Marktoberdorf Summer School 2007

Security Analysis of Network Protocols

John Mitchell

Stanford

Reference: http://www.stanford.edu/class/cs259/

It’s great to be here

- My third Summer School
  - Other two were “red series”
- Some goals
  - Meet old and new friends
  - Sample five main kinds of beer made in Bavaria
  - Swim in lake, after discussion before dinner (T,Th,F)
  - Hike, weather and other factors permitting (W?)

Computer Security

- Cryptography
  - Encryption, signatures, cryptographic hash, …
- Security mechanisms
  - Access control policy
  - Network protocols
- Implementation
  - Cryptographic library
  - Code implementing mechanisms
    - Reference monitor and TCB
    - Protocol
  - Runs under OS, uses program library, network protocol stack

Analyze protocols, assuming crypto, implementation, OS correct

Cryptographic Protocols

- Two or more parties
- Communication over insecure network
- Cryptography used to achieve goal
  - Exchange secret keys
  - Verify identity (authentication)

Crypto (class poll):

Public-key encryption, symmetric-key encryption, CBC, hash, signature, key generation, random-number generators

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
    - More functionality can be *worse*

Security Analysis

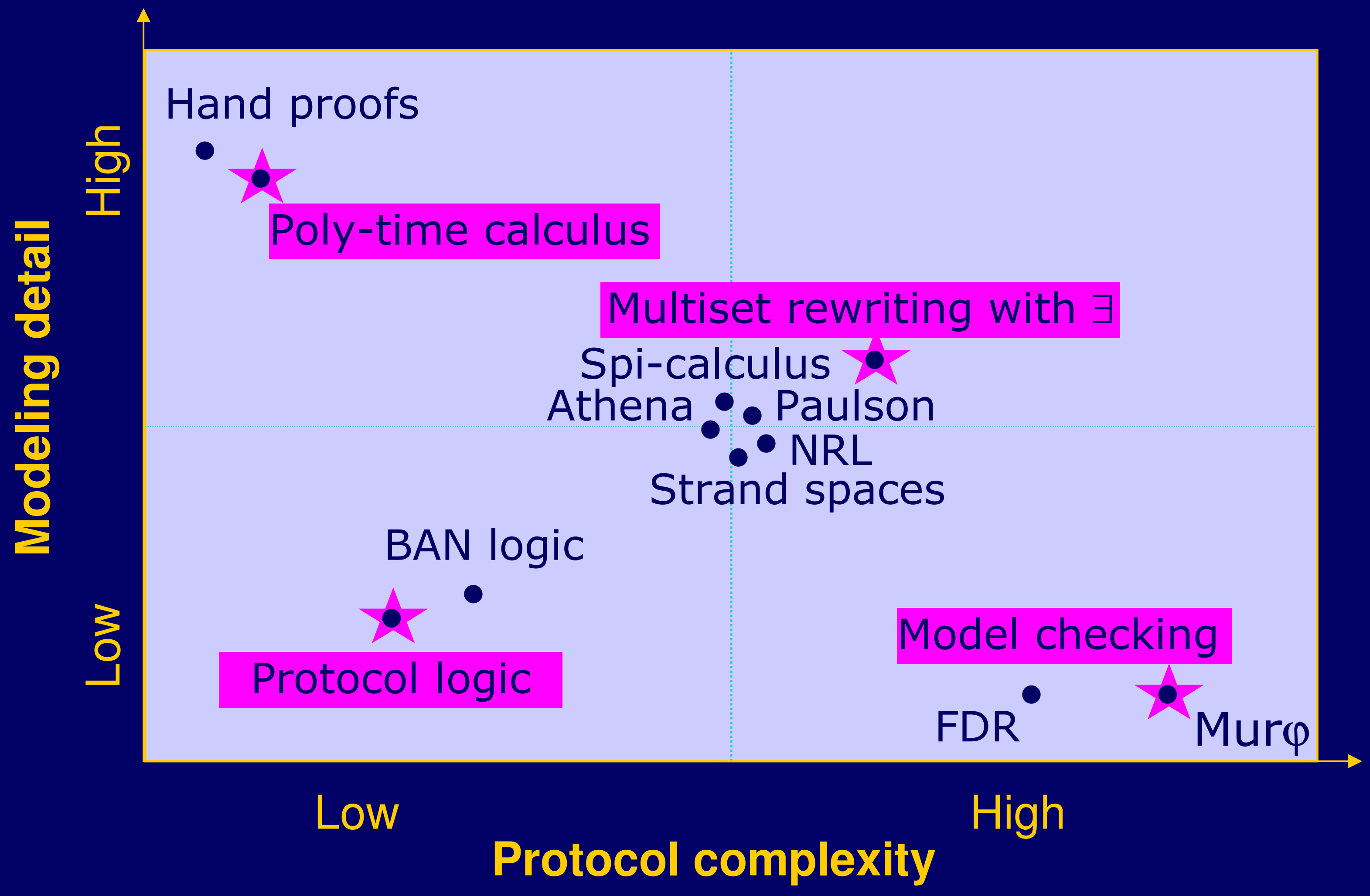
- Model system
- Model adversary
- Identify security properties
- See if properties are preserved under attack

- Result
  - No “absolute security”
  - Security means: under given assumptions about system, no attack of a certain form will destroy specified properties.

Important Modeling Decisions

- How powerful is the adversary?
  - Simple replay of previous messages
  - Block messages; Decompose, 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" cryptography
    - algebraic properties: encr(x*y) = encr(x) * encr(y) for RSA encrypt(k,msg) = msg^k mod N

Protocol analysis spectrum



Four “Stanford” approaches

SRI, U Penn, U Texas, Kiel, INRIA, …

- Finite-state analysis
  - Case studies: find errors, debug specifications
- Symbolic execution model: Multiset rewriting
  - Identify basic assumptions
  - Study optimizations, prove correctness
  - Complexity results
- Process calculus with probability and complexity
  - More realistic intruder model
  - Interaction between protocol and cryptography
  - Equational specification and reasoning methods
- Protocol logic
  - Axiomatic system for modular proofs of protocol properties

Some other projects and tools

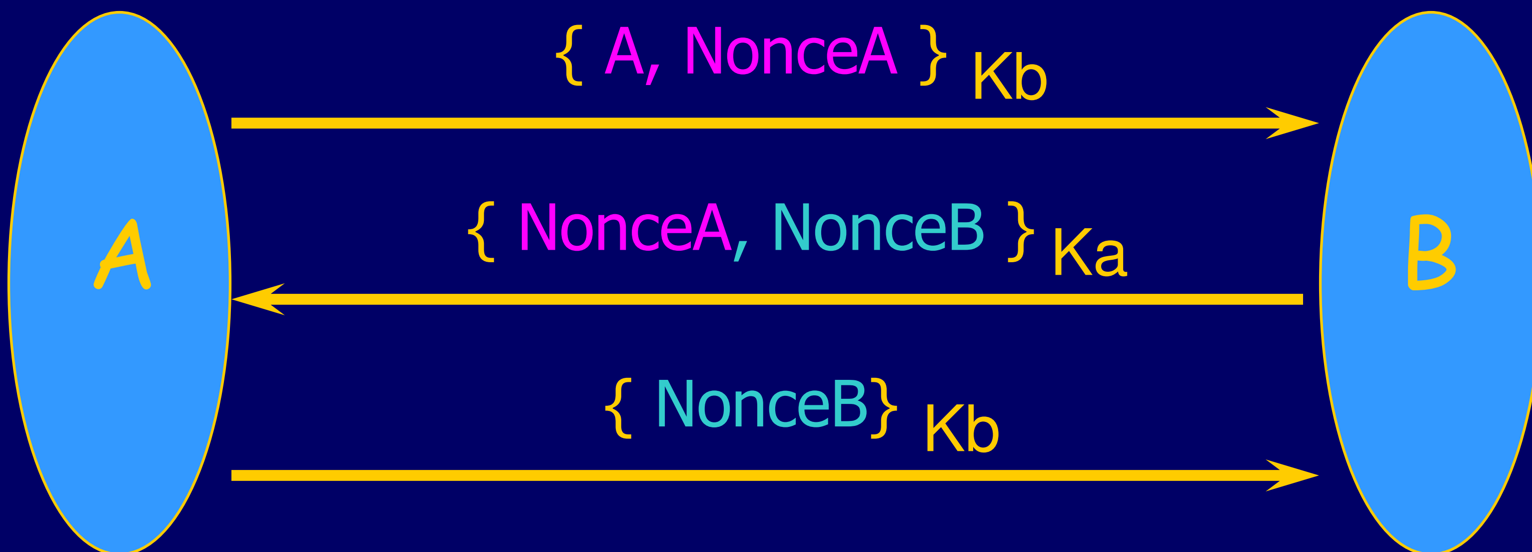
- Exhaustive finite-state analysis
  - FDR, based on CSP [Lowe, Roscoe, Schneider, …]
- Search using symbolic representation of states
  - Meadows: NRL Analyzer, Millen: Interrogator
- Prove protocol correct
  - Paulson's "Inductive method", others in HOL, PVS, …
  - MITRE -- Strand spaces
  - Process calculus approach: Abadi-Gordon spi-calculus, applied pi-calculus, …
  - Type-checking method: Gordon and Jeffreys, …

Many more – this is just a small sample

Example: Needham-Schroeder

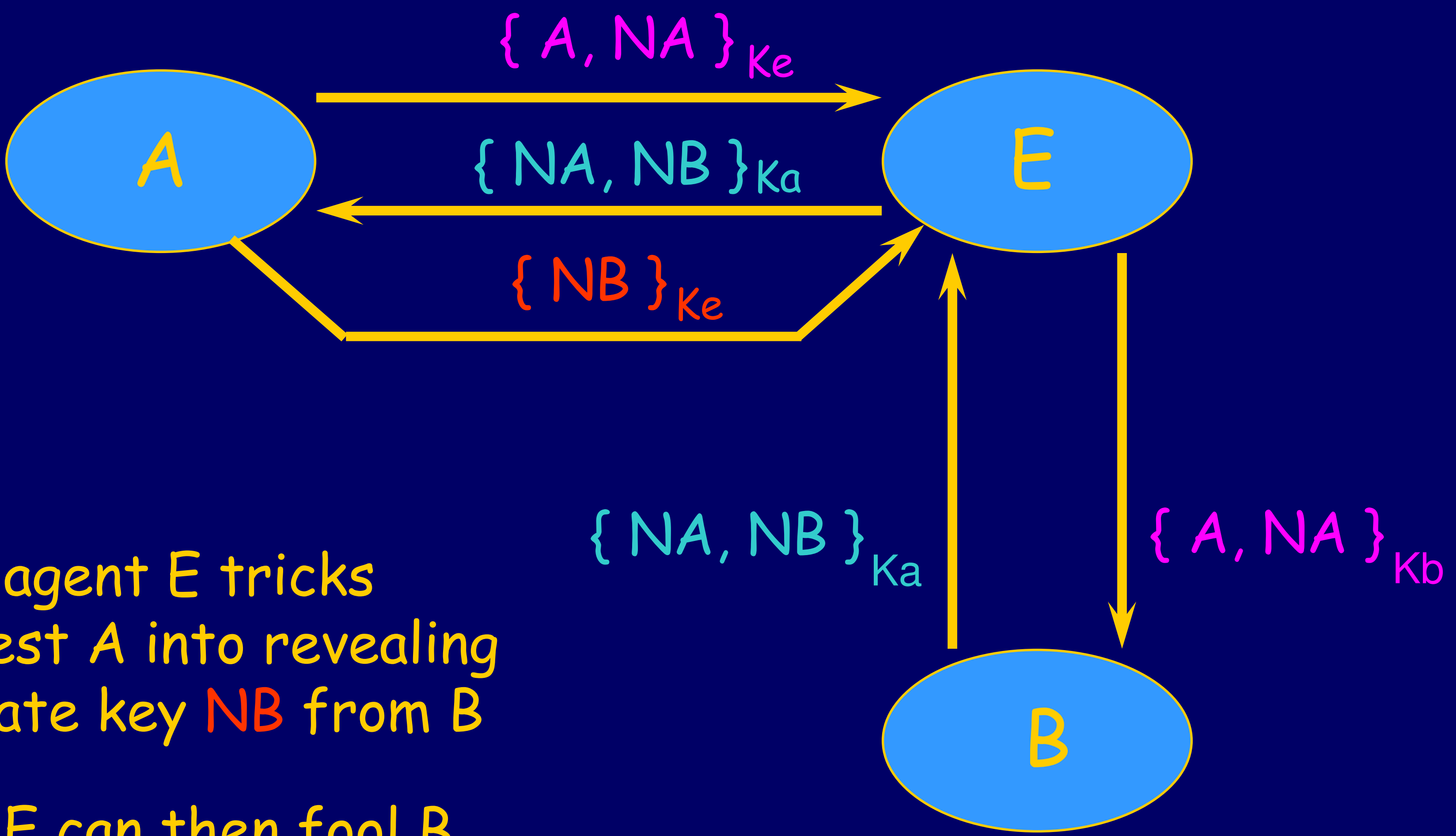
- Famous simple example
  - Protocol published and known for 10 years
  - Gavin Lowe discovered unintended property while preparing formal analysis using FDR system
- Background: Public-key cryptography
  - Every agent A has
    - Public encryption key Ka
    - Private decryption key Ka^{-1}
  - Main properties
    - Everyone can encrypt message to A
    - Only A can decrypt these messages

Needham-Schroeder Key Exchange



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

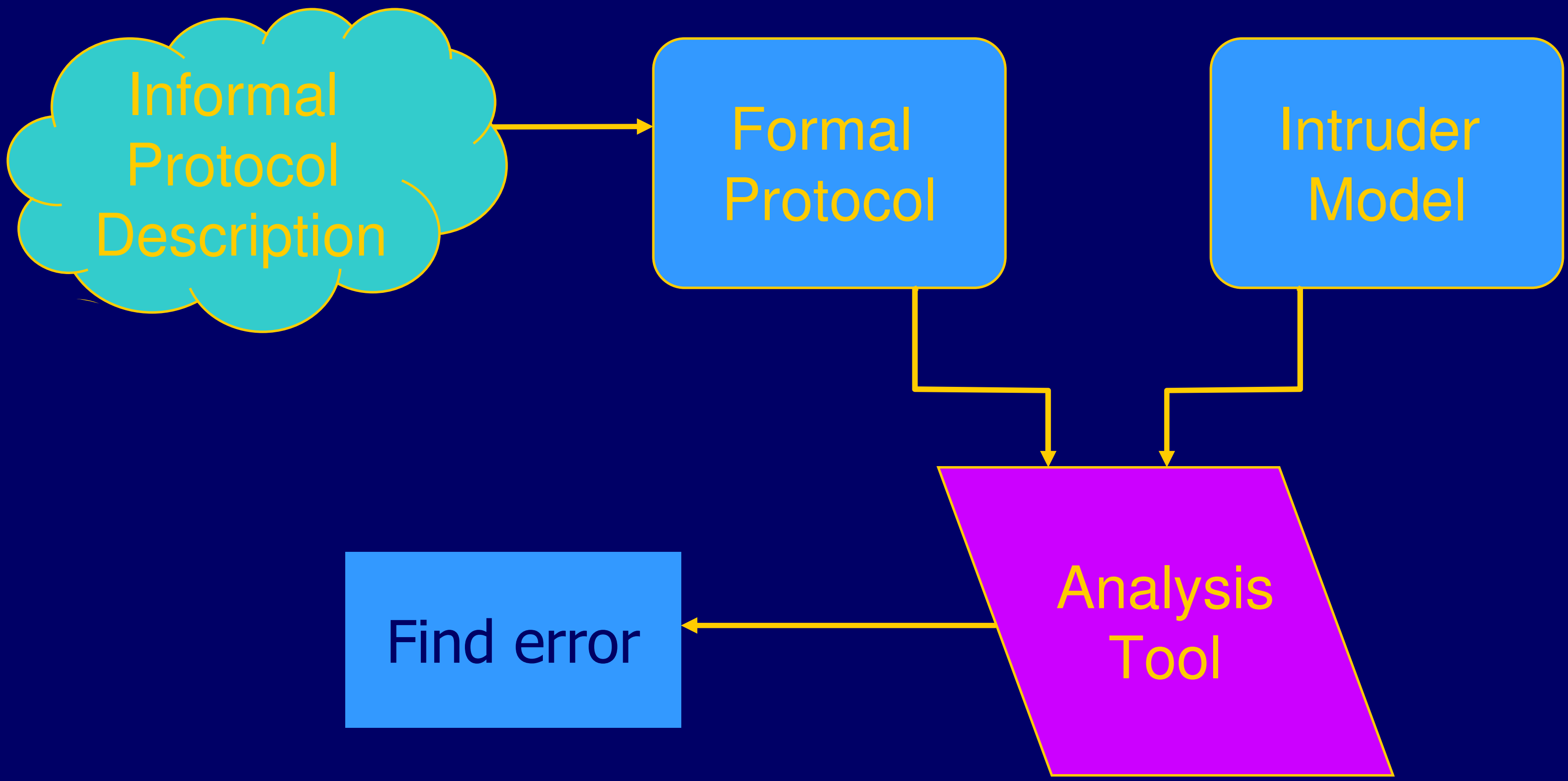
Anomaly in Needham-Schroeder



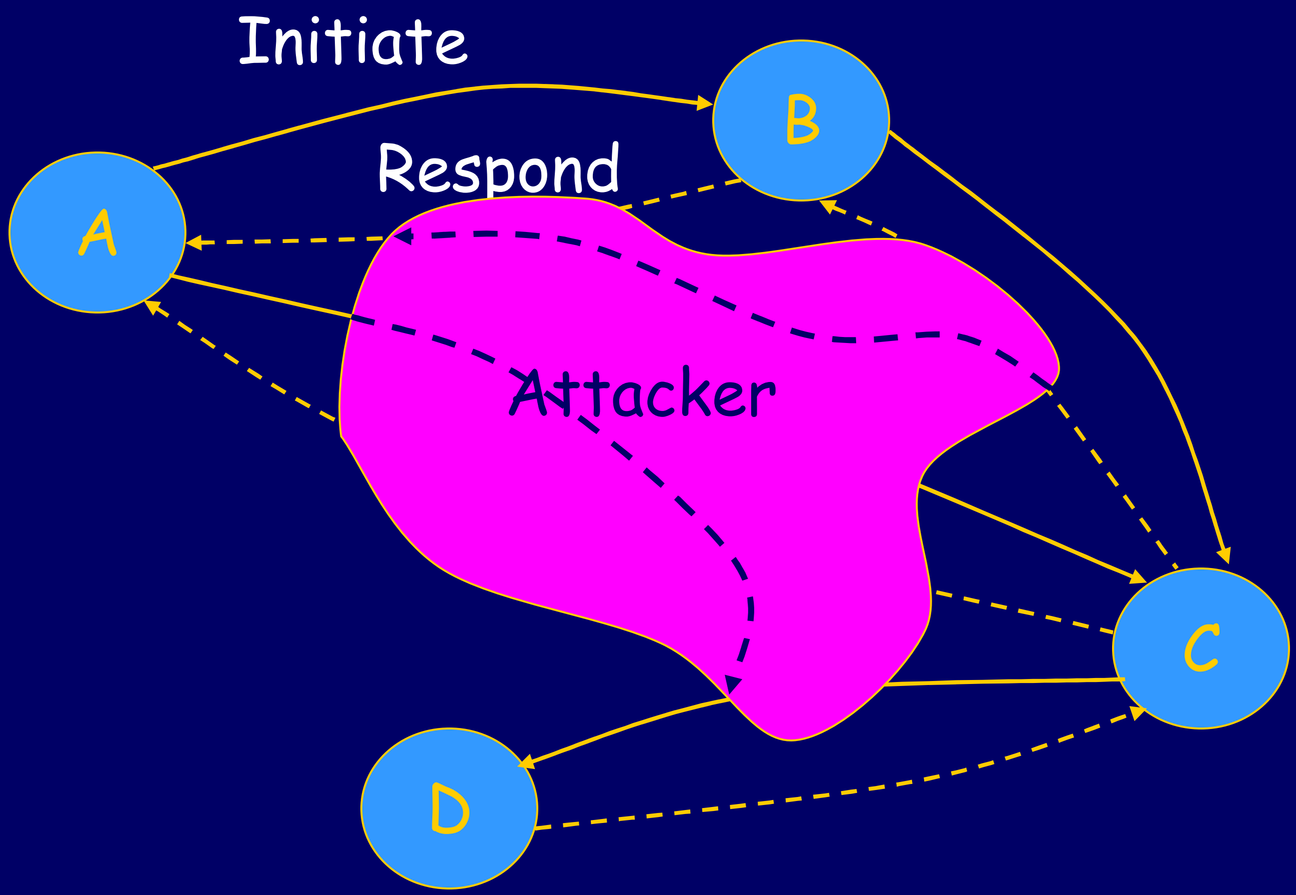
Evil agent E tricks honest A into revealing private key NB from B

Evil E can then fool B

Explicit Intruder Method

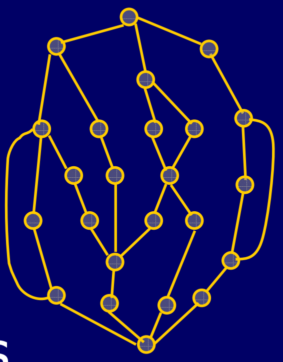


Run of protocol



Correct if no security violation in any run

Automated Finite-State Analysis



- Define finite-state system
  - Bound on number of steps
  - Finite number of participants
  - Nondeterministic adversary with finite options
- Pose correctness condition
  - Can be simple: authentication and secrecy
  - Can be complex: contract signing
- Exhaustive search using "verification" tool
  - Error in finite approximation $\Rightarrow$ Error in protocol
  - No error in finite approximation $\Rightarrow$???

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 r1, r2, r3, …
    - Do not allow unbounded encrypt(encrypt(encrypt(…)))

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
- Research and industrial protocol verification

