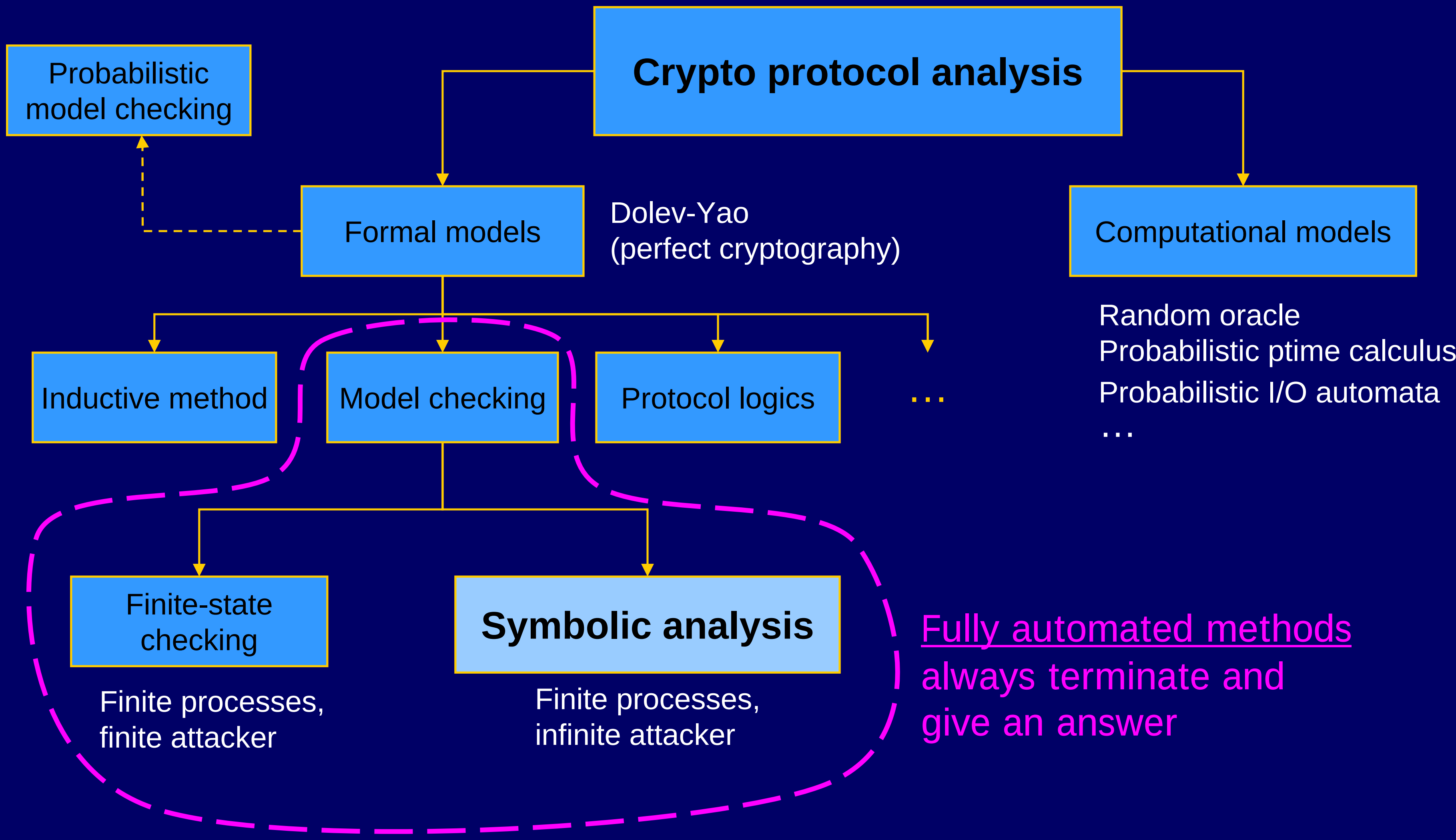
Symbolic Constraint Solving

Overview

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

Protocol Analysis Techniques



Obtaining a Finite Model

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

This restriction is necessary
(or the problem is undecidable)

This restriction is **not** necessary
for fully automated analysis!

Decidable Protocol Analysis

- Eliminate sources of undecidability
  - Bound the number of protocol sessions
    - Artificial bound, no guarantee of completeness
  - Bound structural size of messages by lazy instantiation of variables
  - Loops are simulated by multiple sessions
- Secrecy and authentication are NP-complete if the number of protocol instances is bounded

  [Rusinowitch, Turuani '01]
- Search for solutions can be fully automated
  - Several tools; we'll talk about SRI constraint solver

Strand Space Model

[Thayer, Herzog, Guttman '98]

- A strand is a representation of a protocol "role"
  - Sequence of "nodes"
  - Describes what a participant playing one side of the protocol must do according to protocol specification
- A node is an observable action
  - "+" node: sending a message
  - "-" node: receiving a message
- Messages are ground terms
  - Standard formalization of cryptographic operations: pairing, encryption, one-way functions, ...

