CS 395T

Design and Analysis of Security Protocols

Vitaly Shmatikov

http://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/

Course Logistics

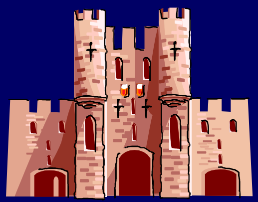
- Lectures
  - Monday, Wednesday 3:30-5pm
  - Project presentations in the last two weeks
- This is a project course
  - The best way to understand security is by getting your hands dirty
  - There will be one short homework and one read-and-present a research paper assignment
  - Most of your work will be project, writeup and in-class presentation

Please enroll!

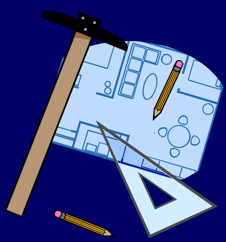
Grading

- Homework: 10%
- Read and present a research paper: 15%
- Project: 75%
  - Projects are best done individually
  - Two-person teams are Ok, but talk to me first
  - Project proposal due around 5th week of the course
    - More details later
  - I'll provide a list of potential project ideas, but don't hesitate to propose your own

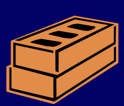
Computer Security



Systems



Blueprints



Building blocks

Implementation

Protocols and policies

Cryptographic primitives

Algorithmic number theory

Computational complexity

Firewalls, intrusion detection…

SSL, IPSec, access control…

RSA, DSS, SHA-1…

Class Poll

- Cryptography?
  - Public-key and symmetric encryption, digital signatures, cryptographic hash, random-number generators?
  - Computational complexity?
- Systems security?
  - Buffer overflows, Web security, sandboxing, firewalls, denial of service?
- Formal methods and verification?
  - Model checking, theorem proving?

… this course doesn't require any of these ☺

Security Protocols

- The focus of this course is on secure communications...
  - Two or more parties
  - Communication over insecure network
  - Cryptography used to achieve some goal
    - Exchange secret keys, verify identity, pay for a service...
- ...and formal analysis techniques for security
  - Analyze protocol design assuming cryptography, implementation, underlying OS are correct
- Later in the course will talk about privacy protection in databases and trusted computing

Correctness vs Security

- Program or system correctness: program satisfies specification
  - For reasonable input, get reasonable output
- Program or system security: program properties preserved in face of attack
  - For unreasonable input, output not completely disastrous
- Main differences
  - Active interference from adversary
  - Refinement techniques may fail
    - Abstraction is very difficult to achieve in security: what if the adversary operates below your level of abstraction?

Security Analysis

- ❶ Model system
- ❷ Model adversary
- ❸ Identify security properties
- ❹ See if properties preserved under attack

Theme #1: there are many notions of what it means for a protocol to be “secure”

Theme #2: there are many ways of looking for security flaws

- Result
  - Under given assumptions about system, no attack of a certain form will destroy specified properties
  - There is no “absolute” security

Theme #1: Protocols and Properties

Some of these are excellent topics for a project or the paper-reading assignment

- Authentication
  - Needham-Schroeder, Kerberos
- Key establishment
  - SSL/TLS, IPSec protocols (IKE, JFK, IKEv2)
- Secure group protocols
  - Group Diffie-Hellman, CLIQUES, key trees and graphs
- Anonymity
  - MIX, Onion routing, Mixmaster and Mixminion
- Electronic payments, wireless security, fair exchange, privacy...

Theme #2: Formal Analysis Methods

- Focus on special-purpose security applications
  - Some techniques are very different from those used in hardware verification
  - In all cases, the main difficulty is modeling the attacker
- Simple, mechanical models of the attacker
- No cryptanalysis!
  - In this course, we'll assume that cryptography is perfect
  - Search for design flaws, not cryptographic attacks
- We'll talk about the relationship between formal and cryptographic models late in the course

Variety of Tools and Techniques

- Secrecy
- Authentication
- Authorization

- Anonymity

- Fairness

◆ Explicit finite-state checking
- Murφ model checker
- There will be a small homework!