Applying Murφ to security protocols

- Formulate protocol
- 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 (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;
```

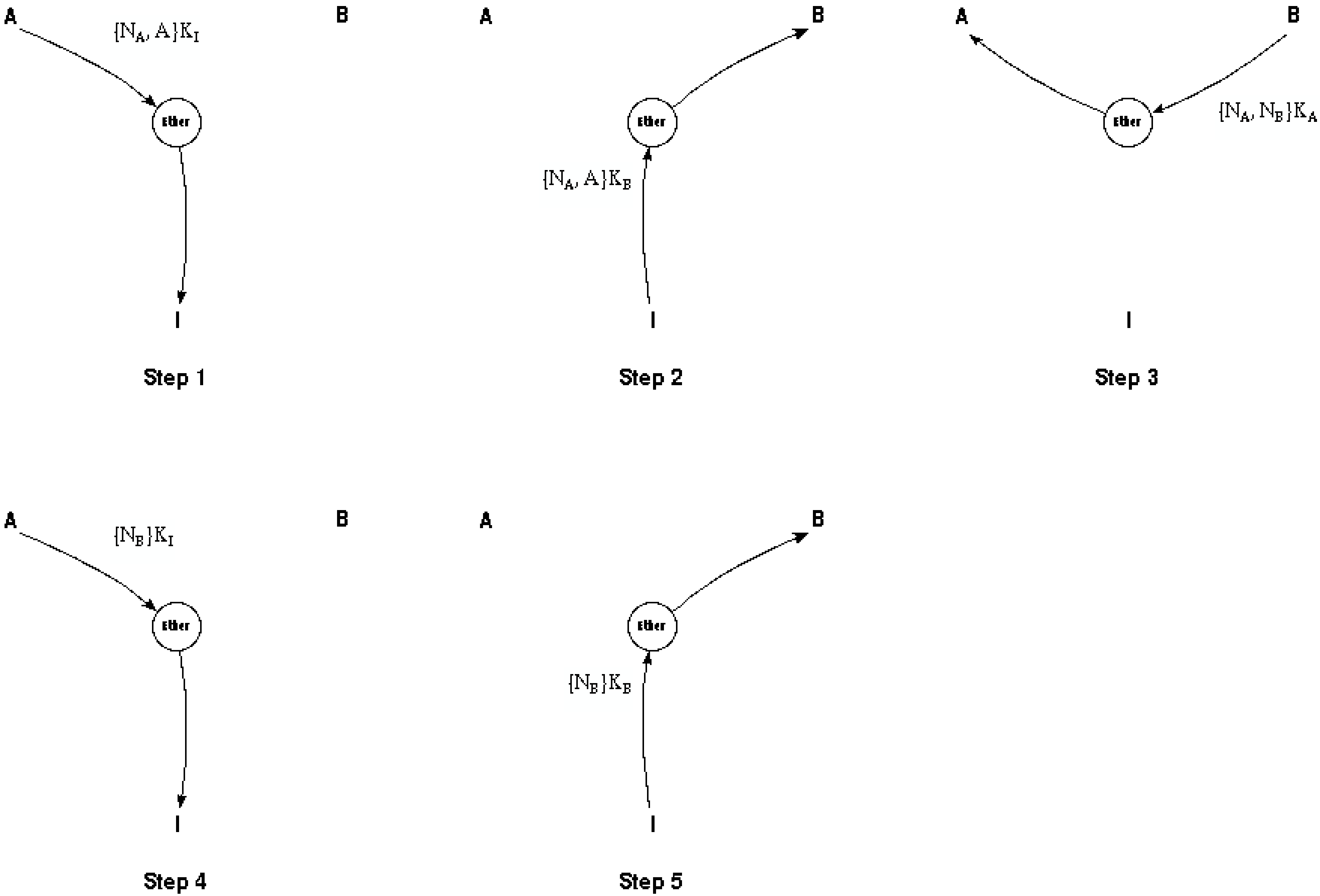
Adversary Model

- 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

Run of Needham-Schroeder

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

number of			size of network	states	time
ini.	res.	int.			
1	1	1	1	1706	3.1s
1	1	1	2	40207	82.2s
2	1	1	1	17277	43.1s
2	2	1	1	514550	5761.1s


A
B
$\{N_A, A\}K_I$
Ether
I
Step 1
A
B
$\{N_A, A\}K_B$
Ether
I
Step 2
A
B
$\{N_A, N_B\}K_A$
Ether
I
Step 3
A
B
$\{N_B\}K_I$
Ether
I
Step 4
A
B
$\{N_B\}K_B$
Ether
I
Step 5

Limitations

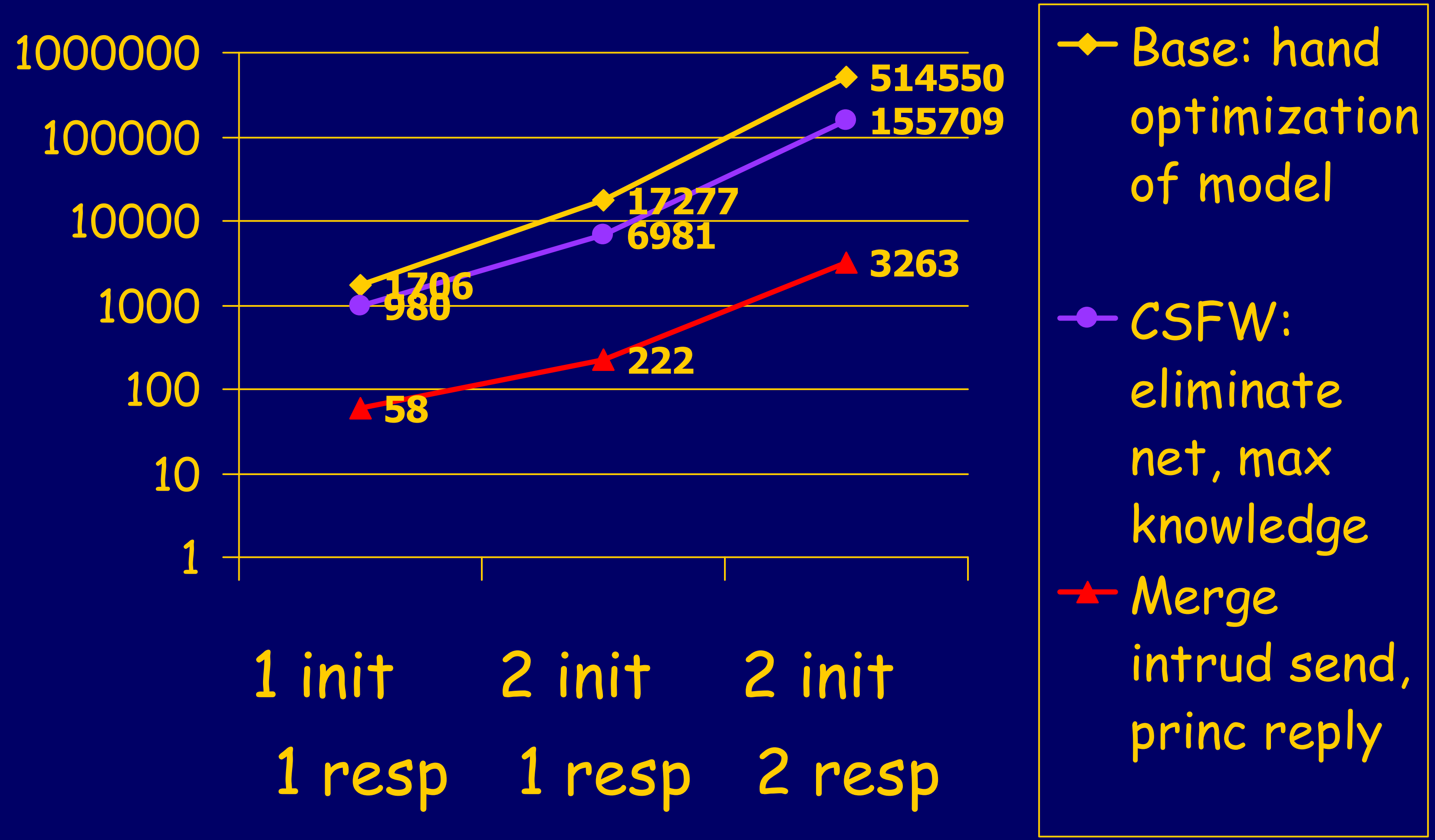
- System size with current methods
  - 2-6 participants
    - Kerberos: 2 clients, 2 servers, 1 KDC, 1 TGS
  - 3-6 steps in protocol
  - May need to optimize adversary
- Adversary model
  - Cannot model randomized attack
  - Do not model adversary running time

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

  … ADD YOUR PROJECT HERE …

State Reduction on N-S Protocol



Plan for ~~this~~ another course

- Protocols
  - Authentication, key establishment, assembling protocols together (TLS ?), fairness exchange, …
- Tools
  - Finite-state and probabilistic model checking, constraint-solving, process calculus, temporal logic, proof systems, game theory, polynomial time …
- Projects (You do this later on your own!)
  - Choose a protocol or other security mechanism
  - Choose a tool or method and carry out analysis
  - Hard part: formulating security requirements

Reference Material (CS259 web site)

- Protocols
  - Clarke-Jacob survey
  - Use Google; learn to read an RFC
- Tools
  - Murphi
    - Finite-state tool developed by David Dill's group at Stanford
  - PRISM
    - Probabilistic model checker, University of Birmingham
  - MOCHA
    - Alur and Henzinger; now consortium
  - Constraint solver using prolog
    - Shmatikov and Millen
  - Isabelle
    - Theorem prover developed by Larry Paulson in Cambridge, UK
    - A number of case studies available on line

Plan for these 4 lectures

- Introduction
  - Simple example, finite-state analysis
- Protocol examples
  - SSL, 802.11i, Kerberos (PKINIT), IKEv2, …
- Security Proofs
  - Symbolic model
    - Paulson's method
    - Protocol composition logic (PCL)
  - Cryptographic soundness
    - Computational model for PCL: challenges, accomplishments

SSL / TLS Case Study

John Mitchell

Stanford

Reference: http://www.stanford.edu/class/cs259/

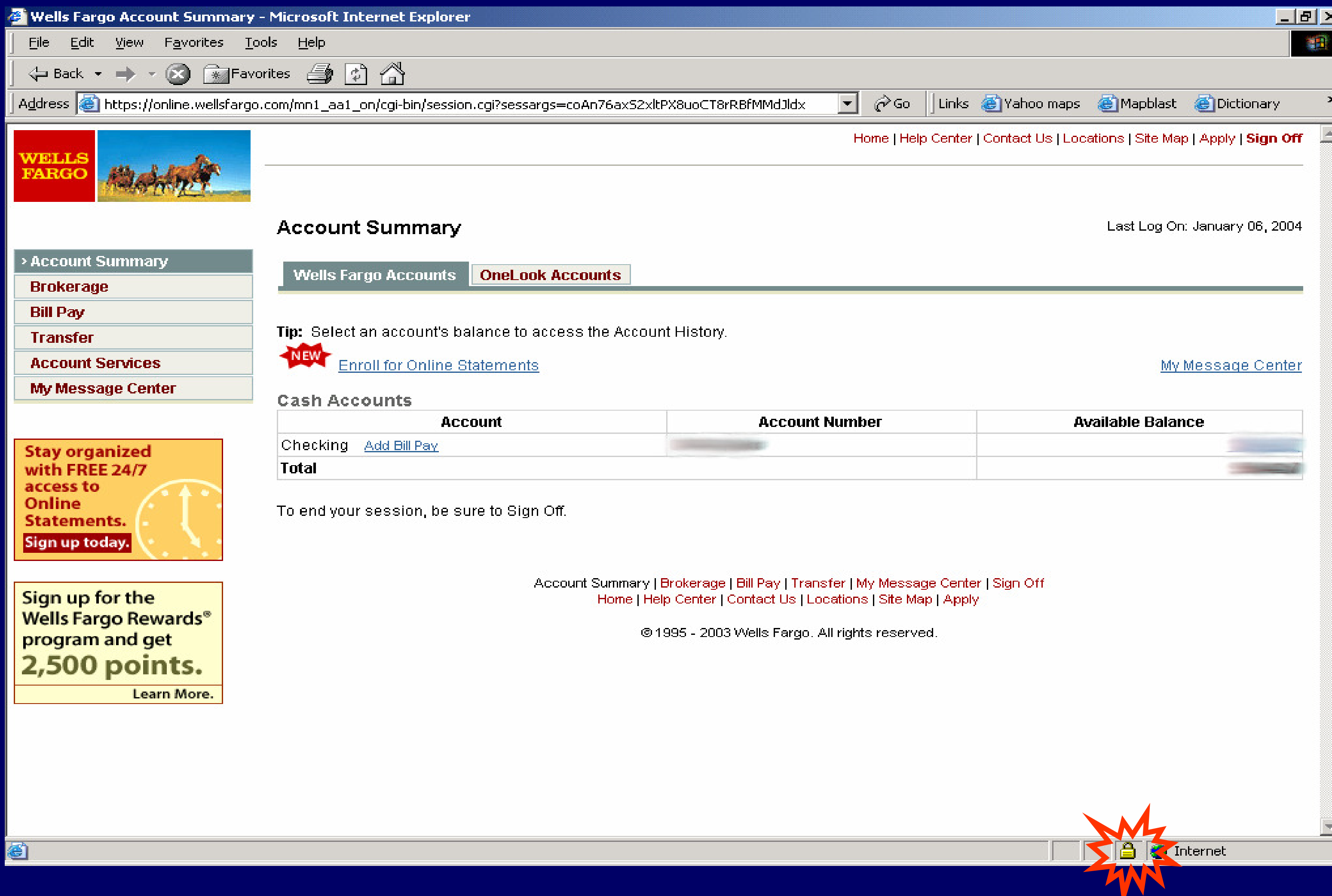
Overview

- Introduction to the SSL / TLS protocol
  - Widely deployed, “real-world” security protocol
- Protocol analysis case study
  - Start with the RFC describing the protocol
  - Create an abstract model and code it up in Murφ
  - Specify security properties
  - Run Murφ to check whether security properties are satisfied

What is SSL / TLS?

- Transport Layer Security protocol, ver 1.0
  - De facto standard for Internet security
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
  - In practice, used to protect information transmitted between browsers and Web servers
- Based on Secure Sockets Layers protocol, ver 3.0
  - Same protocol design, different algorithms
- Deployed in nearly every web browser

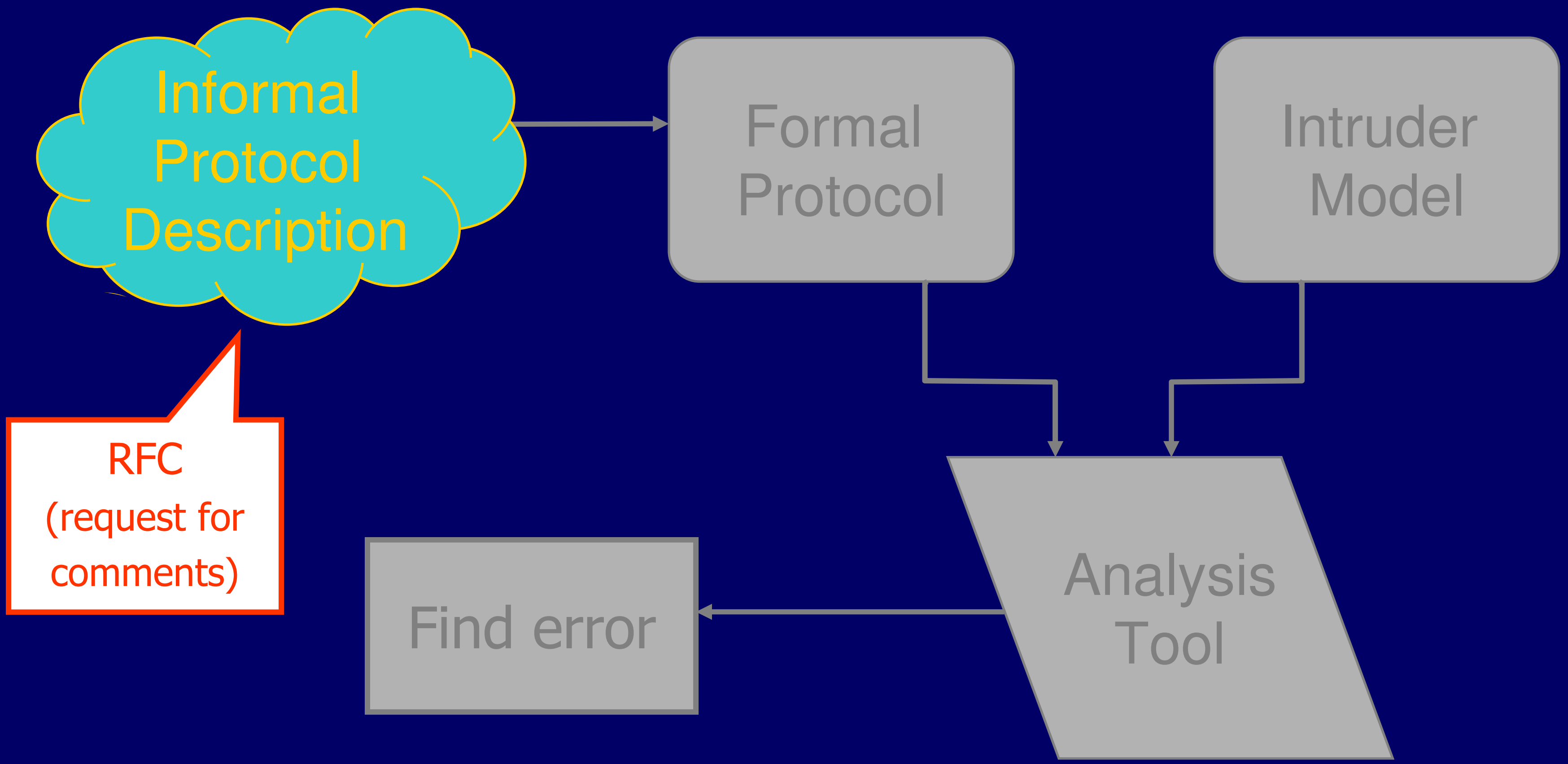
SSL / TLS in the Real World



History of the Protocol

- SSL 1.0
  - Internal Netscape design, early 1994?
  - Lost in the mists of time
- SSL 2.0
  - Published by Netscape, November 1994
  - Badly broken
- SSL 3.0
  - Designed by Netscape and Paul Kocher, November 1996
- TLS 1.0
  - Internet standard based on SSL 3.0, January 1999
  - <u>Not</u> interoperable with SSL 3.0

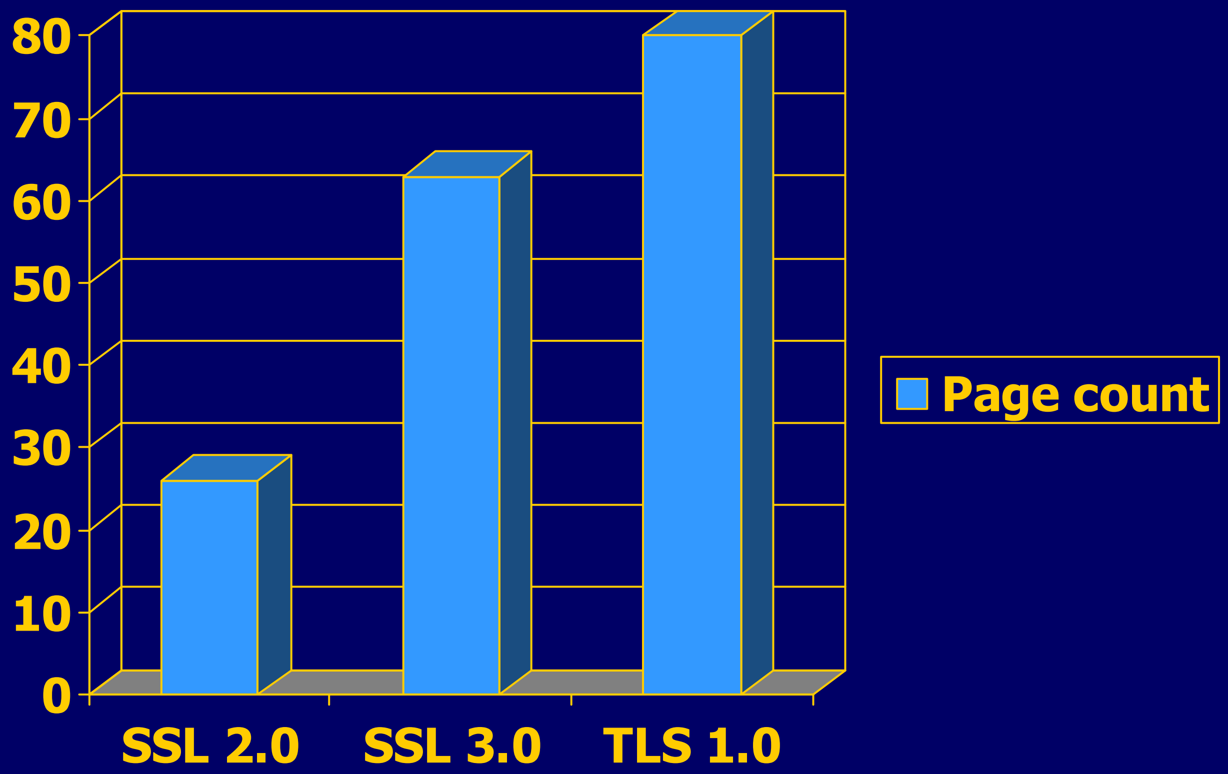
Let's Get Going…



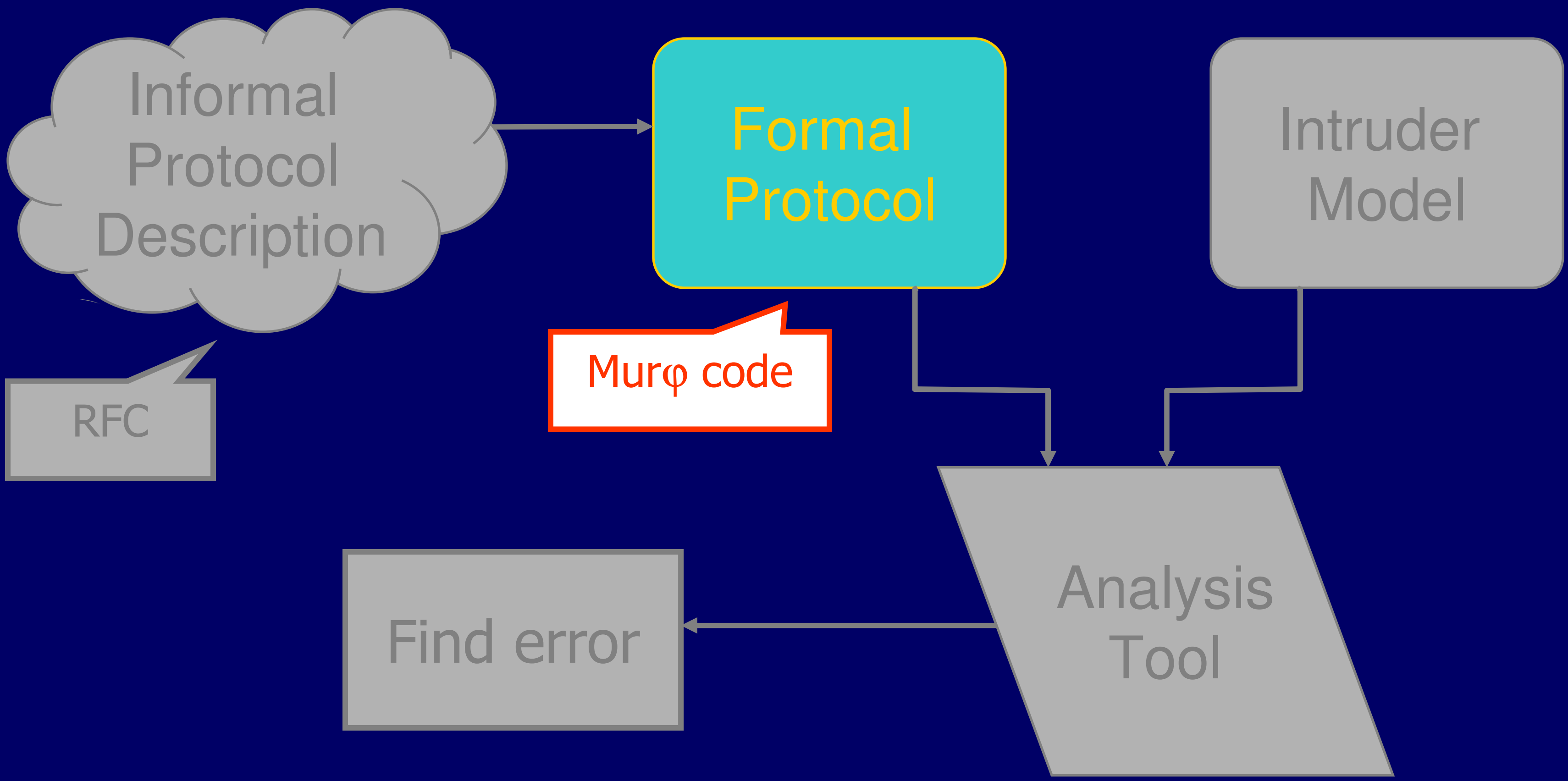
Request for Comments

- Network protocols are usually disseminated in the form of an RFC
- TLS version 1.0 is described in RFC 2246
- Intended to be a self-contained definition
  - Describes the protocol in sufficient detail for readers who will be implementing it and those who will be doing protocol analysis (that's <u>you</u>!)
  - Mixture of informal prose and pseudo-code
- Read some RFCs to get a flavor of what protocols look like when they emerge from the committee

Evolution of the SSL/TLS RFC



From RFC to Murφ Model



TLS Basics

- TLS consists of two protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret key established in the handshake protocol to protect communication between the client and the server
- We will focus on the handshake protocol

TLS Handshake Protocol

- Two parties: client and server
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations of the protocol
- Authenticate client and server (optional)
  - Use digital certificates to learn each other's public keys and verify each other's identity
- Use public keys to establish a shared secret

Handshake Protocol

ClientHello $C \rightarrow S$ $C, Ver_C, Suite_C, N_C$

ServerHello $S \rightarrow C$ $Ver_S, Suite_S, N_S, sign_{CA}\{ S, K_S \}$

ClientVerify $C \rightarrow S$ $sign_{CA}\{ C, V_C \}$

$\{ Ver_C, Secret_C \}_{K_S}$

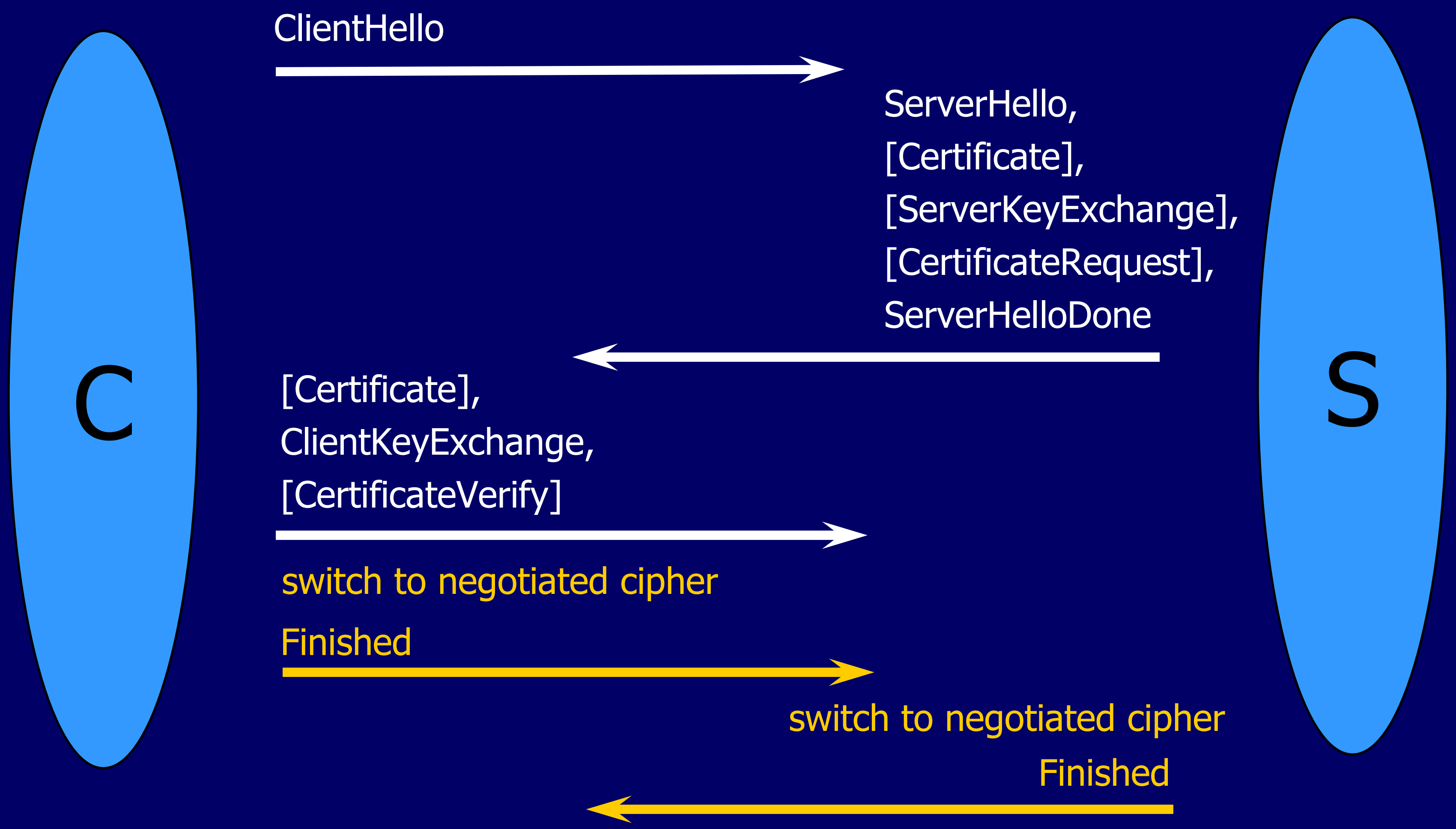
$sign_C \{ Hash(Master(N_C, N_S, Secret_C) + Pad_2 +$
$Hash(Msgs + C + Master(N_C, N_S, Secret_C) + Pad_1)) \}$

(Change to negotiated cipher)

ServerFinished $S \rightarrow C$ $\{ Hash(Master(N_C, N_S, Secret_C) + Pad_2 +$
$Hash(Msgs + S + Master(N_C, N_S, Secret_C) + Pad_1))$
$\}_{Master(N_C, N_S, Secret_C)}$

ClientFinished $C \rightarrow S$ $\{ Hash(Master(N_C, N_S, Secret_C) + Pad_2 +$
$Hash(Msgs + C + Master(N_C, N_S, Secret_C) + Pad_1))$
$\}_{Master(N_C, N_S, Secret_C)}$

Handshake Protocol Structure



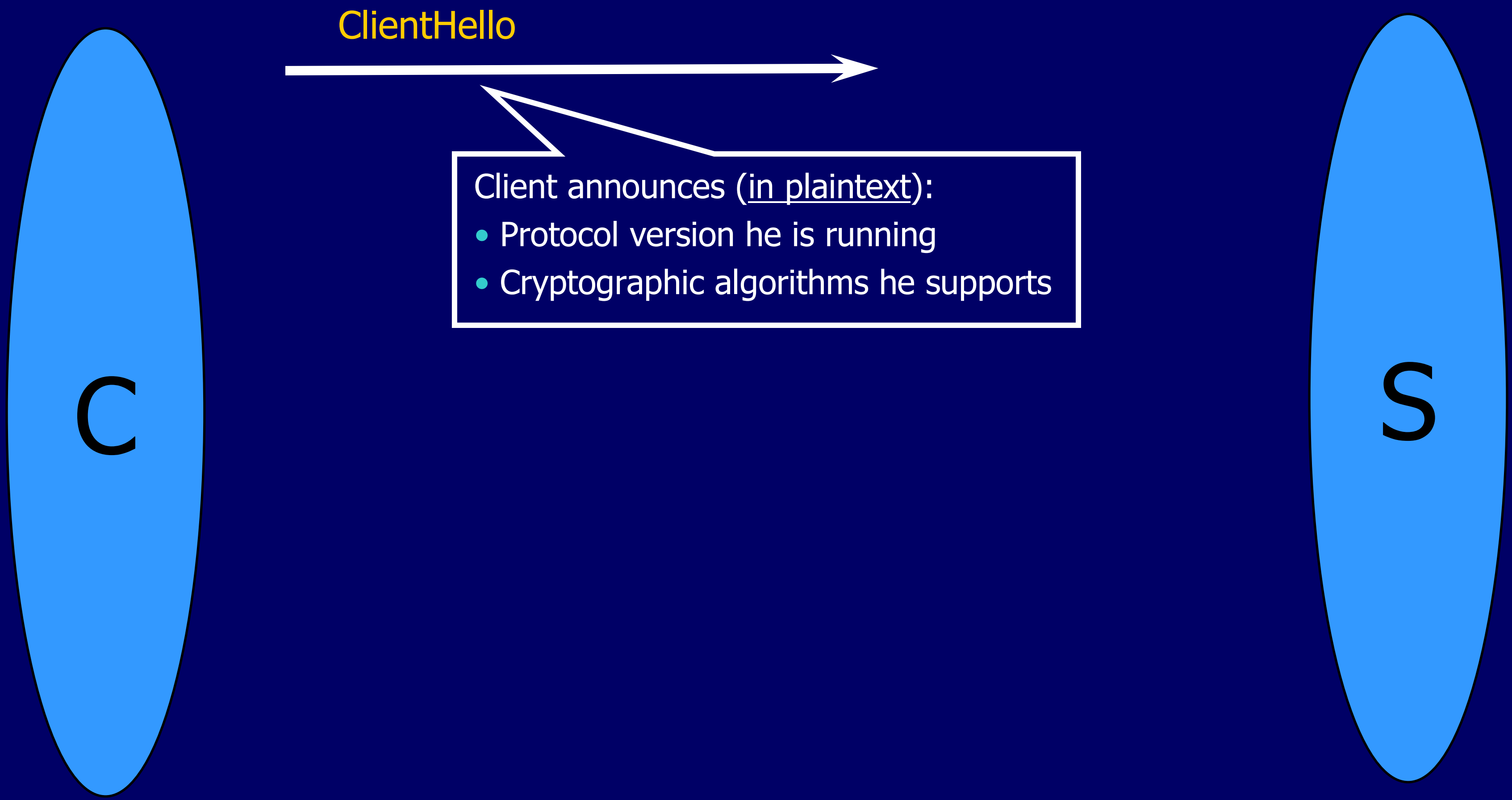
Abbreviated Handshake

- The handshake protocol may be executed in an abbreviated form to resume a previously established session
  - No authentication, key material not exchanged
  - Session resumed from an old state
- For complete analysis, have to model both full and abbreviated handshake protocol
  - This is a common situation: many protocols have several branches, subprotocols for error handling, etc.

Rational Reconstruction

- Begin with simple, intuitive protocol
  - Ignore client authentication
  - Ignore verification messages at the end of the handshake protocol
  - Model only essential parts of messages (e.g., ignore padding)
- Execute the model checker and find a bug
- Add a piece of TLS to fix the bug and repeat
  - Better understand the design of the protocol

Protocol Step by Step: ClientHello



ClientHello (RFC)

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

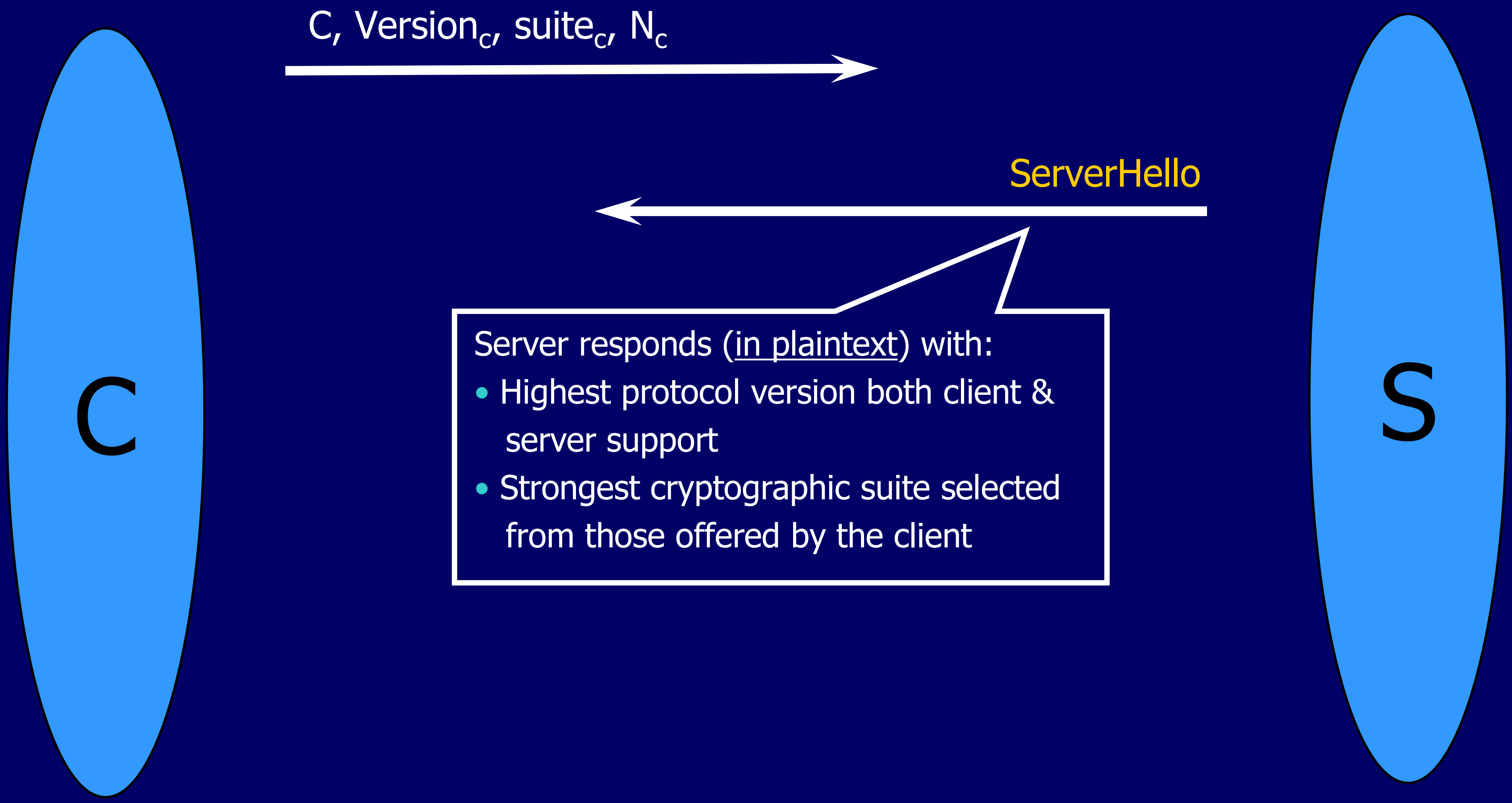
Session id (if the client wants to resume an old session)

Cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)

ClientHello (Murφ)

