Protocol analysis using ProVerif
Attacker model

secretA
ALICE

secretB
BOB
Attacker model

The attacker has full control over the network. He can drop, halt, modify, or substitute messages. The attacker decides who runs the protocols with whom.
Attacker model

The attacker has full control over the network. He can drop, halt, modify, substitute messages. The attacker decides who runs the protocols with whom.
Attacker model

- The attacker has full control over network.
  - He can drop, halt, modify, substitute messages.
Attacker model

- The attacker has full control over network.
- He can drop, halt, modify, substitute messages.
- The attacker decides who runs the protocols with whom.
**Attacker model**

- The attacker has full control over the network.
- He can drop, halt, modify, substitute messages.
- The attacker decides who runs the protocols with whom.
ProVerif

- [https://prosecco.gforge.inria.fr/personal/bblanche/proverif/](https://prosecco.gforge.inria.fr/personal/bblanche/proverif/)
- Static analysis for cryptographic protocols under the perfect cryptography assumption
- Can check secrecy and correspondence properties
- Errors only to the safe side
  - If a protocol is insecure, then says so
  - If a protocol is secure, then sometimes may claim to have found an attack
- Principle: translate the protocol to a set of *Horn clauses*
  - Involves a little bit of abstraction
Horn clauses

\[ p_1(t_{11}, \ldots, t_{1k_1}) \land \cdots \land p_n(t_{n1}, \ldots, t_{nk_n}) \Rightarrow q(t'_1, \ldots, t'_m) \]

- \( p_1, \ldots, p_n, q \) — predicate symbols
  - from a fixed set; each with fixed arity
- \( t_*, t'_* \) — term
  - countable number of atoms (constants)
  - constructors (functional symbols) from a fixed set
- terms may contain term variables as subterms (in these slides, denoted with capital letters)
  - \( \bigwedge_i p_i(\ldots X \ldots) \Rightarrow q(\ldots X \ldots) \) means
    \[ \forall t \in T : \left( \bigwedge_i p_i(\ldots t \ldots) \Rightarrow q(\ldots t \ldots) \right) \]
- \( T \) — the set of all ground terms (without variables)
Examples

- A translation of a protocol always contains a unary predicate \( a \)
- \( a(X) \) means that the attacker can learn \( X \)
- A translation contains rules for composing and decomposing messages:
  - \( a(pair(X, Y)) \Rightarrow a(X) \quad a(pair(X, Y)) \Rightarrow a(Y) \)  //\((X,Y)\)
  - \( a(X) \land a(Y) \Rightarrow a(pair(X, Y)) \)
  - \( a(senc(K, X)) \land a(K) \Rightarrow a(X) \)  //symmetric encryption
  - \( a(penc(pk(K), X)) \land a(K) \Rightarrow a(X) \)  //asymmetric encryption
  - \( a(K) \land a(X) \Rightarrow a(sign(K, X)) \)  //signature
  - \( a(sign(K, X)) \Rightarrow a(X) \)
  - \( a(X) \Rightarrow a(h(X)) \)  //hash
  - ...

- There are also rules for protocol steps
- There is a goal, stated as a boolean formula, whose truthfulness we need to verify.
Logic programming

- A logic program is a set of Horn clauses.
- \( \forall X_1 \cdots \forall X_k (p_1 \land \cdots p_k \Rightarrow q) \equiv \forall X_1 \cdots \forall X_k (\neg p_1 \lor \cdots \neg p_k \lor q) \)
- A formula is in CNF (conjunctive normal form) if it is of the form \( \forall X_1 (L_{11} \lor \cdots \lor L_{1k_1}) \land \cdots \land \forall X_n (L_{n1} \lor \cdots \lor L_{nk_n}) \) where
  - each \( L \) is a literal — a predicate application or its negation.
- Denote this formula with \( \{[L_{11}, \ldots, L_{1k_1}], \ldots, [L_{n1}, \ldots, L_{nk_n}]\} \)
  - A set of sets, actually.
- There are known methods (resolution) that prove whether such a formula is satisfiable.
Recall our example

1. $A \rightarrow B : \{[K_A, N_A, K_{AB}]\}_{K_B}$
2. $B \rightarrow A : \{[N_A, N_B, K_B]\}_{K_A}$
3. $A \rightarrow B : \{[N_A, N_B]\}_{K_B}$
4. $B \rightarrow A : \{M\}_{K_{AB}}$

