Security Protocols IV: Algorithmic analysis methods (Part 1)

Sebastian Mödersheim and Christoph Sprenger

FMSEC Module 5, v.1
19.10.2009

Everything in the free algebra today.

Automated Verification and Decidability

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

- Input:
  - ★ some description of a program P
  - ★ some description of a functional specification S
- Output: *Yes* if P satisfies S, and *No* otherwise.
- Optional extra: in the *No* case, give a counter-example, i.e. an input on which P violates the specification.

Automated Verification and Decidability

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

- Input:
  - ★ some description of a program P
  - ★ some description of a functional specification S
- Output: *Yes* if P satisfies S, and *No* otherwise.
- Optional extra: in the *No* case, give a counter-example, i.e. an input on which P violates the specification.

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.

The Sources of Infinity

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

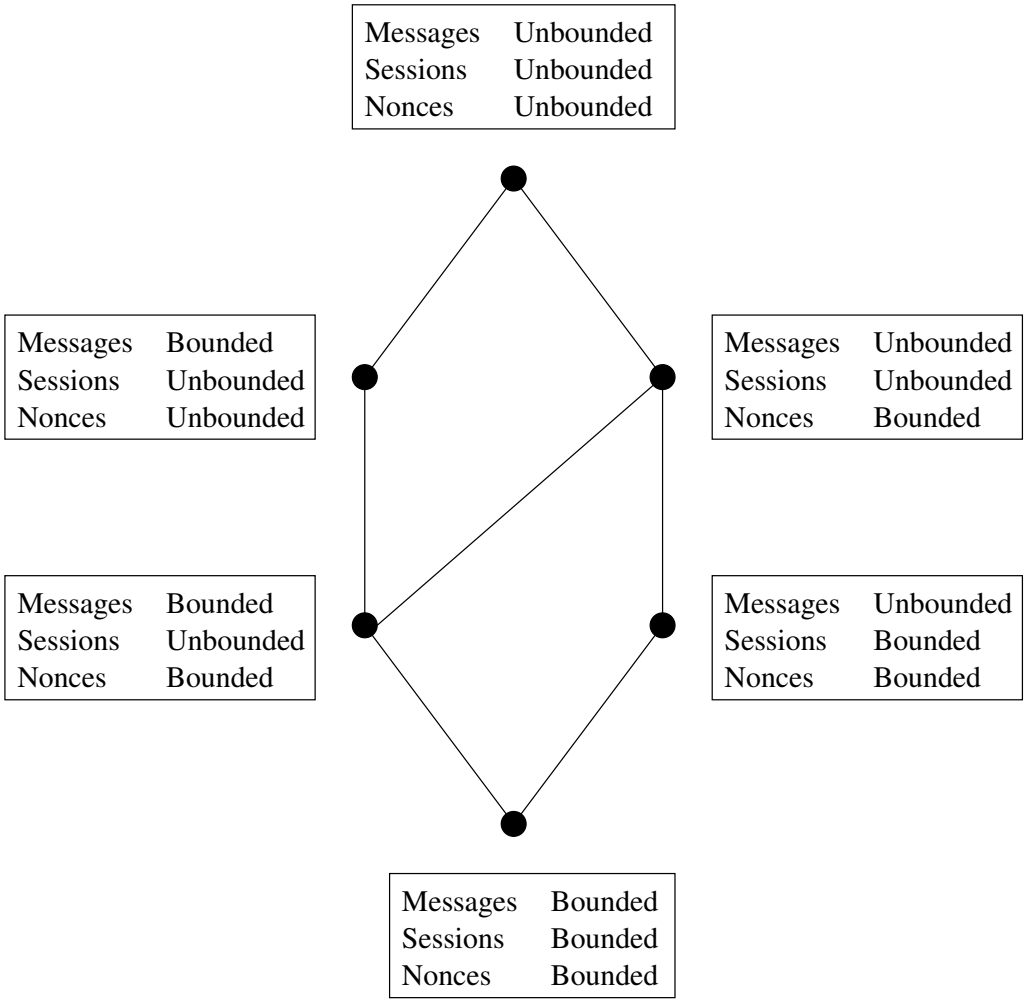
  Messages The intruder can compose arbitrarily complex messages from his knowledge, e.g. $i, h(i), h(h(i)), \ldots$.

  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?