```
ruleset i: ClientId do
  ruleset j: ServerId do
    rule "Client sends ClientHello to server (new session)"
      cli[i].state = M_SLEEP &
      cli[i].resumeSession  = false
    ==>
    var
      outM: Message;  -- outgoing message
    begin
      outM.source := i;
      outM.dest := j;
      outM.session := 0;
      outM.mType := M_CLIENT_HELLO;
      outM.version := cli[i].version;
      outM.suite := cli[i].suite;
      outM.random := freshNonce();
      multisetadd (outM, cliNet);
      cli[i].state  := M_SERVER_HELLO;
    end;
  end;
end;
```

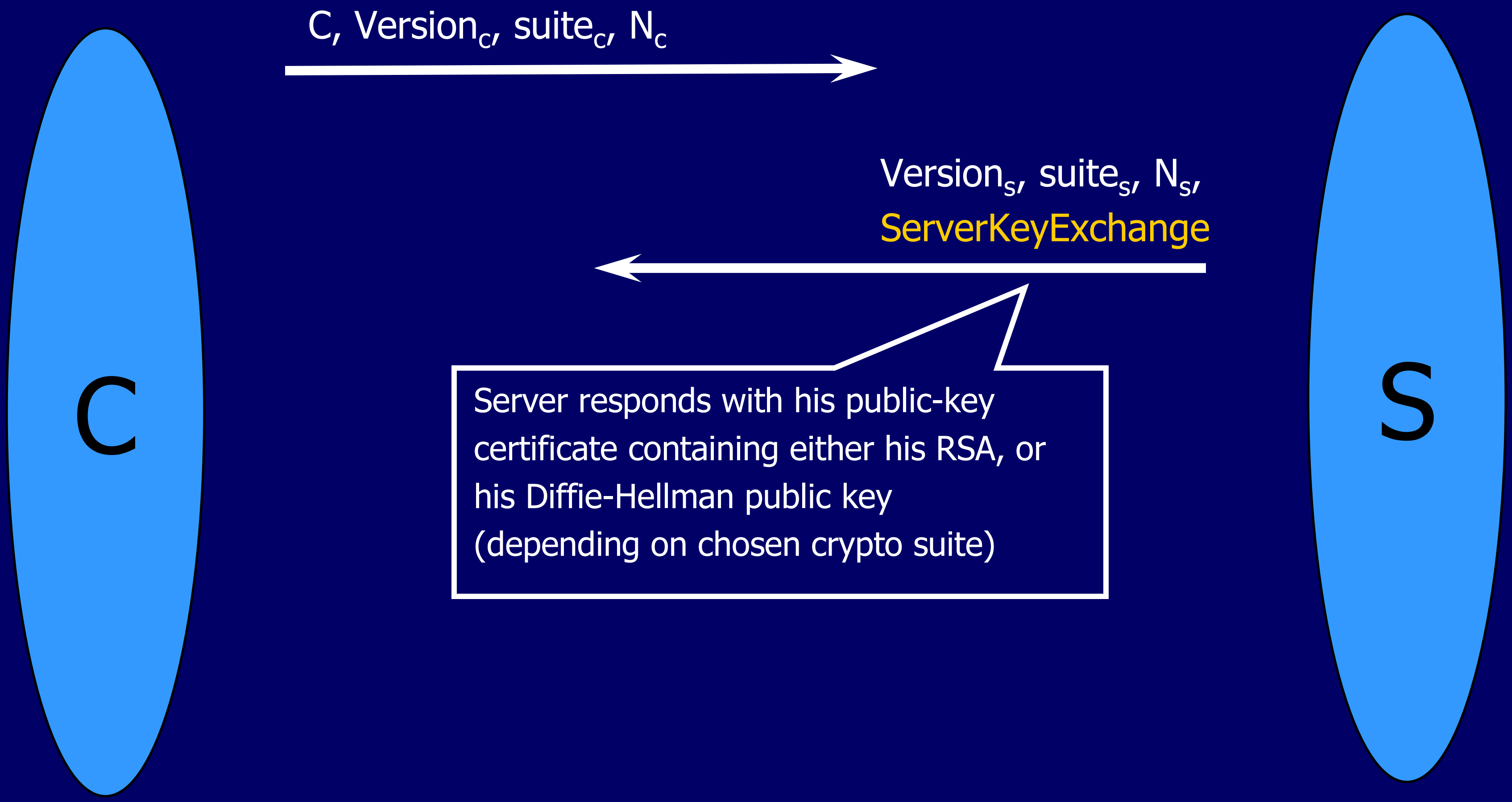
ServerHello



ServerHello (Murφ)

```
ruleset i: ServerId do
  choose l: serNet do
   rule "Server receives ServerHello (new session)"
     ser[i].clients[0].state = M_CLIENT_HELLO &
     serNet[l].dest = i &
     serNet[l].session = 0
    ==>
    var
      inM:   Message;   -- incoming message
      outM: Message;   -- outgoing message
    begin
      inM := serNet[l];  -- receive message
      if inM.mType = M_CLIENT_HELLO then
        outM.source := i;
        outM.dest := inM.source;
        outM.session := freshSessionId();
        outM.mType := M_SERVER_HELLO;
        outM.version := ser[i].version;
        outM.suite := ser[i].suite;
        outM.random := freshNonce();
        multisetadd (outM, serNet);
        ser[i].state  := M_SERVER_SEND_KEY;
    end;  end;  end;
```

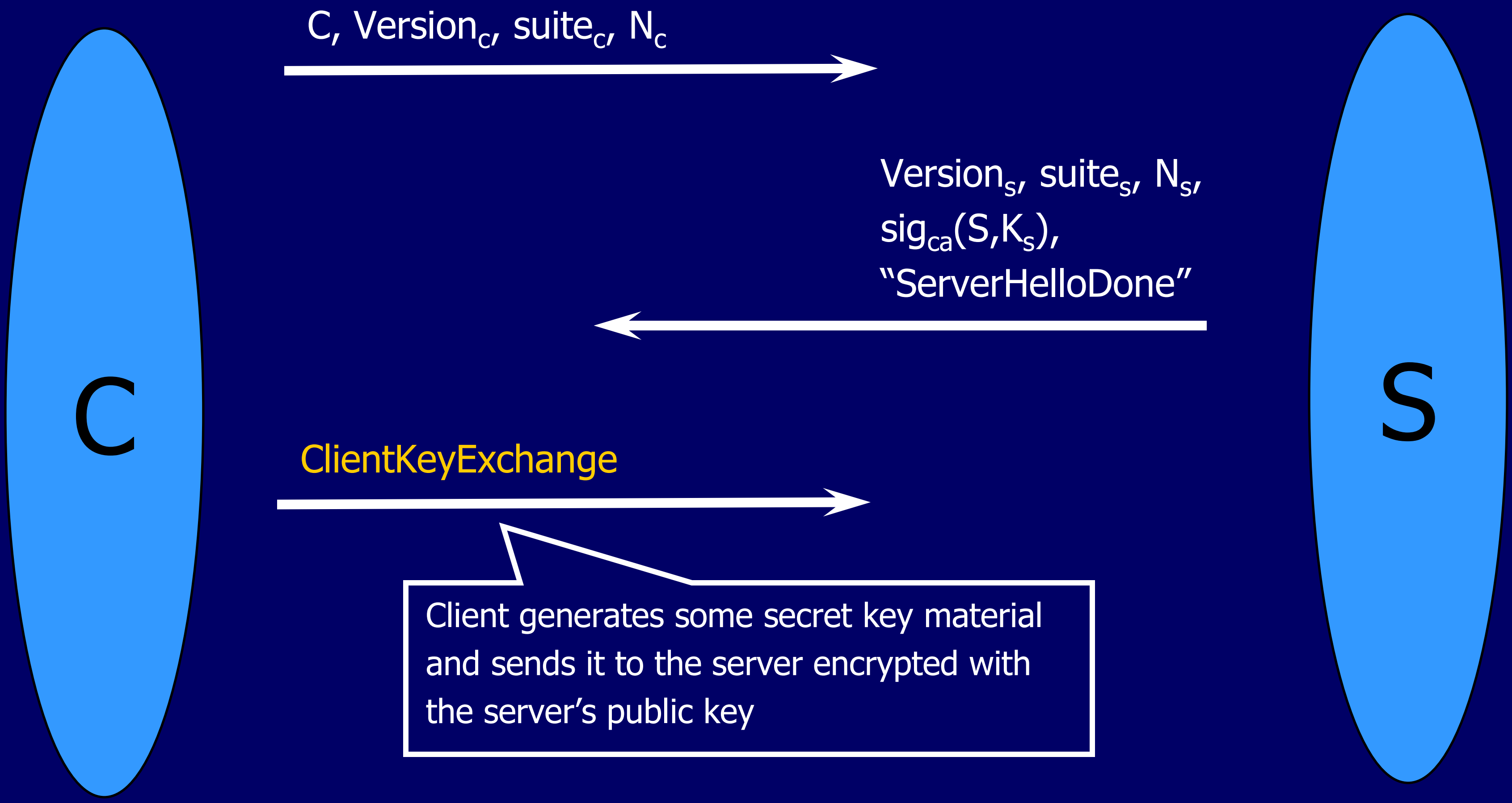
ServerKeyExchange



"Abstract" Cryptography

- We will use abstract data types to model cryptographic operations
  - Assumes that cryptography is perfect
  - No details of the actual cryptographic schemes
  - Ignores bit length of keys, random numbers, etc.
- Simple notation for encryption, signatures, hashes
  - $\{M\}_k$ is message M encrypted with key k
  - $sig_k(M)$ is message M digitally signed with key k
  - hash(M) for the result of hashing message M with a cryptographically strong hash function

ClientKeyExchange

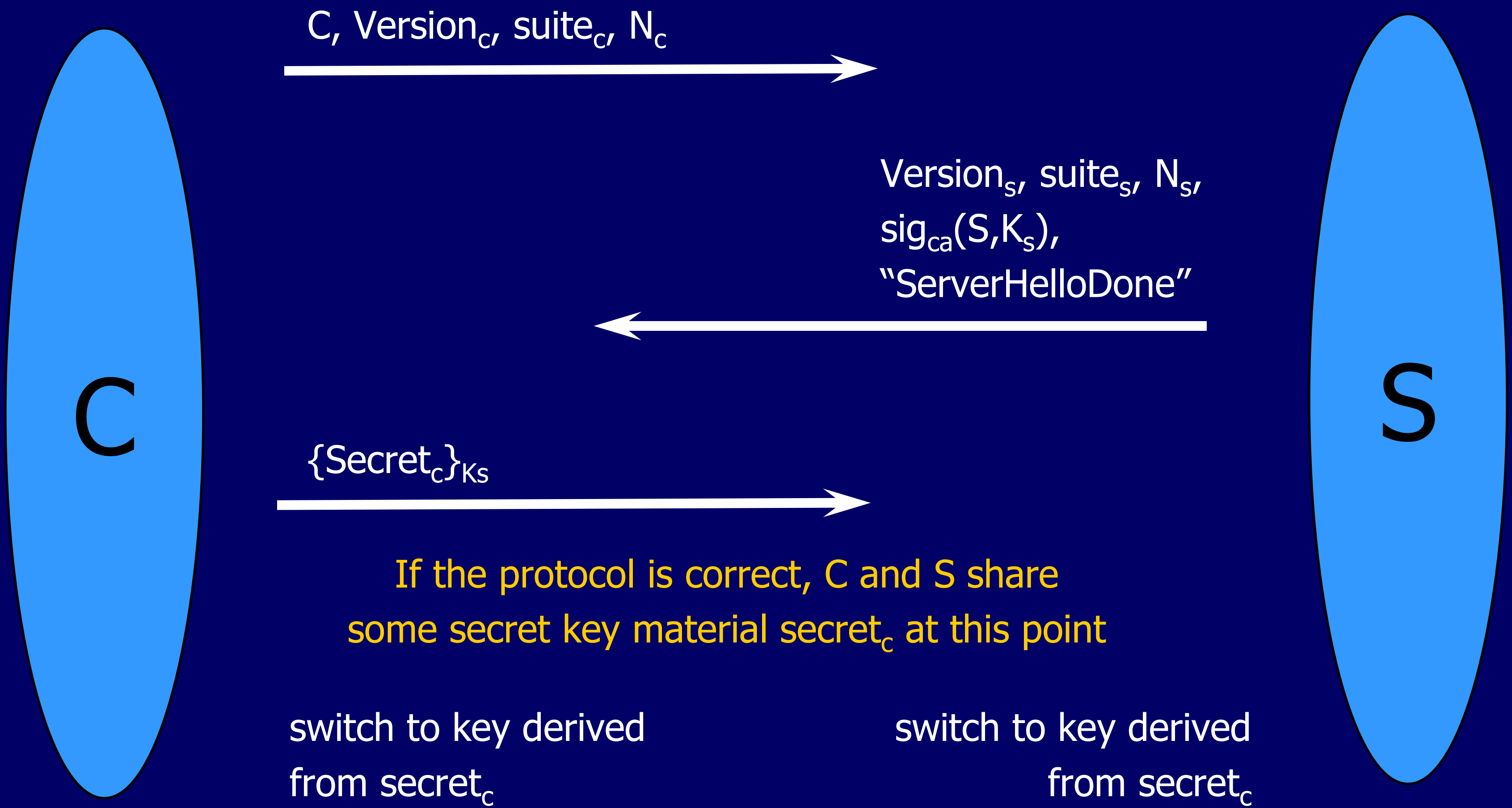


ClientKeyExchange (RFC)

Let's model this as $\{Secret_c\}_{Ks}$

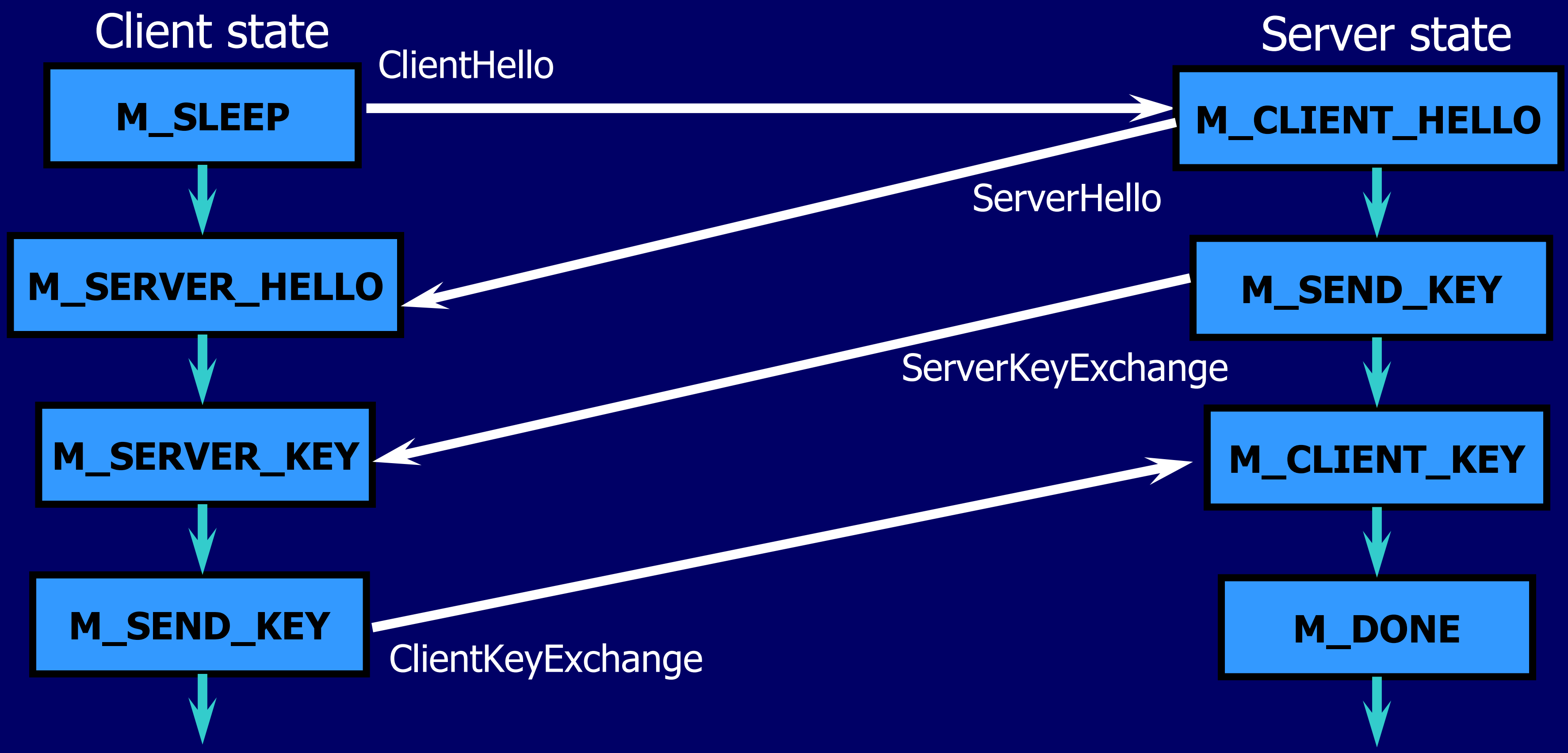
```
struct {
   select (KeyExchangeAlgorithm) {
      case rsa: EncryptedPreMasterSecret;
      case diffie_hellman: ClientDiffieHellmanPublic;
   } exchange_keys
} ClientKeyExchange

