02244 Language-Based Security
Security Protocols
Automated Analysis: Introduction and The Lazy Intruder

Sebastian Mödersheim

April 15, 2018

Automated Verification and Decidability

We would like to have a program V with ...

- Input:
  - some description of a program P
  - some description of a specification: a set S of functions
- Output: *Yes* if P computes a function in S, and *No* otherwise.

Automated Verification and Decidability

We would like to have a program V with ...

- Input:
  - some description of a program P
  - some description of a specification: a set S of functions
- Output: *Yes* if P computes a function in S, and *No* otherwise.

Forget it:

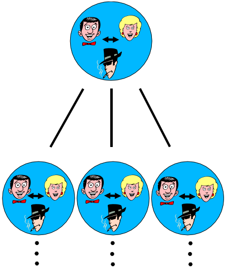
Theorem (Rice)

Let S be any non-empty, proper subset of the computable functions. Then the verification problem for S (the set of programs P that compute a function in S) is undecidable.

Recall: Transition Systems

We can now define an abstract protocol world:

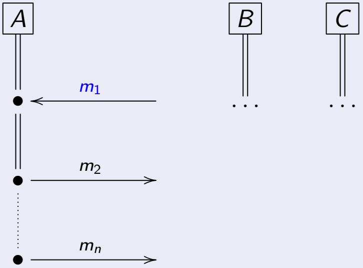
- It has an initial state
- There are several transitions, i.e., ways the world can evolve from one state into a different state
  - ★ An honest agent sending a message
  - ★ An honest agent receiving a message
  - ★ An honest agent checking a condition
  - ★ ...
- Every state consists of
  - ★ Local states of the honest agents
  - ★ The knowledge of the intruder
  - ★ Special events that we use to formulate the goals/attack states
- Define which states count as attack states



Sources of Infinity

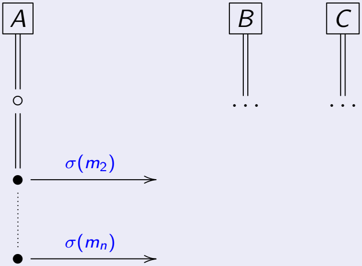
Unbounded Messages

Current State



Possible Next State

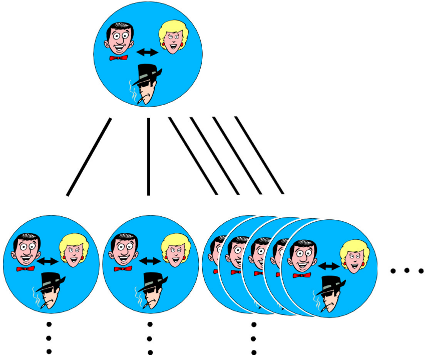
if σ substitutes all variables of m_1 such that $M \vdash \sigma(m_1)$.



- In general there are infinitely many σ such that $M \vdash \sigma(m_1)$

Sources of Infinity

Unbounded Messages



- In general, a single strand can cause infinitely many successor states.
- Can we somehow avoid that?

- If there are finitely many strands/sessions, there is no infinite path in this tree.
- In general there is no bound on the number of protocol sessions.
- If the initial state contains infinitely many strands, then the tree can be both infinitely deep and wide.

Sources of Infinity

Number of Sessions and Constants

- If there are finitely many strands/sessions, there is no infinite path in this tree.
- In general there is no bound on the number of protocol sessions.
- If the initial state contains infinitely many strands, then the tree can be both infinitely deep and wide.
- Additionally, when admitting an infinite number of sessions, we may have an unbounded number of fresh constants.

The Sources of Infinity

- For security protocols, the state space can be infinite for (at least) the following reasons:

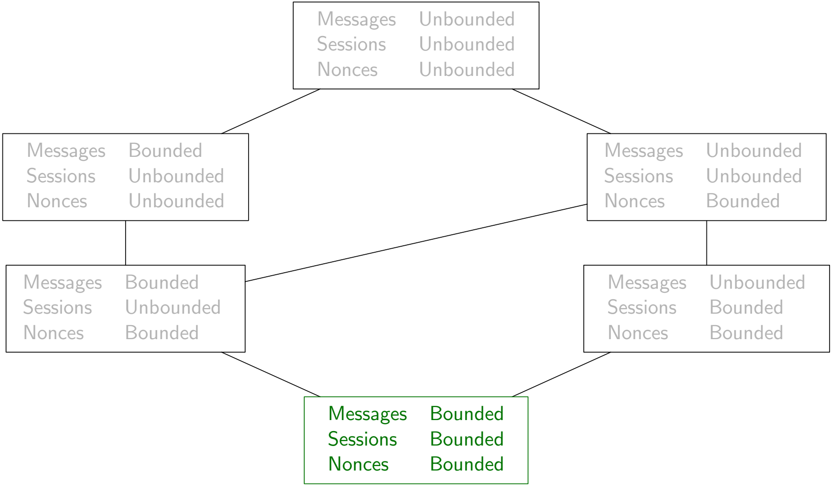
  Messages The intruder can compose arbitrarily complex messages from his knowledge, e.g. $i, h(i), h(h(i)), \ldots$.

  Sessions No bound on the number of executions of the protocol. (In our model: infinitely many threads in the initial state).

  Nonces In an unbounded number of sessions, honest agents create an infinite number of fresh nonces.

- Consider the models that arise from bounding any subset of these parameters:
  - ★ Decidability/Automation?
  - ★ Can we justify the bounds?

Decidability Lattice



⋆ Undecidability

Idea: give a reduction from an undecidable problem like PCP to protocol verification:

Definition (Post's Correspondence Problem)

Input Finite sequence of pairs of strings $(s_1, t_1), \dots, (s_n, t_n)$

Output Yes if there is a finite sequence of indices $i_1, \dots, i_k \in \{1, \dots, n\}$ such that $s_{i_1} \dots s_{i_k} = t_{i_1} \dots t_{i_k}$; and No otherwise.

Example

The correspondence problem

$$\begin{array}{lll} s_1 = 1 & s_2 = 10 & s_3 = 011 \\ t_1 = 101 & t_2 = 00 & t_3 = 11 \end{array}$$

Has a solution: $s_1 s_3 s_2 s_3 = 101110011 = t_1 t_3 t_2 t_3$.

