Symbolic Protocol Analysis

Overview
- Strand space model
- Protocol analysis with unbounded attacker
  - Parametric strands
  - Symbolic attack traces
  - Protocol analysis via constraint solving
- SRI constraint solver

Protocol Analysis Techniques
- Crypto protocol analysis
- Formal models
  - Dolev-Yao model (perfect cryptography)
  - Random oracle
- Probabilistic model checking
- Inductive method
  - Model checking
  - Probabilistic process calculi
  - Probabilistic I/O automata
- Probabilistic I/O automata
- Finite-state checking
- Symbolic analysis
  - Finite processes, finite attacker
  - Finite processes, infinite attacker
  - Fully automated methods, always terminate and give an answer

Obtaining a Finite Model
- Two sources of infinite behavior
  - Multiple protocol sessions, multiple participants
  - Message space or data space may be infinite
- Finite approximation
  - Assume finite sessions
    - Example: 2 clients, 2 servers
  - Assume finite message space
    - Represent random numbers by r1, r2, r3, ...
    - Do not allow encrypt(encrypt(encrypt(...)))

Decidable Protocol Analysis
- Eliminate sources of undecidability
  - Bound the number of protocol sessions
    - Artificial bound, no guarantee of completeness
  - Bound structural size of messages by lazy instantiation of variables
  - Loops are simulated by multiple sessions
- Secrecy and authentication are NP-complete if the number of protocol instances is bounded
  - [Rusinowitch, Turuani '01]
- Search for solutions can be fully automated
  - Several tools; we’ll talk about SRI constraint solver

Strand Space Model
- A strand is a representation of a protocol “role”
  - Sequence of “nodes”
  - Describes what a participant playing one side of the protocol must do according to protocol specification
- A node is an observable action
  - “*” node: sending a message
  - “*” node: receiving a message
- Messages are ground terms
  - Standard formalization of cryptographic operations: pairing, encryption, one-way functions, ...
Participant Roles in NSPK

```
Protocol
A → B {n, A}_kb
B → A {n, r}_ka
A → B {r}_kb

"A" role
A → {n, A}_kb
A ← {n, r}_ka
A → {r}_kb

"B" role
B ← {n, A}_kb
B → {n, r}_ka
B ← {r}_kb

Controls network and can schedule any consistent interleaving of these roles
```

Bundles

- A bundle combines strands into a partial ordering
  - Nodes are ordered by internal strand order
  - "Send message" nodes of one strand are matched up with "receive message" nodes of another strand
- Infinitely many possible bundles for any given set of strands
  - No bound on the number of times any given attacker strand may be used
- Each bundle corresponds to a particular execution trace of the protocol
  - Conceptually similar to a Murϕ trace

Parametric Strands

- Use a variable for every term whose value is not known to recipient in advance

```
Parametric "A" strand
+ {n, A}_kb
+ {n, r}_ka
+ {r}_kb

Parametric "B" strand
+ {n, A}_kb
+ {n, r}_ka
+ {r}_kb
```

Properties of Parametric Strands

- Variables are untyped
  - Attacker may substitute a nonce for a key, an encrypted term for a nonce, etc.
  - More flexible; can discover more attacks
- Compound terms may be used as symmetric keys
  - Useful for modeling key establishment protocols
    - Keys constructed by exchanging and hashing random numbers
    - Public keys constructed with pk(A)
- Free term algebra
  - Simple, but cannot model some protocols
  - No explicit decryption, no cryptographic properties
**Attack Scenario**

- Partial bundle corresponding to attack trace
  - By contrast, in Mur $\varphi$ need to specify attack state
  - Assume that the attacker will intercept all messages

```
+ (n, A)_{pk}(X)
+ (Z)_{pk}(A)
```

Is there a way to insert attacker strands here so that attacker learns secret $r$ in the resulting bundle?

```
- (n, Z)_{pk}(A)
+ (Z)_{pk}(X)
```

secret

**Attack Scenario: Example**

**A’s role**
- “Talk to $X$”
- $A \rightarrow \{(n, A)_{pk}(X)\}$
- $A \rightarrow \{(n, Z)_{pk}(A)\}$
- $A \rightarrow \{(Z)_{pk}(X)\}$

**B’s role**
- “Talk to $B$”
- $B \rightarrow \{(Y)_{pk}(A)\}$
- $B \rightarrow \{(Y)_{pk}(A)\}$
- $B \rightarrow \{(Y, r)_{pk}(A)\}$