Roadmap


Messages Unbounded
Sessions Unbounded
Nonces Unbounded
Messages Bounded
Sessions Unbounded
Nonces Unbounded
Messages Unbounded
Sessions Unbounded
Nonces Bounded
Messages Bounded
Sessions Unbounded
Nonces Bounded
Messages Unbounded
Sessions Bounded
Nonces Bounded
Messages Bounded
Sessions Bounded
Nonces Bounded

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 $(s_1, t_1), \ldots, (s_n, t_n)$

Output Yes if there is a finite sequence of indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $s_{i_1} \ldots s_{i_k} = t_{i_1} \ldots t_{i_k}$; 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

Reduction: give a computable translation f

- that translates every correspondence problem
- into a security protocol (with a secrecy goal)
- such that x has a solution iff $f(x)$ has an attack.
- This shows: if secrecy in security protocols is decidable, then also PCP. And PCP is not decidable ...

Reducing PCP to Protocol Security

For the given problem $x = ((s_1, t_1), \ldots, (s_n, t_n))$:

Roles

- R_1 : $\mathrm{rcv}(\langle\langle X, Xs\rangle, \langle \$, \$\rangle\rangle) \cdot \mathrm{snd}(\{\!|\langle\langle X, Xs\rangle, \langle \$, \$\rangle\rangle|\!\}_k)$
- R_2^j for each (s_j, t_j) of the correspondence problem, $j \in \{1, \ldots, n\}$: $\mathrm{rcv}(\{\!|\langle\langle j, Xs\rangle, \langle Y_l, Y_r\rangle\rangle|\!\}_k) \cdot \mathrm{snd}(\{\!|\langle Xs, \langle s_j \cdot Y_l, t_j \cdot Y_r\rangle\rangle|\!\}_k)$
- R_3: $\mathrm{rcv}(\{\!|\langle \$, \langle X, X\rangle\rangle|\!\}_k) \cdot \mathrm{snd}(secret)$.

Consider an initial state that contains

- one thread of R_1 and R_3 each and
- an infinite number of threads of each R_2^j.
- Initial intruder knowledge: all constants but k and $secret$.
- For $s = c_1 c_2 \ldots c_n$, let $s \cdot Y$ denote $\langle c_1, \langle c_2, \langle \ldots, \langle c_n, Y\rangle\rangle\rangle\rangle$.
- Note: $\langle \cdot, \cdot \rangle$ is not associative.
- The goal is that the intruder never obtains $secret$.

Reducing PCP to Protocol Security: Example

Example

$s_1 = 1 \quad s_2 = 10 \quad s_3 = 011$
$t_1 = 101 \quad t_2 = 00 \quad t_3 = 11$

- $R_1 : \mathrm{rcv}(\langle\langle X, Xs\rangle, \langle \$, \$\rangle\rangle) \cdot \mathrm{snd}(\{\!|\langle\langle X, Xs\rangle, \langle \$, \$\rangle\rangle|\!\}_k)$
- R_2^j for $j \in \{1, \ldots, n\}$:
  $\mathrm{rcv}(\{\!|\langle\langle j, Xs\rangle, \langle Y_l, Y_r\rangle\rangle|\!\}_k) \cdot \mathrm{snd}(\{\!|\langle Xs, \langle s_j \cdot Y_l, t_j \cdot Y_r\rangle\rangle|\!\}_k)$
- R_3: $\mathrm{rcv}(\{\!|\langle \$, \langle X, X\rangle\rangle|\!\}_k) \cdot \mathrm{snd}(secret)$.