⋆ Reducing PCP to Protocol Security

[Even and Goldreich 1983]

Every PCP problem can be translated to the following protocol:

- The intruder generates a sequence of indices
- Honest agents compute the two strings that correspond to the sequence.
- If the two strings are equal, they give the intruder a secret s.
- Goal: secrecy of s

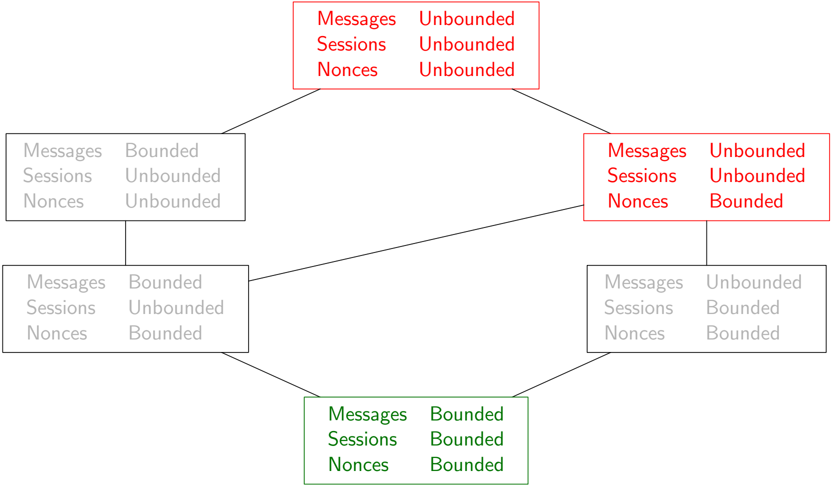
This protocol has an attack, iff the given PCP has a solution.

Thus: if protocol security were decidable, then also PCP would be — which is we know is not the case.

Protocol security is undecidable, even if participants do not generate fresh nonces.

- The construction however requires an unbounded number of sessions and unbounded depth of terms.

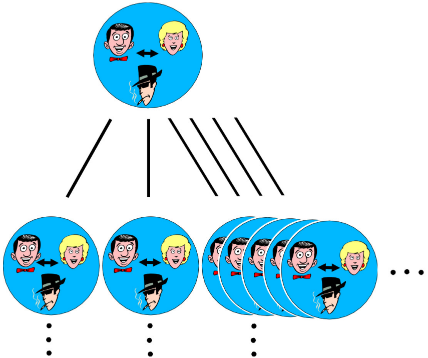
Decidability Lattice



In the next two lectures, we will further fill this lattice.

Sources of Infinity

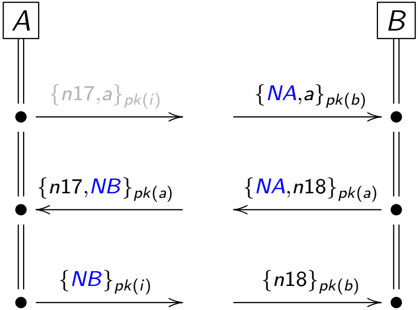
Unbounded Messages



- In general, a single strand can cause infinitely many successor states.
  - ★ Infinitely many messages the intruder can construct.
- Even bounding the messages, this is not efficient.
- Can we somehow avoid that?

Example From Last Week

Last week for NSPK we got into the following state:

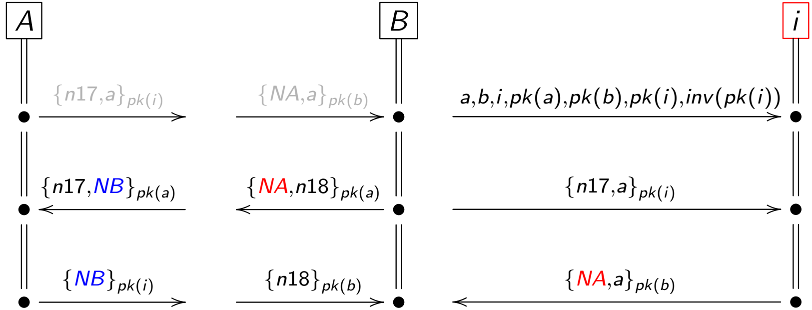


Intruder knowledge:

$M = \{a, b, i, pk(a), pk(b), pk(i), inv(pk(i)), \{n17, a\}_{pk(i)}\}$

- What messages can the intruder send to B?
- Find all solutions σ for $M \vdash \sigma(\{NA, a\}_{pk(b)})$.
- There are infinitely many such σ.
- Idea: let's not go through that and use a constraint instead.

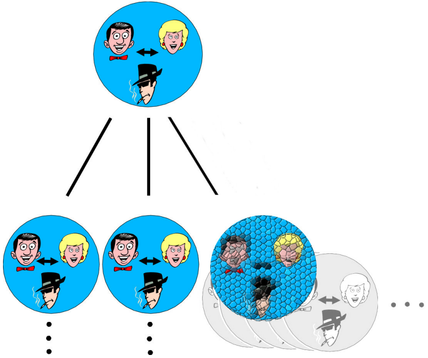
Idea: A Symbolic State with Constraints



Meaning of the intruder strand:

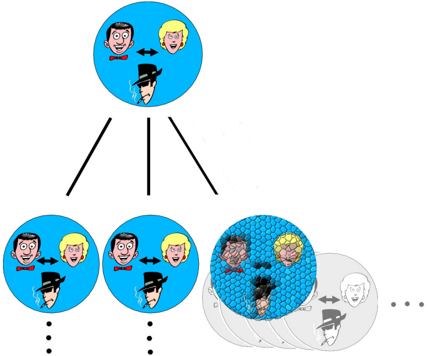
- Incoming messages: messages that the intruder has learned
- Outgoing messages: messages that the intruder has produced
- Represents all solutions σ of the free variables such that:
  - ★ the intruder can generate every outgoing message (under σ) when knowing all previous incoming messages (under σ)
  - ★ we are just lazily procrastinating that choice of σ!

Idea: Symbolic Representation



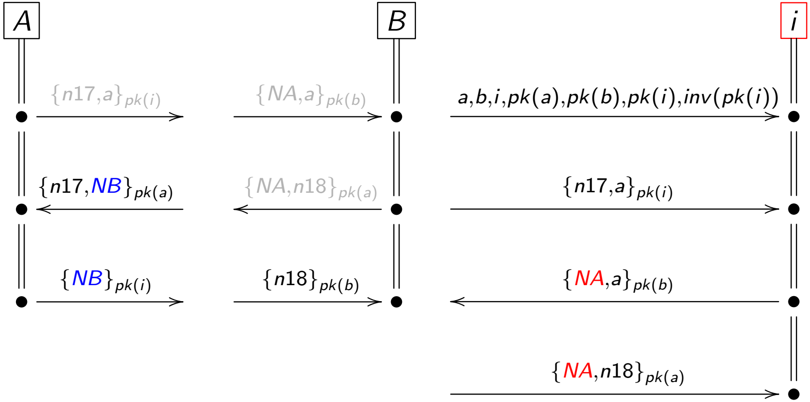
- Instead of infinitely many successor states, just have one symbolic states that represent the infinitely many concrete states.

Idea: Symbolic Representation



- Instead of infinitely many successor states, just have one symbolic states that represent the infinitely many concrete states.
- Lazy: we postpone the decision what concrete message the intruder sends.