◆ Infinite-state symbolic model checking
- SRI constraint solver

◆ Process algebras
- Applied pi-calculus

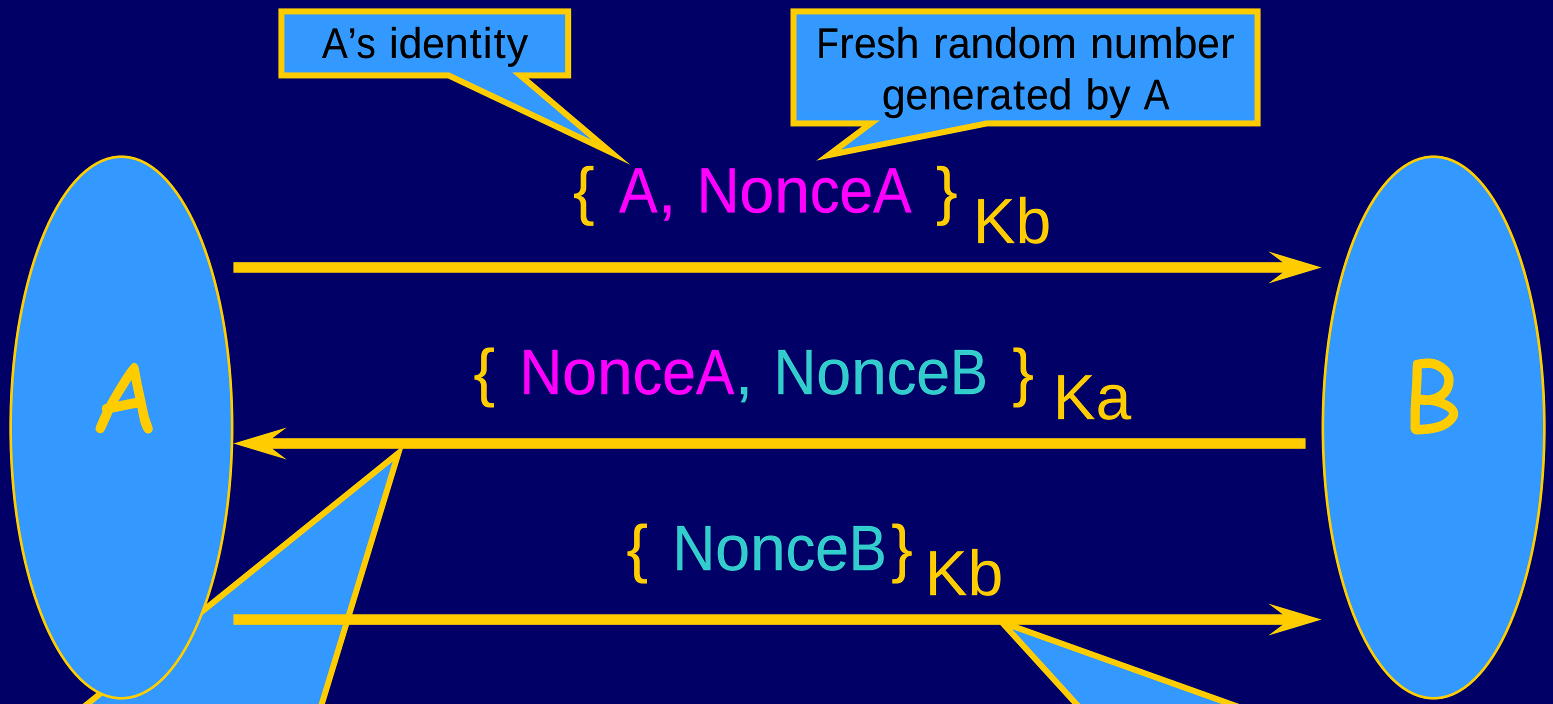
◆ Probabilistic model checking
- PRISM probabilistic model checker