Participant Roles in NSPK

Protocol

$A \rightarrow B \quad \{n,A\}_{kb}$

$B \rightarrow A \quad \{n,r\}_{ka}$

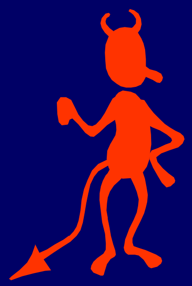
$A \rightarrow B \quad \{r\}_{kb}$

"A" role

$A \rightarrow \quad \{n,A\}_{kb}$

$A \leftarrow \quad \{n,r\}_{ka}$

$A \rightarrow \quad \{r\}_{kb}$



Controls network and can schedule any <u>consistent</u> interleaving of these roles

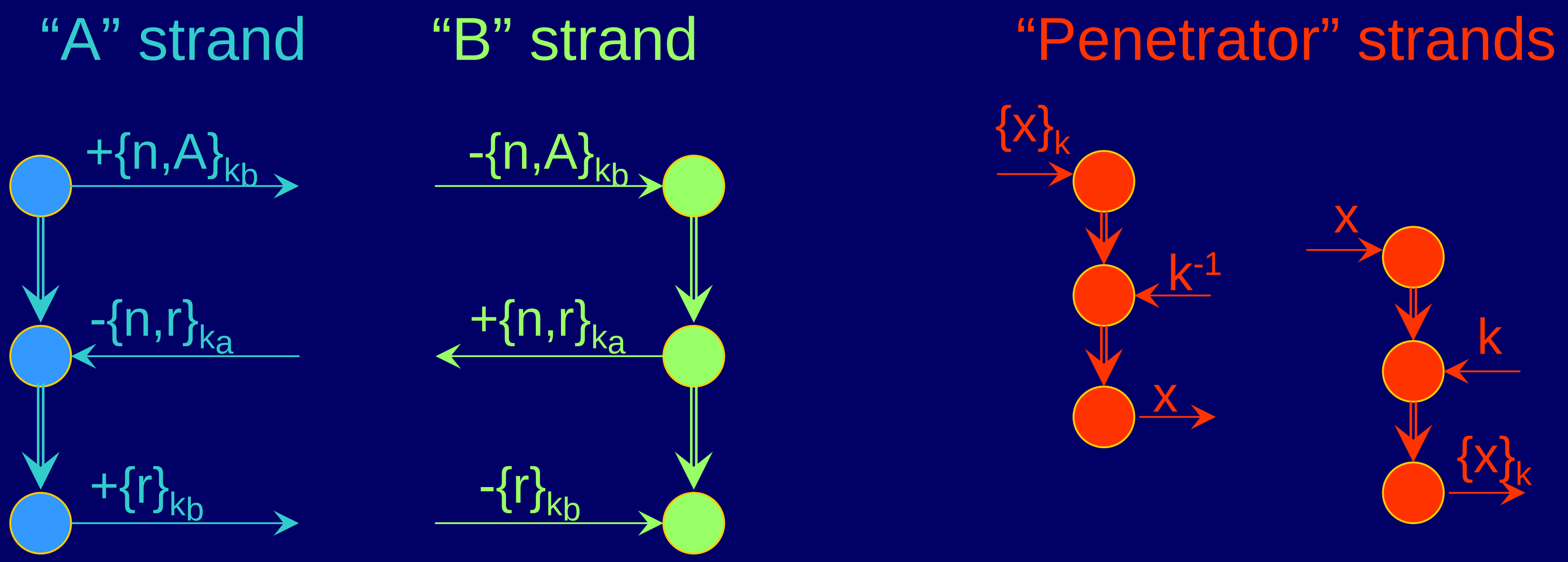
"B" role

$B \leftarrow \quad \{n,A\}_{kb}$

$B \rightarrow \quad \{n,r\}_{ka}$

$B \leftarrow \quad \{r\}_{kb}$

NSPK in Strand Space Model

