- Assume that locks are unbreakable, and that for every lock, there exists a unique key that opens it.
- The secret reaches B securely.
- But the key is with A; by assumption, B cannot open it.
- Should A send the key in *another locked box* !?

One-way functions

- Given x, finding $f(x)$ is *easy* to compute.
- Given y, finding $f^{-1}(y)$ is *hard*.
- Easy / hard in what sense ?
- The idea of trapdoor: helps to find the inverse.

Diffie - Hellman key exchange

- The general one-way function is $Q^x \ mod \ P$, where P is prime; both Q and P are public.
- A chooses x_A, secret to her, and computes $Q^{x_A} \ mod \ P = y_A$.
- B chooses x_B, secret to him, and computes $Q^{x_B} \ mod \ P = y_B$.
- A sends y_A to B, while B sends y_B to A.
- A computes $K_A = y_B^{x_A} \ mod \ P$, while B computes $K_B = y_A^{x_B} \ mod \ P$.
- Note that $K_A = K_B$, so they have a secret shared key.

The RSA system

- A picks two giant primes p and q, computes $N = p \cdot q$.
- She then picks e such that $gcd(e, (p-1) \cdot (q-1)) = 1$.
- e and N together describe A's public key.
- To send a message M to her, B computes $C = M^e \ mod \ N$, and sends C to A.
- Now A can compute d, where $e \cdot d = 1 \ mod \ ((p-1) \cdot (q-1))$.
- A finds M by observing that $C^d \ mod \ N = M$.

Key establishment

- A and B wish to generate a *session key* to be used subsequently. The need a protocol to establish a new key K_{AB}.
- Assume that a reliable, trustworthy *key server* S exists.
- At the end of the protocol, K_{AB} should be known to both A and B but to nobody else, except perhaps S.
- Both A and B should know that K_{AB} is newly generated.

First attempt

- Here is a protocol:
  1. $A \rightarrow S : A, B$
  2. $S \rightarrow A : K_{AB}$
  3. $A \rightarrow B : K_{AB}, A$
- Remarks about notation:
  - No internal actions are specified.
  - Only the communications in successful runs are specified.
  - It is assumed that A and B understand that these are protocol messages.

Insecurity in it

- Confidentiality is compromised.
- Security Assumption: The adversary can eavesdrop on all messages sent on public channels.
- Pragmatic assumption: Assume that the server has an initial *shared key* with every principal in the system.

Second attempt

- 1. $A \rightarrow S : A, B$
  2. $S \rightarrow A : \{K_{AB}\}_{K_{AS}}, \{K_{AB}\}_{K_{BS}}$
  3. $A \rightarrow B : \{K_{AB}\}_{K_{BS}}, A$
- The notation $\{x\}_k$ denotes a message x encrypted with key k.
- It does not matter which encryption algorithm is used.
- We make the *perfect encryption* assumption.

An attack

- Security Assumption: The adversary can also capture messages, alter them at will, reroute them (but cannot break encryption).
- Here is the attack:
  1. $A \rightarrow S : A, B$
  2. $S \rightarrow A : \{K_{AB}\}_{K_{AS}}, \{K_{AB}\}_{K_{BS}}$
  3. $A \rightarrow B(I) : \{K_{AB}\}_{K_{BS}}, A$
  3'. $(I)C \rightarrow B : \{K_{AB}\}_{K_{BS}}, C$
- Now B believes that he is sharing a key with C, whereas he is actually sharing it with A.
- Some secrets meant only for A can be leaked to C.

Another attack

- In the attack we saw, I does not get hold of K_{AB}. Here is one where he does.
- 1. $A \rightarrow S(I) : A, B$
  1'. $I \rightarrow S : A, I$
  2. $S \rightarrow I : \{K_{AI}\}_{K_{AS}}, \{K_{AI}\}_{K_{IS}}$
  2'. $(I)S \rightarrow A : \{K_{AI}\}_{K_{AS}}, \{K_{AI}\}_{K_{IS}}$
  3. $A \rightarrow B(I) : \{K_{AI}\}_{K_{IS}}, A$
- A believes that the protocol is completed with B, and hence I can masquerade as B.

Insiders

- In the attack we saw, I could have been a legitimate user known to S.
- Security Assumption: The adversary may be a legitimate participant in the protocol (insider), or an external party, or a combination of both.
- In fact, security threats from insiders are more of a problem than from outsiders, in many systems.
- Typically referred to as the "person in the middle" attack.
- Solution ?

Another attempt

- Bind keys to agent names.
- 1. $A \rightarrow S : A, B$
  2. $S \rightarrow A : \{K_{AB}, B\}_{K_{AS}}, \{K_{AB}, A\}_{K_{BS}}$
  3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
- It can be checked that neither of the attacks considered earlier are possible any longer.

Replay

- The generated key is supposed to be *new* !
- Security Assumption: The adversary may be able to obtain the value of the session key K'_{AB} used in any sufficiently old earlier run of the protocol.
- A replay attack:
  1. $A \rightarrow S(I) : A, B$
  2. $(I)S \rightarrow A : \{K'_{AB}, B\}_{K_{AS}}, \{K'_{AB}, A\}_{K_{BS}}$
  3. $A \rightarrow B : \{K'_{AB}, A\}_{K_{BS}}$
- Note that I need not actually have the value of K'_{AB} but simply replay the earlier message, forcing A and B to accept an old, dated key.
- Now I can replay an earlier (legitimate) message, which can be, say: "Deposit Rs 1 lakh into I's account" !

Nonces

- How do we ensure that old keys are not replayed ?
- A generates a new *random* value N_A called nonce.
- A starts the protocol by sending a nonce N_A. If by the end of the protocol, the same value returns, she knows that the key has not been replayed.
- What about B ? He generates his own nonce as well.
- This is typically called a *challenge - response* technique.