◆ Game-based verification
- MOCHA model checker

Example: Needham-Schroeder

- Very (in)famous example
  - Appeared in a 1979 paper
  - Goal: authentication in a network of workstations
  - In 1995, Gavin Lowe discovered unintended property while preparing formal analysis using FDR system
- Background: public-key cryptography
  - Every agent A has a key pair Ka, Ka^{-1}
  - Everybody knows public key Ka and can encrypt messages to A with it (we'll use $\{m\}_{Ka}$ notation)
  - Only A knows secret key Ka^{-1}, therefore, only A can decrypt messages encrypted with Ka

Needham-Schroeder Public-Key Protocol



A's reasoning:

- The only person who could know NonceA is the person who decrypted 1^{st} message
- Only B can decrypt message encrypted with Kb
- Therefore, B is on the other end of the line

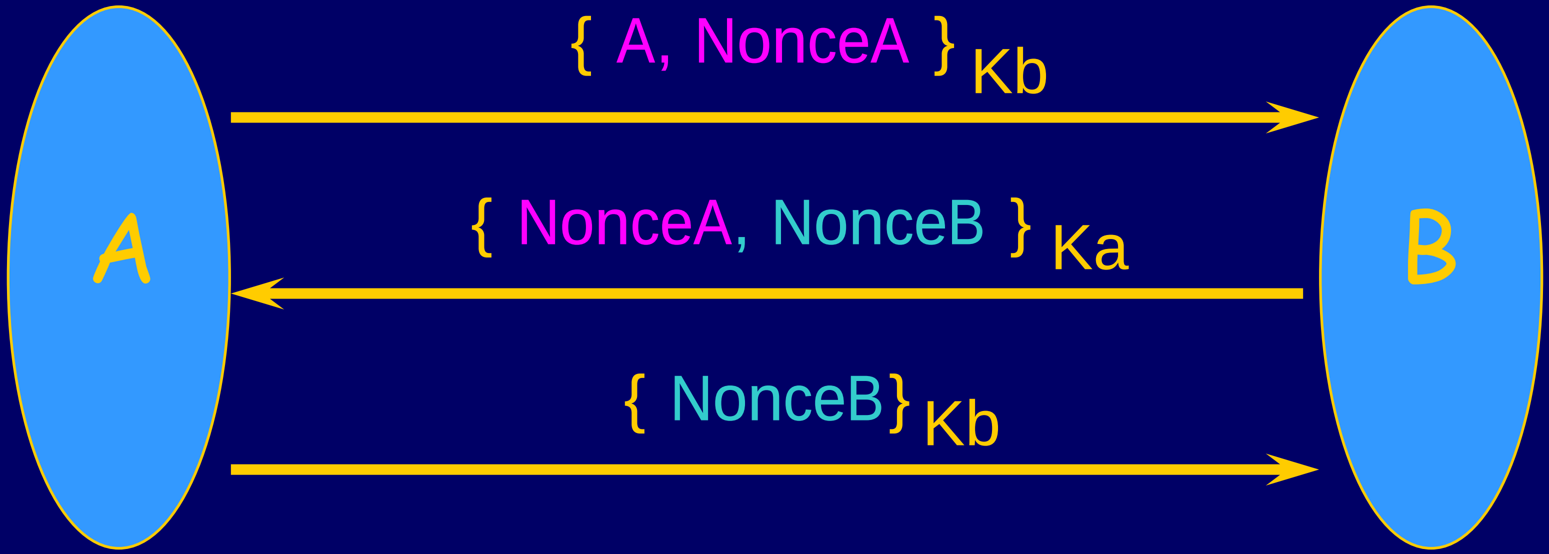
B is authenticated!

B's reasoning:

- The only way to learn NonceB is to decrypt 2^{nd} message
- Only A can decrypt 2^{nd} message
- Therefore, A is on the other end

A is authenticated!