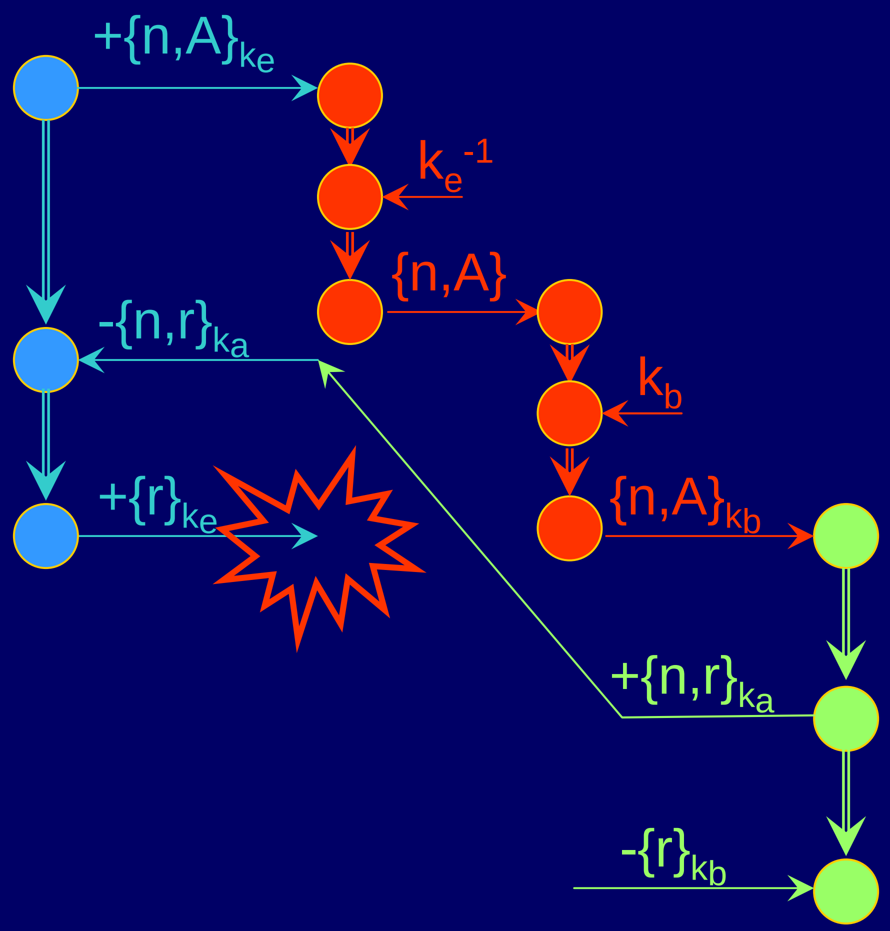


- Each primitive capability of the attacker is a "penetrator" strand
- Same set of attacker strands for every protocol

Bundles

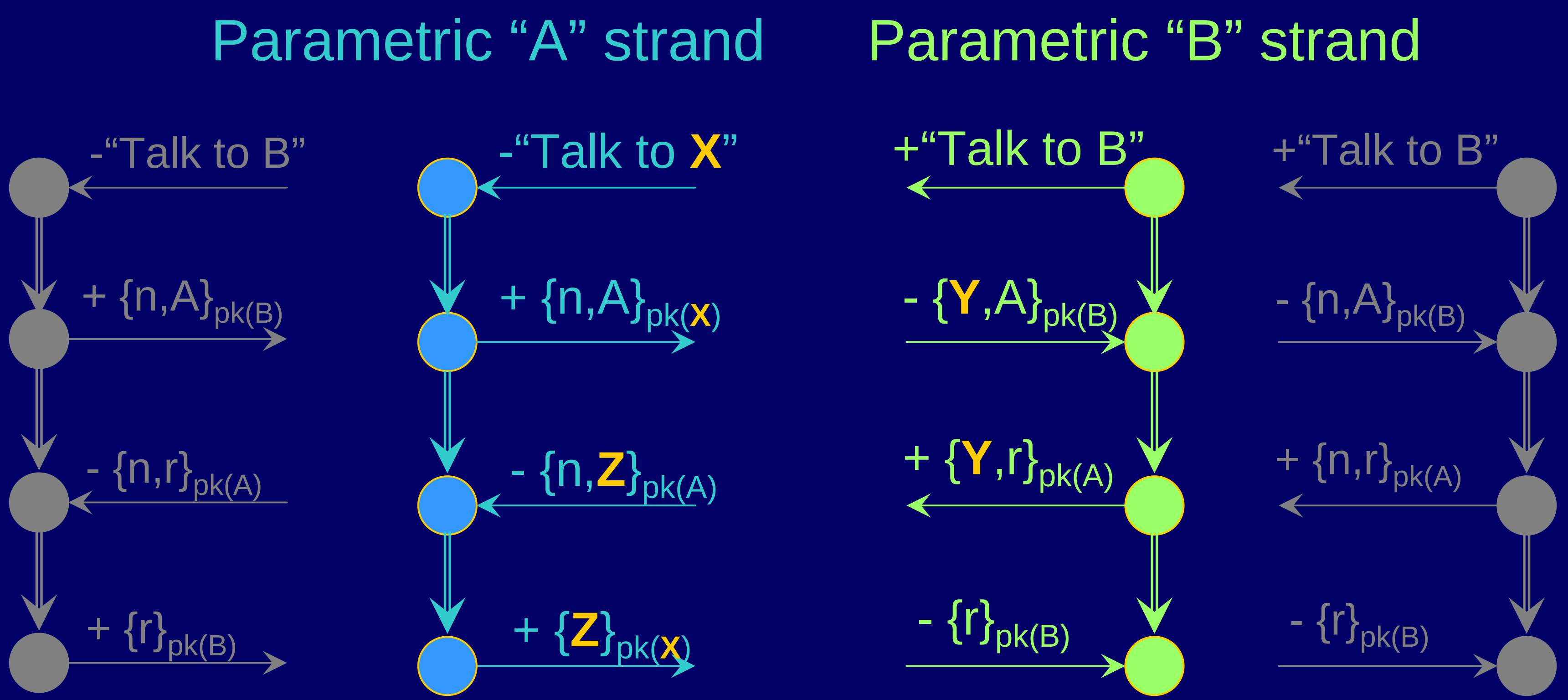
- A bundle combines strands into a partial ordering
  - Nodes are ordered by internal strand order
  - "Send message" nodes of one strand are matched up with "receive message" nodes of another strand
- Infinitely many possible bundles for any given set of strands
  - No bound on the number of times any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