- The attacker can have the 1st message by starting a new session

$$a(pk(A)) \land a(pk(B)) \Rightarrow a(penc(pk(B), triple(pk(A), na, k)))$$
Recall our example

1. \( A \rightarrow B : \{[K_A, N_A, K_{AB}]\}_{K_B} \)
2. \( B \rightarrow A : \{[N_A, N_B, K_B]\}_{K_A} \)
3. \( A \rightarrow B : \{[N_A, N_B]\}_{K_B} \)
4. \( B \rightarrow A : \{M\}_{K_{AB}} \)

The attacker can have the 1st message by starting a new session

\[ a(pk(A)) \land a(pk(B)) \Rightarrow a(penc(pk(B), triple(pk(A), na, k))) \]

Something is very wrong here... What \( na \)? What \( k \)?

\( na \) and \( k \) would be different in each session. There must be a parameter “session ID”.

7
The first message

1. $A \rightarrow B : \{[K_A, N_A, K_{AB}]\}_{K_B}$
2. $B \rightarrow A : \{[N_A, N_B, K_B]\}_{K_A}$
3. $A \rightarrow B : \{[N_A, N_B]\}_{K_B}$
4. $B \rightarrow A : \{M\}_{K_{AB}}$

The attacker can have the 1st message by starting a new session

$$a(pk(A)) \land a(pk(B)) \land a(id) \Rightarrow a(penc(pk(B), \text{triple}(pk(A), na[id], k[id])))$$
The first message

1. \( A \rightarrow B : \{ [K_A, N_A, K_{AB}] \}_{K_B} \)
2. \( B \rightarrow A : \{ [N_A, N_B, K_B] \}_{K_A} \)
3. \( A \rightarrow B : \{ [N_A, N_B] \}_{K_B} \)
4. \( B \rightarrow A : \{ M \}_{K_{AB}} \)

The attacker can have the 1st message by starting a new session

\[
a(pk(A)) \land a(pk(B)) \land a(Id) \Rightarrow a(penc(pk(B), triple(pk(A), na[Id], k[Id])))
\]

Attacker: “Dear Alice, please start session 5 with Bob”

\( k(5) \) will be exchanged

Attacker “Dear Alice, please start session 5 with me”

Attacker learns \( k(5) \)
The first message (let us try again)

1. $A \rightarrow B : \{[K_A, N_A, K_{AB}]\}_{K_B}$
2. $B \rightarrow A : \{[N_A, N_B, K_B]\}_{K_A}$
3. $A \rightarrow B : \{[N_A, N_B]\}_{K_B}$
4. $B \rightarrow A : \{M\}_{K_{AB}}$

(cd) Session ID must contain the roles of the parties.

$$a(pk(A)) \land a(pk(B)) \land a(Id) \Rightarrow$$

$$a(penc(pk(B), triple(pk(A),$$

$$na[pk(A), pk(B), Id], k[pk(A), pk(B), Id]))))$$
The second message

1. $A \rightarrow B : \{[K_A, N_A, K_{AB}]\}_{K_B}$
2. $B \rightarrow A : \{[N_A, N_B, K_B]\}_{K_A}$
3. $A \rightarrow B : \{[N_A, N_B]\}_{K_B}$
4. $B \rightarrow A : \{M\}_{K_{AB}}$

When Bob gets the 1st message, he responds with the 2nd

\[ a(Id) \land a(penc(pk(B), triple(pk(A), Na, K))) \Rightarrow \]
\[ a(penc(pk(A), triple(Na, nb[pk(A), pk(B), Id], pk(B)))) \]
The third message

1. \( A \rightarrow B : \{ [K_A, N_A, K_{AB}] \}_{K_B} \)
2. \( B \rightarrow A : \{ [N_A, N_B, K_B] \}_{K_A} \)
3. \( A \rightarrow B : \{ [N_A, N_B] \}_{K_B} \)
4. \( B \rightarrow A : \{ M \}_{K_{AB}} \)

When Alice gets the 2nd message, she responds with the 3rd

\[ \text{a}(\text{penc}(pk(A), \text{triple}(na[pk(A), pk(B), Id], Nb, pk(B)))) \Rightarrow \]
\[ \text{a}(\text{penc}(pk(B), \text{pair}(na[pk(A), pk(B), Id], Nb))) \]
The fourth message