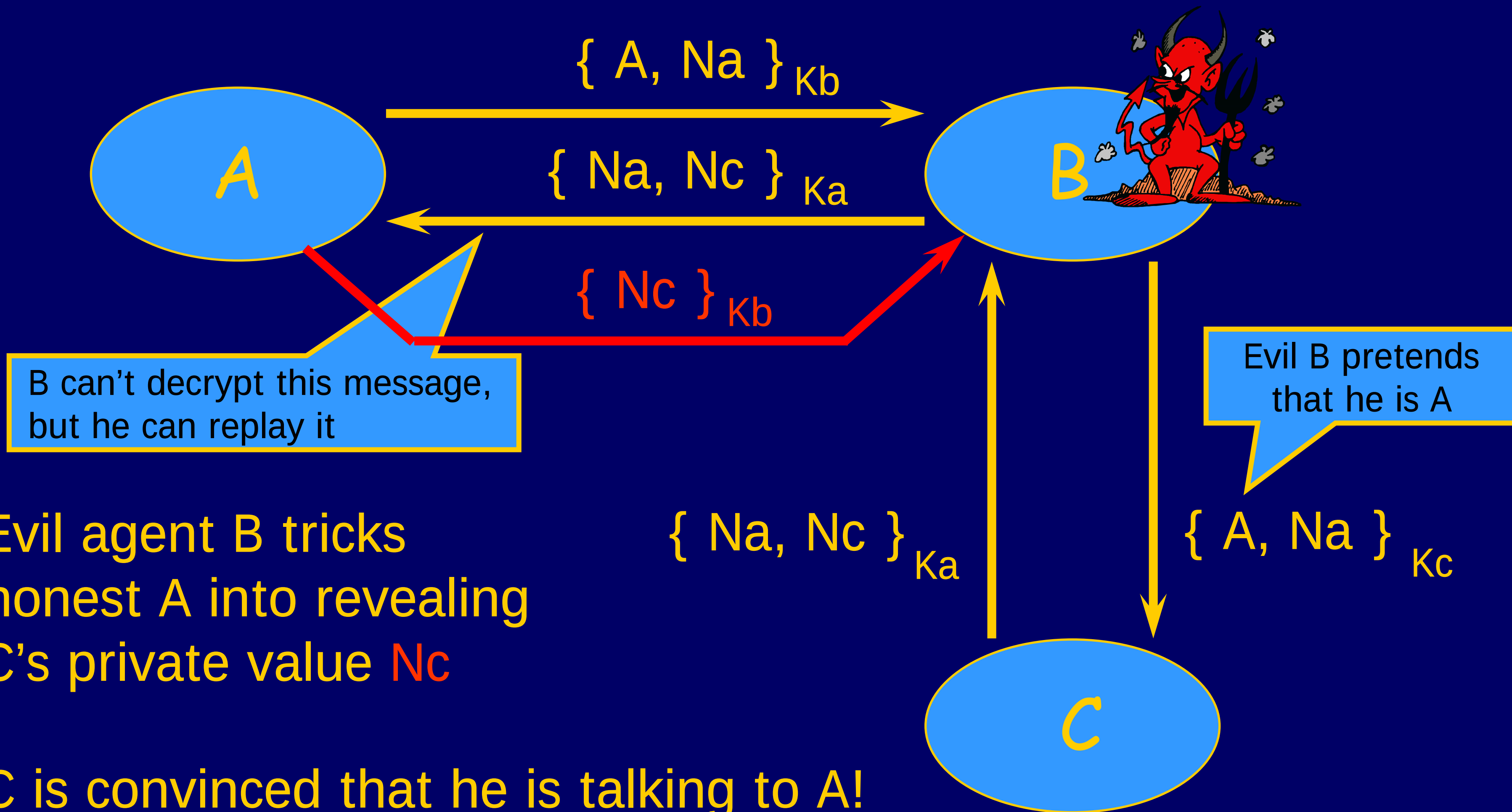
What Does This Protocol Achieve?



- Protocol aims to provide both authentication and secrecy
- After this the exchange, only A and B know Na and Nb
- Na and Nb can be used to derive a shared key

Anomaly in Needham-Schroeder

[published by Lowe]



Evil agent B tricks honest A into revealing C's private value Nc

C is convinced that he is talking to A!