struct {
   ProtocolVersion client_version;
   opaque random[46];
} PreMasterSecret
```

"Core" SSL

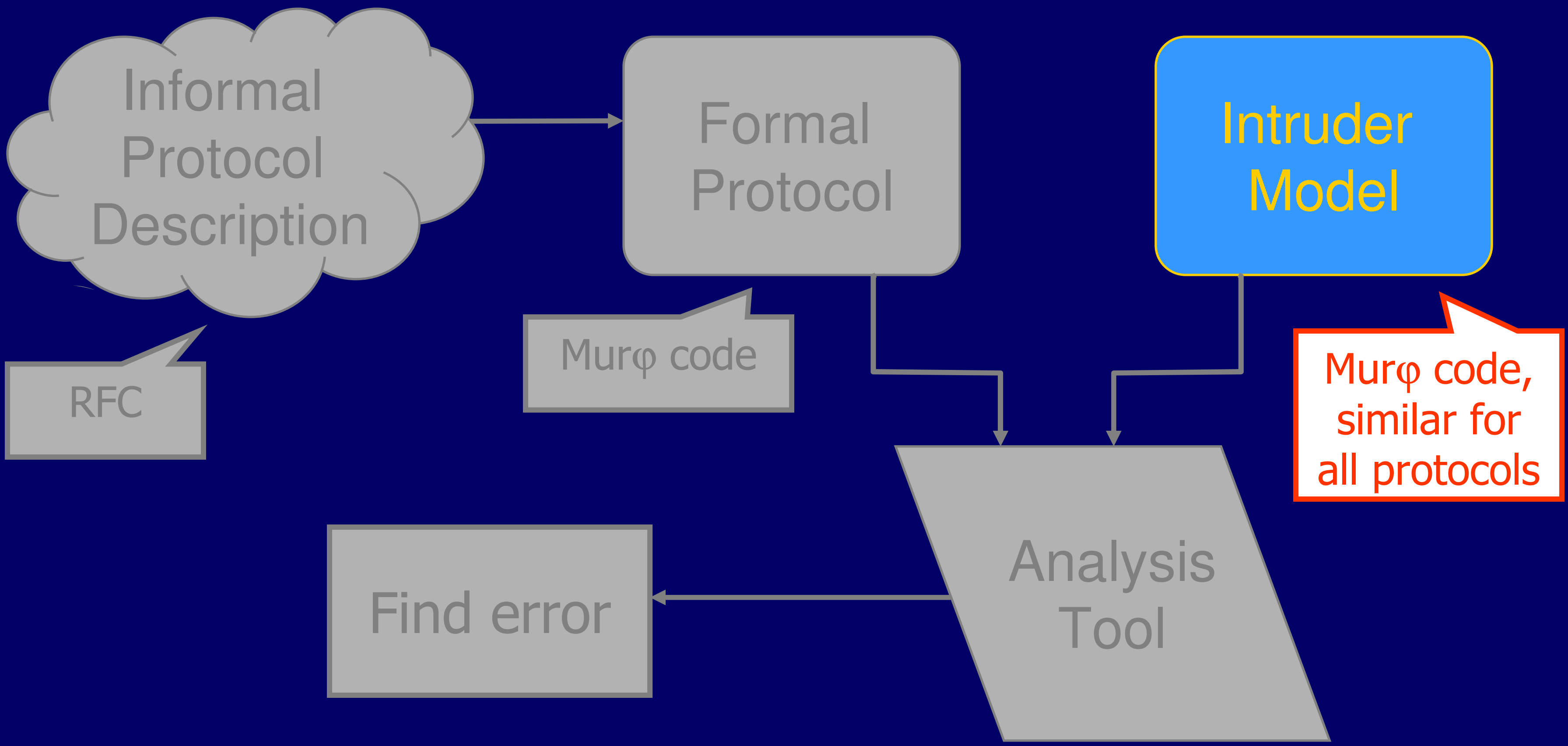

C, $Version_c$, $suite_c$, N_c
$Version_s$, $suite_s$, N_s,
$sig_{ca}(S,K_s)$,
"ServerHelloDone"
C
S
$\{Secret_c\}_{Ks}$
If the protocol is correct, C and S share
some secret key material $secret_c$ at this point
switch to key derived
from $secret_c$
switch to key derived
from $secret_c$

Participants as Finite-State Machines

Murφ rules define a finite-state machine for each protocol participant



Intruder Model



Intruder Can Intercept

- Store a message from the network in the data structure modeling intruder's "knowledge"

```
ruleset i: IntruderId do
  choose l: cliNet do
    rule "Intruder intercepts client's message"
      cliNet[l].fromIntruder = false
    ==>
    begin
      alias msg: cliNet[l] do   -- message from the net
      ...
      alias known: int[i].messages do
         if multisetcount(m: known,
                          msgEqual(known[m], msg)) = 0 then
            multisetadd(msg, known);
         end;
      end;
    end;
```

Intruder Can Decrypt if Knows Key

- If the key is stored in the data structure modeling intruder's "knowledge", then read message

```
ruleset i: IntruderId do
  choose l: cliNet do
    rule "Intruder intercepts client's message"
      cliNet[l].fromIntruder = false
    ==>
    begin
      alias msg: cliNet[l] do   -- message from the net
      …
      if msg.mType = M_CLIENT_KEY_EXCHANGE then
           if keyEqual(msg.encKey, int[i].publicKey.key) then
              alias sKeys: int[i].secretKeys do
                if multisetcount(s: sKeys,
                   keyEqual(sKeys[s], msg.secretKey)) = 0 then
                   multisetadd(msg.secretKey, sKeys);
                end;
           end;
      end;
```

Intruder Can Create New Messages

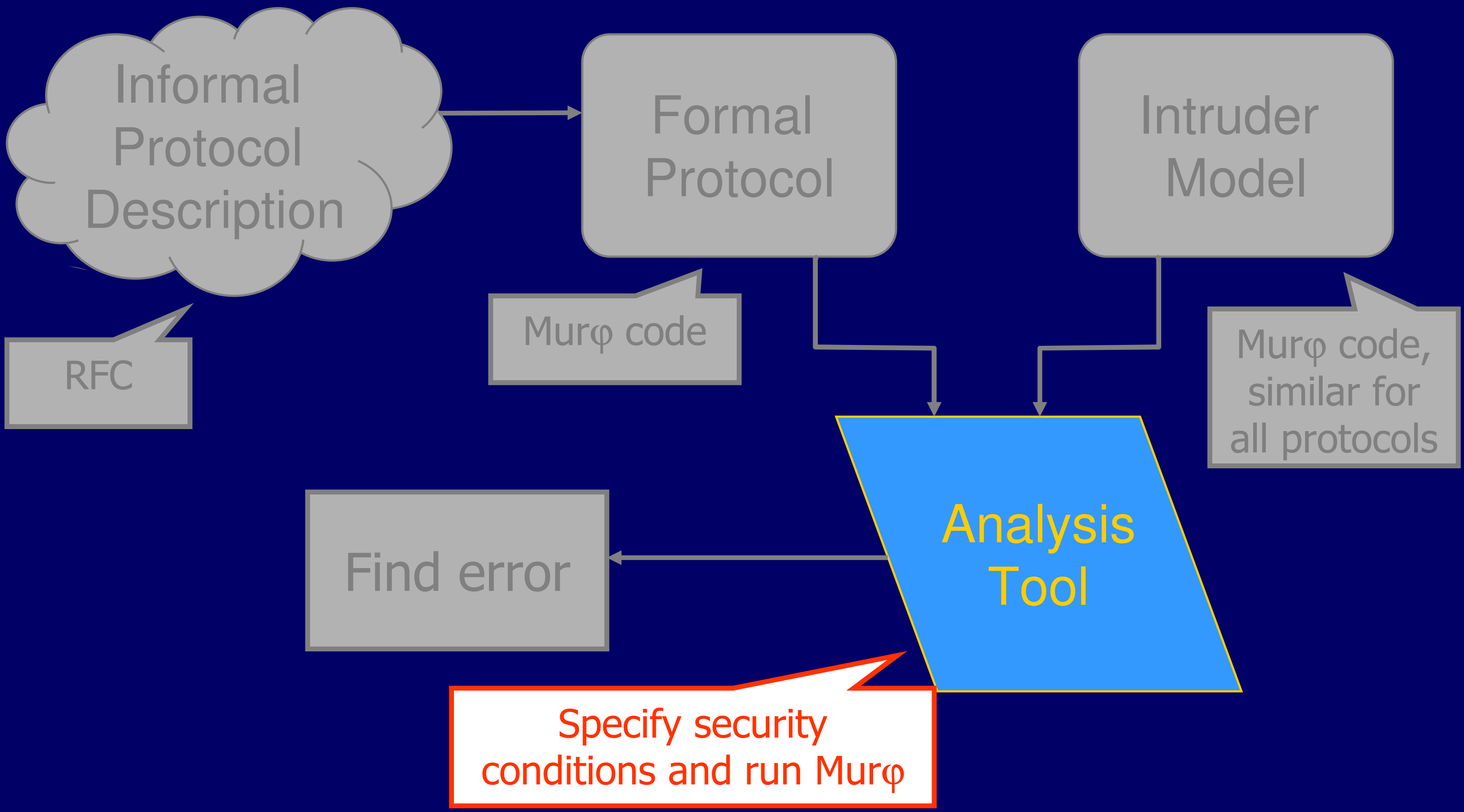
- Assemble pieces stored in the intruder's "knowledge" to form a message of the right format

```
ruleset i: IntruderId do
  ruleset d: ClientId do
    ruleset s: ValidSessionId do
      choose n: int[i].nonces do
      ruleset version: Versions do
      rule "Intruder generates fake ServerHello"
          cli[d].state = M_SERVER_HELLO
        ==>
        var
          outM: Message;  -- outgoing message
        begin
          outM.source := i; outM.dest := d; outM.session := s;
          outM.mType := M_SERVER_HELLO;
          outM.version := version;
          outM.random := int[i].nonces[n];
          multisetadd (outM, cliNet);
        end; end; end; end;
```

Intruder Model and Cryptography

- There is no actual cryptography in our model
  - Messages are marked as “encrypted” or “signed”, and the intruder rules respect these markers
- Our assumption that cryptography is perfect is reflected in the absence of certain intruder rules
  - There is no rule for creating a digital signature with a key that is not known to the intruder
  - There is no rule for reading the contents of a message which is marked as “encrypted” with a certain key, when this key is not known to the intruder
  - There is no rule for reading the contents of a “hashed” message

Running Murφ Analysis



Secrecy

- Intruder should not be able to learn the secret generated by the client

```
ruleset i: ClientId do
  ruleset j: IntruderId do
    rule "Intruder has learned a client's secret"
      cli[i].state = M_DONE &
      multisetcount(s: int[j].secretKeys,
         keyEqual(int[j].secretKeys[s], cli[i].secretKey)) > 0
    ==>
    begin
      error "Intruder has learned a client's secret"
    end;
  end;
end;
```

Shared Secret Consistency

- After the protocol has finished, client and server should agree on their shared secret

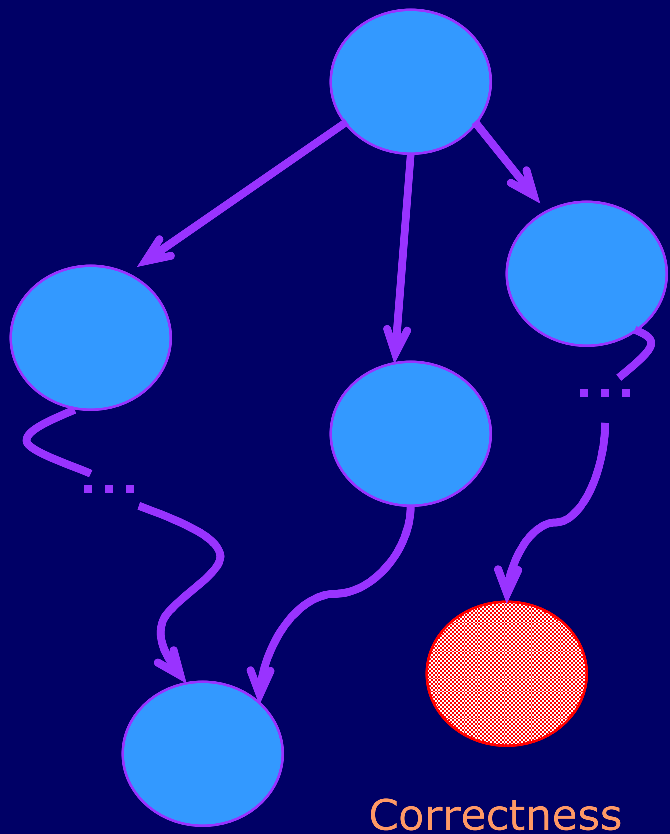
```
ruleset i: ServerId do
  ruleset s: SessionId do
    rule "Server's shared secret is not the same as its client's"
      ismember(ser[i].clients[s].client, ClientId) &
      ser[i].clients[s].state = M_DONE &
      cli[ser[i].clients[s].client].state = M_DONE &
      !keyEqual(cli[ser[i].clients[s].client].secretKey,
                ser[i].clients[s].secretKey)
    ==>
    begin
      error "S's secret is not the same as C's"
    end;
  end;
end;
```

Version and Crypto Suite Consistency

- Client and server should be running the highest version of the protocol they both support

```
ruleset i: ServerId do
  ruleset s: SessionId do
    rule "Server has not learned the client's version or suite correctly"
      !ismember(ser[i].clients[s].client, IntruderId) &
      ser[i].clients[s].state = M_DONE &
      cli[ser[i].clients[s].client].state = M_DONE &
      (ser[i].clients[s].clientVersion != MaxVersion |
       ser[i].clients[s].clientSuite.text != 0)
    ==>
    begin
      error "Server has not learned the client's version or suite correctly"
    end;
  end;
end;
```

Finite-State Verification



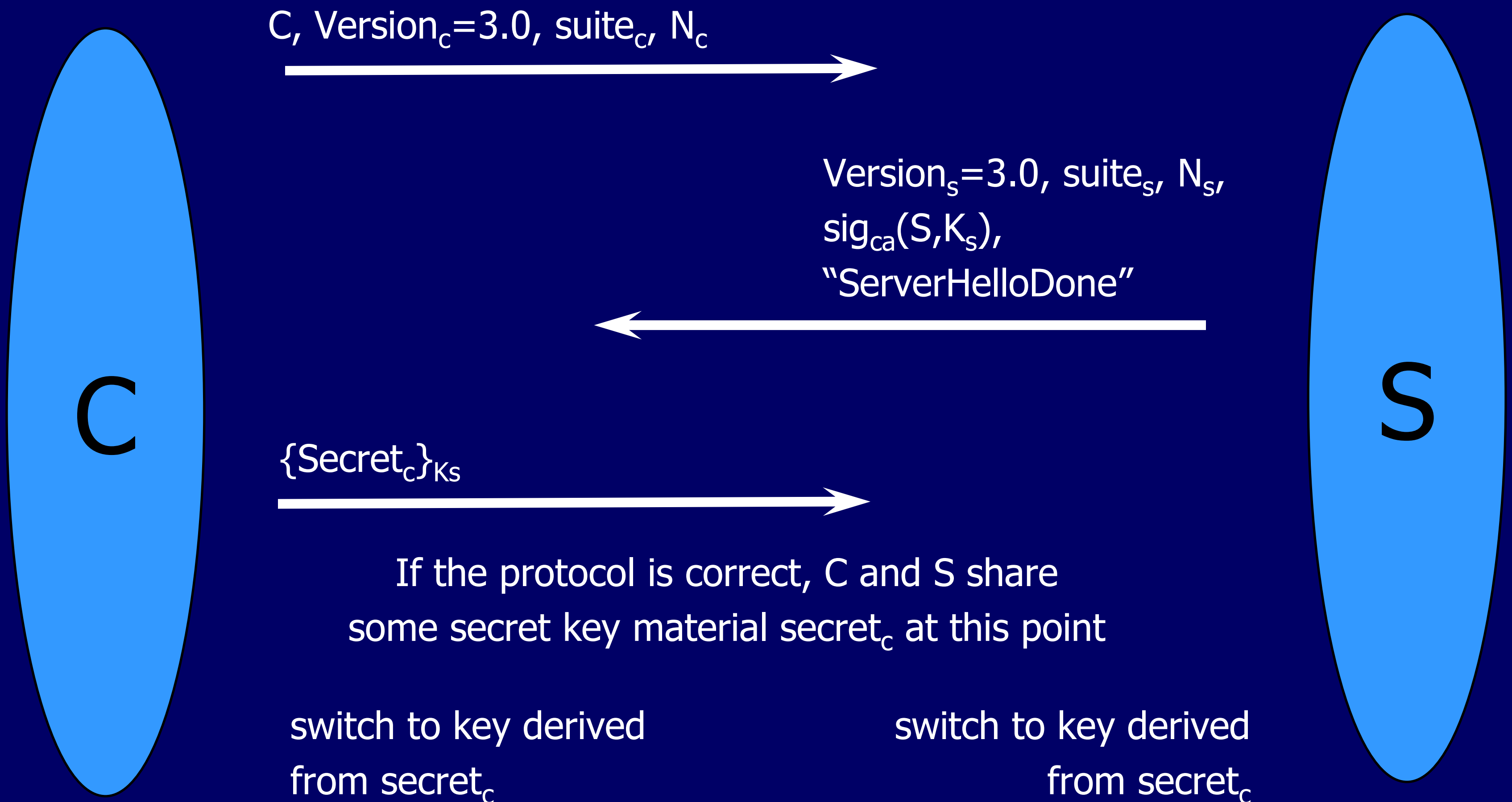
Correctness condition violated

- Murφ rules for protocol participants and the intruder define a nondeterministic state transition graph
- Murφ will exhaustively enumerate all graph nodes
- Murφ will verify whether specified security conditions hold in every reachable node
- If not, the path to the violating node will describe the attack

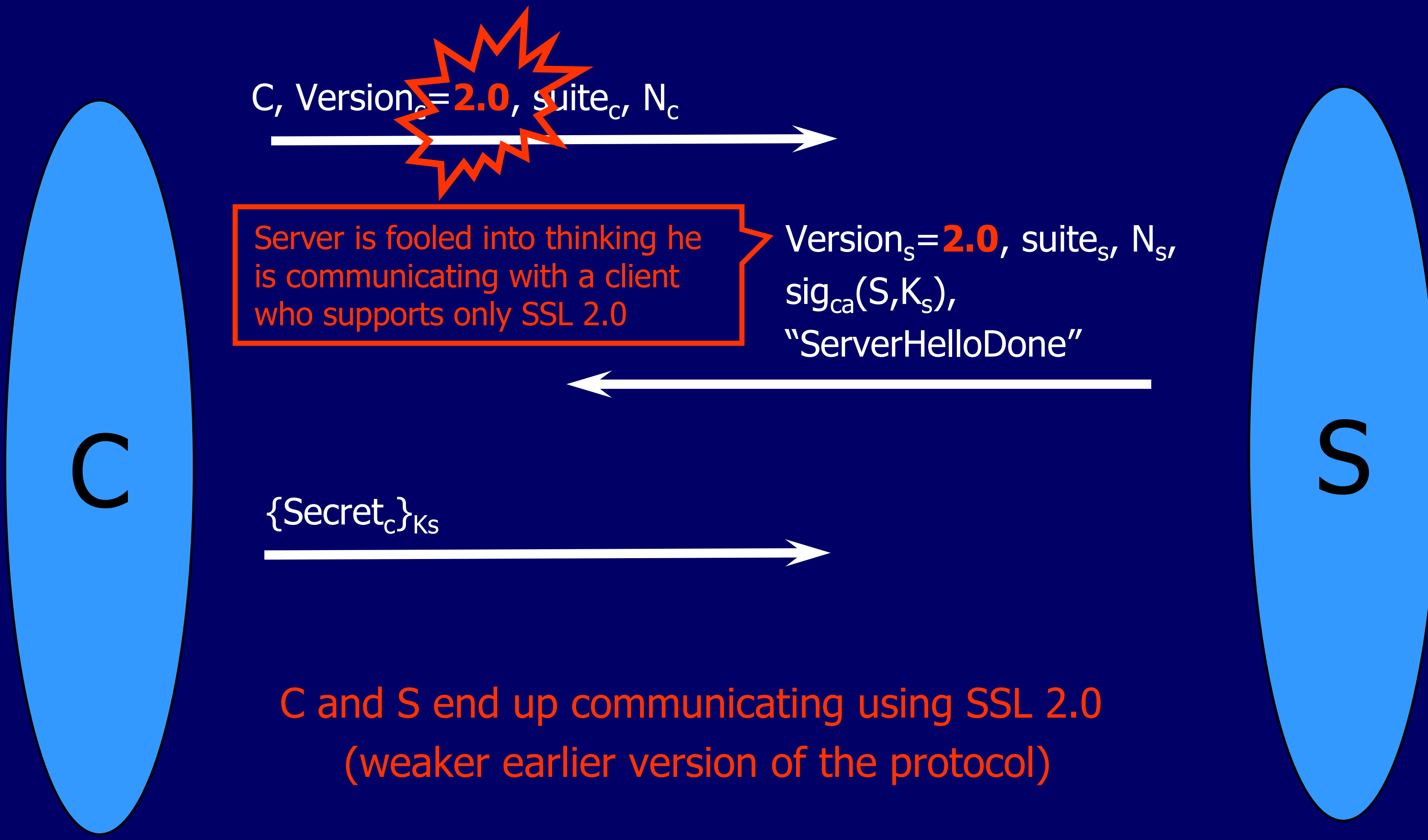
When Does Murφ Find a Violation?

- Bad abstraction
  - Removed too much detail from the protocol when constructing the abstract model
  - Add the piece that fixes the bug and repeat
  - This is part of the rational reconstruction process
- Genuine attack
  - Yay! Hooray!
  - Attacks found by formal analysis are usually quite strong: independent of specific cryptographic schemes, OS implementation, etc.
  - Test an implementation of the protocol, if available

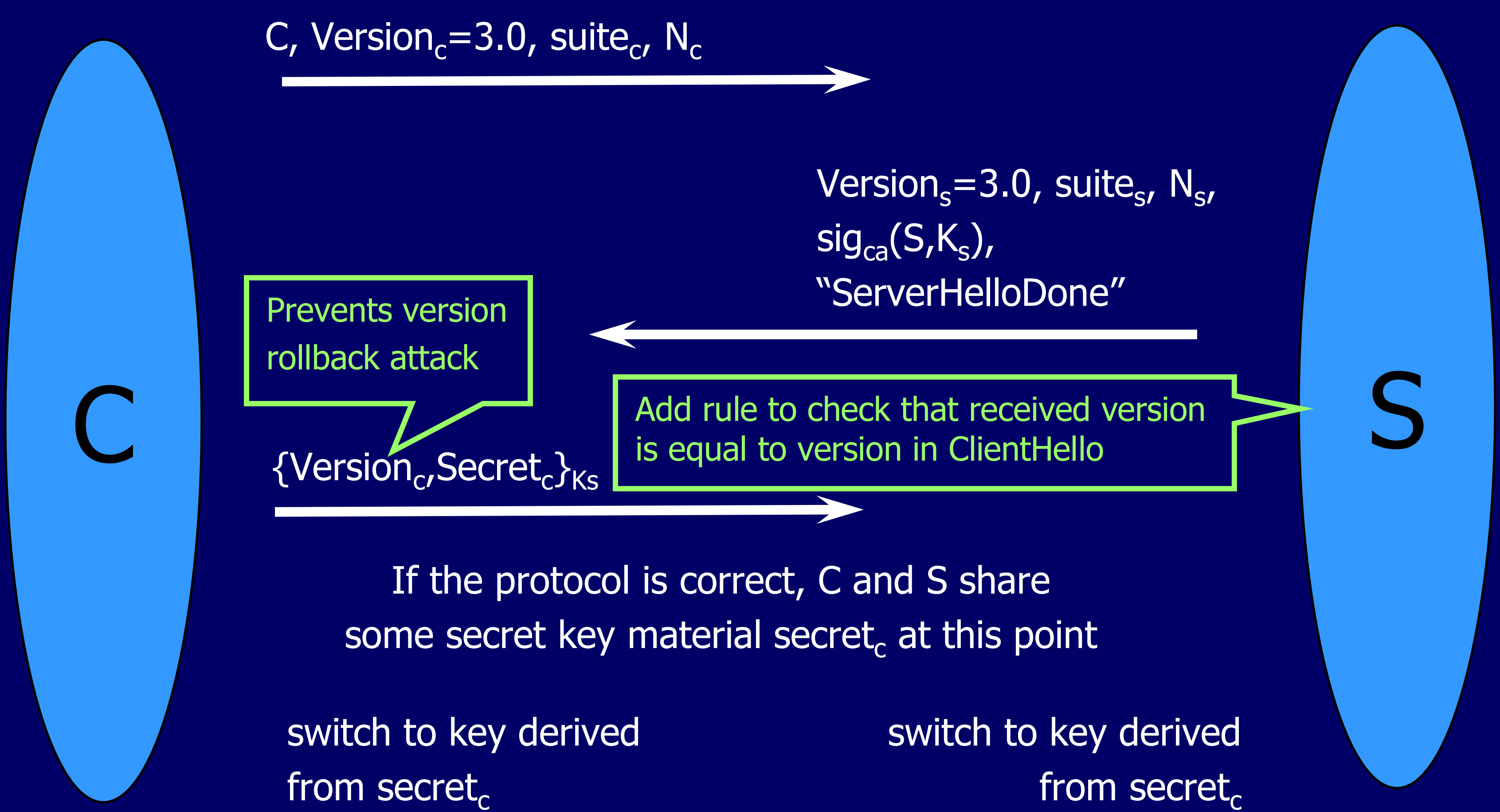
“Core” SSL 3.0


C, $Version_c$=3.0, $suite_c$, N_c
$Version_s$=3.0, $suite_s$, N_s,
$sig_{ca}(S,K_s)$,
“ServerHelloDone”
$\{Secret_c\}_{Ks}$
If the protocol is correct, C and S share
some secret key material $secret_c$ at this point
switch to key derived
from $secret_c$
switch to key derived
from $secret_c$
C
S

Version Consistency Fails!



Fixed “Core” SSL



A Case of Bad Abstraction