1. $A \rightarrow B : \{[K_A, N_A, K_{AB}]\}_{K_B}$
2. $B \rightarrow A : \{[N_A, N_B, K_B]\}_{K_A}$
3. $A \rightarrow B : \{[N_A, N_B]\}_{K_B}$
4. $B \rightarrow A : \{M\}_{K_{AB}}$

- When Bob gets the 3rd message, he responds with the 4th...
- But only if he has participated in the session from the beginning
The fourth message

1. \( A \rightarrow B : \{ [K_A, N_A, K_{AB}] \}_{K_B} \)
2. \( B \rightarrow A : \{ [N_A, N_B, K_B] \}_{K_A} \)
3. \( A \rightarrow B : \{ [N_A, N_B] \}_{K_B} \)
4. \( B \rightarrow A : \{ M \}_{K_{AB}} \)

- When Bob gets the 3rd message, he responds with the 4th...
- But only if he has participated in the session from the beginning
- When Bob has received the 1st and 3rd messages, he can respond with the 4th.

\[
\begin{align*}
& a( penc(pk(B), triple(pk(A), Na, K))) \land \\
& a( penc(pk(B), pair(Na, nb[pk(A), pk(B), Id]))) \rightarrow \\
& \quad a( senc(K, m))
\end{align*}
\]
Solving the system

◎ Is $a(m)$ derivable?
◎ You may ask a Prolog system (traditional logic programming). And it will answer...
Solving the system

- Is $a(m)$ derivable?
- You may ask a Prolog system (traditional logic programming). And it will answer...
  - ...infinite loop.
    - To get $a(m)$, we could use some $a(f(m))$
    - To get $a(f(m))$, we could use some $a(f(f(m)))$
    - To get...

- The unification strategy of ProVerif is more geared towards such protocol representations.
Try to run ProVerif

- Demo
  - Invoking the analyzer: ./proverif file
Try to run ProVerif

- Demo
  Invoking the analyzer: ./proverif file
- Try to reconstruct the attack
What went wrong

- The attacker gained access to the secret key of Alice and could decrypt her messages.
- Actually, ProVerif tells that the attacker generated himself the secret key of Alice.
- How could that have happened?
What went wrong

- The attacker gained access to the secret key of Alice and could decrypt her messages.
- Actually, ProVerif tells that the attacker generated himself the secret key of Alice.
- How could that have happened?
- Since A and B are term variables (i.e. can represent any party, as well as the attacker), the attacker will learn the secret if he takes the role A.
- We are interested in privacy only if Alice is an honest user.
The fourth message (revisited)

1. $A \rightarrow B : \{ [K_A, N_A, K_{AB}] \}_{K_B}$
2. $B \rightarrow A : \{ [N_A, N_B, K_B] \}_{K_A}$
3. $A \rightarrow B : \{ [N_A, N_B] \}_{K_B}$
4. $B \rightarrow A : \{ M \}_{K_{AB}}$

Let $s_A$ and $s_B$ be the secret keys (unknown to the attacker) of actual Alice and Bob (i.e. not the roles, but some honest users)

Only Bob will send $m$, and only to Alice.

$\text{a}(\text{penc}(pk(s_B), \text{triple}(pk(s_A), Na, K))) \land$

$\text{a}(\text{penc}(pk(s_B), \text{pair}(Na, nb[pk(s_A), pk(s_B), Id]))) \Rightarrow$

$\text{a}(\text{senc}(K, m))$
Try to run ProVerif

- Demo
Try to run ProVerif

- Demo
- Try to reconstruct the attack
What went wrong

- Attacker plays Alice sending the first message to Bob
- Bob received it twice, responding to it both times
  - Fair enough
What went wrong

- Attacker plays Alice sending the first message to Bob
- Bob received it twice, responding to it both times
  - Fair enough
- But the adversary repeated the session identifier
  - Not good
  - To avoid that, newly generated values must contain all received messages so far.
The second message

1. \(A \rightarrow B : \{ [K_A, N_A, K_{AB}] \}_{K_B} \)
2. \(B \rightarrow A : \{ [N_A, N_B, K_B] \}_{K_A} \)
3. \(A \rightarrow B : \{ [N_A, N_B] \}_{K_B} \)
4. \(B \rightarrow A : \{ M \}_{K_{AB}} \)