NSPK Attack Bundle



Parametric Strands

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

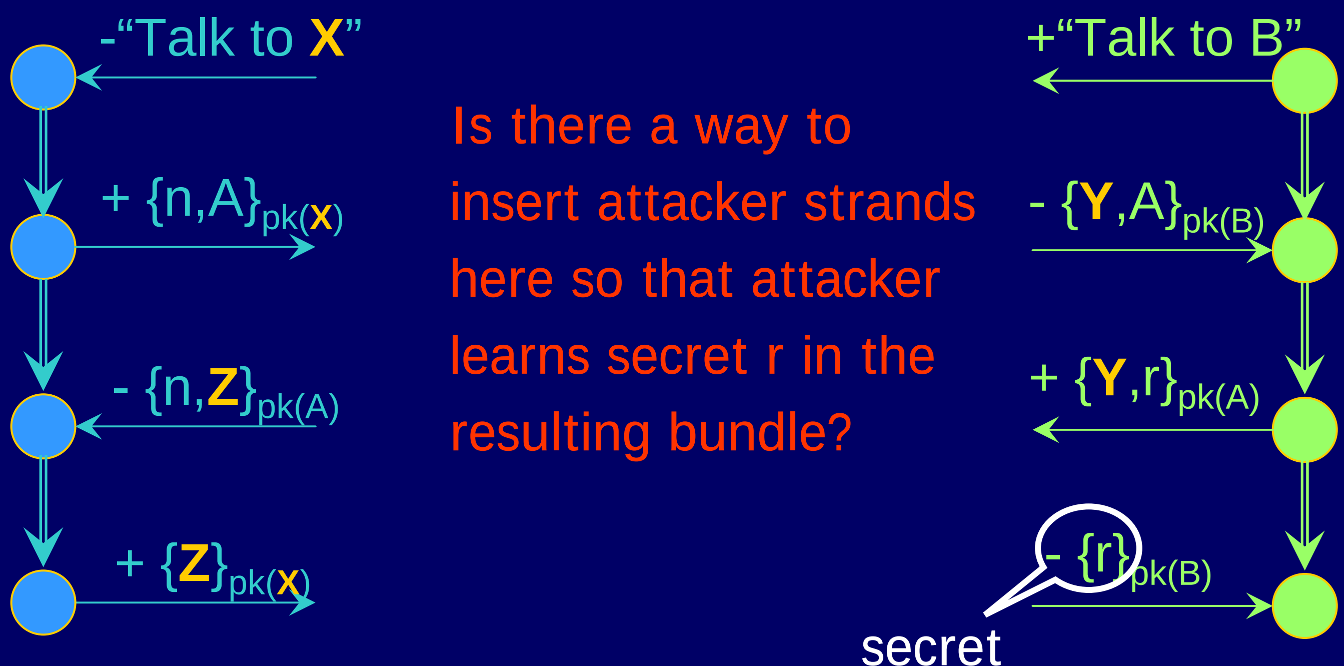


Properties of Parametric Strands

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

Attack Scenario

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



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

Attack Scenario Generation

- Choose a finite number of strands
- Try all combinations respecting partial order imposed by individual strands
  - If node L appears after node K in the same strand, then L must appear after K in the combination bundle
  - Two strands of size m & n $\Rightarrow$ choose(m+n,n) variants
- Optimization to reduce number of variants
  - The order of “send message” nodes doesn’t matter: attacker will intercept all sent messages anyway
  - If this is the only difference between two combinations, throw one of them away