```
struct {
   select (KeyExchangeAlgorithm) {
      case rsa: EncryptedPreMasterSecret;
      case diffie_hellman: ClientDiffieHellmanPublic;
   } exchange_keys
} ClientKeyExchange
```

Model this as $\{Version_c, Secret_c\}_{Ks}$

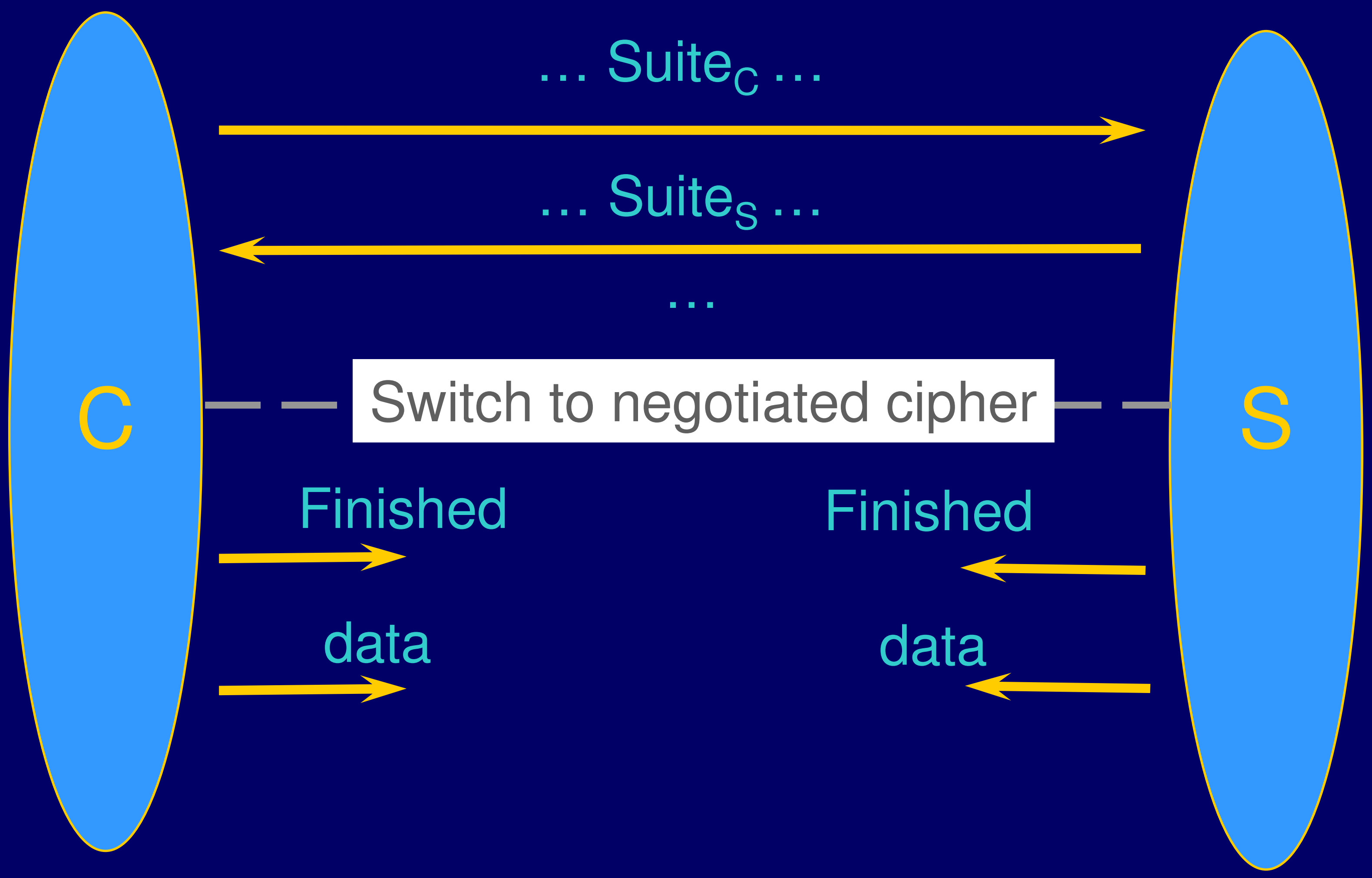
```
struct {
   ProtocolVersion client_version;
   opaque random[46];
} PreMasterSecret
```

This piece matters! Need to add it to the model.

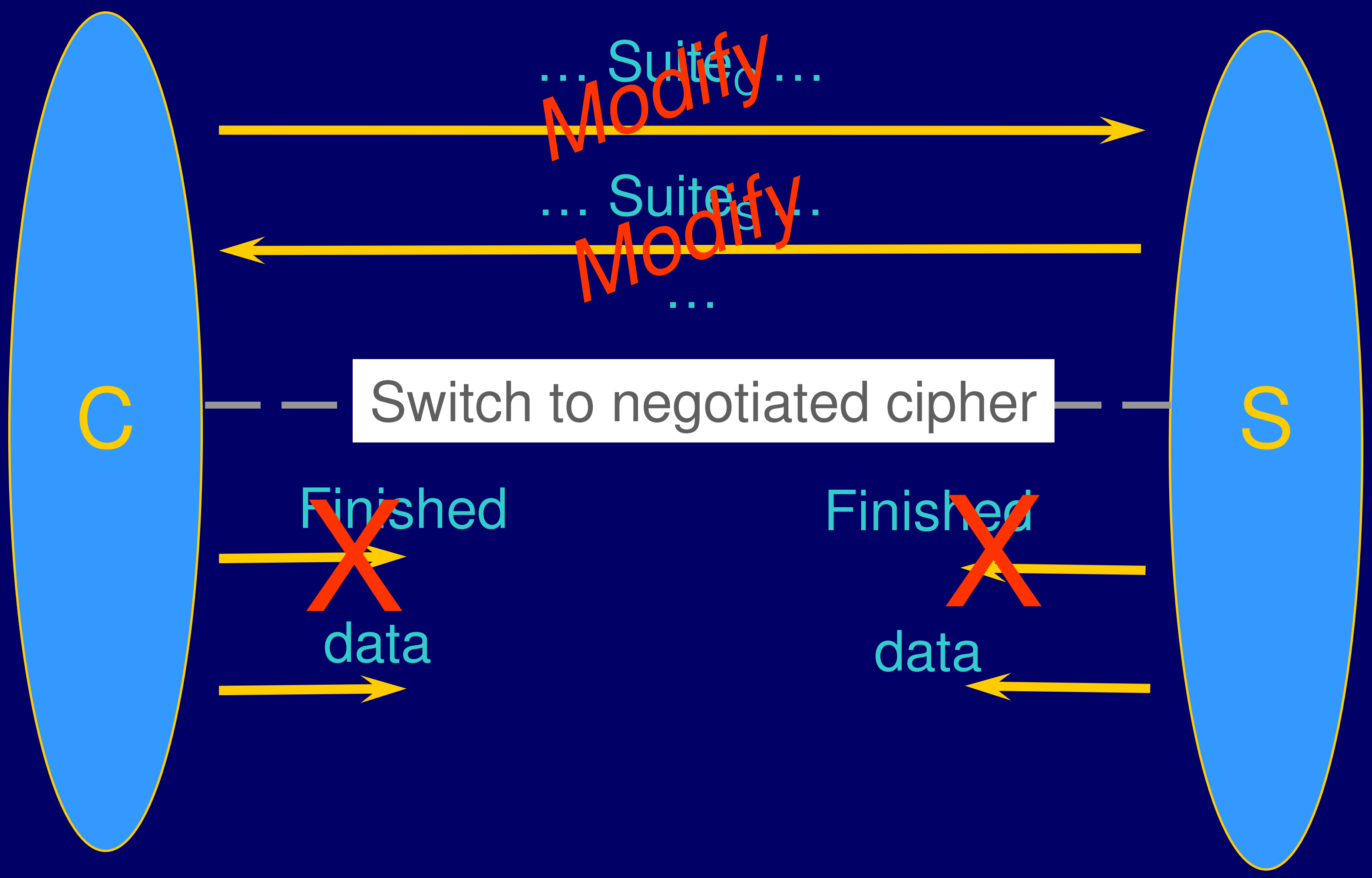
Summary of Reconstruction

- A = Basic protocol
- C = A + certificates for public keys
  - Authentication for client and server
- E = C + verification (Finished) messages
  - Prevention of version and crypto suite attacks
- F = E + nonces
  - Prevention of replay attacks
- Z = “Correct” subset of SSL

Anomaly (Protocol F)

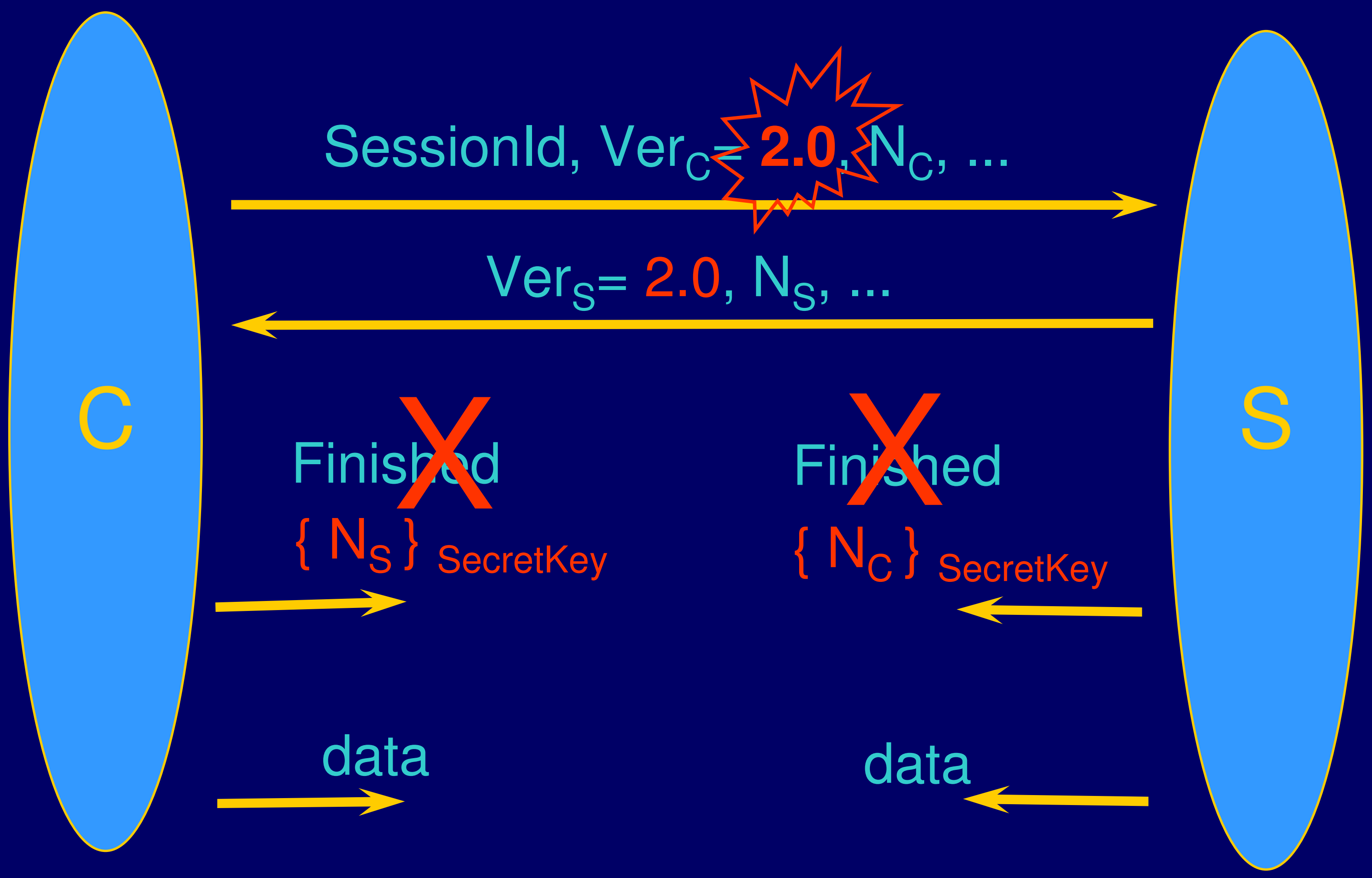


Anomaly (Protocol F)



Protocol Resumption

Version Rollback Attack



Basic Pattern for Doing This Yourself

- Read and understand protocol specification
  - Typically an RFC or a research paper
  - We'll have a few on the CS259 website: take a look!
- Choose a tool
  - Murφ works, also many other tools
  - Play with Murφ now to get some experience (installing, running simple models, etc.)
- Start with a simple (possibly flawed) model
  - Rational reconstruction is a good way to go
- Give careful thought to security conditions

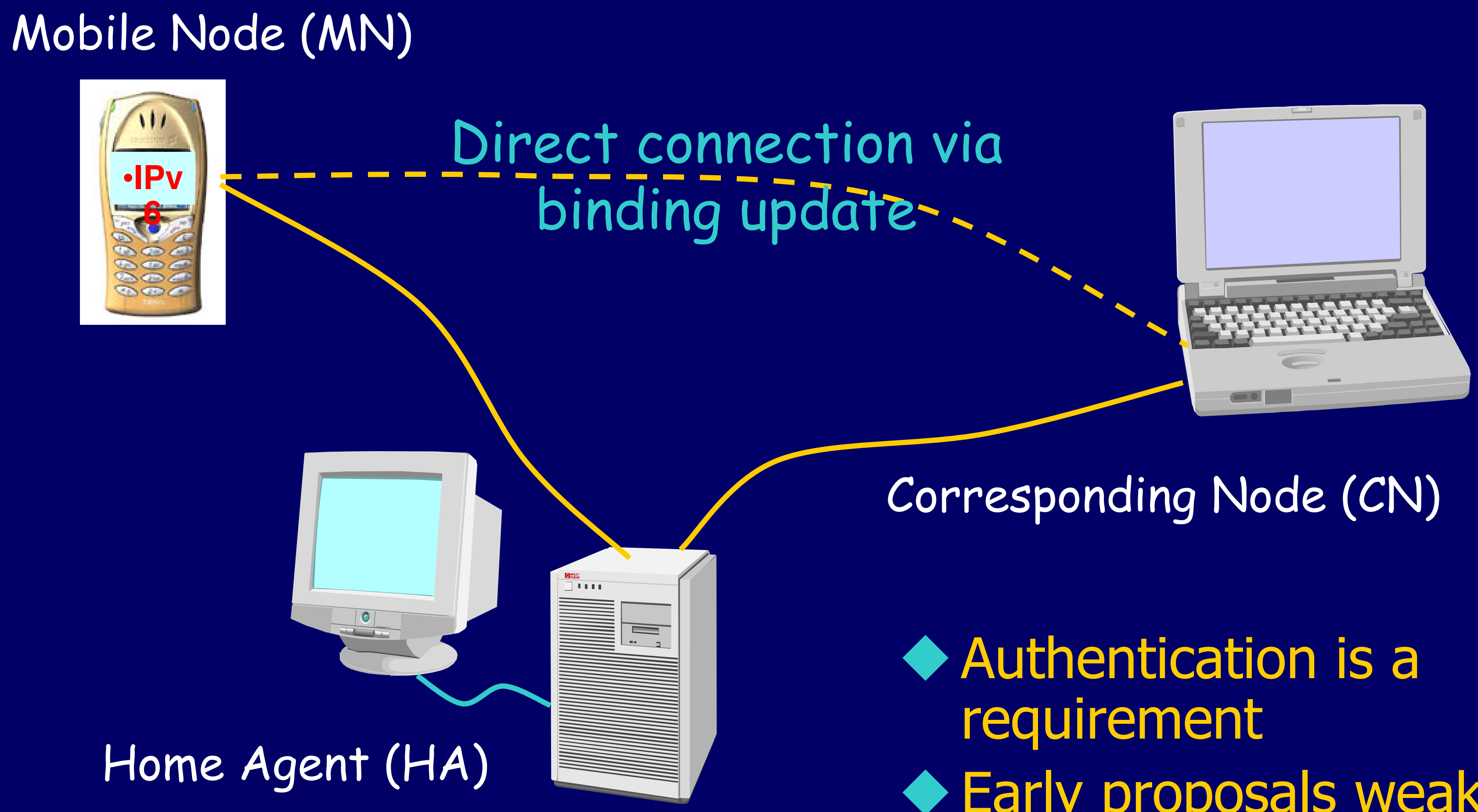
Additional Reading on SSL 3.0

- D. Wagner and B. Schneier. "Analysis of the SSL 3.0 protocol." USENIX Electronic Commerce '96.
  - Nice study of an early proposal for SSL 3.0
- J.C. Mitchell, V. Shmatikov, U. Stern. "Finite-State Analysis of SSL 3.0". USENIX Security '98.
  - Murφ analysis of SSL 3.0 (similar to this lecture)
  - Actual Murφ model available
- D. Bleichenbacher. "Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1". CRYPTO '98.
  - Cryptography is <u>not</u> perfect: this paper breaks SSL 3.0 by directly attacking underlying implementation of RSA

Many security protocols

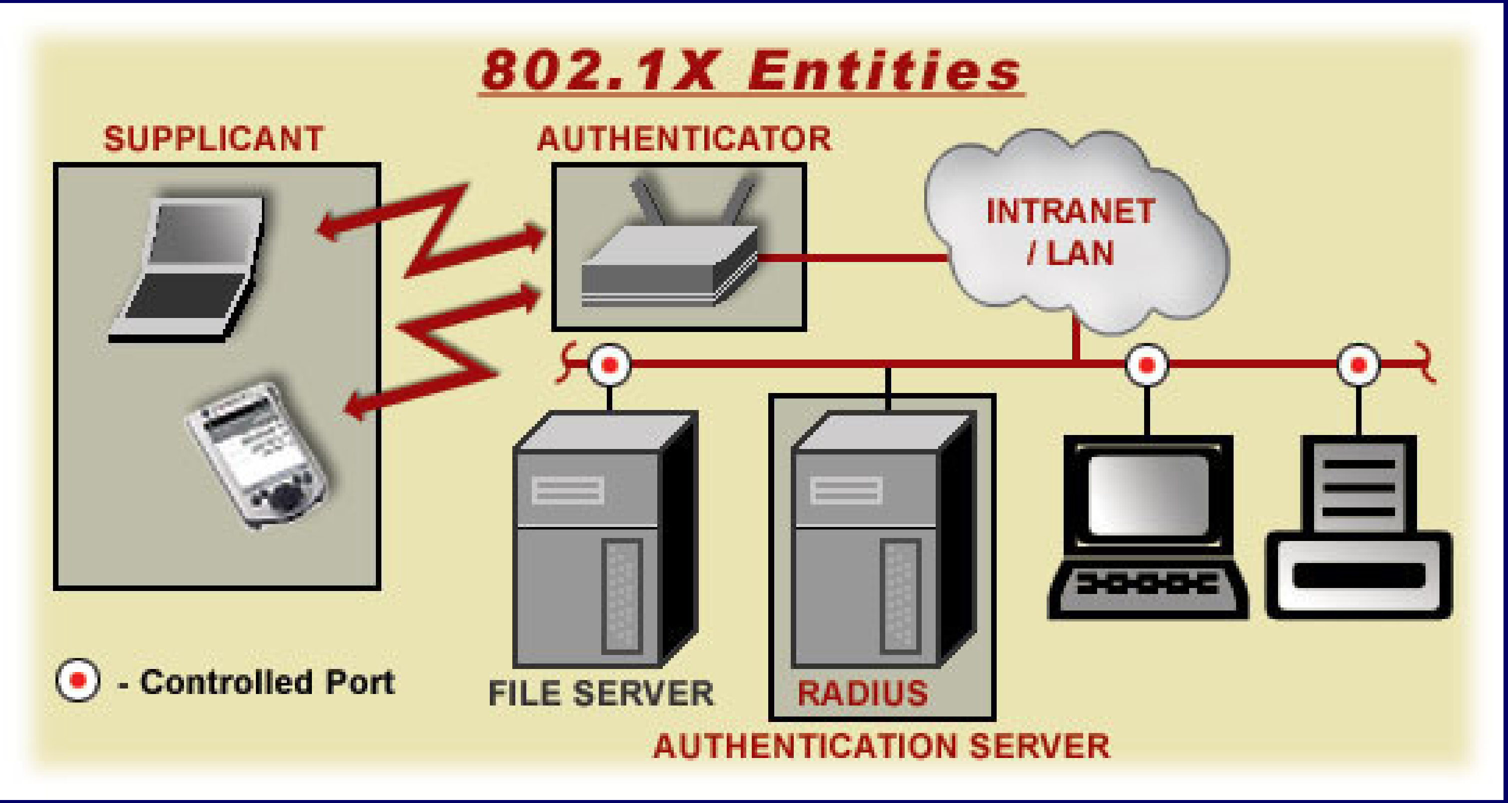
- Challenge-response
  - ISO 9798-1,2,3; Needham-Schroeder, …
- Authentication
  - Kerberos
- Key Exchange
  - SSL handshake, IKE, JFK, IKEv2,
- Wireless and mobile computing
  - Mobile IP, WEP, 802.11i
- Electronic commerce
  - Contract signing, SET, electronic cash, …

Mobile IPv6 Architecture

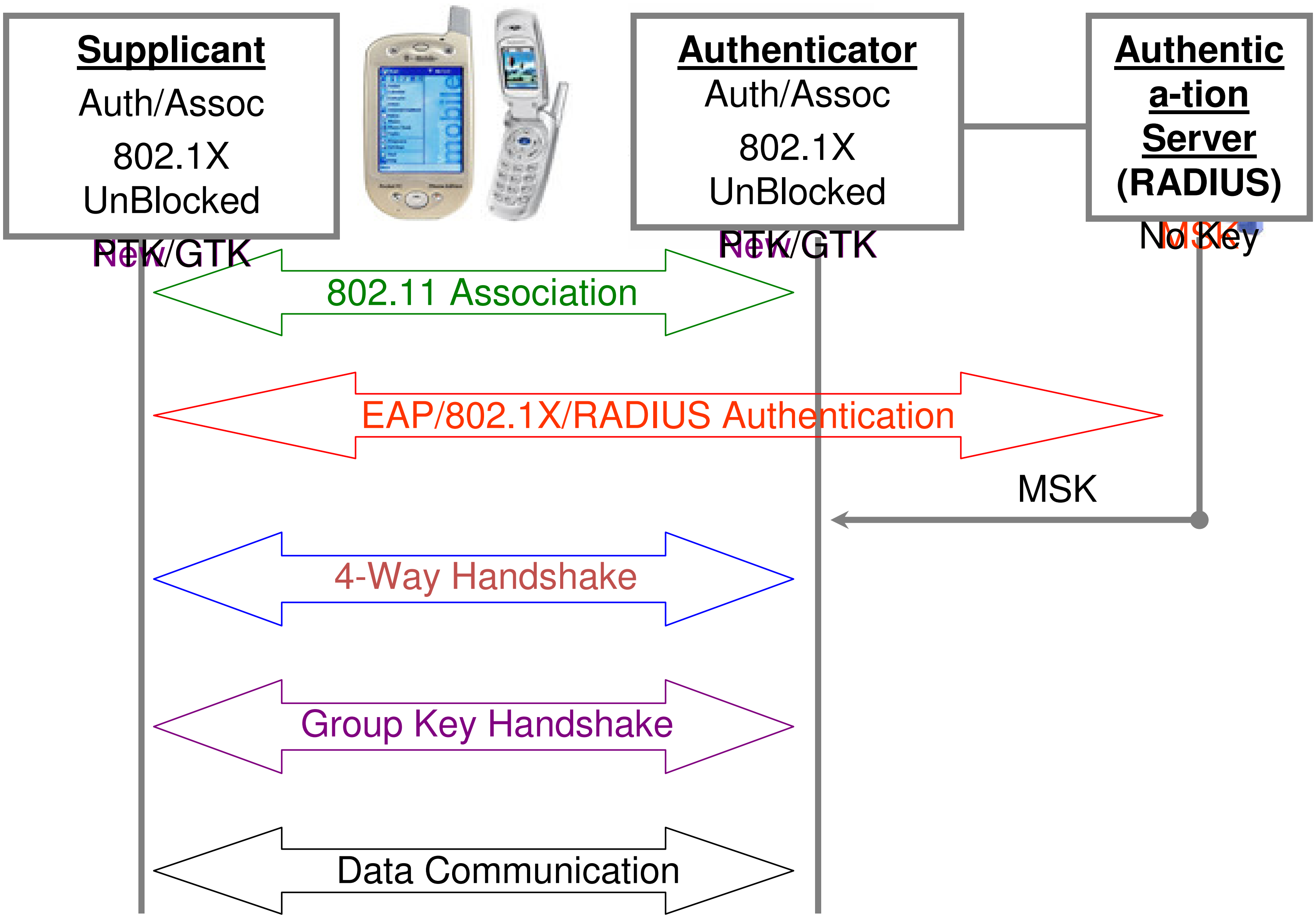


- Authentication is a requirement
- Early proposals weak

802.11i Wireless Authentication



802.11i Protocol



Protocol Verification

Proofs of Correctness

John Mitchell
Stanford

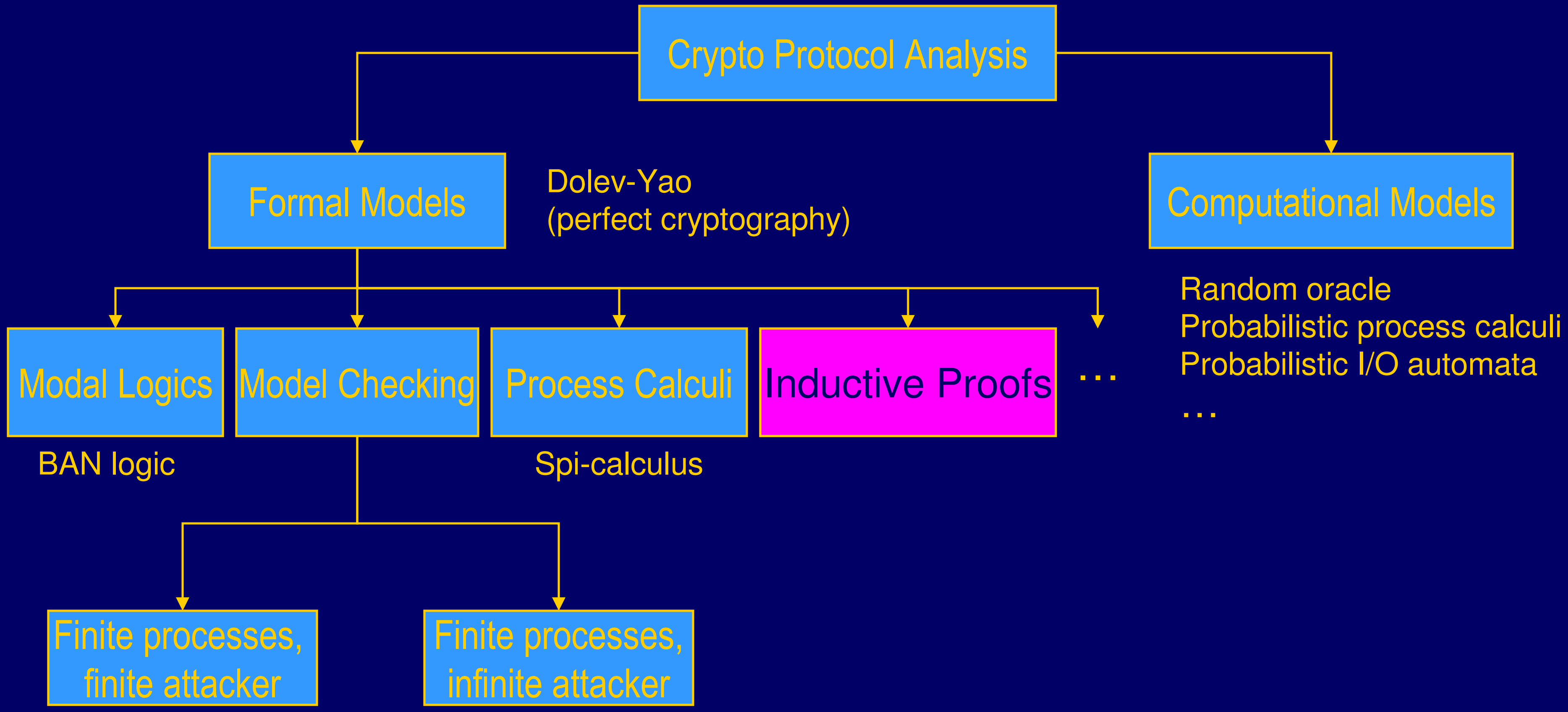
Your mountains

Our mountains

Next picture from here

Our mountains

Analysis Techniques



Reference Material (CS259 web site)

- Protocols
  - Clarke-Jacob survey
  - Use Google; learn to read an RFC
- Tools
  - Murphi
    - Finite-state tool developed by David Dill's group at Stanford
  - PRISM
    - Probabilistic model checker, University of Birmingham
  - MOCHA
    - Alur and Henzinger; now consortium
  - Constraint solver using prolog
    - Shmatikov and Millen
  - Isabelle
    - Theorem prover developed by Larry Paulson in Cambridge, UK
    - A number of case studies available on line

Avispa Project

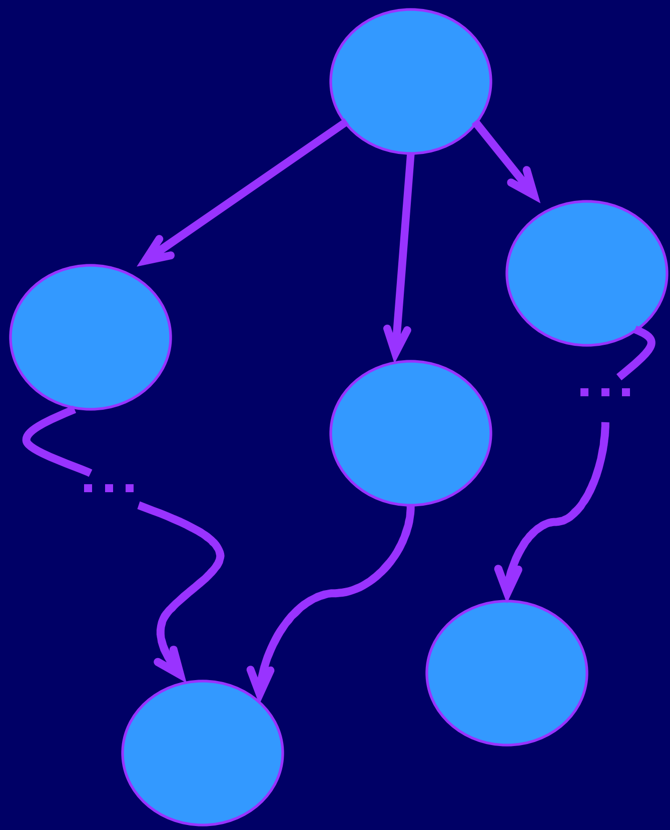
- Convenient web interface
- Several analysis methods
  - Model checker, constraint checker, …
- Single input language
  - Straightforward protocol definition
  - Attacker is built-in
    - Advantage: no need to specify
    - Disadvantage: not easy to change
      - Example: Mobile IPv6 security against "local" attacker – requires a different attacker model

[Paulson]

Analysis using theorem proving

- Correctness instead of bugs
  - Use higher-order logic to reason about possible protocol executions
- No finite bounds
  - Any number of interleaved runs
  - Algebraic theory of messages
  - No restrictions on attacker
- Mechanized proofs
  - Automated tools can fill in parts of proofs
  - Proof checking can prevent errors in reasoning

Recall: protocol state space



- Participant + attacker actions define a state transition graph
- A path in the graph is a trace of the protocol
- Graph can be
  - Finite if we limit number of agents, size of message, etc.
  - Infinite otherwise

Inductive proofs

- Define set of traces
  - Given protocol, a trace is one possible sequence of events, including attacks
- Prove correctness by induction
  - For every state in every trace, no security condition fails
    - Works for safety properties only
  - Proof by induction on the length of trace

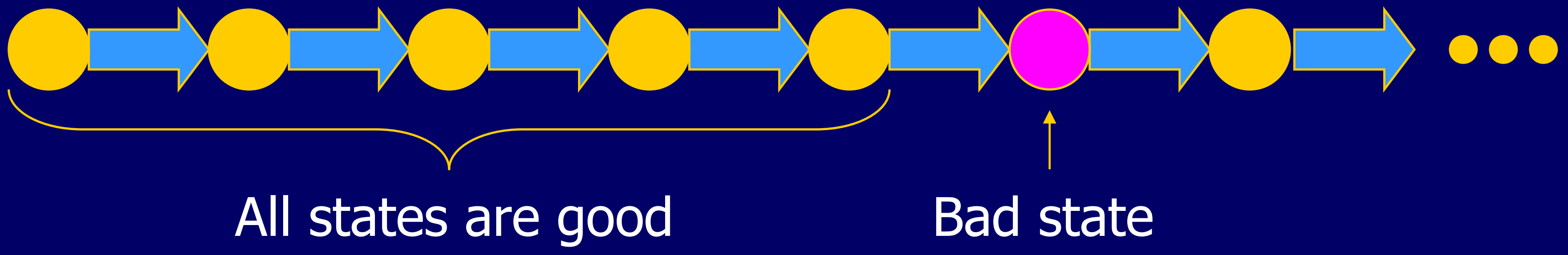
Two forms of induction

- Usual form for $\forall n \in Nat.\ P(n)$
  - Base case: $P(0)$
  - Induction step: $P(x) \Rightarrow P(x+1)$
  - Conclusion: $\forall n \in Nat.\ P(n)$
- Minimal counterexample form
  - Assume: $\exists x\ [\ \neg P(x) \wedge \forall y{<}x.\ P(y)\ ]$
  - Prove: contraction
  - Conclusion: $\forall n \in Nat.\ P(n)$

Both equivalent to "the natural numbers are well-ordered"

Use second form

- Given set of traces
  - Choose shortest sequence to bad state
  - Assume all steps before that OK
  - Derive contradiction
    - Consider all possible steps

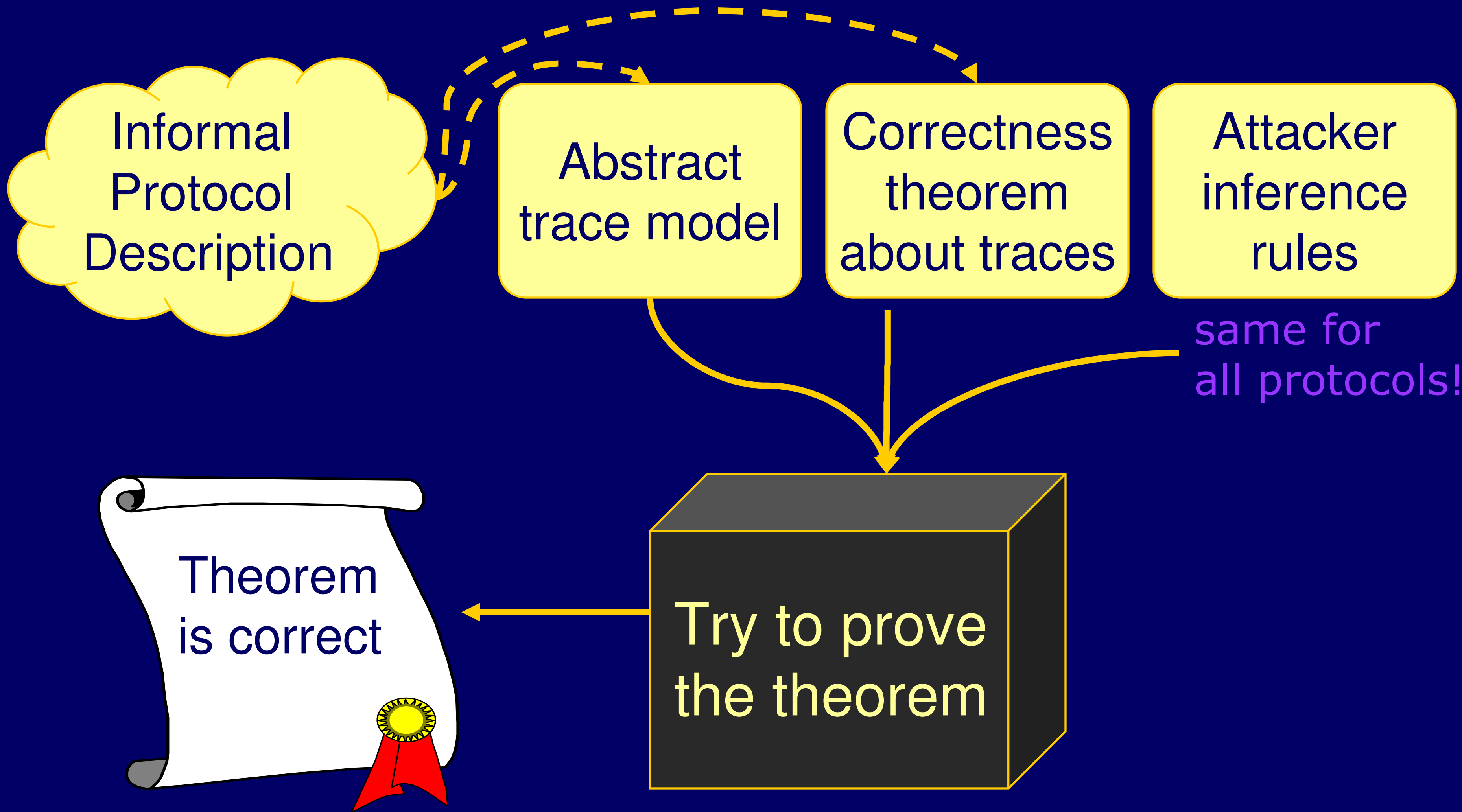


Sample Protocol Goals

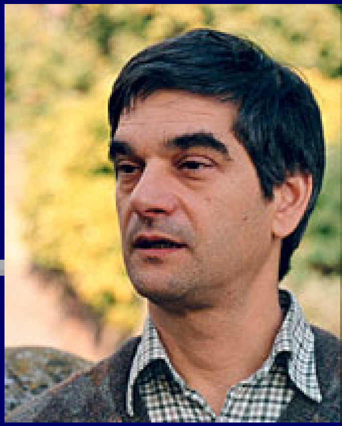
- Authenticity: who sent it?
  - Fails if A receives message from B but thinks it is from C
- Integrity: has it been altered?
  - Fails if A receives message from B but message is not what B sent
- Secrecy: who can receive it?
  - Fails if attacker knows message that should be secret
- Anonymity
  - Fails if attacker or B knows action done by A

These are all safety properties

Inductive Method in a Nutshell



Work by Larry Paulson



- Isabelle theorem prover
  - General tool; protocol work since 1997
- Papers describing method
- Many case studies
  - Verification of SET protocol (6 papers)
  - Kerberos (3 papers)
  - TLS protocol
  - Yahalom protocol, smart cards, etc

http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html

Verifying Security Protocols Using Isabelle - Microsoft Internet Explorer

http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html

Verifying Security Protocols Using Isabelle

- Introductory papers
- Verification of the SET protocol
- Other results
- The original papers *(subsumed by the first paper below)*

INTRODUCTORY PAPERS