Lessons of Needham-Schroeder

- Classic man-in-the-middle attack
- Exploits participants' reasoning to fool them
  - A is correct that B must have decrypted $\{A,Na\}_{Kb}$ message, but this does not mean that $\{Na,Nb\}_{Ka}$ message came from B
  - The attack has nothing to do with cryptography!
- It is important to realize limitations of protocols
  - The attack requires that A willingly talk to adversary
  - In the original setting, each workstation is assumed to be well-behaved, and the protocol is correct!
- Wouldn't it be great if one could discover attacks like this automatically?

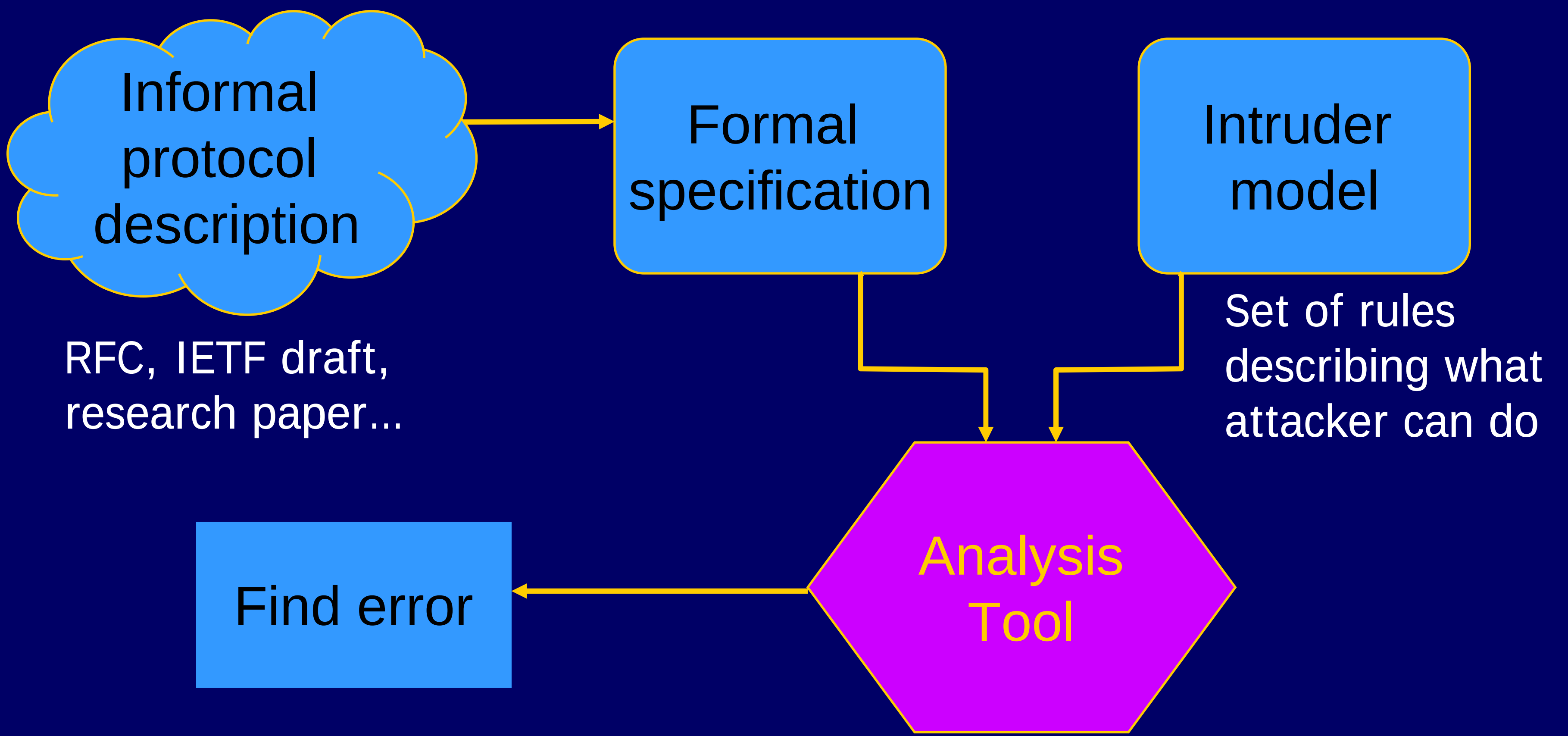
Important Modeling Decisions

- How powerful is the adversary?
  - Simple replay of previous messages
  - Decompose into pieces, reassemble and resend
  - Statistical analysis, partial info from network traffic
  - Timing attacks
- How much detail in underlying data types?
  - Plaintext, ciphertext and keys
    - Atomic data or bit sequences?
  - Encryption and hash functions
    - Perfect ("black-box") cryptography
    - Algebraic properties: encr(x+y) = encr(x) * encr(y) for RSA
      because encrypt(k,msg) = msg^k mod N

Fundamental Tradeoff

- Formal models are abstract and greatly simplified
  - Components modeled as finite-state machines
  - Cryptographic functions modeled as abstract data types
  - Security property stated as unreachability of “bad” state
- Formal models are tractable...
  - Lots of verification methods, many automated
- ...but not necessarily sound
  - Proofs in the abstract model are subject to simplifying assumptions which ignore some of attacker’s capabilities
- Attack in the formal model implies actual attack

Explicit Intruder Method



Murφ [Dill et al.]

- Describe finite-state system
  - State variables with initial values
  - Transition rules for each protocol participant
  - Communication by shared variables
- Specify security condition as a state invariant
  - Predicate over state variables that must be true in every state reachable by the protocol
- Automatic exhaustive state enumeration
  - Can use hash table to avoid repeating states
- Research and industrial protocol verification

Making the Model Finite

- Two sources of infinite behavior
  - Many instances of participants, multiple runs
  - Message space or data space may be infinite
- Finite approximation
  - Assume finite number of participants
    - For example, 2 clients, 2 servers
    - Murφ is scalable: can choose system size parameters
  - Assume finite message space
    - Represent random numbers by constants r1, r2, r3, ...