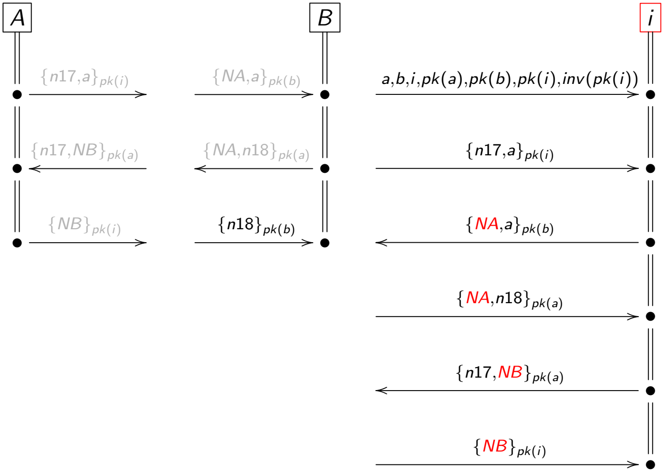
Transitions on Symbolic States – Example



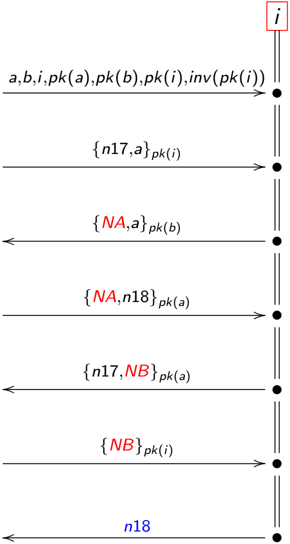
What happened here:

- the intruder learned an answer from B to his message.
- this answer now contains the free variable NA, i.e., depends on what the intruder sends before.

If the intruder now talks to A and gets an answer:

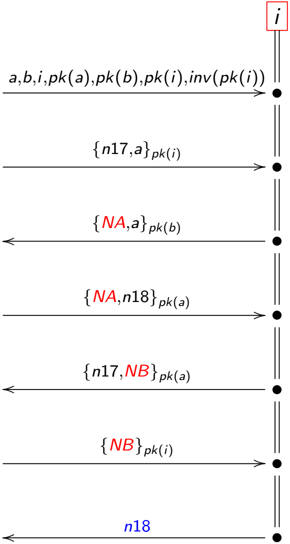


- To check whether the intruder can produce the secret $n18$ right now:
- We put the constraint that the intruder can generate $n18$.
- Secrecy is violated, if we can find a solution for this constraint.



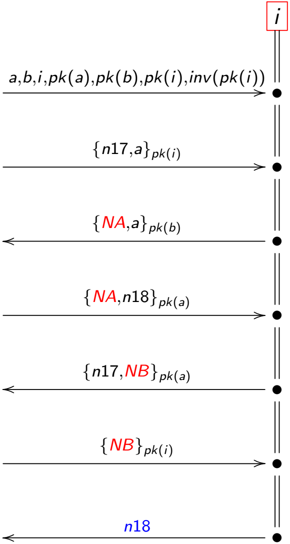
Goals on Symbolic States – Example

- To check whether the intruder can produce the secret $n18$ right now:
- We put the constraint that the intruder can generate $n18$.
- Secrecy is violated, if we can find a solution for this constraint.
- There is only one such solution: $\sigma(NA) = n17$ and $\sigma(NB) = n18$.



Goals on Symbolic States – Example

- To check whether the intruder can produce the secret $n18$ right now:
- We put the constraint that the intruder can generate $n18$.
- Secrecy is violated, if we can find a solution for this constraint.
- There is only one such solution: $\sigma(NA) = n17$ and $\sigma(NB) = n18$.
- We give a procedure below to solve such constraints.

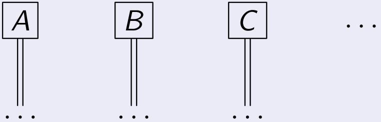


Roadmap for the Lazy Intruder

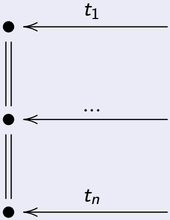
- Definition of Symbolic Transition Systems
- Meaning of Intruder Constraints
- Solving Procedure for Constraints
  - ★ Example: solving the NSPK example constraint

Initial State

Honest Agents:



Intruder Strand:



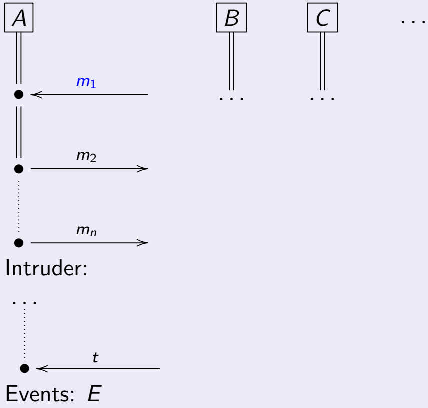
Events $\emptyset$

where $t_1, \ldots, t_n$ are the messages initially known to the intruder

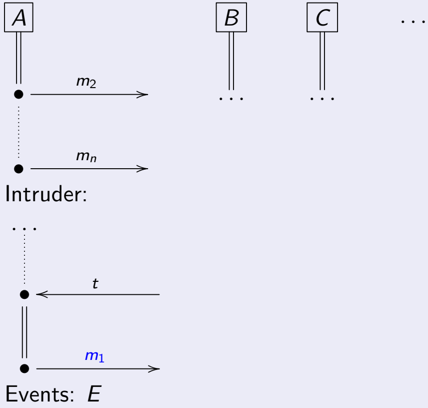
A Symbolic Transition System

Transition: honest agents receiving

Current State



Possible Next State

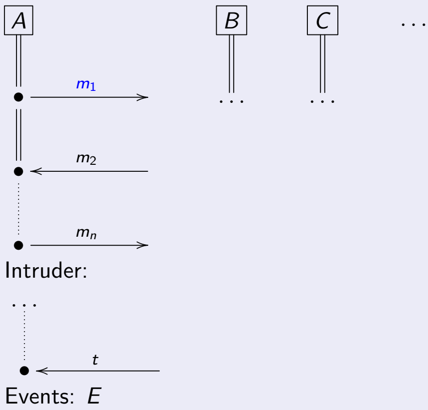


Note that we do not subsitute the variables of m_1.

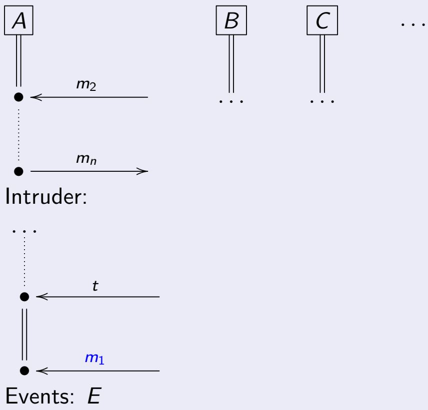
A Symbolic Transition System

Transition: honest agents sending

Current State



Possible Next State



Just as before: the intruder immediately learns the sent message m_1.
Note it may contain variables now.