- L C Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security* **6** (1998), 85–128.
- Giampaolo Bella
  Inductive Verification of Cryptographic Protocols
  PhD Thesis, University of Cambridge (2000)
- L C Paulson. Security protocols and their correctness.
  Automated Reasoning Workshop 1998. St. Andrews, Scotland (1998)
  Slides available
- L C Paulson. Proving security protocols correct.
  IEEE Symposium on Logic in Computer Science. Trento, Italy (1999).
  Slides available
- L C Paulson. Seven Years of Verifying Security Protocols
  Schloß Dagstuhl Seminar 03451: *Applied Deductive Verification* (2003)
  Slides available

VERIFICATION OF THE SET PROTOCOL

Isabelle

- Automated support for proof development
  - Higher-order logic
  - Serves as a logical framework
  - Supports ZF set theory & HOL
  - Generic treatment of inference rules
- Powerful simplifier & classical reasoner
- Strong support for inductive definitions

Agents and Messages

agent A,B,…	=	Server | Friend *i* | Spy
msg X,Y,…	=	Agent *A*
	|	Nonce *N*
	|	Key *K*
	|	{ *X*, *Y*}
	|	Crypt *X K*

Typed, free term algebra, …

Protocol semantics

- Traces of events:
  - *A sends X to B*
- Operational model of agents
- Algebraic theory of messages (derived)
- A general attacker
- Proofs mechanized using Isabelle/HOL

Define sets inductively

- Traces
  - Set of sequences of events
  - Inductive definition involves implications

    if $ev_1, \ldots, ev_n \in evs$, then add ev′ to evs
- Information from a set of messages
  - parts H : parts of messages in H
  - analz H : information derivable from H
  - synth H : msgs constructible from H

Protocol events in trace

- Several forms of events
  - *A sends B message X*
  - *A receives X*
  - *A stores X*

$A \rightarrow B \quad \{A, N_A\}_{pk(B)}$

If `ev` is a trace and `Na` is unused, add
  `Says A B Crypt(pk B){A,Na}`

$B \rightarrow A \quad \{N_B, N_A\}_{pk(A)}$

If **`Says A' B Crypt(pk B){A,X}`** ∈ `ev` and `Nb` is unused, add
  `Says B A Crypt(pk A){Nb,X}`

$A \rightarrow B \quad \{N_B\}_{pk(B)}$

If **`Says ...{X,Na}...`** ∈ `ev`, add
  `Says A B Crypt(pk B){X}`

Dolev-Yao Attacker Model

- Attacker is a nondeterministic process
- Attacker can
  - Intercept any message, decompose into parts
  - Decrypt if it knows the correct key
  - Create new message from data it has observed
- Attacker cannot
  - Gain partial knowledge
  - Perform statistical tests
  - Stage timing attacks, …

Attacker Capabilities: Analysis

analz H is what attacker can learn from H

$$X \in H \quad \Rightarrow \quad X \in \text{analz } H$$

$$\{X, Y\} \in \text{analz } H \quad \Rightarrow \quad X \in \text{analz } H$$

$$\{X, Y\} \in \text{analz } H \quad \Rightarrow \quad Y \in \text{analz } H$$

$$\text{Crypt } X\, K \in \text{analz } H$$

$$\& \quad K^{-1} \in \text{analz } H \quad \Rightarrow \quad X \in \text{analz } H$$

Attacker Capabilities: Synthesis

synth H is what attacker can create from H

infinite set!

$X \in H \Rightarrow X \in \text{synth } H$

$X \in \text{synth } H \ \& \ Y \in \text{synth } H$

$\Rightarrow \{X, Y\} \in \text{synth } H$

$X \in \text{synth } H \ \& \ K \in \text{synth } H$

$\Rightarrow \text{Crypt } X \, K \in \text{synth } H$

Equations and implications

$\text{analz}(\text{analz } H) = \text{analz } H$

$\text{synth}(\text{synth } H) = \text{synth } H$

$\text{analz}(\text{synth } H) = \text{analz } H \cup \text{synth } H$

$\text{synth}(\text{analz } H) = ???$

$\text{Nonce } N \in \text{synth } H \quad \Rightarrow \quad \text{Nonce } N \in H$

$\text{Crypt } K\, X \in \text{synth } H \quad \Rightarrow \quad \text{Crypt } K\, X \in H$

$\text{or} \quad X \in \text{synth } H \;\&\; K \in H$

Attacker and correctness conditions

If $X \in$ synth(analz(spies *evs*)),
add *Says Spy B X*

X is not secret because attacker can construct it from the parts it learned from *events*

If $\mathit{Says}\ B\ A\ \{N_b, X\}_{pk(A)} \in \mathit{evs}$ &
$\mathit{Says}\ A'\ B\ \{N_b\}_{pk(B)} \in \mathit{evs}$,
Then $\mathit{Says}\ A\ B\ \{N_b\}_{pk(B)} \in \mathit{evs}$

If B thinks he's talking to A,
then A must think she's talking to B

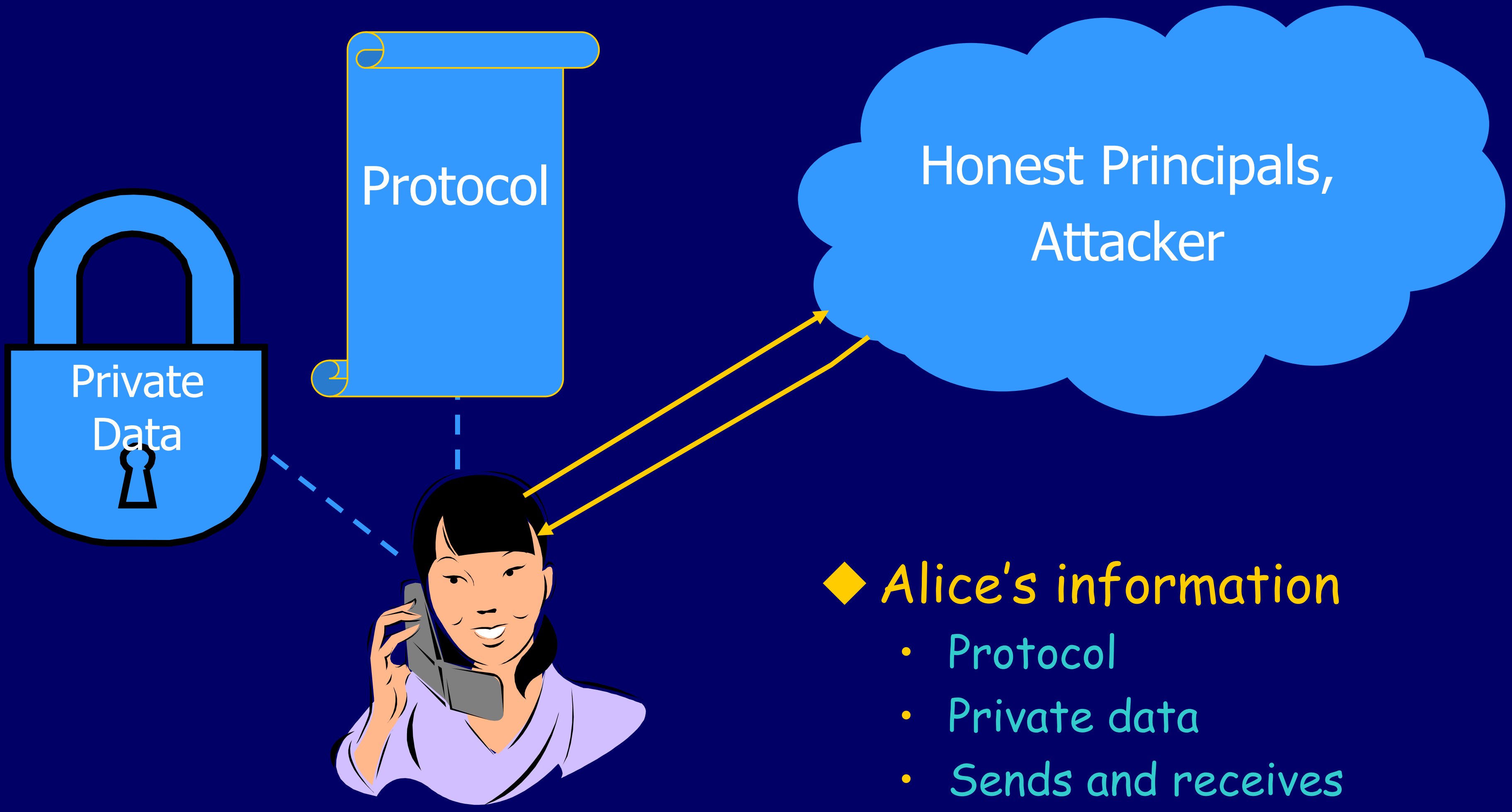
Inductive Method: Pros & Cons

- Advantages
  - Reason about infinite runs, message spaces
  - Trace model close to protocol specification
  - Can "prove" protocol correct
- Disadvantages
  - Does not always give an answer
  - Failure does not always yield an attack
  - Still trace-based properties only
  - Labor intensive
    - Must be comfortable with higher-order logic

Intuition for protocol logic

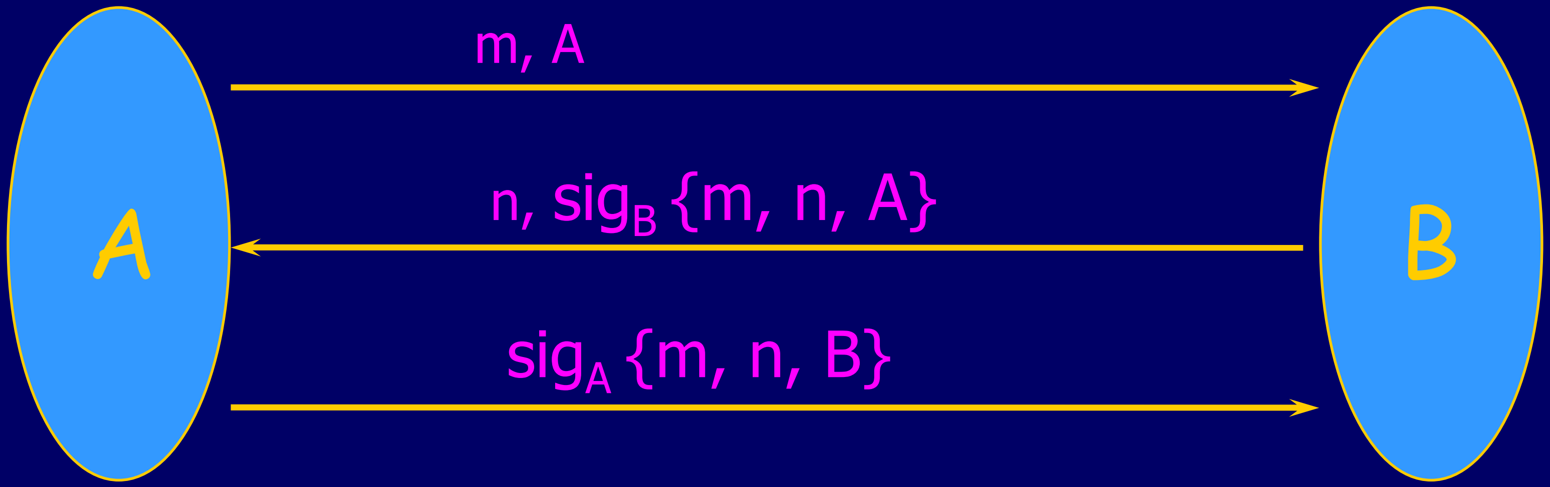
- Reason about local information
  - I chose a new number
  - I sent it out encrypted
  - I received it decrypted
  - Therefore: someone decrypted it
- Incorporate knowledge about protocol
  - Protocol: Server only answers if sent a request
  - If *server not corrupt* and
    - I receive an answer from the server, then
    - the server must have received a request

Intuition: Picture



- Alice's information
  - Protocol
  - Private data
  - Sends and receives

Example: Challenge-Response



- Alice reasons: if Bob is honest, then:
  - only Bob can generate his signature. [protocol independent]
  - if Bob generates a signature of the form $sig_B\{m, n, A\}$,
    - he sends it as part of msg2 of the protocol and
    - he must have received msg1 from Alice [protocol dependent]
  - Alice deduces: Received (B, msg1) $\wedge$ Sent (B, msg2)

Formalizing the Approach

- Language for protocol description
  - Write program for each role of protocol
- Protocol logic
  - State security properties
  - Specialized form of temporal logic
- Proof system
  - Formally prove security properties
  - Supports modular proofs

Cords

- Protocol programming language
  - Server = [receive x; new n; send {x, n}]
- Building blocks
  - Terms
    - names, nonces, keys, encryption, …
  - Actions
    - send, receive, pattern match, …

Terms

t ::=	c	constant term
	x	variable
	N	name
	K	key
	t, t	tupling
	$sig_K\{t\}$	signature
	$enc_K\{t\}$	encryption

Example: $x, sig_B\{m, x, A\}$ is a term

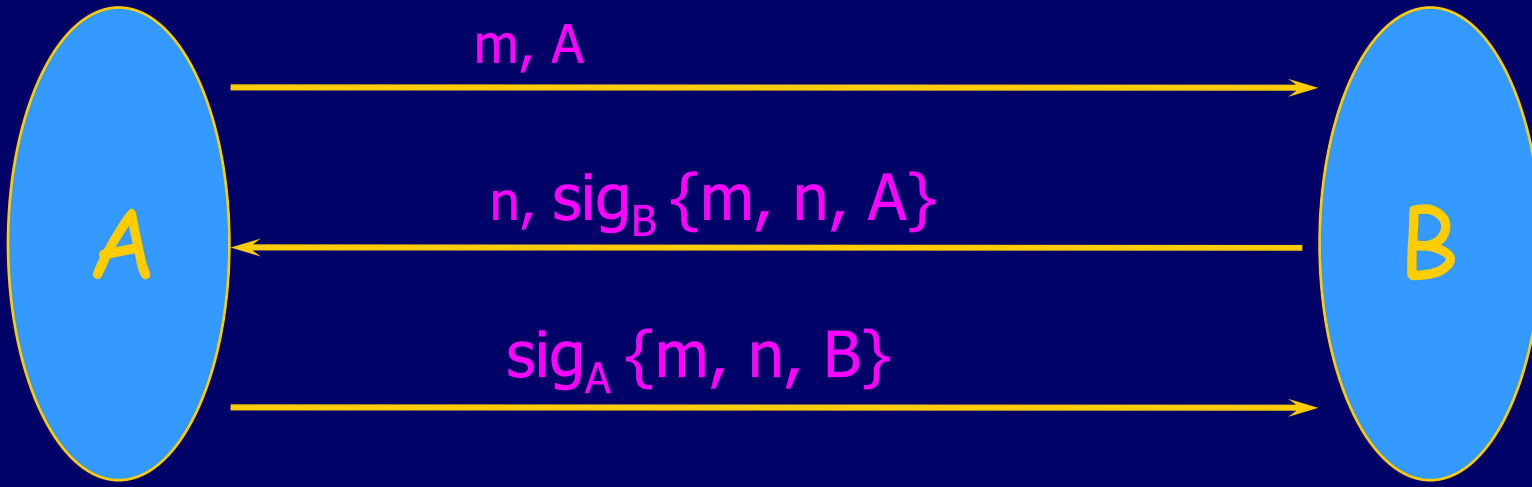
Actions and Cords

- Actions
  - send t; send a term t
  - receive x; receive a term into variable x
  - match t/p(x); match term t against p(x)
- Cord
  - Sequence of actions
- Notation
  - Some match actions are omitted in slides

    receive sigB{A, n} means

    receive x; match x/sigB{A, n}

