Logistics

- Scribing
  - Notes due a week after the lecture
  - Sign-up!

- Blackboard access for people who are auditing, but are not officially enrolled:
  - send email to Joe

- Homework 1 is out; due Sept 18 before class

- Project discussion:
  - Sept 18, 20 (Tue-Th): 4:30-6PM in CIC 2118
  - Sign up for slots (15 min per group)
This Lecture

- Model checking overview
- Model checking security protocols using Murphi
  - Example: Needham-Schroeder public key protocol

- Note:
  - More on Murphi this Friday
  - HW1 will use Murphi; possibly projects
Model Checking: Basics

- Model the system
  - Set of states + Start state
  - Transition relation between states

- Specify properties
  - Special case: Invariants (properties that are true in all states)

- Model checking
  - Check that every reachable state in the graph satisfies the invariants
  - Graph search done using BFS or DFS

Developed by [Clarke, Emerson, Sistla] and [Quille, Sifakis] around 1980
Model Checking: Issue

- State space explosion
  - The number of reachable states quickly blows up

- Techniques to combat problem
  - Symmetry reduction
  - Partial order reduction
  - Symbolic representation of transition relation

More details in 15-817A: Introduction to Model Checking
Model checking security protocols

- Many tools (see “Tools” page on course web page)
  - Murphi – Explicit state model-checker
    - Authentication, key exchange protocols
  - PRISM – Probabilistic model checker
    - Protocols for anonymity
  - MOCHA – Game properties specified in ATL and ATL*
    - Contract signing protocols
  - AVISPA – Specialized symbolic model checker for security protocols
    - Authentication, key exchange

Tools can be used for course projects
Explicit intruder model

Informal Protocol Description → Formal Protocol → Intruder Model → Specify Property

Find error → Analysis Tool
Example: Needham-Schroeder

- Famous simple example
  - Protocol published and known for 10 years
  - Gavin Lowe discovered unintended property while preparing formal analysis using FDR system

- Subsequently rediscovered by every analysis method
Needham-Schroeder Crypto

- Nonces
  - Fresh, random numbers

- Public-key cryptography
  - Every agent A has
    - Public encryption key Ka
    - Private decryption key Ka^{-1}

- Main properties
  - Everyone can encrypt message to A
  - Only A can decrypt these messages

Recall how we modeled encryption in the symbolic model
Needham-Schroeder Key Exchange

A \rightarrow B: \{ \text{A, NonceA} \}^{K_b}

B \rightarrow A: \{ \text{NonceA, NonceB} \}^{K_a}

A \leftarrow B: \{ \text{NonceB} \}^{K_b}

Security properties: On execution of the protocol, A and B are guaranteed mutual authentication and secrecy
NS security properties

- Responder correctly authenticated
  - When initiator A completes the protocol apparently with honest responder B, it must be that B believes he ran the protocol with A

- Initiator correctly authenticated
  - When responder B completes the protocol apparently with honest initiator A, it must be that A believes she ran the protocol with B

- Initiator Nonce secrecy
  - When honest initiator completes the protocol with honest peer, intruder does not know initiator’s nonce.
Anomaly in Needham-Schroeder

Evil agent E tricks honest A into revealing private key NB from B

Evil E can then fool B

[Lowe96]
Murphi

- Describe finite-state system
  - State variables with initial values
  - Transition rules
  - Communication by shared variables
- Scalable: choose system size parameters
- Automatic exhaustive state enumeration
  - Space limit: hash table to avoid repeating states
- Research and industrial protocol verification
- Method for security protocol analysis developed by Mitchell et al
Limitations of Finite State Methods

- Two sources of infinite behavior
  - Many instances of participants, multiple runs
  - Message space or data space may be infinite

- Finite approximation
  - Assume finite participants
    - Example: 2 clients, 2 servers
  - Assume finite message space
    - Represent random numbers by $r_1, r_2, r_3, \ldots$
    - Do not allow $\text{encrypt}(	ext{encrypt}(	ext{encrypt}(\ldots)))$
Applying Murϕ to security protocols

- **Formulate protocol**
  - Model initiator, responder state machines
  - Model n/w as a shared variable
  - Model properties using invariants

- **Add adversary**
  - Control over “network”
  - Possible actions
    - Intercept any message
    - Remember parts of messages
    - Generate new messages, using observed data and initial knowledge (e.g. public keys)
const
    NumInitiators: 1;  -- number of initiators
    NumResponders: 1;  -- number of responders
...
type
    InitiatorId: scalarset (NumInitiators);
    InitiatorStates: enum{I_SLEEP, I_WAIT, I_COMMIT};
    Initiator: record
        state: InitiatorStates
        responder: AgentId
    end;
var
    ini: array [InitiatorId] of Initiator
MessageType : enum {              -- types of messages
    M_NonceAddress,          -- {Na, A}Kb  nonce and addr
    M_NonceNonce,            -- {Na,Nb}Ka  two nonces
    M_Nonce                  -- {Nb}Kb    one nonce
};