- “Talk to $X$”
- “Talk to $B$”
- $A \rightarrow \{(n, A)_{pk}(X)\}$
- $B \rightarrow \{(A, Y)_{pk}(B)\}$
- $A \rightarrow \{(n, Z)_{pk}(A)\}$
- $B \rightarrow \{(Y, r)_{pk}(A)\}$
- $A \rightarrow \{(Z)_{pk}(X)\}$
- $B \rightarrow \{(Y, r)_{pk}(A)\}$

- $r$ is a possible way to plug attacker in the middle, for example:
- $B \rightarrow \{(Y, r)_{pk}(A)\}$
- $A \rightarrow \{(n, A)_{pk}(X)\}$
- $B \rightarrow \{(A, Y)_{pk}(B)\}$
- $A \rightarrow \{(n, Z)_{pk}(A)\}$
- $B \rightarrow \{(Y, r)_{pk}(A)\}$
- $A \rightarrow \{(Z)_{pk}(X)\}$

- $r$ is a symbolic attack trace
- Variables are uninstantiated
- It may or may not correspond to a concrete trace

**Symbolic Analysis Problem**

- **Attack modeled as a symbolic trace**
  - Sequence of protocol messages with variables
  - Represents a successful attack
    - For example, attacker learns secret in the end
  - Adequate for secrecy, authentication, fairness

- **Equivalent to a sequence of symbolic constraints**

```
Can the attacker learn message from terms $t_1, \ldots, t_n$?
```

- This constraint is satisfiable if and only if there exists substitution $\sigma$ such that attacker can derive $m\sigma$ from $t_1\sigma, \ldots, t_n\sigma$

**Constraint Generation**

- **For each message attacker sends in the attack trace, create symbolic constraint $m$ from $T_i$**
  - $m$ is the message attacker needs to send
  - $T_i$ is set of messages previously observed by attacker
  - $m, T_i$ may contain variables

- **Attack is feasible if and only if all constraints are satisfiable simultaneously**
  - There exists a substitution $\sigma$ such that $t_i\sigma$ attacker can derive $m\sigma$ from $T_i\sigma$ using Dolev-Yao rules
  - Variables must be instantiated consistently in all terms

**From Protocols to Constraints**

- Formal specification of protocol roles
- Choose finite number of role instances
- Choose an interleaving corresponding to an attack
- Sequence of symbolic constraints
- Constraint solving procedure
Constraint Generation: Example

<table>
<thead>
<tr>
<th>Attack Trace</th>
<th>Symbolic Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>B→E</td>
<td>B → E (&quot;Talk to B&quot;)</td>
</tr>
<tr>
<td>A→E</td>
<td>A → E (&quot;Talk to X&quot;)</td>
</tr>
<tr>
<td>B→R</td>
<td>B → R (A, Y)_{pk(B)}</td>
</tr>
<tr>
<td>A→E</td>
<td>A → E (n, Z)_{mk(A)}</td>
</tr>
<tr>
<td>E→f</td>
<td>E → f (X)_{mk(X)}</td>
</tr>
</tbody>
</table>

"Talk to X" from T0 (attacker's initial knowledge)

\{A,Y\}_{pk(B)} from T0, \{n,A\}_{pk(X)}

\{n,Z\}_{mk(A)} from T0, \{n,A\}_{mk(X)}, \{Y,r\}_{mk(A)}

r from T0, \{n,A\}_{mk(X)}, \{Y,r\}_{mk(A)}, \{Z\}_{mk(X)}

"Talk to X" from T0 (attacker's initial knowledge)

Solving Constraint Sequence

Initial constraint sequence

Non-deterministically apply special transformation rules to first m from T where m is not a variable

No rule is applicable

var, from T0

var, from T0

if reduction tree has at least one such sequence as a leaf, there is a solution, and attack scenario is feasible

SRI Constraint Solver

- Easy protocol specification
  - Specify only protocol rules and correctness condition
  - No explicit intruder rules!
- Fully automated protocol analysis
  - Generates all possible attack scenarios
  - Converts scenario into a constraint solving problem
  - Automatically solves the constraint sequence
- Fast implementation
  - Three-page program in standard Prolog (SWI, XSB, etc.)