Challenge-Response as Cords

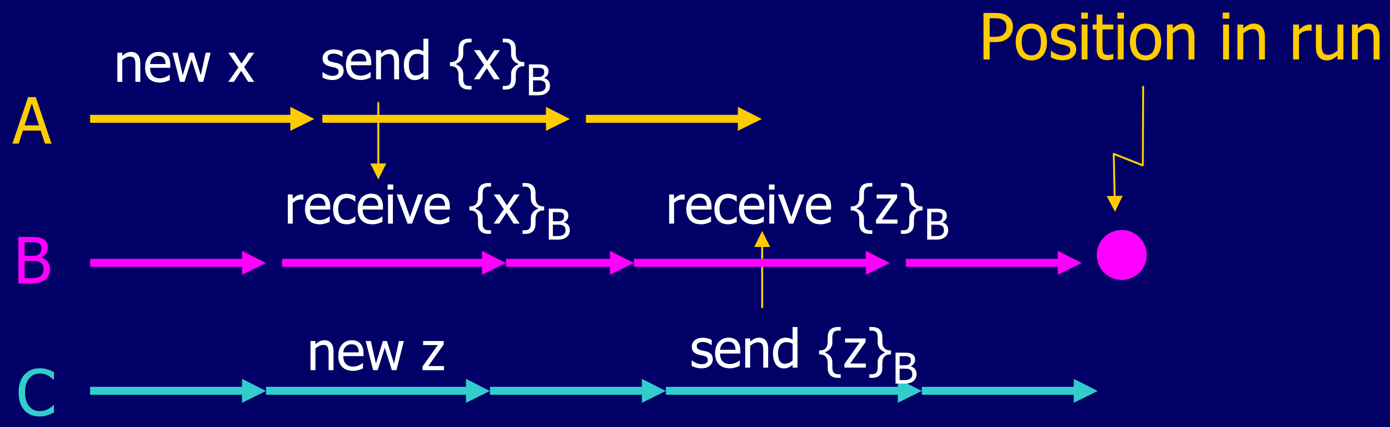


InitCR(A, X) = [
  new m;
  send A, X, {m, A};
  receive X, A, {x, sig_X{m, x, A}};
  send A, X, sig_A{m, x, X}};
]

RespCR(B) = [
  receive Y, B, {y, Y};
  new n;
  send B, Y, {n, sig_B{y, n, Y}};
  receive Y, B, sig_Y{y, n, B}};
]

Execution Model

- Protocol
  - Cord gives program for each protocol role
- Initial configuration
  - Set of principals and keys
  - Assignment of ≥1 role to each principal
- Run



Formulas true at a position in run

- **Action formulas**

  a ::= Send(P,m) | Receive (P,m) | New(P,t)
  | Decrypt (P,t) | Verify (P,t)

- **Formulas**

  φ ::= a | Has(P,t) | Fresh(P,t) | Honest(N)
  | Contains(t_1, t_2) | $\neg\varphi$ | $\varphi_1 \wedge \varphi_2$ | $\exists x\ \varphi$
  | $\bigcirc\varphi$ | $\Diamond\varphi$

- **Example**

  After(a,b) = $\Diamond$ (b $\wedge$ $\bigcirc\Diamond$a)

Modal Formulas

- After actions, postcondition

  $[\text{ actions }]_P\ \varphi$ where $P = \langle \text{princ, role id} \rangle$

- Before/after assertions

  $\varphi\ [\text{ actions }]_P\ \psi$

- Composition rule

$$\frac{\varphi\,[\,S\,]_P\,\psi \qquad \psi\,[\,T\,]_P\,\theta}{\varphi\,[\,ST\,]_P\,\theta}$$

Note: same P
in all formulas

Security Properties

- **Authentication for Initiator**

  $CR \models [\ InitCR(A, B)\ ]_A\ Honest(B) \supset$
  ActionsInOrder(
  Send(A, {A,B,m}),
  Receive(B, {A,B,m}),
  Send(B, {B,A,{n, sig_B {m, n, A}}}),
  Receive(A, {B,A,{n, sig_B {m, n, A}}})
)

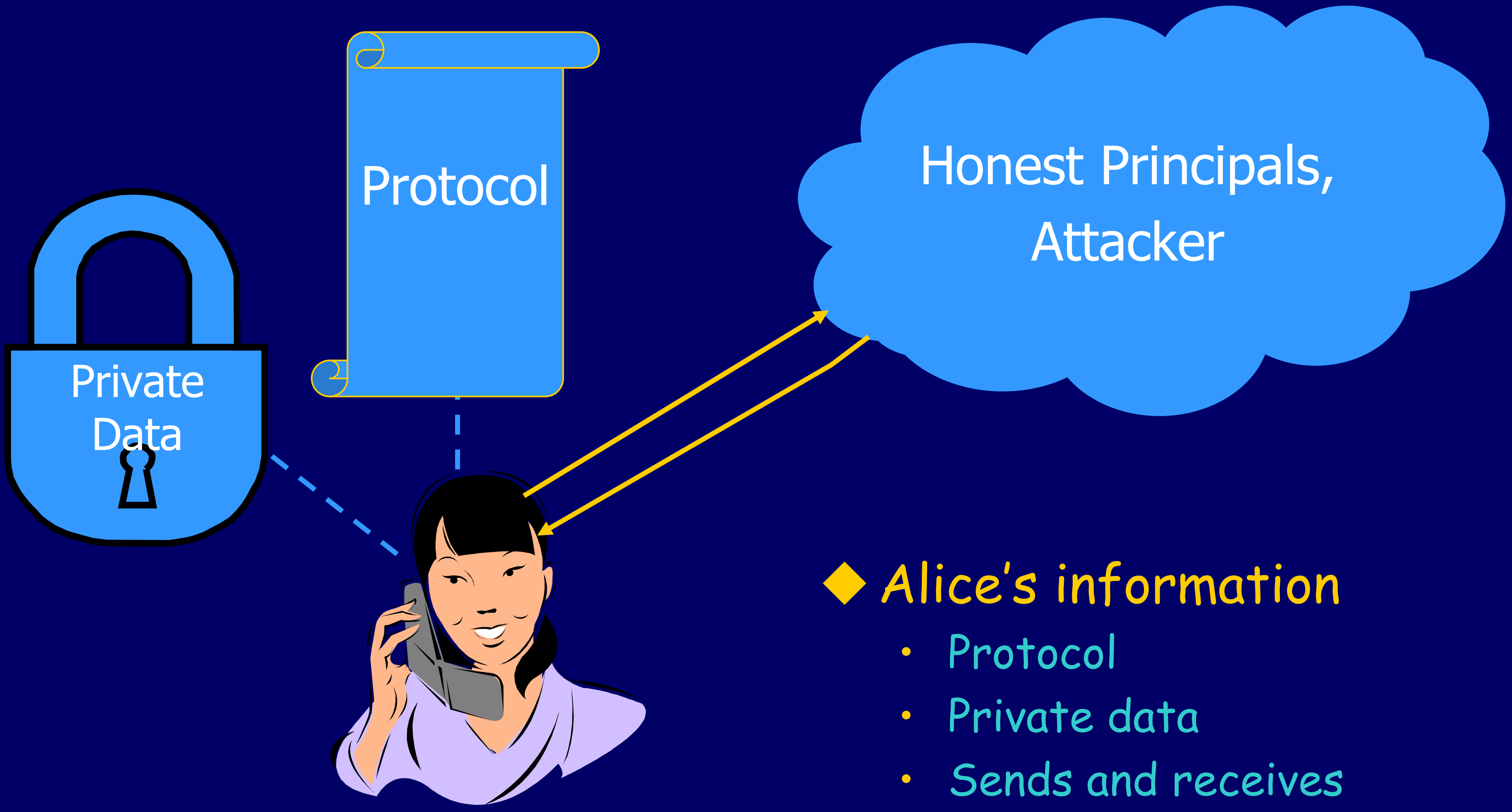
- **Shared secret**

  $NS \models [\ InitNS(A, B)\ ]_A\ Honest(B) \supset$
  $(\ Has(X, m) \supset X{=}A \wedge X{=}B\ )$

Protocol Composition Logic

John Mitchell
Stanford

Intuition: Picture

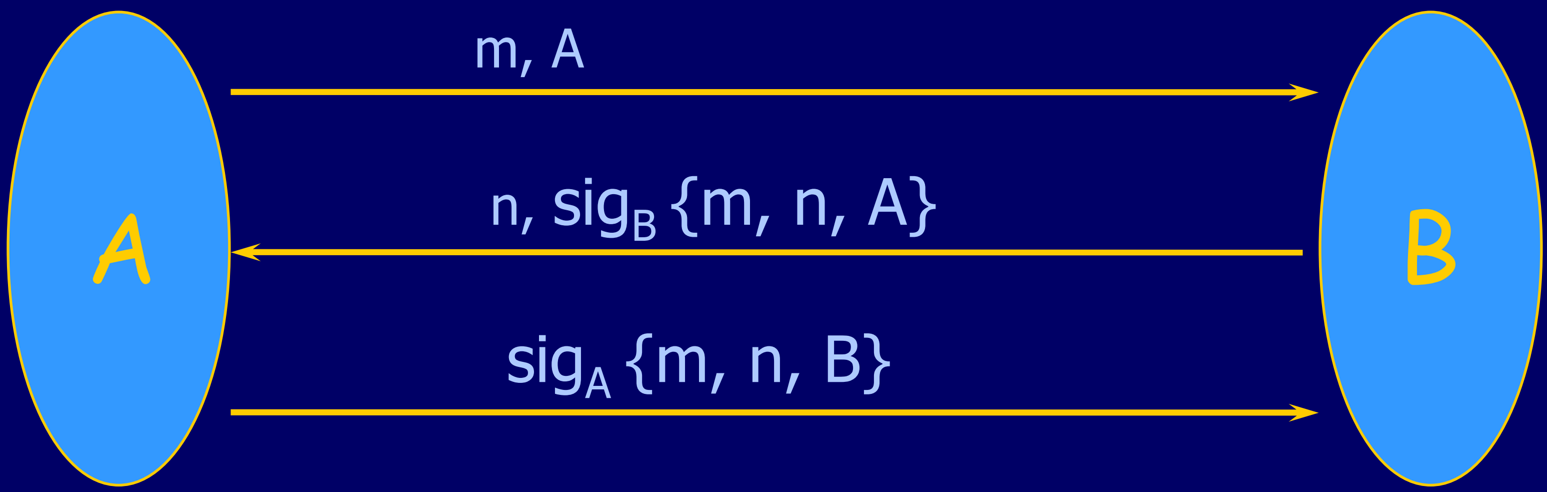


- Alice's information
  - Protocol
  - Private data
  - Sends and receives

Formalization

- Language for protocol description
  - Write program for each role of protocol
- Protocol logic
  - State security properties
  - Specialized form of temporal logic
- Proof system
  - Formally prove security properties
  - Supports modular proofs

Challenge-Response roles

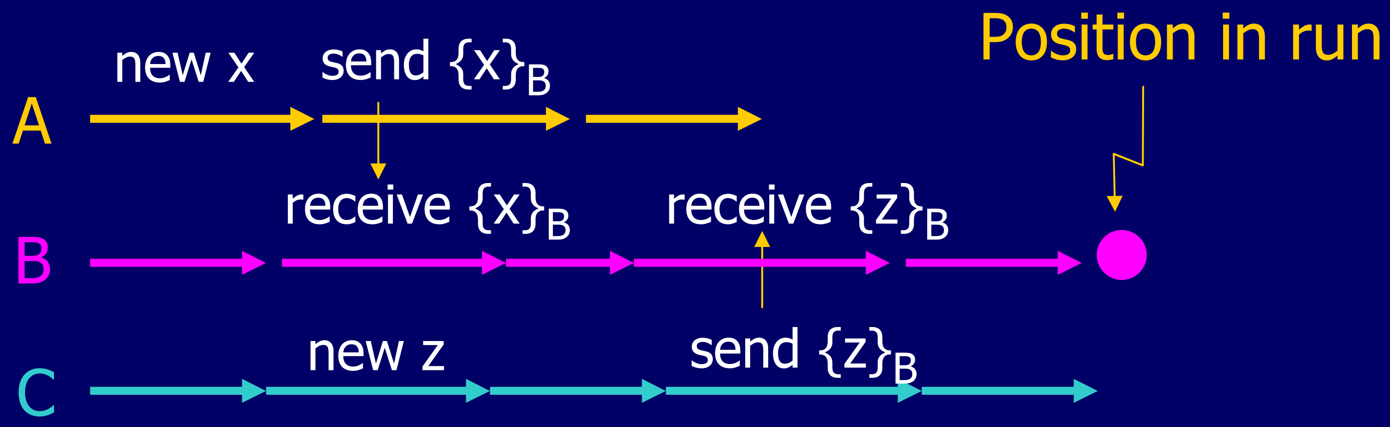


InitCR(A, X) = [
new m;
send A, X, {m, A};
receive X, A, {x, sig_X{m, x, A}};
send A, X, sig_A{m, x, X}};
]

RespCR(B) = [
receive Y, B, {y, Y};
new n;
send B, Y, {n, sig_B{y, n, Y}};
receive Y, B, sig_Y{y, n, B}};
]

Execution Model

- Protocol
  - Sequential program for each protocol role
- Initial configuration
  - Set of principals and keys
  - Assignment of ≥1 role to each principal
- Run



Security Properties

- Authentication for Initiator

  $CR \models [\ \mathrm{InitCR}(A, B)\ ]_A\ \mathrm{Honest}(B) \supset$
  $\mathrm{ActionsInOrder}($
  $\quad \mathrm{Send}(A, \{A,B,m\}),$
  $\quad \mathrm{Receive}(B, \{A,B,m\}),$
  $\quad \mathrm{Send}(B, \{B,A,\{n, sig_B\{m, n, A\}\}\}),$
  $\quad \mathrm{Receive}(A, \{B,A,\{n, sig_B\{m, n, A\}\}\})$
  $)$

- Shared secret

  $NS \models [\ \mathrm{InitNS}(A, B)\ ]_A\ \mathrm{Honest}(B) \supset$
  $(\ \mathrm{Has}(X, m) \supset X{=}A \wedge X{=}B\ )$

Semantics

- Protocol Q
  - Defines set of roles (e.g, initiator, responder)
  - Run R of Q is sequence of actions by principals following roles, plus attacker
- Satisfaction
  - $Q, RS \models \varphi$ [*actions*]$_P$ ψ

    If φ at the end of trace R, and some role of P does exactly *actions* in S, then ψ is true after RS
  - $Q \models \varphi$ [*actions*]$_P$ ψ

    $Q, R \models \varphi$ [*actions*]$_P$ ψ for all runs R of Q

Sample axioms about actions

- New data
  - $[\text{ new } x\ ]_P\ \ Has(P,x)$
  - $[\text{ new } x\ ]_P\ \ Has(Y,x) \supset Y{=}P$
- Actions
  - $[\text{ send } m\ ]_P\ \ Send(P,m)$
- Knowledge
  - $[\text{ receive } m\ ]_P\ \ Has(P,m)$
- Verify
  - $[\text{ match } x/sig_X\{m\}\ ]_P\ \ Verify(P,m)$

Reasoning about posession

- **Pairing**
  - $Has(X, \{m,n\}) \supset Has(X, m) \wedge Has(X, n)$
- **Encryption**
  - $Has(X, enc_K(m)) \wedge Has(X, K^{-1}) \supset Has(X, m)$

Encryption and signature

- Public key encryption

  $\text{Honest}(X) \wedge \text{Decrypt}(Y, \text{enc}_X\{m\}) \supset X=Y$

- Signature

  $\text{Honest}(X) \wedge \text{Verify}(Y, \text{sig}_X\{m\}) \supset$

  $\exists\, m' \,(\text{Send}(X, m') \wedge \text{Contains}(m', \text{sig}_X\{m\}))$

Sample inference rules

- Preservation rules

$$\frac{\psi\ [\ \text{actions}\ ]_P\ Has(X, t)}{\psi\ [\ \text{actions; action}\ ]_P\ Has(X, t)}$$

- Generic rules

$$\frac{\psi\ [\ \text{actions}\ ]_P\ \phi \qquad \psi\ [\ \text{actions}\ ]_P\ \varphi}{\psi\ [\ \text{actions}\ ]_P\ \phi \wedge \varphi}$$

Bidding conventions (motivation)

- Blackwood response to 4NT
  - 5♣ : 0 or 4 aces
  - 5♦ : 1 ace
  - 5♥ : 2 aces
  - 5♠ : 3 aces
- Reasoning
  - If my partner is following Blackwood, then if she bid 5♥, she must have 2 aces

Honesty rule (rule scheme)

$\forall$roles R of Q. $\forall$ initial segments $A \subseteq$ R.

$$\frac{Q \;\vdash\; [\,A\,]_X\, \phi}{Q \;\vdash\; \text{Honest}(X) \supset \phi}$$

- This is a finitary rule:
  - Typical protocol has 2-3 roles
  - Typical role has 1-3 receives
  - Only need to consider *A* waiting to receive

Honesty rule (example use)

$\forall$roles R of Q. $\forall$ initial segments $A \subseteq$ R.

$$\frac{Q \;\vdash\; [\,A\,]_X\, \phi}{Q \;\vdash\; \mathrm{Honest}(X) \supset \phi}$$

- Example use:
  - If Y receives a message from X, and Honest(X) $\supset$ (Sent(X,m) $\supset$ Received(X,m')) then Y can conclude

    Honest(X) $\supset$ Received(X,m'))

Correctness of CR

InitCR(A, X) = [
    new m;
    send A, X, {m, A};
    receive X, A, {x, sig_X{m, x, A}};
    send A, X, sig_A{m, x, X}};
]

RespCR(B) = [
    receive Y, B, {y, Y};
    new n;
    send B, Y, {n, sig_B{y, n, Y}};
    receive Y, B, sig_Y{y, n, B}};
]

CR |- [InitCR(A, B) $]_A$ Honest(B) $\supset$
    ActionsInOrder(
        Send(A, {A,B,m}),
        Receive(B, {A,B,m}),
        Send(B, {B,A,{n, sig_B {m, n, A}}}),
        Receive(A, {B,A,{n, sig_B {m, n, A}}})
)

Correctness of CR - step 1

InitCR(A, X) = [
    new m;
    send A, X, {m, A};
    receive X, A, {x, sig_X{m, x, A}};
    send A, X, sig_A{m, x, X}};
]

RespCR(B) = [
    receive Y, B, {y, Y};
    new n;
    send B, Y, {n, sig_B{y, n, Y}};
    receive Y, B, sig_Y{y, n, B}};
]

1. A reasons about it's own actions

CR |- [InitCR(A, B)]$_A$ Verify(A, sig_B {m, n, A})

Correctness of CR - step 2