Attack Scenario: Example

A's role

$A \leftarrow$ "Talk to X"
$A \rightarrow \{n,A\}_{pk(X)}$
$A \leftarrow \{n,Z\}_{pk(A)}$
$A \rightarrow \{Z\}_{pk(X)}$

B's role

$B \rightarrow$ "Talk to B"
$B \leftarrow \{A,Y\}_{pk(B)}$
$B \rightarrow \{Y,r\}_{pk(A)}$
$B \leftarrow \{r\}_{pk(B)}$

Try all possible ways to plug attacker in the middle, for example:

$B \rightarrow E$ "Talk to B"
$A \leftarrow E$ "Talk to X"
$A \rightarrow E \quad \{n,A\}_{pk(X)}$
$B \leftarrow E \quad \{A,Y\}_{pk(B)}$
$B \rightarrow E \quad \{Y,r\}_{pk(A)}$
$A \leftarrow E \quad \{n,Z\}_{pk(A)}$
$A \rightarrow E \quad \{Z\}_{pk(X)}$
$\leftarrow E \quad r$

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

Symbolic Attack Scenarios

- Attack is modeled as a symbolic execution trace
  - Trace is a sequence of message send and receive events
  - Attack trace ends in a violation
    - E.g., attacker outputs the secret
  - Messages contain variables
    - Variables represent data controlled by attacker
- Adequate for trace-based security properties
  - Secrecy, authentication, some forms of fairness...
- A symbolic trace may or may not have a feasible concrete instantiation
  - Goal: discover whether a feasible instantiation exists

From Attack Traces to Constraints

- Any symbolic execution trace is equivalent to a sequence of symbolic constraints

$\boxed{m \ \texttt{from}\ t_1, \ldots, t_n}$

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

- A constraint is satisfiable if and only if m can be derived from $t_1, \ldots, t_n$ in attacker term algebra
  - Attacker term algebra is an abstract representation of what the attacker can do

Constraint Generation: Example

Attack Trace

$B \to E$ "Talk to B"
$A \leftarrow E$ "Talk to X"
$A \to E$ $\{n,A\}_{pk(X)}$
$B \leftarrow E$ $\{A,Y\}_{pk(B)}$
$B \to E$ $\{Y,r\}_{pk(A)}$
$A \leftarrow E$ $\{n,Z\}_{pk(A)}$
$A \to E$ $\{Z\}_{pk(X)}$
$\leftarrow E$ r

Symbolic Constraints

"Talk to X" **from** T_0 (attacker's initial knowledge)

$\{A,Y\}_{pk(B)}$ **from** T_0, $\{n,A\}_{pk(X)}$

$\{n,Z\}_{pk(A)}$ **from** T_0, $\{n,A\}_{pk(X)}$, $\{Y,r\}_{pk(A)}$

r **from** T_0, $\{n,A\}_{pk(X)}$, $\{Y,r\}_{pk(A)}$, $\{Z\}_{pk(X)}$

Symbolic Constraint Generation

- For each message sent by the attacker in the attack trace, create symbolic constraint

$$\boxed{m_i \ \mathbf{from}\ t_1, \ldots, t_n}$$

  - m_i is the message attacker needs to send
  - $t_1,\ldots,t_n$ are the messages observed by attacker up to this point (may contain variables)
- Attack is feasible if and only if all constraints are satisfiable simultaneously
  - There exists an instantiation σ such that $\forall i\ m_i\sigma$ can be derived from $t_1\sigma, \ldots, t_n\sigma$ in attacker's term algebra

Dolev-Yao Term Algebra

Attacker's term algebra is a set of derivation rules

$$\frac{v \in T}{T \triangleright u} \text{ if } u = v\sigma \text{ for some } \sigma \qquad \frac{T \triangleright u \quad T \triangleright v}{T \triangleright [u,v]} \qquad \frac{T \triangleright u \qquad T \triangleright v}{T \triangleright \mathrm{crypt}_u[v]}$$

$$\frac{T \triangleright [u,v]}{T \triangleright u} \qquad \frac{T \triangleright [u,v]}{T \triangleright v} \qquad \frac{T \triangleright \mathrm{crypt}_u[v] \quad T \triangleright u}{T \triangleright v}$$