A Tiny Bit of Prolog (I)

- Atoms
  - a, foo_bar, 23, 'any.string'
- Variables
  - A, Foo, _G456
- Terms
  - f(N), [a,B], N+1

Using Prolog

- Put definitions in a text file
  - ...actdef or ...actdef.pl
  - swipl, pl or plwin.exe
- Start Prolog
  - Prolog prompt
- Load definitions file
  - consult(factdef). consult(factdef) in SWI-Prolog
  - Both UNIX and Windows
- subdirectory, need quotes
  - Start search for true instance
- Execute query
  - M=6
  - Yes
  - M=6
  - Yes
  - M=6
  - Yes
  - .../factdef or ...actdef.pl
  - swipl, pl or plwin.exe
  - Prolog prompt
  - consult(factdef) in SWI-Prolog
  - Both UNIX and Windows
  - subdirectory, need quotes
  - Start search for true instance
  - Prolog responds
  - Quit Protocol session.
Defining a Protocol: Terms

- **Constants**
  - \(a, b, e, na, k, \ldots\)
  - \(e\) is the name of the attacker

- **Variables**
  - \(A, M, \ldots\)
  - by convention, names capitalized

- **Compound terms**
  - \([A,B,C]\) \(n\)-ary concatenation, for all \(n > 1\)
  - \(A+K\) symmetric encryption
  - \(A*pk(B)\) public-key encryption
  - \(sha(X)\) hash function
  - \(f(X,Y)\) new function unknown to attacker

Specifying Protocol Roles

- **Specifying Protocol Roles**

  - **Special secrecy test strand**
    - Forces analysis to stop as soon as the strand is executed
    ```
    strand(secrecytest,X,[recv(X),send(stop)]).
    ```

  - **When the attacker has learned the secret, he’ll pass it to this strand to “announce” that the attack has succeeded**

Choosing Number of Sessions

- **Choose number of instances for each role**
  - For example, one sender and two recipients

- **In each instance, use different constants to instantiate nonces and keys created by that role**
  - Each nonce modeled by a separate constant
  - 1 instance of role A, 2 instances of role B

Verifying Secrecy

- **Add secrecy test strand to the bundle**
  ```
  nspk0([Sa,Sb1,Sb2]) :-
  strand(roleA(a,b,na,Nb,Sa)),
  strand(roleB,a,b,Na1,nb1,Sb1),
  strand(roleB,A3,b,Na2,nb2,Sb2),
  strand(secrecytest,nb1,St).
  ```

- **This bundle is solvable if and only if the attacker can learn secret \(nb1\) and pass it to test strand**

- **Run the constraint solver to find out**
  ```
  :- nspk0(B),search(B,[{}]).
  ```

- **This is it! Will print the attack if there is one.**

Specifying Authentication Condition

- **What is authentication?**
  - If B completes the protocol successfully, then there is or was an instance of A that agrees with B on certain values (each other’s identity, some key, some nonce)

- **Use a special authentication message**
  ```
  send(roleA(a,b,nb))
  ```

  “A believes he is talking to B and B’s nonce is \(nb\)”

- **Attack succeeds if B completes protocol, but A’s doesn’t send authentication message**
  - B thinks he is talking to A, but not vice versa
NSPK Strands for Authentication

\[
\text{strand}(\text{roleA}, A, B, Na, Nb, [} \\
\text{send}\ {[A, Na] * pk (B)}, \\
\text{recv}\ {[Na, Nb] * pk (A)}, \\
\text{send}\ {[Nb, Nb] * pk (B)}\ ]).
\]

\[
\text{strand}(\text{roleB}, A, B, Na, Nb, [} \\
\text{recv}\ {[A, Na] * pk (B)}, \\
\text{send}\ {[Na, Nb] * pk (A)}, \\
\text{recv}\ {Nb} * pk (B)}, \\
\text{send}\ {[roleB(A, B, Na)]}\ ]).
\]

Verifying Authentication

\[
\text{nspk0}([Sa,Sb,St], \text{roleA}(a,b,nb)) \rightarrow \text{nspk0}([Sa,Sb,St], \text{roleB}(a,b,na)).
\]

\[
\text{test} = \text{roleB}(a,b,na) \rightarrow \text{roleA}(a,b,nb).
\]

This bundle is solvable if and only if the attacker can cause \text{roleA}(a,b,nb) to appear in a trace that does not contain \text{roleB}(a,b,na).

Convince B that he is talking A when A does not think he is talking to B.

Symbolic Analysis in a Nutshell

Informal protocol description

Participant roles

All possible attack traces

Symbolic constraints for each trace

Automated constraint solving procedure

If constraints are satisfied, then there is an attack