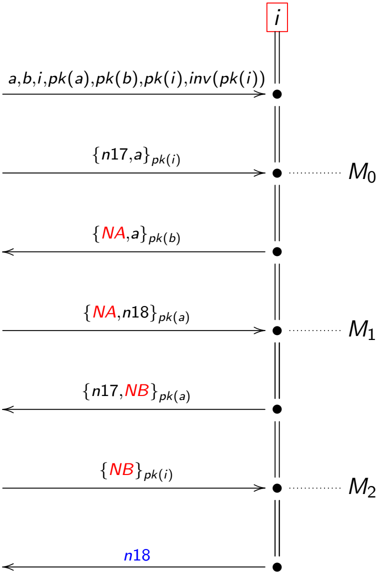
Meaning of Intruder Constraints

Definition

An intruder constraint is a strand with only free variables, i.e., all variables first occur in an outgoing message.
At any point $\bullet$ in the constraint, the intruder knowledge at that point is the set of all messages received so far.
Given an intruder constraint C, and σ a substitution of all its free variables with ground terms. Then σ is called a solution of C iff:

- for every outgoing message m of C, it holds that $\sigma(M) \vdash \sigma(m)$ where M is the intruder knowledge at that point.

Meaning of Intruder Constraints – Example



Any solution σ such that:

$$
\begin{aligned}
M_0 &:= \{a, b, i, pk(a), pk(b), pk(i), \\
&\qquad inv(pk(i)), \{n17, a\}_{pk(i)}\} \\
\sigma(M_0) &\vdash \sigma(\{NA, a\}_{pk(b)}) \\
M_1 &:= M_0 \cup \{\{NA, n18\}_{pk(a)}\} \\
\sigma(M_1) &\vdash \sigma(\{n17, NB\}_{pk(a)}) \\
M_2 &:= M_1 \cup \{\{NB\}_{pk(i)}\} \\
\sigma(M_2) &\vdash \sigma(n18)
\end{aligned}
$$

Solving Constraints

We give a procedure for solving constraints with rules of the form:

$$\boxed{\text{S}} \rightsquigarrow \boxed{\text{S'}}$$

Which means:

- to solve S, one way is to try to solve S'.

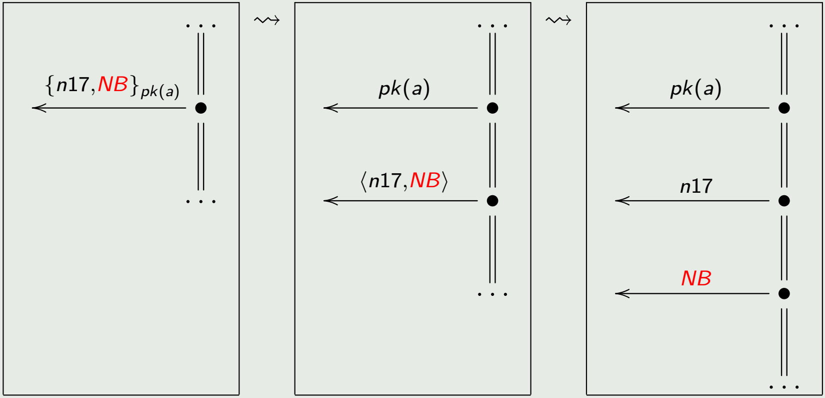
For a given constraint S_0, compute the following tree:

- The root node is the constraint S_0
- Every node S has as children the strands that are reachable in one step with $\rightsquigarrow$:
  - ★ i.e., if $S \rightsquigarrow S'$ then S' is a child of S.
- It will be ensured that this tree is finite.
- Every leaf of the tree is either simple or unsolvable. (Explained below, easy to check.)
- The root has a solution iff any leaf has a solution.

Solving Constraints: (I) Composition

Composing

To construct an outgoing message of the form $f(t_1, \ldots, t_n)$ it is sufficient that f is a public symbol and the intruder can construct the submessages t_i:



Solving Constraints: (I) Composition

Definition (Lazy Intruder Composition Rule)

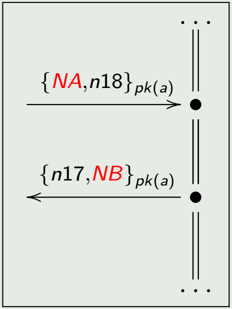
$$S_1.\mathrm{Snd}(f(t_1,\ldots,t_n)).S_2 \rightsquigarrow S_1.\mathrm{Snd}(t_1).\ldots.\mathrm{Snd}(t_n).S_2$$

if $f/n \in \Sigma_p$, i.e., f is a public function symbol.

Solving Constraints: (II) Unification

Unification

Another way to construct an outgoing message is to use a previously received message that has the right "shape".

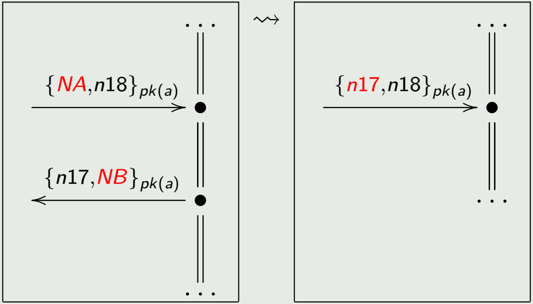


These messages can be unified under unifier
$\sigma(NA) = n17$ and
$\sigma(NB) = n18$.

The unifier has to be applied to the entire intruder constraint.

Unification

Another way to construct an outgoing message is to use a previously received message that has the right "shape".



For that we need a well-known algorithm: computing the most general unifier.

mgu: Computing the Most General Unifier

Definition (Unification)

A unification problem is a set $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ of pairs of terms.
A unifier σ for this unification problem is a substitution such that

$$\sigma(s_1) = \sigma(t_1) \text{ and } \ldots \text{ and } \sigma(s_n) = \sigma(t_n) .$$

There is a unification algorithm that always produces the most general unifier, if any unifier exists, and otherwise returns failure.

Unification Algorithm (Free Algebra)

Definition (Algorithm $mgu(U, \sigma)$)

Input: a unification problem U, a substitution σ (initially empty).
Output: A substitution or answer failure.