```
InitCR(A, X) = [
    new m;
    send A, X, {m, A};
    receive X, A, {x, sig_X{m, x, A}};
    send A, X, sig_A{m, x, X}};
]
```

```
RespCR(B) = [
    receive Y, B, {y, Y};
    new n;
    send B, Y, {n, sig_B{y, n, Y}};
    receive Y, B, sig_Y{y, n, B}};
]
```

2. Properties of signatures

$$CR \vdash [\ InitCR(A, B)\ ]_A\ Honest(B) \supset$$
$$\exists\ m'\ (Send(B, m') \wedge Contains(m', sig_B\ \{m, n, A\}))$$

Correctness of CR – Honesty

InitCR(A, X) = [
    new m;
    send A, X, {m, A};
    receive X, A, {x, sig_X{m, x, A}};
    send A, X, sig_A{m, x, X}};
]

RespCR(B) = [
    receive Y, B, {y, Y};
    new n;
    send B, Y, {n, sig_B{y, n, Y}};
    receive Y, B, sig_Y{y, n, B}};
]

Honesty invariant

CR |- Honest(X) ∧
Send(X, m') ∧ Contains(m', sig_x {y, x, Y}) ∧ ¬ New(X, y) ⊃
m= X, Y, {x, sig_B{y, x, Y}} ∧ Receive(X, {Y, X, {y, Y}})

"If an honest X sends m containing sig_x {y, x, Y}, and X did not create y, then m is responders message and X receive initiators message 1"

Correctness of CR - step 3

InitCR(A, X) = [
new m;
send A, X, {m, A};
receive X, A, {x, sig_X{m, x, A}};
send A, X, sig_A{m, x, X}};
]

RespCR(B) = [
receive Y, B, {y, Y};
new n;
send B, Y, {n, sig_B{y, n, Y}};
receive Y, B, sig_Y{y, n, B}};
]

3. From Honesty rule

CR |- [InitCR(A, B) $]_A$ Honest(B) ⊃
Receive(B, {A,B,m}),

Correctness of CR - step 4

InitCR(A, X) = [
    new m;
    send A, X, {m, A};
    receive X, A, {x, sig_X{m, x, A}};
    send A, X, sig_A{m, x, X}};
]

RespCR(B) = [
    receive Y, B, {y, Y};
    new n;
    send B, Y, {n, sig_B{y, n, Y}};
    receive Y, B, sig_Y{y, n, B}};
]

4. Use properties of nonces for temporal ordering

$CR \vdash [\ InitCR(A, B)\ ]_A\ Honest(B) \supset Auth$

Complete formal proof

AM1	$(A\ B\ \eta)[\,]_{A,\eta}\ \mathsf{Has}(A,A,\eta) \wedge \mathsf{Has}(A,B,\eta)$
AN3	$[(\nu m)]_{A,\eta}\ \mathsf{Fresh}(A,m,\eta)$
AA1	$[\langle A,B,m\rangle]_{A,\eta}\ \Diamond \mathsf{Send}(A,\{A,B,m\},\eta)$
AA1	$[(B,A,n,\{\!\vert m,n,A\vert\!\}_{\overline{B}})]_{A,\eta}$ $\Diamond \mathsf{Receive}(A,\{B,A,n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\},\eta)$
AA1	$[(\{\!\vert m,n,A\vert\!\}_{\overline{B}}/\{\!\vert m,n,A\vert\!\}_{B})]_{A,\eta}\ \Diamond \mathsf{Verify}(A,\{\!\vert m,n,A\vert\!\}_{\overline{B}},\eta)$
AA1	$[\langle A,B,\{\!\vert m,n,B\vert\!\}_{\overline{A}}\rangle]_{A,\eta}\ \Diamond \mathsf{Send}(A,\{A,B,\{\!\vert m,n,B\vert\!\}_{\overline{A}}\},\eta)$
AF1, AF2	$(A\ B\ \eta)[(\nu m)\langle A,B,m\rangle(x)(x/B,A,n,\{\!\vert m,n,A\vert\!\}_{\overline{B}})$ $(\{\!\vert m,n,A\vert\!\}_{\overline{B}}/\{\!\vert m,n,A\vert\!\}_{B})\langle A,B,\{\!\vert m,n,B\vert\!\}_{\overline{A}}\rangle]_{A,\eta}$ $\mathsf{ActionsInOrder}(\mathsf{Send}(A,\{A,B,m\},\eta), \mathsf{Receive}(A,\{B,A,n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\},\eta),$ $\mathsf{Send}(A,\{A,B,\{\!\vert m,n,B\vert\!\}_{\overline{A}}\},\eta))$
N1	$\Diamond \mathsf{New}(A,m,\eta) \supset \neg \Diamond \mathsf{New}(B,m,\eta')$
5, **VER**	$\mathsf{Honest}(B) \wedge \Diamond \mathsf{Verify}(A,\{\!\vert m,n,A\vert\!\}_{\overline{B}},\eta) \supset$ $\exists \eta'.\exists m'.(\Diamond \mathsf{CSend}(B,m',\eta') \wedge (\{\!\vert m,n,A\vert\!\}_{\overline{B}} \subseteq m'))$
HON	$\mathsf{Honest}(B) \supset (\exists \eta'.\exists m'.((\Diamond \mathsf{CSend}(B,m',\eta') \wedge$ $\{\!\vert m,n,A\vert\!\}_{\overline{B}} \subseteq m' \wedge \neg \Diamond \mathsf{New}(B,m,\eta')) \supset$ $(m' = \{B,A,\{n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\}\} \wedge \Diamond \mathsf{Receive}(B,\{A,B,m\},\eta') \wedge$ $\mathsf{ActionsInOrder}(\mathsf{Receive}(B,\{A,B,m\},\eta'), \mathsf{New}(B,n,\eta'),$ $\mathsf{Send}(B,\{B,A,\{n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\}\},\eta')))))$
2, 3, 11, **AF3**	$\mathsf{Honest}(B) \supset \mathsf{After}(\mathsf{Send}(A,\{A,B,m\},\eta),$ $\mathsf{Receive}(B,\{A,B,m\},\eta'))$
11, **AF2**	$\mathsf{Honest}(B) \supset \mathsf{After}(\mathsf{Receive}(B,\{A,B,m\},\eta'),$ $\mathsf{Send}(B,\{B,A,\{n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\}\},\eta'))$
11, 4, **AF3**	$\mathsf{Honest}(B) \supset \mathsf{After}(\mathsf{Send}(B,\{B,A,\{n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\}\},\eta'),$ $\mathsf{Receive}(A,\{B,A,\{n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\}\},\eta))$
10 − 13, **AF2**	$\mathsf{Honest}(B) \supset \exists \eta'.(\mathsf{ActionsInOrder}(\mathsf{Send}(A,\{A,B,m\},\eta),$ $\mathsf{Receive}(B,\{A,B,m\},\eta'), \mathsf{Send}(B,\{B,A,\{n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\}\},\eta'),$ $\mathsf{Receive}(A,\{B,A,\{n,\{\!\vert m,n,A\vert\!\}_{\overline{B}}\}\},\eta))$

Table 8. Deductions of A executing $\mathtt{Init}$ role of CR

Composition Rules

- Prove assertions from invariants

$$\Gamma \mid\!- \ \varphi\ [\ldots]_P\ \psi$$

- Invariant weakening rule

$$\frac{\Gamma \mid\!- \ \varphi\ [\ldots]_P\ \psi}{\Gamma \cup \Gamma' \mid\!- \ \varphi\ [\ldots]_P\ \psi}$$

If combining protocols, extend assertions to combined invariants

- Prove invariants from protocol

$$\frac{Q \blacktriangleright \Gamma \qquad Q' \blacktriangleright \Gamma}{Q \bullet Q' \blacktriangleright \Gamma}$$

Use honesty (invariant) rule to show that both protocols preserve assumed invariants

Combining protocols

Γ $\qquad\qquad$ Γ'

DH ▶ Honest(X) $\supset$ … $\qquad$ CR ▶ Honest(X) $\supset$ …

Γ |- Secrecy $\qquad$ Γ' |- Authentication

$\Gamma \cup \Gamma'$ |- Secrecy $\qquad$ $\Gamma \cup \Gamma'$ |- Authentication

$\Gamma \cup \Gamma'$ |- Secrecy $\wedge$ Authentication

DH • CR ▶ $\Gamma \cup \Gamma'$

||

ISO ▶ Secrecy $\wedge$ Authentication

Protocol Templates

- Protocols with function variables instead of specific operations
  - One template can be instantiated to many protocols
- Advantages:
  - proof reuse
  - design principles/patterns

Example

Challenge-Response Template

$A \rightarrow B$: m
$B \rightarrow A$: $n, F(B,A,n,m)$
$A \rightarrow B$: $G(A,B,n,m)$

Abstraction

ISO-9798-2

$A \rightarrow B$: m
$B \rightarrow A$: $n, E_{KAB}(n,m,B)$
$A \rightarrow B$: $E_{KAB}(n,m)$

SKID3

$A \rightarrow B$: m
$B \rightarrow A$: $n, H_{KAB}(n,m,B)$
$A \rightarrow B$: $H_{KAB}(n,m,A)$

ISO-9798-3

$A \rightarrow B$: m
$B \rightarrow A$: $n, sig_B(n,m,A)$
$A \rightarrow B$: $sig_A(n,m,B)$

Instantiation

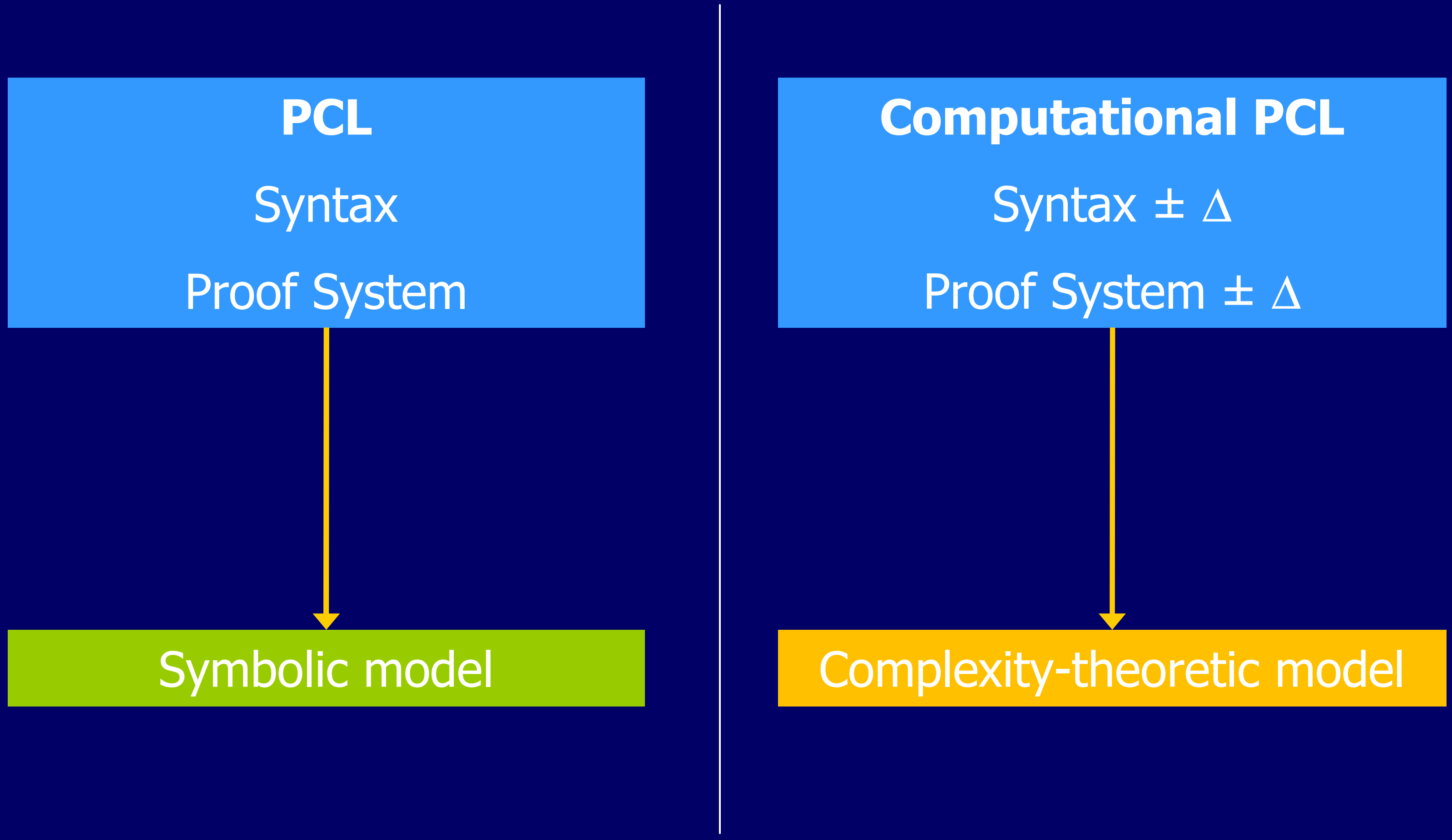
Sample projects using PCL

- Simple key exchange
  - STS family
  - Diffie-Hellman -> MQV
  - GDOI [Meadows, Pavlovic]
- Larger protocols
  - SSL verification
  - Wireless 802.11i
  - JFK, IKEv2
  - Kerberos, including PKINIT, DHINIT

Symbolic vs Computational model

- Suppose Γ |- $[\text{actions}]_X\, \varphi$
  - If a protocol P satisfies invariants Γ, then if X does *actions*, φ will be true
- Symbolic soundness
  - No idealized adversary acting against "perfect" cryptography can make φ fail
- Computational soundness
  - No probabilistic polytime adversary can make φ fail with nonnegligible probability

PCL → Computational PCL



Some general issues

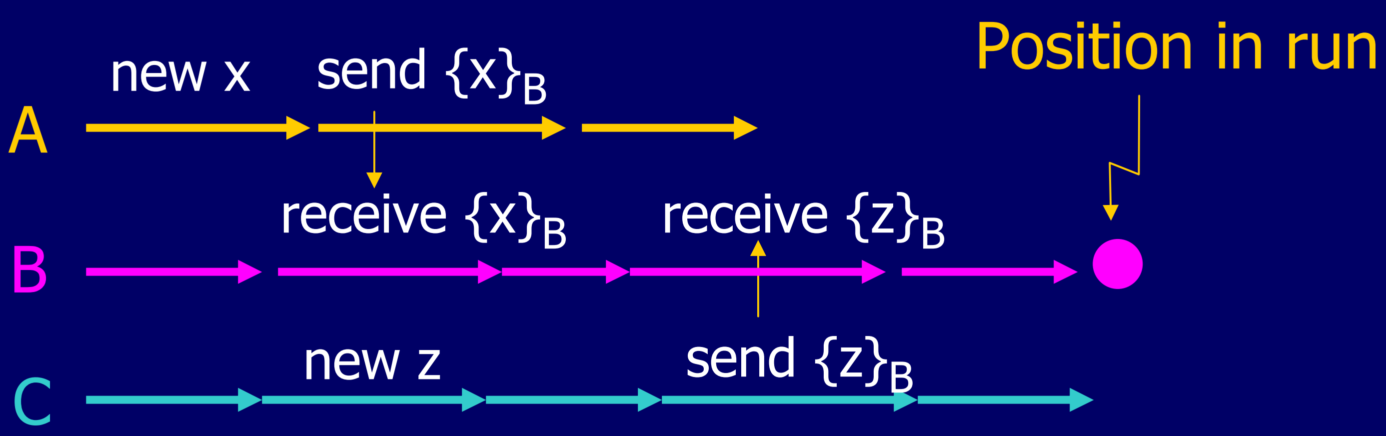
- Computational PCL
  - Symbolic logic for proving security properties of network protocols that use cryptography
- Soundness Theorem:
  - If a property is provable in CPCL, then property holds in computational model with overwhelming asymptotic probability
- Benefits
  - Retain compositionality
  - Symbolic proofs about computational model
  - Probability, complexity theory in soundness proof (only!)
  - Different axioms rely on different crypto assumptions
    - Competing symbolic ≈ computational methods generally requires *strong* crypto assumptions

PCL → Computational PCL

- Syntax, proof rules mostly the same
  - Retain compositional approach
  - But some issues with propositional connectives…
- Significant differences
  - Symbolic "knowledge"
    - Has(X,t) : X can produce t from msgs that have been observed, by symbolic algorithm
  - Computational "knowledge"
    - Possess(X,t) : can produce t by ppt algorithm
    - Indist(X,t) : cannot distinguish from rand value in ppt
  - More subtle system
    - Some axioms rely on CCA2, some info-theoretically sound, etc.

Recall Execution Model

- Protocol
  - Sequential program for each protocol role
- Initial configuration
  - Set of principals and keys
  - Assignment of ≥1 role to each principal
- Run



Computational Traces

- Computational trace contains
  - Symbolic actions of honest parties
  - Mapping of symbolic variables to bitstrings
  - Send-receive actions (only) of the adversary
- Runs of the protocol
  - Set of all possible traces
    - Each tagged with random bits used to generate trace
    - Tagging $\Rightarrow$ set of equi-probable traces

Complexity-theoretic semantics

- Given protocol Q, adversary A, security parameter n, define
  - T=T(Q,A,n), set of all possible traces
  - [[φ]](T) a subset of T that respects φ in a specific way
- Intuition: φ valid when [[φ]](T) is an asymptotically overwhelming subset of T

Semantics of trace properties

- Defined in a straight forward way

$$[[\ \text{Send}(X, m)\ ]](T)$$

All traces $t \in T$ such that

- t contains a Send(msg) action by X
- the bistring value of msg is the bitstring value of m

Inductive Semantics

- $[[\ \varphi_1 \wedge \varphi_2\ ]]\ (T) = [[\ \varphi_1\ ]]\ (T) \cap [[\ \varphi_2\ ]]\ (T)$
- $[[\ \varphi_1 \vee \varphi_2\ ]]\ (T) = [[\ \varphi_1\ ]]\ (T) \cup [[\ \varphi_2\ ]]\ (T)$
- $[[\ \neg\ \varphi\ ]]\ (T) = T - [[\ \varphi\ ]]\ (T)$

Implication uses *form* of conditional probability

- $[[\varphi_1 \Rightarrow \varphi_2]]\ (T) = [[\neg\varphi_1]]\ (T)$

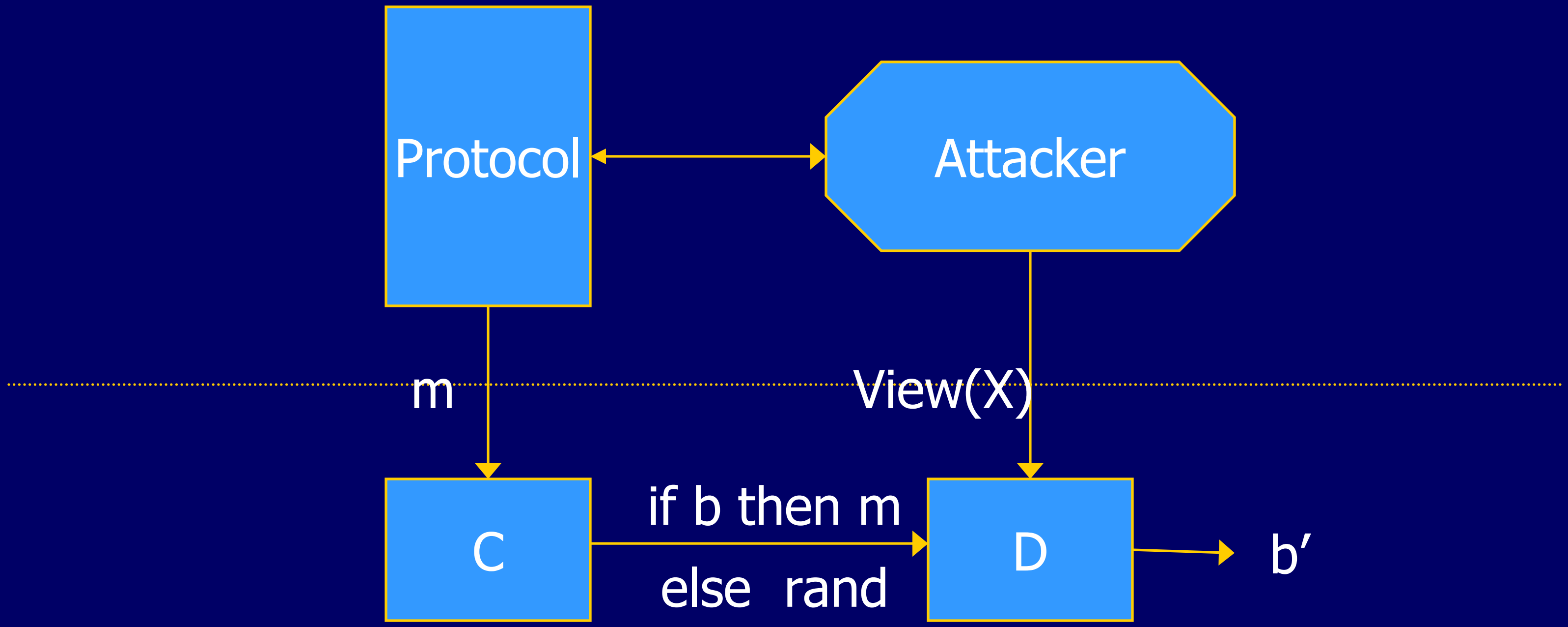
  $\cup\ [[\varphi_2]]\ (T')$

  where $T' = [[\varphi_1]]\ (T)$

This seems needed for reduction proofs. What is logic of this $\Rightarrow$?

Semantics of Indistinguishable

- Not a trace property
- Intuition: Indist(X, m) holds if no algorithm can distinguish m from a random value, given X's view of the run



$[[\text{Indist}(X, m)]]\ (T, D, \varepsilon) = T \text{ if } \left|\ \#(t: b=b') - |T|/2\ \right| < \varepsilon$

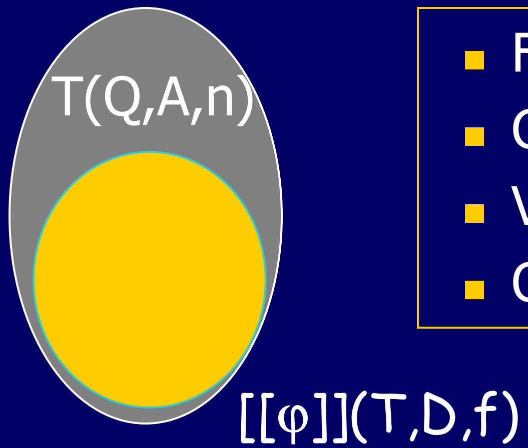
Validity of a formula

$Q \models \varphi$ if $\forall$ adversary A $\forall$ distinguisher D
$\exists$ negligible function f $\exists$ n_0 s.t. $\forall n > n_0$

$$|[[\varphi]](T,D,f(n))| \,/\, |T| > 1 - f(n)$$

Fraction of traces where "φ is true"

- Fix protocol Q, PPT adversary A
- Choose value of security parameter n
- Vary random bits used by all programs
- Obtain set T=T(Q,A,n) of equi-probable traces



Advantages of Computational PCL

- High-level reasoning, sound for "real crypto"
  - Prove properties of protocols without explicit reasoning about probability, asymptotic complexity
- Composability
  - PCL is *designed* for protocol composition
  - Composition of individual steps
    - Not just coarser composition available with UC/RSIM
- Can identify crypto assumptions needed
  - ISO-9798-3 [DDMW2006]

Note: existing deployed protocols may have weak security properties, assuming realistic but weak security properties of primitives they use

CPCL analysis of Kerberos V5

- Kerberos has a staged architecture
  - First stage generates a nonce and sends it encrypted
  - Second stage uses nonce as key to encrypt another nonce
  - Third stage uses second-stage nonce to encrypt other msgs
- Secrecy
  - Logic proves “GoodKey” property of both nonces
- Authentication
  - Proved assuming encryption provides *ciphertext integrity*
- Modular proofs using composition theorems
  - Applies to DHINIT, which is outside scope of competing approaches

Challenges for computational reasoning

- More complicated adversary
  - Actions of computational adversary do not have a simple inductive characterization
- More complicated messages
  - Computational messages are arbitrary sequences of bits, without an inductively defined syntactic structure
- Different scheduler
  - Simpler "non-preemptive" scheduling is typically used in computational models (change symbolic model for equiv)
- Power of induction ?
  - Indistinguishability, other non-trace-based properties appear unsuitable as inductive hypotheses
  - Solution: prove trace property inductively and derive secrecy

Current and Future Work

- Investigate nature of propositional fragment
  - Non-classical implication related to conditional probability
    - complexity-theoretic reductions
    - connections with probabilistic logics (e.g. Nilsson86)
- Generalize reasoning about secrecy
  - Work in progress, thanks to Arnab
  - Need to incorporate insight of "Rackoff's attack"
- Extend logic
  - More primitives: signature, hash functions,…
- Complete case studies
  - Produce correctness proofs for all widely deployed standards
- Collaborate on
  - Foundational work – please join us !
  - Implementation and case studies – please help us !

Conclusions

- Protocol design is tricky and error-prone
  - Model checking can find errors
  - Proof method can show correctness
- Modular analysis is a challenge
- Closing gap between logical analysis and cryptography
  - Symbolic model supports useful tools
  - Computational model more informative
    - Includes probability, complexity
    - Does not require strong cryptographic assumptions
  - Two approaches can be combined
    - Several current projects and approaches [BPW, MW, Blan, CH, …]
    - One example: computational semantics for symbolic protocol logic
- Research area coming of age
  - Interactions with and impact on industry

Credits

- **Collaborators**
  - M. Backes, A. Datta, A. Derek, N. Durgin, C. He, R. Kuesters, D. Pavlovic, A. Ramanathan, A. Roy, A. Scedrov, V. Shmatikov, M. Sundararajan, V. Teague, M. Turuani, B. Warinschi, …
- **More information**
  - Web page on Protocol Composition Logic
    - http://www.stanford.edu/~danupam/logic-derivation.html

Science is a social process