When Bob gets the first message, he responds with the second

\[ a(Id) \land a(\text{penc}(pk(B), \text{triple}(pk(A), N, K))) \Rightarrow \]

\[ a(\text{penc}(pk(A), \text{triple}(Na, nb[pk(A), pk(B), Id, \text{penc}(pk(B), \text{triple}(pk(A), Na, K))], pk(B)))) \]
The fourth message

1. $A \rightarrow B : \{[K_A, N_A, K_{AB}]\}_{K_B}$
2. $B \rightarrow A : \{[N_A, N_B, K_B]\}_{K_A}$
3. $A \rightarrow B : \{[N_A, N_B]\}_{K_B}$
4. $B \rightarrow A : \{M\}_{K_{AB}}$

\[
a(penc(pk(sB), triple(pk(sA), Na, K))) \land a(penc(pk(sB), pair(Na, nb[pk(sA), pk(sB), Id, penc(pk(sB), triple(pk(sA), Na, K)])))) \Rightarrow a(senc(K, m))\]
Try to run ProVerif

- Demo
Try to run ProVerif

- Demo
- A similar-looking attack...
Try to run ProVerif

- Demo
- A similar-looking attack...
  - The attacker messed up 1st and 2nd messages of different sessions.
  - This is actually a type flaw, as the attacker needs to make a key look like a nonce, and a symmetric key like an asymmetric key.
Try to run ProVerif

- Demo
- A similar-looking attack...
  - The attacker messed up 1st and 2nd messages of different sessions.
  - This is actually a type flaw, as the attacker needs to make a key look like a nonce, and a symmetric key like an asymmetric key.
- How to fix it?
  - We can use typed version of Horn clauses
  - We can add constants (e.g. fst() and snd()) to the first and the second messages respectively.
Correspondence assertions

- So far, we have analysed message secrecy.
- We may need some other important properties like "Does Bob always accept the same shared key as Alice does?"
  - The event **Bob accepts K** should happen only if **Alice accepts K** happened.
  - More generally, an event **end(M)** should happen only if **begin(M)** has happened (for some **M**).
  - ... take into account session IDs etc.
Correspondence assertions as Horn clauses

- Two more predicates, \( b \) and \( e \), for \textbf{begin} and \textbf{end}.
- After a party has executed \textbf{begin}(M), its following messages are translated with \( b(M) \) as a premise.
  - \( b(M) \land a(\cdots) \Rightarrow a(\cdots) \)
  - \( \cdots \) contains session IDs and received messages.
- Emitting \textbf{end}(M) is adversary’s goal, hence it is the conclusion of a rule.
  - \( a(\cdots) \Rightarrow e(M) \)
- If \( b(M) \) is necessary for \( e(M) \), then we have (non-injective) agreement.
ISO 3-pass mutual authentication

1. $A \rightarrow B: N_A$
2. $B \rightarrow A: [\{N_A, N_B, K_A\}]_{K_B}$
3. $A \rightarrow B: [\{N_B, N_A, K_B\}]_{K_A}$

- From signature find the message.
- Public key $\equiv$ principal’s name.
- $\textbf{end}(K_A, K_B)$ executed by $B$ in the very end.
- $\textbf{begin}(K_A, K_B)$ executed by $A$ before 3rd message.
Injective agreement

- An agreement is injective if no two instances of end event can share the same begin event.
- Add the session identifier Z to the argument of e.
- Add the session identifier and received messages Y to the argument of b.
- If \( b((X, Y)) \) is necessary for \( e((X, Z)) \), and Z appears in Y, then we have injective agreement.
Injective agreement (example)

Example that has agreement, which is not injective:

1. \( A \rightarrow B : (A, B) \)
2. \( B \rightarrow A : [[N]]_{KB} \)

Let \textbf{begin} event be executed by \( B \) after 1st step, and \textbf{end} executed by \( A \) after the 2nd step.

- There is agreement, as \( A \)'s signature verification fails, if \( B \) has never signed anything.
- It is non-injective, as the attacker may resend the second message multiple times in different sessions.
Try to run ProVerif

- Demo
Conclusion

- Writing down protocols in Horn clauses is a non-trivial task.
- Technical transformations that we had to do (e.g. including previously received messages everywhere) could be done automatically.
- There exists more user-friendly pi-calculus interface of ProVerif.