If $U = \emptyset$ then return σ. Otherwise pick any pair (s, t) in U and

- if $s = t$, return $mgu(U \setminus \{(s, t)\}, \sigma)$.
- if s is a variable:
  - if $s \in vars(t)$: return failure
  - otherwise: let σ' be the extension of σ with $[s \mapsto t]$ and return $mgu(\sigma'(U \setminus \{(s, t)\}), \sigma')$
- if t is a variable: analogous to previous case
- otherwise, i.e., $s = f(s_1, \ldots, s_n)$ and $t = g(t_1, \ldots, t_m)$:
  - if $f \neq g$: return failure.
  - if $f = g$ (and thus $n = m$): return $mgu(U \setminus \{(s, t)\} \cup \{(s_1, t_1), \ldots, (s_n, t_n)\}, \sigma)$.

We sometimes just write $mgu(s, t)$ for $mgu(\{(s, t)\}, [])$

mgu: Example

$$\begin{aligned} & mgu(\{\ (\{NA, n18\}_{pk(a)}\ ,\ \{n17, NB\}_{pk(a)})\ \}, []) \\ = \ & ? \end{aligned}$$

Solving Constraints: (II) Unification

Definition (Lazy Intruder Unification Rule)

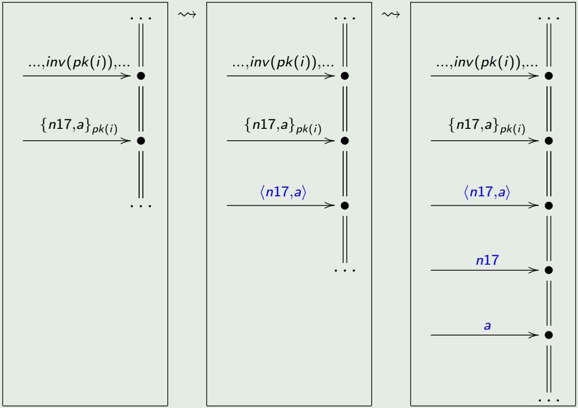
$$S_1.\mathrm{Rcv}(s).S_2.\mathrm{Snd}(t).S_3 \rightsquigarrow \sigma(S_1.\mathrm{Rcv}(s).S_2.S_3)$$

if s and t are not variables, and $\sigma = mgu(s, t)$.

Solving Constraints: (III) Simple Analysis

Analysis

The intruder received an encrypted message and has the decryption key in his knowledge. Then we can also add the decrypted message. Similar for pairs, he can obtain the components immediately.



Solving Constraints: (III) Simple Analysis

Definition (Lazy Intruder Simple Analysis Rules)

$$S_1.\mathrm{Rcv}(\mathrm{inv}(k)).S_2.\mathrm{Rcv}(\{m\}_k).S_3 \rightsquigarrow S_1.\mathrm{Rcv}(\mathrm{inv}(k)).S_2.\mathrm{Rcv}(\{m\}_k).\mathrm{Rcv}(m).S_3$$

$$S_1.\mathrm{Rcv}(k).S_2.\mathrm{Rcv}(\{|m|\}_k).S_3 \rightsquigarrow S_1.\mathrm{Rcv}(k).S_2.\mathrm{Rcv}(\{|m|\}_k).\mathrm{Rcv}(m).S_3$$

$$S_1.\mathrm{Rcv}(\langle m_1, m_2\rangle).S_2 \rightsquigarrow S_1.\mathrm{Rcv}(\langle m_1, m_2\rangle).\mathrm{Rcv}(m_1).\mathrm{Rcv}(m_2).S_2$$

$$S_1.\mathrm{Rcv}(\{m\}_{inv(k)}).S_2 \rightsquigarrow S_1.\mathrm{Rcv}(\{m\}_{invk}).\mathrm{Rcv}(m).S_2$$

Note: this now can lead to non-termination, if you repeatedly apply a rule to the same term. But since this is redundant (not adding new knowledge), we can exclude repeated application to the same term.

$\star$ Solving Constraints: (III) Full Analysis

In general, analysis is more difficult, since:

- the key-term may contain variables, like $\{m\}_{pk(A)}$.
  - $\star$ the intruder can decrypt with $inv(pk(i))$ iff $A = i$.
- the key-term may be composed, like $\{|m|\}_{h(n1,n2)}$.
  - $\star$ the intruder may first need to compose the key term.
- the intruder may receive a decrypted message that he cannot decrypt at first, but learn the decryption key in a later step.

⋆ Solving Constraints: (III) Full Analysis

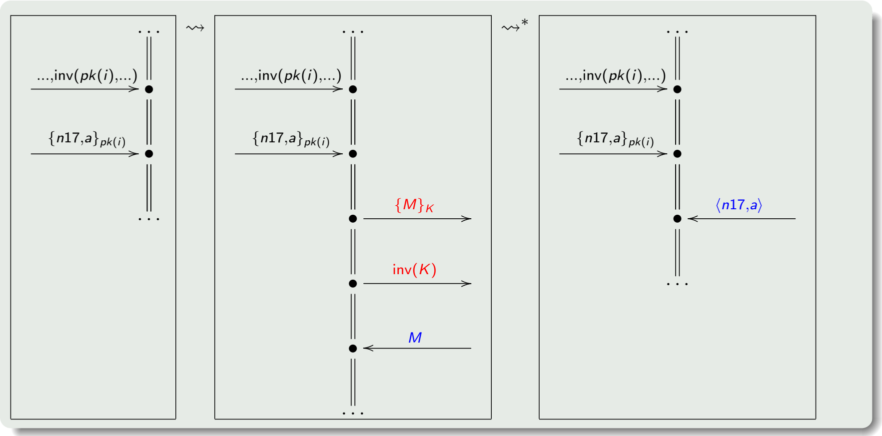
Idea: insert analysis strands into the intruder constraint:

Definition (Analysis Strands)

$$\begin{tikzcd} \bullet \arrow[r, "\{M\}_K"] \arrow[d, equal] & {} \\ \bullet \arrow[r, "\mathrm{inv}(K)"] \arrow[d, equal] & {} \\ \bullet & {} \arrow[l, "M"'] \end{tikzcd} \quad \begin{tikzcd} \bullet \arrow[r, "\{\!|M|\!\}_K"] \arrow[d, equal] & {} \\ \bullet \arrow[r, "K"] \arrow[d, equal] & {} \\ \bullet & {} \arrow[l, "M"'] \end{tikzcd} \quad \begin{tikzcd} \bullet \arrow[r, "\langle M_1, M_2 \rangle"] \arrow[d, equal] & {} \\ \bullet \arrow[d, equal] & {} \arrow[l, "M_1"'] \\ \bullet & {} \arrow[l, "M_2"'] \end{tikzcd} \quad \begin{tikzcd} \bullet \arrow[r, "\{M\}_{inv(K)}"] \arrow[d, equal] & {} \\ \bullet & {} \arrow[l, "M"'] \end{tikzcd}$$

- These analysis strands force the intruder to produce a message of a form that can be decrypted and produce the decryption key (where necessary).
- As a result the intruder obtains the decrypted message.