    - Do not allow encrypt(encrypt(encrypt(...)))

Applying Murφ to Security Protocols

- **Formulate the protocol**
  - Define a datatype for each message format
  - Describe finite-state behavior of each participant
    - If received message M3, then create message M4, deposit it in the network buffer, and go to state WAIT
  - Describe security condition as state invariant
- **Add adversary**
  - Full control over the "network" (shared buffer)
  - Nondeterministic choice of actions
    - Intercept a message and split it into parts; remember parts
    - Generate new messages from observed data and initial knowledge (e.g., public keys)

Murφ will try all possible combinations

Needham-Schroeder in Murφ (1)

```
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};
```

Needham-Schroeder in Murφ (2)

```
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;
```

Needham-Schroeder in Murφ (3)

```
-- 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;
```

Try Playing With Murφ

- You'll need to use Murφ for your first homework
- The input language is easy to understand, but ask me if you are having problems
  - Simple IF... THEN... guarded commands
  - Attacker is nondeterministic, not sequential
- Local Murφ installation is in
  `/projects/shmat/Murphi3.1`

  Some security examples are in
  `/projects/shmat/Murphi3.1/ex/secur`
  - Needham-Schroeder, SSL (ignore rule priorities!)

Start Thinking About the Project

- I'll post a list of ideas soon
- Four ways to go about it
  - Use one of the tools we'll discuss in class to analyze an existing or proposed protocol
    - Learn to read an RFC
    - Check out reference materials on the class website
  - Extend a tool to handle a new class of properties
  - Do a theoretical project
    - Example: algorithmic properties of verification techniques; relationship between cryptographic and formal models
  - Invent something of your own (but talk to me first!)

Some Ideas

- E-commerce protocols
  - Micropayment schemes, secure electronic transactions
- Wireless security
  - Ad-hoc routing, WiFi security, location security
- Trusted Computing Base / Palladium
- Electronic voting
- Group key management protocols
- Anonymity networks
- Censorship-resistant Web publishing
- Choose something that interests you!

Watch This Space

http://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/

- Already contains pointers to several tools, some with online demos
- I'll be constantly adding new references
- Start poking around in protocol libraries
  - Clark-Jacob survey is a good start