Needham - Schroeder

- We now have the following version of the protocol:
  1. $A \rightarrow S : A, B, N_A$
  2. $S \rightarrow A : \{K_{AB}, B, N_A, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
  3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
  4. $B \rightarrow A : \{N_B\}_{K_{AB}}$
  5. $A \rightarrow B : \{N_B + 1\}_{K_{AB}}$
- This is the famous Needham - Schroeder protocol, 1978.
- Denning and Sacco found an attack in 1986.

The attack and its fix

- A replay attack:
  3'. $(I)A \rightarrow B : \{K'_{AB}, A\}_{K_{BS}}$
  4. $B \rightarrow A(I) : \{N_B\}_{K'_{AB}}$
  5. $(I)A \rightarrow B : \{N_B + 1\}_{K'_{AB}}$
- The fix is quite simple:
  1. $B \rightarrow A : B, N_B$
  2. $A \rightarrow S : A, B, N_A, N_B$
  3. $S \rightarrow A : \{K_{AB}, B, N_A\}_{K_{AS}}, \{K_{AB}, A, N_B\}_{K_{BS}}$
  4. $A \rightarrow B : \{K_{AB}, A, N_B\}_{K_{BS}}$

Are we done?

- Is this protocol secure ? Can we prove it secure ?
- Yes, but this takes a great deal of effort and machinery.
- Note that it still achieves less. At the end of the run, neither A nor B can actually deduce that the other has actually received K_{AB} !

Infrastructure

The architecture of a security protocol crucially depends on:

- Existing cryptographic keys (shared, public)
- Key generation mechanisms
- Nonce generation mechanisms
- Number of users
- Number of multisessions concurrently active

Security properties

- Confidentiality: Data is available only to those authorized to obtain it.
- Integrity: Data has not been altered by unauthorised entities.
- Authentication: Data has indeed originated from the purported sender.
- Non-repudiation: Entities cannot deny sending data they have committed to.
- Freshness: Nonces are unguessable, and never re-used.

Attacks

- Eavesdropping
- Modification
- Replay
- Preplay
- Denial of service
- Type flaws
- Cryptanalysis

Contract signing

- Two parties agree on the contract text
- Each will sign if the other will.
- Physical solution is easy: sit at a table.
- Over a network ?

Why is this difficult ?

- Cannot trust communication channels. (Intruder may block or insert messages.)
- Cannot trust the other party in the protocol !
  Public key certificate does not certify honesty.
- Even if a trustworthy judge exists, she may become a communication bottleneck.

Optimistic contract signing

- A tells B: "I am going to sign the contract"
- B tells A: "I am going to sign the contract"
- A sends her signature to B
- B sends his signature to A

A judge may declare contract binding if given first two messages.

Requirements

- Fairness: If A cannot obtain a contract, B should not be able to obtain a contract either, and vice versa.
  If A cannot get a deed for the house, B should not be able to collect A's money.
- No advantage: No party may solely determine the outcome of the protocol.
  At no stage should A be able to decide by herself whether the house is sold or not.
- No provable advantage: No party may prove that it can solely determine the outcome of the protocol.
  B should not be able to show A's offer to C and convince C to pay more.

Negative result

- Dishonest party has advantage in any fixed-round optimistic fair exchange protocol.
  - Dishonest party always has a strategy to reach a state where it can unilaterally force an outcome.
  - Similar to impossibility result for distributed consensus.
  - Cryptography cannot help.
- Need a trusted party for every transaction: bad news for e-commerce (or maybe good news !?).

Electronic voting

- Secrecy: Every voter's choice should be private, and others should not be able to figure out how she voted.
- Individual verifiability: Each voter should be able to check whether her vote has been counted properly.
- Universal verifiability: It should be possible to check whether every voter's vote has been counted properly.
- Fairness: Voters do not have any knowledge of the distribution of votes until the tallies are finally announced.
- Receipt-freeness: No voter has any means of proving to another that he has voted in a particular manner.

Receipt-freeness

- A problem characteristic of elections, and an important one.
- Lack of receipt-freeness (that is, the presence of a receipt) allows vote buying and coercion which has the potential to drastically affect the election process.
- When a receipt exists, it could be constructed in any manner that convinces the sceptical second party.
- Demonstrating that no such way exists is highly demanding, and is seen as a significant challenge to formal models.

Protocol structure

- Three kinds of agents: voters are one kind.
- Administrators know the voters' identity, but cannot see the votes, and their job is to check voters' eligibility to vote.
- Talliers see the votes, but not the voters' identities, and their job is to count votes for each candidate and announce the result.

Homomorphic encryption

- Based on Adi Shamir's secret sharing technique.
- Voters split their votes into several shares which they send to different administrators.
- Unless many administrators collude, it is difficult to reconstruct the vote.

Blind signature

- Voter sends an encrypted vote $\{t\}_k$ to the administrator.
- The administrator A cannot decrypt the vote, but checks the voter's eligibility, "blindly" signs the message and returns $\{t_A\}_k$ to the voter.
- The voter, who generated k, can strip it off from $\{t_A\}_k$ obtaining t_A, which is the vote t duly attested by A.
- Now the voter sends t_A anonymously to the tallier.
- The tallier verifies the attestation, and gets the vote t (while not knowing the origin).

Summary

- Security protocols are difficult to design and analyse, bugs hard to attack.
- We need a development methodology which would always generate provably correct protocols.
- We need automaated tools for discovering attacks.
- Our main contribution: we identify subclasses of security protocols for which it is possible to automatically verify many security properties.