⋆ Solving Constraints: (III) Full Analysis

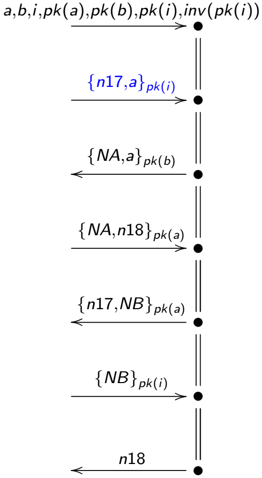


⋆ Solving Constraints: (III) Full Analysis

The insertion of analysis strands is tricky:

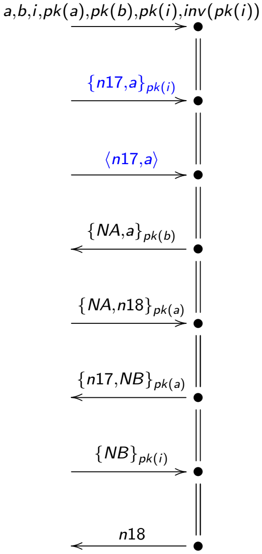
- For each destructor in a message that the intruder receives, one may insert one corresponding analysis rule.
- . . . at any point after the receive step.
- The simple analysis rules are a special case, namely when the decryption key is literally in the intruder knowledge already.
- For all examples in the course, the simple analysis rules suffice.
  - ★ In your report you can restrict yourself to simple analysis rules.

Lazy Intruder Constraint Solving – Example



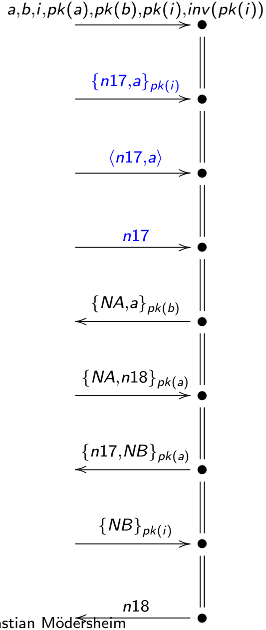
- Here we can decrypt $\{n17, a\}_{pk(i)}$ since we know the private key $inv(pk(i))$
- We can also decompose the resulting pair $\langle n17, a\rangle$.
- Since a is already known, we only really learn $n17$ from this.

Lazy Intruder Constraint Solving – Example



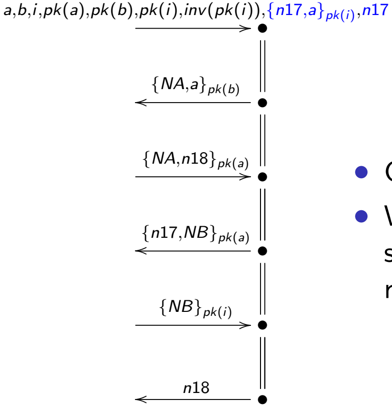
- Here we can decrypt $\{n17, a\}_{pk(i)}$ since we know the private key $inv(pk(i))$
- We can also decompose the resulting pair $\langle n17, a \rangle$.
- Since a is already known, we only really learn $n17$ from this.

Lazy Intruder Constraint Solving – Example



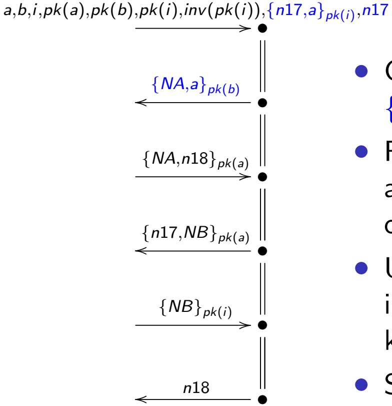
- Here we can decrypt $\{n17, a\}_{pk(i)}$ since we know the private key $inv(pk(i))$
- We can also decompose the resulting pair $\langle n17, a \rangle$.
- Since a is already known, we only really learn $n17$ from this.

Lazy Intruder Constraint Solving – Example



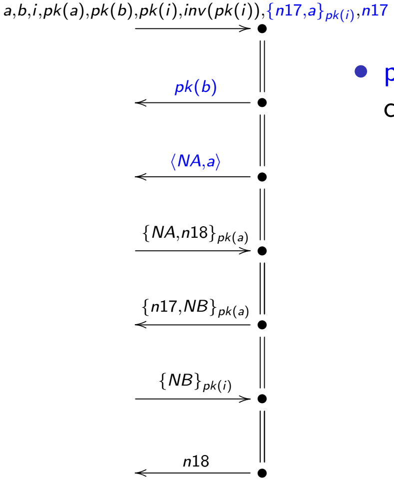
- Compressing notation a bit :-)
- We can also forget the pair $\langle n17, a\rangle$ since we have the components and can reconstruct the pair if necessary.

Lazy Intruder Constraint Solving – Example



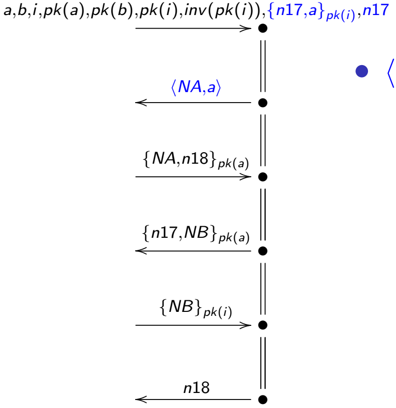
- Consider the first outgroing message $\{NA, a\}_{pk(b)}$
- For each outgoing message, there are always basically two possibilities: composition or unification.
- Unification does not work here since the intruder has nothing fitting in his knowledge.
- So let us go for composition.

Lazy Intruder Constraint Solving – Example



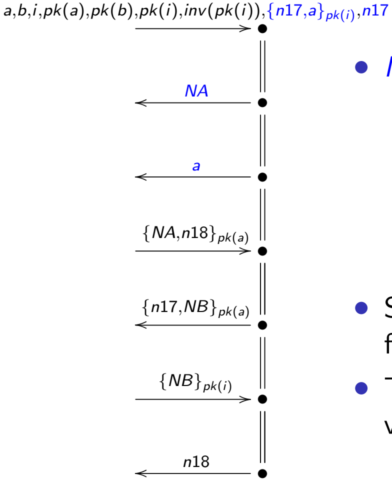
- pk(b) is directly in our knowledge – we can remove it using unification.

Lazy Intruder Constraint Solving – Example



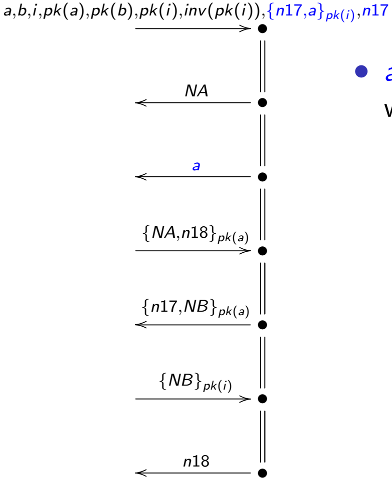
- $\langle NA, a \rangle$ must be done with composition.