Symbolic constraint m **from** $t_1, \ldots, t_n$ is satisfiable

if and only if there is a substitution σ such that

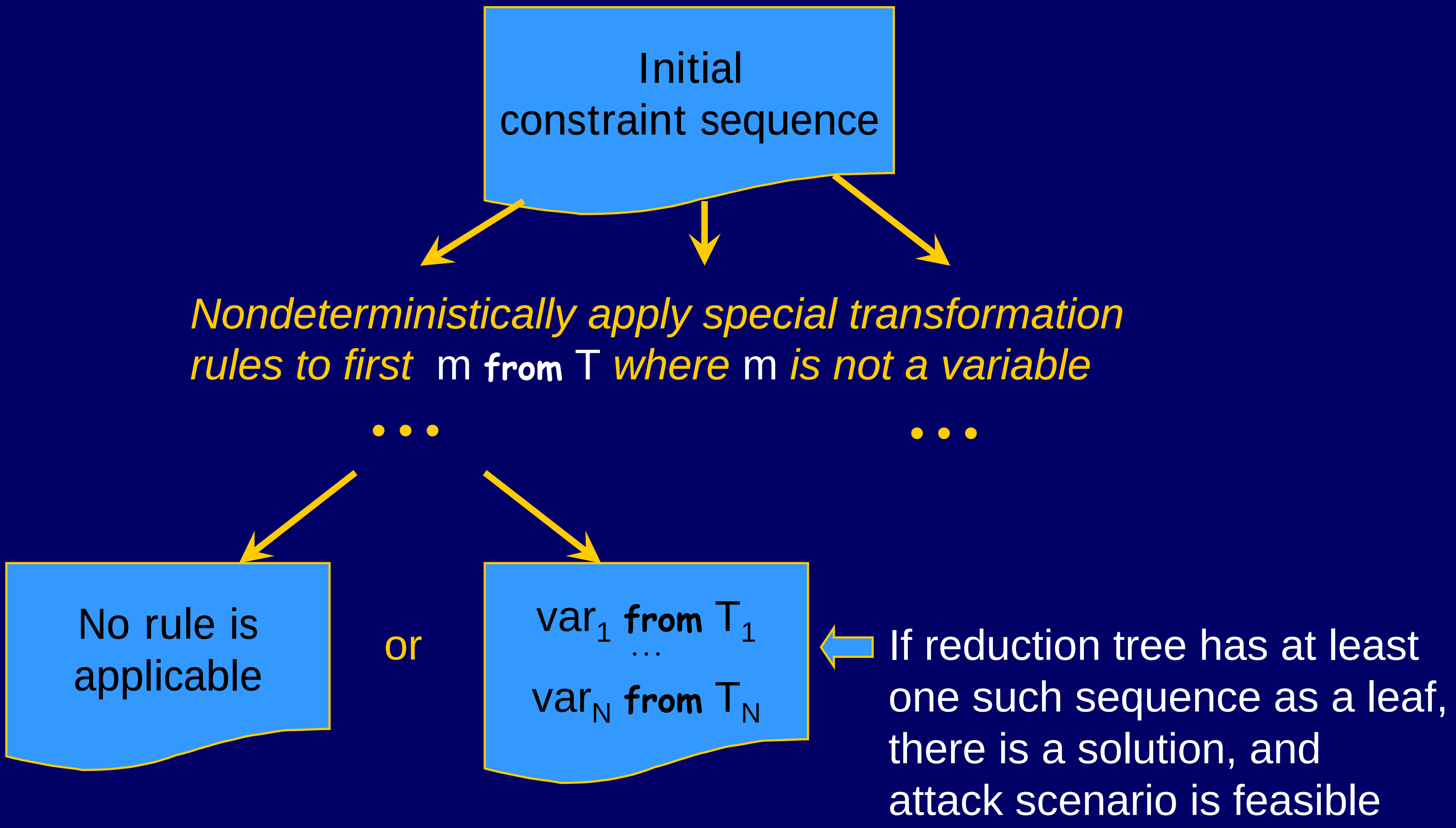
$t_1\sigma, \ldots, t_n\sigma \triangleright m\sigma$ is derivable using these rules

Solving Symbolic Constraints

[Millen and Shmatikov CCS '01]

- Constraint reduction rules
  - Replace each m_i **`from`** T_i with one or more simpler constraints
  - Preserve essential properties of the constraint sequence
- Nondeterministic reduction procedure
  - Structure-driven, but several rules may apply in any state
  - Exponential in the worst case (the problem is NP-complete)
- The procedure is terminating and <u>complete</u>
  - If $T\sigma \triangleright m\sigma$ is derivable in attacker's term algebra,
    1. There exists reduction rule $r=r(\sigma)$ which is applicable to m **`from`** T and produces some m' **`from`** T' such that
    2. $T'\sigma \triangleright m'\sigma$ is derivable in attacker's term algebra

Reduction Procedure



From Protocols to Constraints

Formal specification of protocol roles

This is the only thing the user has to specify!