Message : record
    source:   AgentId;      -- source of message
    dest:     AgentId;      -- intended destination of msg
    key:      AgentId;      -- key used for encryption
    mType:    MessageType;  -- type of message
    nonce1:   AgentId;      -- nonce1
    nonce2:   AgentId;      -- nonce2 OR sender id OR empty
end;

var
    net: multiset[NetworkSize] of Message;       -- state variable for n/w
Modeling Protocol Actions

ruleset i: InitiatorId do
  ruleset j: AgentId do
    rule 20 "initiator starts protocol"
      ini[i].state = I_SLEEP & multisetcount (l:net, true) < NetworkSize
      ==> var
        outM: Message;  -- outgoing message
      begin
        undefine outM;
        outM.source := i; outM.dest := j;
        outM.key := j; outM.mType := M_NonceAddress;
        outM.nonce1 := i; outM.nonce2 := i;
        multisetadd (outM, net); ini[i].state := I_WAIT;
        ini[i].responder := j;
      end; end; end;
invariant "responder correctly authenticated"

forall i: InitiatorId do
    ini[i].state = I_COMMIT &
    ismember(ini[i].responder, ResponderId)
    ->
    res[ini[i].responder].initiator = i &
    ( res[ini[i].responder].state = R_WAIT |
    res[ini[i].responder].state = R_COMMIT )
end;
Adversary Model

- Formalize “knowledge”
  - initial data
  - observed message fields
  - results of simple computations

- Optimization
  - only generate messages that others read
  - time-consuming to hand simplify

- Possibility: automatic generation
Attacker capabilities

(1) $S \vdash m \land S \vdash k \Rightarrow S \vdash \{m\}_k$
(2) $S \vdash \{m\}_k \land S \vdash k \Rightarrow S \vdash m$
(3) $S \vdash m \land S \vdash K \Rightarrow S \vdash \{m\}_K$
(4) $S \vdash \{m\}_K \land S \vdash K^{-1} \Rightarrow S \vdash m$
(5) $S \vdash \{m\}_K \land S \vdash s^{-1} \Rightarrow S \vdash \text{sig}(s^{-1}, m)$
(6) $S \vdash m \Rightarrow S \vdash h(m)$
(7) $S \vdash m \land S \vdash k \Rightarrow S \vdash h(k,m)$
(8) $S \vdash m_1 \land S \vdash m_2 \Rightarrow S \vdash <m_1, m_2>$
(9) $S \vdash <m_1, m_2> \Rightarrow S \vdash m_1 \land S \vdash m_2$
(10) $S \vdash n$ (new symbol denoting random number)

$S$ includes initial knowledge of attacker + messages sent on the network
Modeling the attacker

-- intruder i sends recorded message
ruleset i: IntruderId do -- arbitrary choice of
  choose j: int[i].messages do -- recorded message
  ruleset k: AgentId do -- destination
    rule "intruder sends recorded message"
      !ismember(k, IntruderId) & -- not to intruders
      multisetcount (l:net, true) < NetworkSize
    =>
    var outM: Message;
    begin
      outM := int[i].messages[j];
      outM.source := i;
      outM.dest := k;
      multisetadd (outM, net);
    end; end; end; end;
Run of Needham-Schroeder

- Find error after 1.7 seconds exploration
- Output: trace leading to error state
- Murphi times after correcting error:

<table>
<thead>
<tr>
<th>number of</th>
<th>size of network</th>
<th>states</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ini. res int.</td>
<td>1</td>
<td>1706</td>
<td>3.1s</td>
</tr>
<tr>
<td>1 1 1</td>
<td>2</td>
<td>40207</td>
<td>82.2s</td>
</tr>
<tr>
<td>2 1 1</td>
<td>1</td>
<td>17277</td>
<td>43.1s</td>
</tr>
<tr>
<td>2 2 1</td>
<td>1</td>
<td>514550</td>
<td>5761.1s</td>
</tr>
</tbody>
</table>
Homework #1

- Investigate the NS flaw and the fixed Needham Schroeder Lowe protocol
- Investigate conditions under which attack succeeds: adversary power, initiator behavior and crypto
Limitations

- System size with current methods
  - 2-6 participants
    - Kerberos: 2 clients, 2 servers, 1 KDC, 1 TGS
  - 3-6 steps in protocol
  - May need to optimize adversary

- Adversary model
  - Cannot model randomized attack
  - Do not model adversary running time
State Reduction on N-S Protocol

- **Base:** hand optimization of model
- **CSFW:** eliminate net, max knowledge
- **Merge:** intrud send, princ reply
Security Protocols in Murφ

- Standard “benchmark” protocols
  - Needham-Schroeder, TMN, …
  - Kerberos

- Study of Secure Sockets Layer (SSL)
  - Versions 2.0 and 3.0 of handshake protocol
  - Include protocol resumption

- Tool optimization

- Additional protocols
  - Contract-signing
  - Wireless networking (IEEE 802.11i)
Questions?