Lazy Intruder Constraint Solving – Example



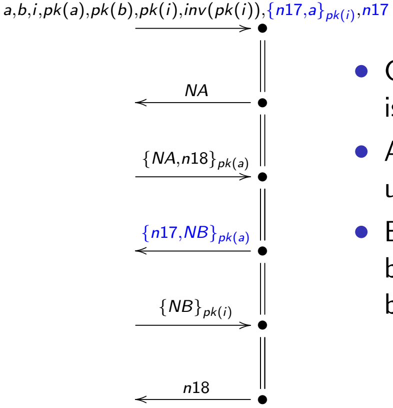
- *NA* is a variable:
  - ★ The composition rule cannot be applied, because it is not of the form $f(\ldots)$ for a public function f.
  - ★ The unification rule can only be applied between two terms s and t that are not variables.
- So no rule can be applied – we leave *NA* for now.
- This is why the intruder is lazy – any value for *NA* will do, so why bother.

Lazy Intruder Constraint Solving – Example



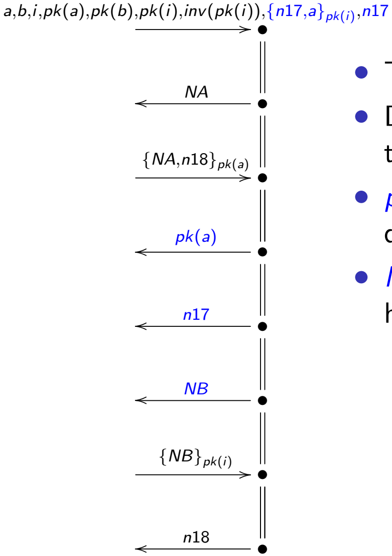
- a is already known and can be removed with unification

Lazy Intruder Constraint Solving – Example

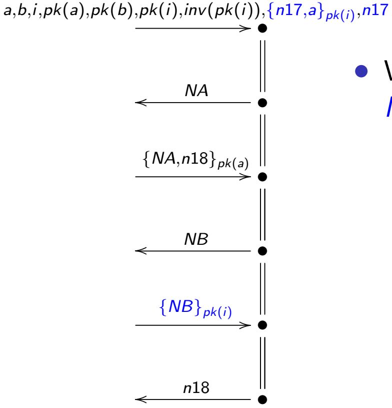


- Consider the first outgoing message that is not a variable: $\{n17, NB\}_{pk(a)}$
- Again two general possibilities: unification or composition.
- Both cases are possible here, and to build the $\rightsquigarrow$ tree, we have to follow both.

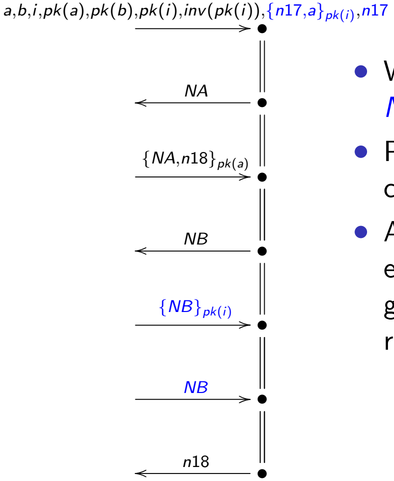
Lazy Intruder Constraint Solving – Example



- The composition case first.
- Directly applying composition also to the pair.
- $pk(a)$ and $n17$ are known and can be done with unification.
- NB is a variable and we are lazy again here, just leave it for now.

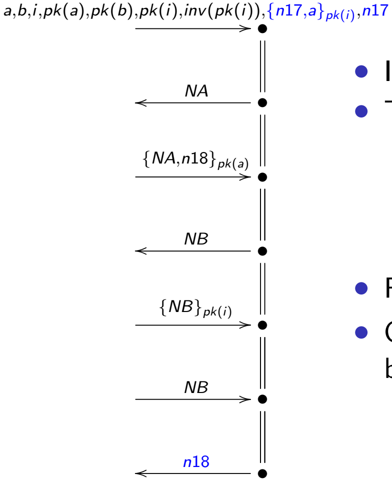


- We can apply decryption here, giving us *NB*.



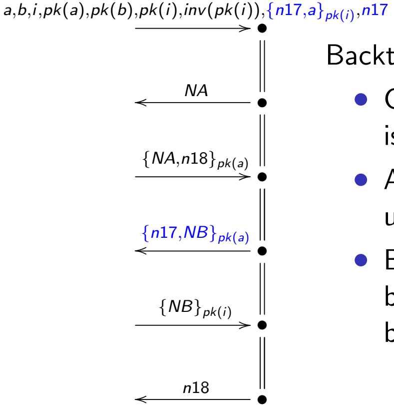
- We can apply decryption here, giving us *NB*.
- Pretty useless: whatever *NB* is, we have constructed that ourselves earlier.
- Actually – this is the normal protocol execution where the intruder has generated some value *NB* and now received it back from Alice.

Lazy Intruder Constraint Solving – Example



- It remains to construct $n18$.
- That's impossible:
  - ★ Composition impossible: it is not a public function
  - ★ Unification impossible: we don't have it in our knowledge.
- Fail!
- One leaf node of our $\rightsquigarrow$ tree, let's backtrack to the last branching point.

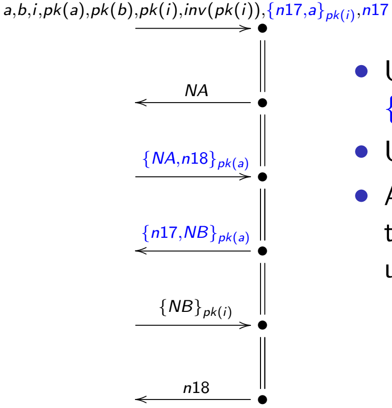
Lazy Intruder Constraint Solving – Example



Backtracking to this point.

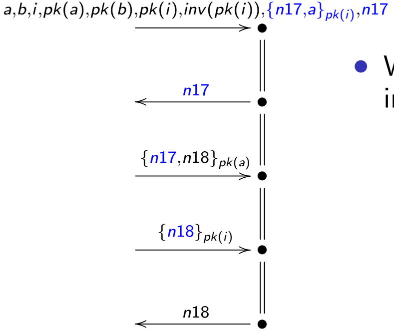
- Consider the first outgoing message that is a variable: $\{n17, NB\}_{pk(a)}$
- Again two general possibilities: unification or composition.
- Both cases are possible here, and to build the $\rightsquigarrow$ tree, we have to follow both.