↓

Choose finite number of role instances

↓

Choose an interleaving corresponding to an attack

contains variables & may not have a feasible instantiation

↓

Sequence of symbolic constraints

satisfiable ⇔ there exists a feasible instantiation

↓

Constraint solving procedure

SRI Constraint Solver

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

http://www.csl.sri.com/users/millen/capsl/constraints.html

A Tiny Bit of Prolog (I)

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

A Tiny Bit of Prolog (II)

- Clauses define terms as relations or predicates
  - factorial(1,1). *Fact, true as given*
  - factorial(N,M) :- *...is true if...*

```
      N>1,               condition for this case
     N1 is N-1,          "is" to do arithmetic
     factorial(N1,M1),   recursive call to find (N-1)!
     M is N*M1.          M = N! = N(N-1)!
```

Using Prolog

- Put definitions in a text file .../factdef or ...\factdef.pl
- Start Prolog swipl, pl or plwin.exe

  ?- *Prolog prompt*
- Load definitions file

  ?- reconsult(factdef). consult(factdef) *in SWI-Prolog*

  ?- [factdef]. *Both UNIX and Windows*

  ?- ['examples/factdef']. *subdirectory, need quotes*
- Execute query

  ?- factorial(3,M). *Start search for true instance*

  M=6 *Prolog responds*

  Yes

  ?- halt. *Quit Protocol session.*

Defining a Protocol: Terms

- Constants
  - a, b, e, na, k, ... *e is the name of the attacker*
- Variables
  - A, M, ... *by convention, names capitalized*
- Compound terms
  - [A,B,C] *n-ary concatenation, for all n > 1*
  - A+K *symmetric encryption*
  - A*pk(B) *public-key encryption*
  - sha(X) *hash function*
  - f(X,Y) *new function unknown to attacker*

Specifying Protocol Roles

Name of the role

Parameters of the role

```
strand(roleA,A,B,Na,Nb,[
  send([A,Na]*pk(B)),
  recv([Na,Nb]*pk(A)),
  send(Nb*pk(B))
]).

strand(roleB,A,B,Na,Nb,[
  recv([A,Na]*pk(B)),
  send([Na,Nb]*pk(A)),
  recv(Nb*pk(B))
]).
```

$A \rightarrow B$: $\{A,Na\}_{pk(B)}$

$B \rightarrow A$: $\{Na,Nb\}_{pk(A)}$

$A \rightarrow B$: $\{Nb\}_{pk(B)}$

Sending and receiving messages (just like in Murφ)

- No need to specify rules for the intruder
- No need to check that messages have correct format

Specifying Secrecy Condition

- Special secrecy test strand

Forces analysis to stop as soon as this strand is executed

```
strand(secrecytest,X,[recv(X),send(stop)]).
```

- When the attacker has learned the secret, he'll pass it to this strand to "announce" that the attack has succeeded

Choosing Number of Sessions

- Choose number of instances for each role
  - For example, one sender and two recipients
- In each instance, use different constants to instantiate nonces and keys created by that role

```
nspk0([Sa,Sb1,Sb2]) :-
      strand(roleA,a,B1,na,Nb,Sa),
      strand(roleB,a,b, Na1,nb1,Sb1),
      strand(roleB,A3,b,Na2,nb2,Sb2).
```

1 instance of role A,
2 instances of role B

Each nonce modeled by a separate constant

Each instance has its own name

Verifying Secrecy

- Add secrecy test strand to the bundle

```
nspk0([Sa,Sb1,Sb2,St]) :-
        strand(roleA,a,B1,na,Nb,Sa),
        strand(roleB,a,b,Na1,nb1,Sb1),
        strand(roleB,A3,b,Na2,nb2,Sb2),
        strand(secrecytest,nb1,St).
```

- This bundle is solvable if and only if the attacker can learn secret nb1 and pass it to test strand
- Run the constraint solver to find out

```
:- nspk0(B),search(B,[]).
```

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

Specifying Authentication Condition

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

  `send(roleA(a,b,nb))`

  "A believes he is talking to B and B's nonce is nb"
- Attack succeeds if B completes protocol, but A's doesn't send authentication message
  - B thinks he is talking to A, but not vice versa

NSPK Strands for Authentication

```
strand(roleA,A,B,Na,Nb,[
  send([A,Na]*pk(B)),
  recv([Na,Nb]*pk(A)),
  send(roleA(A,B,Nb)),
  send(Nb*pk(B))
]).
```

A announces who he thinks he is talking to

```
strand(roleB,A,B,Na,Nb,[
  recv([A,Na]*pk(B)),
  send([Na,Nb]*pk(A)),
  recv(Nb*pk(B)),
  send(roleB(A,B,Na))
]).
```

