CS259: Security Analysis of Network Protocols

Overview of Murphi

Arnab Roy

Running Murphi

- Elaine Machines
  - Murphi available at /usr/class/cs259/Murphi3.1/
  - HW1 code available at /usr/class/cs259/hw1/
- Any issues so far?

Running Murphi

- If you are using another linux machine or cygwin
    - Copy the /usr/class/cs259/Murphi3.1/ directory to your home, lets say /home/cs259/Murphi3.1/
    - Copy the files ‘ns.m’ and ‘Makefile’ in /usr/class/cs259/hw1 to /home/cs259/hw1/
    - Modify paths in Makefile to reflect changes:
        - MURPHI = /home/cs259/Murphi3.1/bin/mu
        - INCLUDE = /home/cs259/Murphi3.1/include/

Running Murphi

- If you are using cygwin or a different distribution of Linux, you might have to recompile Murphi. To do this,
  - ‘cd’ to /home/cs259/Murphi3.1/src and do ‘make’
- In the hw1 directory, modify paths in Makefile to reflect changes, e.g.:
    - MURPHI = /home/cs259/Murphi3.1/bin/mu
    - INCLUDE = /home/cs259/Murphi3.1/include/

Murφ

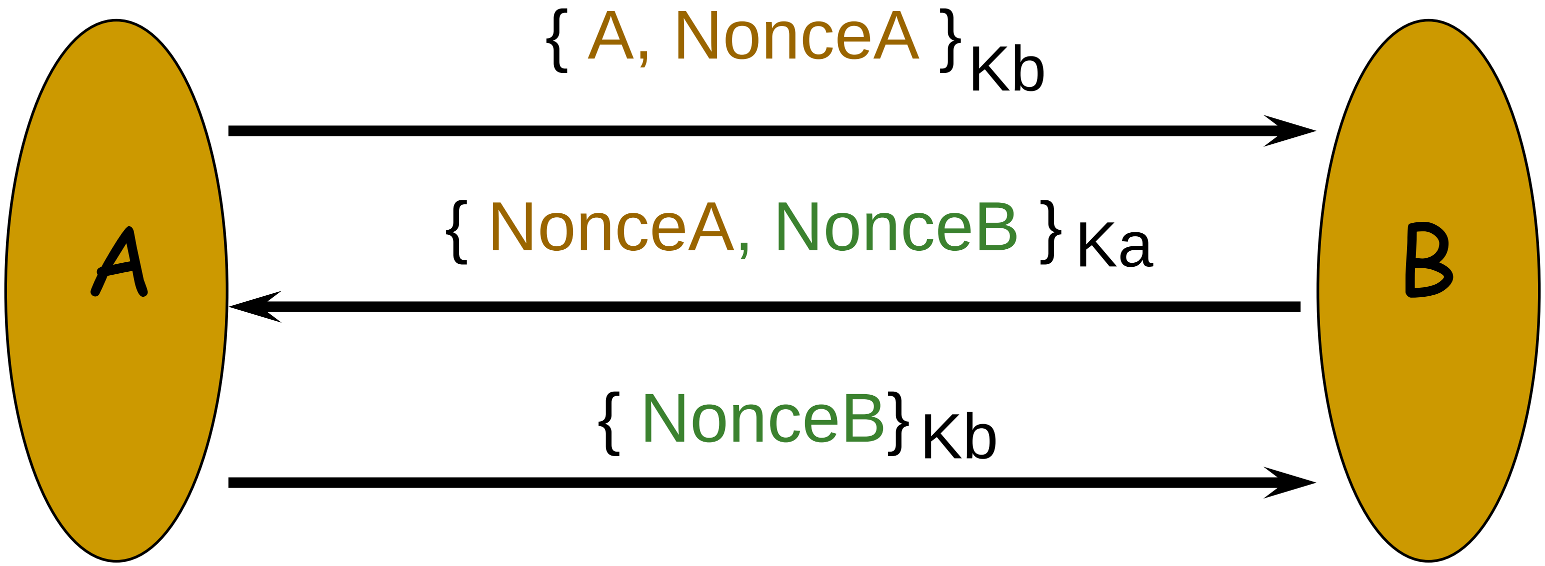
[Dill et al.]

- 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

Caveat Emptor!

- A Murphi analysis coming up with no errors
  - does not prove security of the protocols
  - only provides the limited assurance that protocol secure with fixed limits on number of participants and operations
- However, errors found are most likely real bugs!

Needham-Schroeder Key Exchange



$\{ \text{A, NonceA} \}_{Kb}$

$\{ \text{NonceA, NonceB} \}_{Ka}$

$\{ \text{NonceB} \}_{Kb}$

Result: A and B share two private numbers not known to any observer without Ka^{-1}, Kb^{-1}

Applying Murφ to security protocols

- Formulate protocol
  - Model the honest party roles
- Add adversary
  - Control over “network”   (shared variables)
  - Possible actions
    - Intercept any message
    - Remember parts of messages
    - Generate new messages, using observed data and initial knowledge (e.g. public keys)

Needham-Schroeder in Murφ

```
const
  NumInitiators:  1;   -- number of initiators
  NumResponders:  1;   -- number of responders
  NumIntruders:   1;   -- number of intruders
  NetworkSize:    1;   -- max. outstanding msgs in network
  MaxKnowledge:  10;   -- number msgs intruder can remember

type
  InitiatorId:  scalarset (NumInitiators);
  ResponderId:  scalarset (NumResponders);
  IntruderId:   scalarset (NumIntruders);

  AgentId:   union {InitiatorId, ResponderId, IntruderId};
```

N-S message format in Murφ

```
MessageType : enum {          -- types of messages
  M_NonceAddress,             --  {Na, A}Kb  nonce and addr
  M_NonceNonceAddress,        --  {Na,Nb,B}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
    address:  AgentId;          -- sender identifier
end;
```

Participant states

```
InitiatorStates : enum {
  I_SLEEP,                      -- state after initialization
  I_WAIT,                       -- waiting for response from responder
  I_COMMIT                      -- initiator commits to session
};                              -- (thinks responder is authenticated)

Initiator : record
  state:     InitiatorStates;
  responder: AgentId;            -- agent with whom the initiator
end;                             -- starts the protocol

Intruder : record
  nonces:   array[AgentId] of boolean;              -- known nonces
  messages: multiset[MaxKnowledge] of Message;      -- known msgs
end;
```

N-S protocol action in Murφ

```
ruleset i: InitiatorId do
  ruleset j: AgentId do
    rule "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;
```

Adversary Model

- Formalize “knowledge”
  - initial data
  - observed message fields
  - results of simple computations
- Optimization
  - only generate messages that others read

N-S attacker action in Murφ

```
-- 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;
```

Start State

```
startstate
  -- initialize initiators
  undefine ini;
  for i: InitiatorId do
    ini[i].state     := I_SLEEP;
    ini[i].responder := i;
  end;

  -- initialize responders
  undefine res;
  for i: ResponderId do
    res[i].state     := R_SLEEP;
    res[i].initiator := i;
  end;

  -- initialize intruder, network
  ...
end;
```

Modeling Properties

```
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;
```

Questions?