Lazy Intruder Constraint Solving – Example



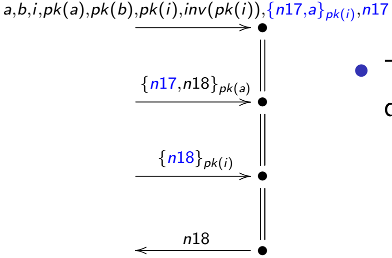
- Unify $\{n17, NB\}_{pk(a)}$ with $\{NA, n18\}_{pk(a)}$
- Unifier $\sigma = [NA \mapsto n17, NB \mapsto n18]$.
- Apply to entire constraint and remove the outgoing message we have just unified.

Lazy Intruder Constraint Solving – Example



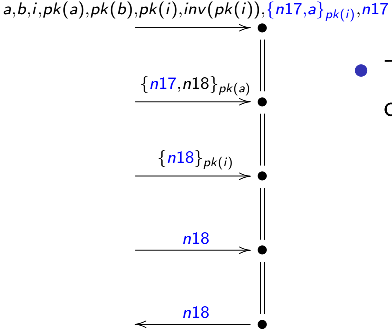
- We "retro-actively" decided that the intruder used $n17$ as nonce NA.
  - ★ Requires that the intruder knows $n17$.
  - ★ He does – so one unification step.

Lazy Intruder Constraint Solving – Example



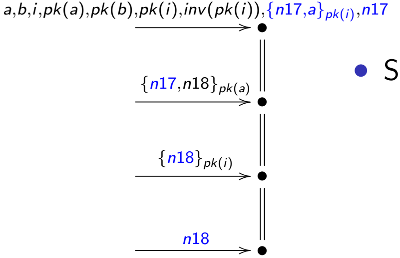
- The message $\{n18\}_{pk(i)}$ can be decrypted, so we get $n18$.

Lazy Intruder Constraint Solving – Example



- The remaining outgoing message $n18$ is one simple unification step.

Lazy Intruder Constraint Solving – Example



- Solved!

Lazy Intruder Synopsis

Finding all solutions of a constraint (with simple analysis)

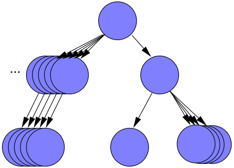
- Always start with the first step that has not been considered yet.
- If incoming: can simple decryption be applied?
- If outgoing:
  - ★ If it is a variable, leave it for now and continue with the next step.
  - ★ If it is not a variable, consider independently (with backtracking!) the following cases:
    - ▶ Composition (if it is a public operator)
    - ▶ Unification – with any incoming message that is not a variable.
- Whenever a unification is done where variables are substituted, apply to the entire constraint and go back to the first message that was affected!
- When all remaining outgoing messages are variables, the constraint is solved. We call that a simple constraint.

Lazy Intruder Synopsis

- Termination: every unification and composition step makes the constraint simpler, this cannot go on forever. The analysis steps can only produce subterms of terms we already have.
- Soundness: the lazy intruder procedure finds only correct solutions (covered by the Dolev-Yao model)
- Completeness: if a constraint has a solution, the lazy intruder will find it:
  - ⋆ Consider any solution of a constraint.
  - ⋆ Then the constraint is either already simple or one of the lazy intruder steps gets us to a new constraint that still supports that solution.
  - ⋆ By termination, we eventually arrive at a simple constraint that supports the considered solution.

Lazy Intruder: Summary

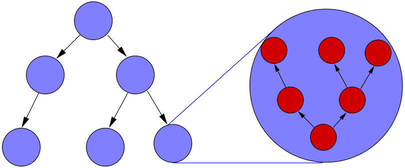
- With the naive approach, most states of our search tree have infinitely many successors, because for $\mathsf{Rcv}(t)$ there are usually infinitely many σ with $M \vdash \sigma(t)$.



- We avoid the enumeration by using symbolic states with constraints. This gives us at most one successor per strand.
- We have now two layers of search:

  Layer 1: search in the tree of symbolic states

  Layer 2: constraint reduction (satisfiability)



Lazy Intruder: Summary

- The constraint reduction produces finitely many simple constraints by a terminating algorithm.
- If the number of sessions is bounded, we now have a decision procedure even without bounding the messages:

Theorem (Rusinowitch & Turuani 2001)

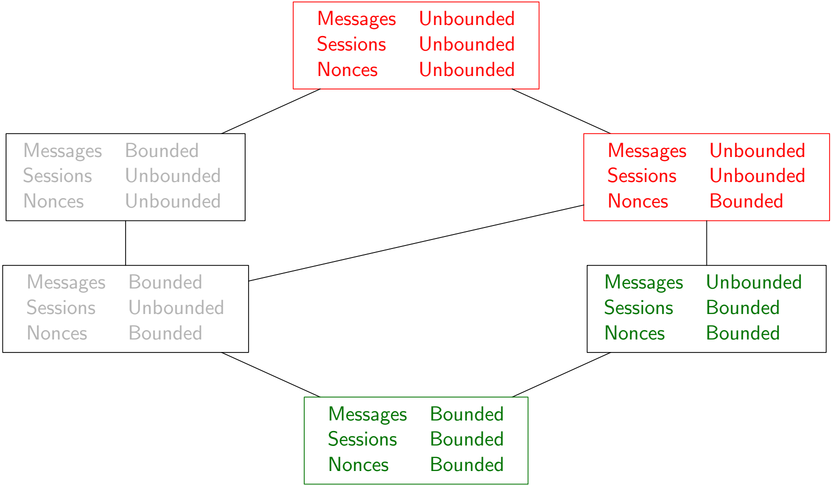
Protocol insecurity for a bounded number of sessions is NP-complete.

Proof Sketch.

In NP: given a finite set of threads in the initial state, guess a symbolic trace for them and a sequence of reduction steps for the resulting constraints. Check that we have reached a valid attack state. (All this can be polynomially bounded.)

NP-hard: Polynomial reduction for boolean formulae to security protocols such that formula satisfiable iff protocol has an attack. □

Decidability Lattice



In the next lecture, we will fill the rest of this lattice.

Bibliography

- Hubert Comon, Véronique Cortier, John Mitchell. Tree Automata with One Memory, Set Constraints, and Ping-Pong Protocols. ICALP 2001. Springer-Verlag, 2001.
- Shimon Even and Oded Goldreich. On the Security of Multi-Party Ping-Pong Protocols, Symposium on Foundations of Computer Science, IEEE Computer Society, 1983.
- John Hopcroft, Rajeev Motwani, Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. Second Edition. Addison-Wesley, 2001.
- Sebastian Mödersheim and Luca Viganò: The Open-source Fixed-point Model Checker for Symbolic Analysis of Security Protocols. Fosad 2007-2008-2009, LNCS 5705, 166-194. Springer-Verlag, 2009.
- Michaël Rusinowitch and Mathieu Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. Computer Security Foundations Workshop, IEEE Computer Society, 2001.