B announces who he thinks he is talking to

Verifying Authentication

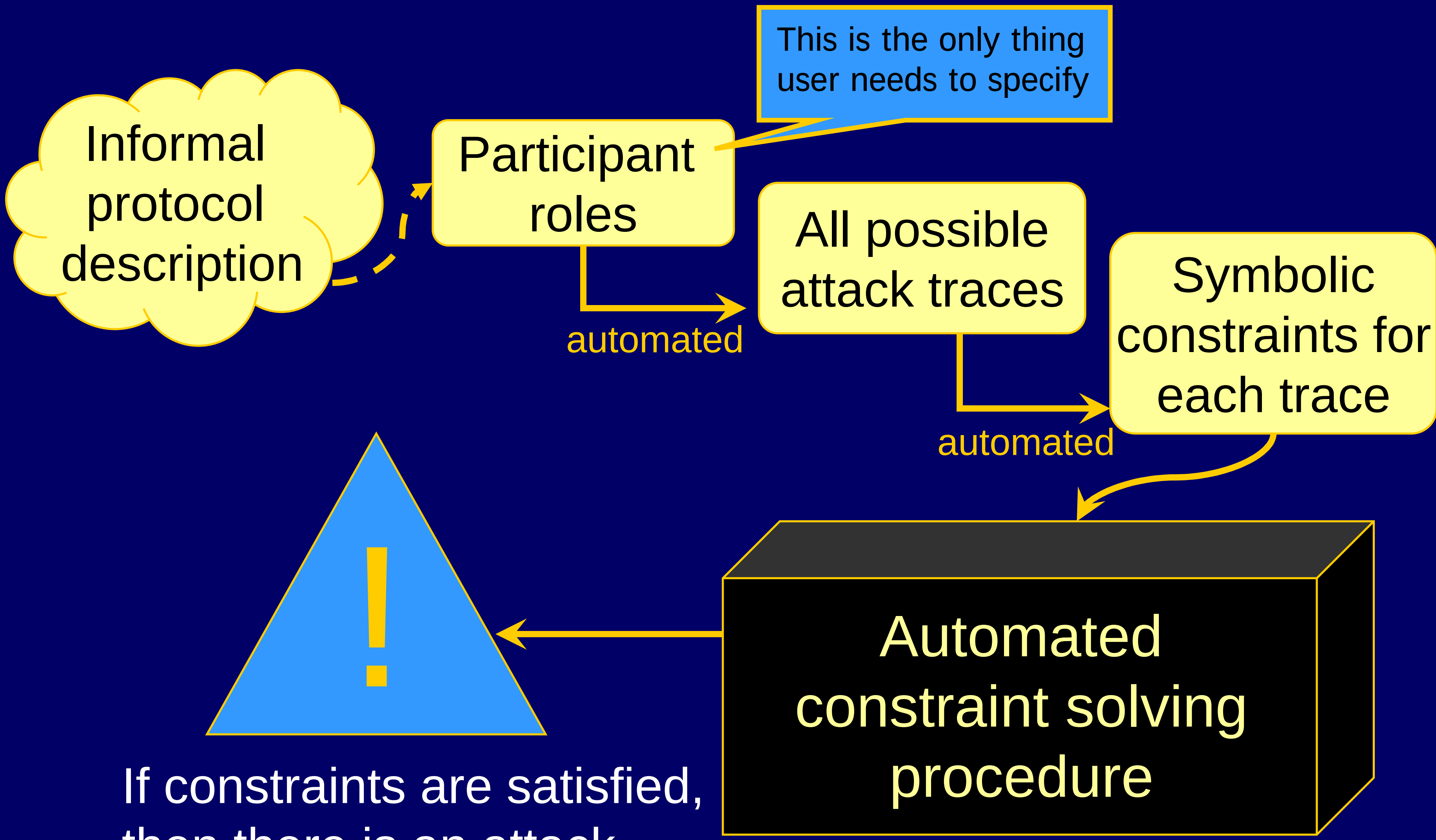
- Test for presence of authentication message

Only look at bundles where this message doesn't occur

```
nspk0([Sa,Sb,St],roleA(a,b,nb)) :-
      strand(roleA,a,B,na,Nb,Sa),
      strand(roleB,a,b,Na,nb,Sb),
      strand(secrecytest,roleB(a,b,na),St).
```

- This bundle is solvable if and only if the attacker can cause `roleB(a,b,na)` to appear in a trace that does <u>not</u> contain `roleA(a,b,nb)`
  - Convince B that he is talking A when A does not think he is talking to B.

Symbolic Analysis in a Nutshell



If constraints are satisfied, then there is an attack