Attack

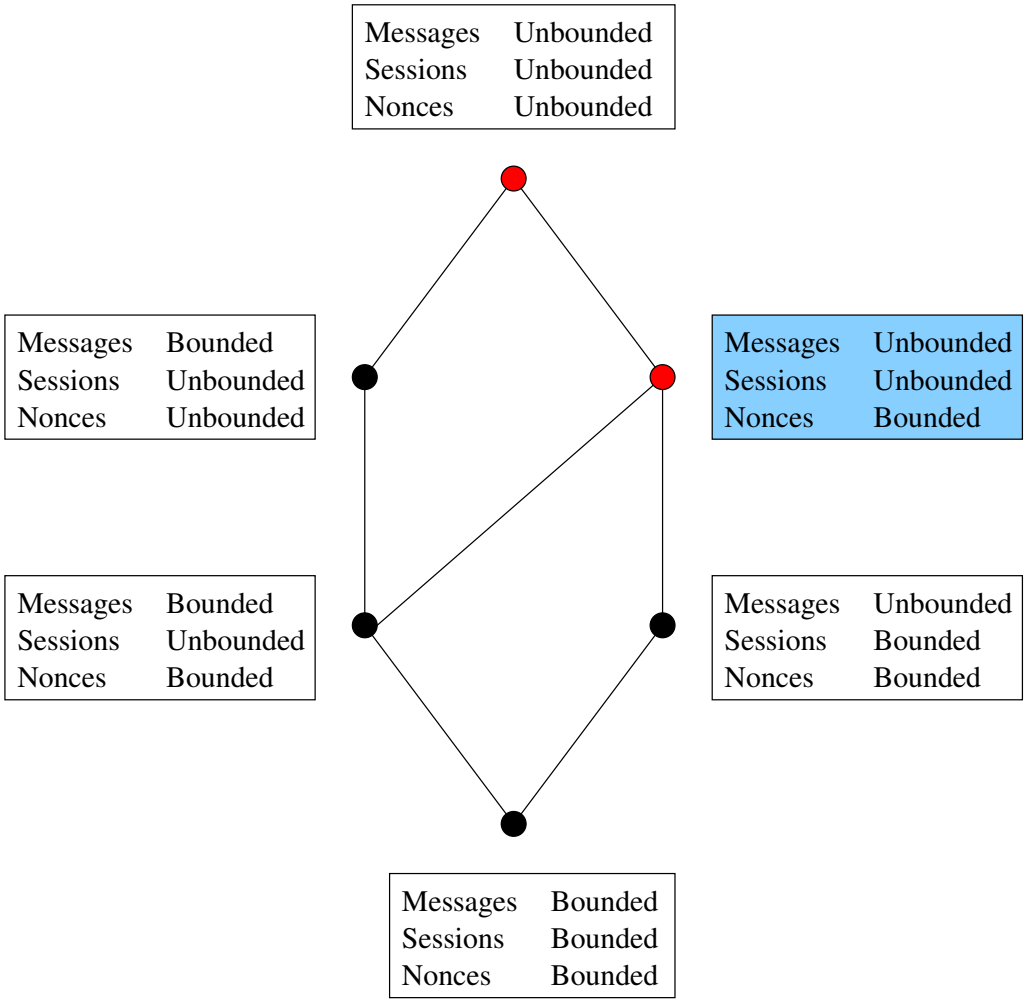
- The intruder "guesses" the solution 1323.
- He sends $\langle\langle 3, \langle 2, \langle 3, \langle 1, \$\rangle\rangle\rangle\rangle, \langle \$, \$\rangle\rangle$ to R_1,
- R_1 replies: $\{\!|\langle\langle 3, \langle 2, \langle 3, \langle 1, \$\rangle\rangle\rangle\rangle, \langle \$, \$\rangle\rangle|\!\}_k$
- Using the R_2^j we get in several steps: $\{\!|\langle \$, \langle t, t\rangle\rangle|\!\}_k$ for $t = \langle 1, \langle 0, \langle 1, \langle 1, \langle 1, \langle 0, \langle 0, \langle 1, \langle 1, \$\rangle\rangle\rangle\rangle\rangle\rangle\rangle\rangle\rangle$
- Using R_3, we get *secret*.

More generally, there is an attack iff there is a solution to the PCP.

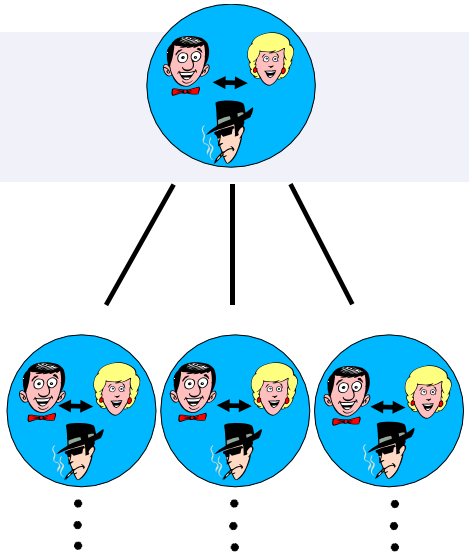
Undecidability (Summary)

- First shown similarly by [Even and Goldreich 1983].
- Basic idea of the proof: we let the intruder "guess" a solution and use the honest agents as a machine to "check" the solutions.
- As the attack is equivalent to the existence of a solution, the protocol verifier can thus be employed to solve the PCP problem.
- However, the protocols generated for PCP are very artificial:
  - ★ They are not even executable without an intruder.
  - ★ Thus they cannot be described in AnB.
  - ★ [Comon and Cortier 2001]: even when restricting to executable protocols, secrecy is undecidable.
- The proof requires unbounded messages and unbounded sessions (as the length of the solution and its check cannot be bounded). There are no fresh nonces at all, thus we have ...

Roadmap


Messages Unbounded
Sessions Unbounded
Nonces Unbounded
Messages Bounded
Sessions Unbounded
Nonces Unbounded
Messages Unbounded
Sessions Unbounded
Nonces Bounded
Messages Bounded
Sessions Unbounded
Nonces Bounded
Messages Unbounded
Sessions Bounded
Nonces Bounded
Messages Bounded
Sessions Bounded
Nonces Bounded

Search Tree



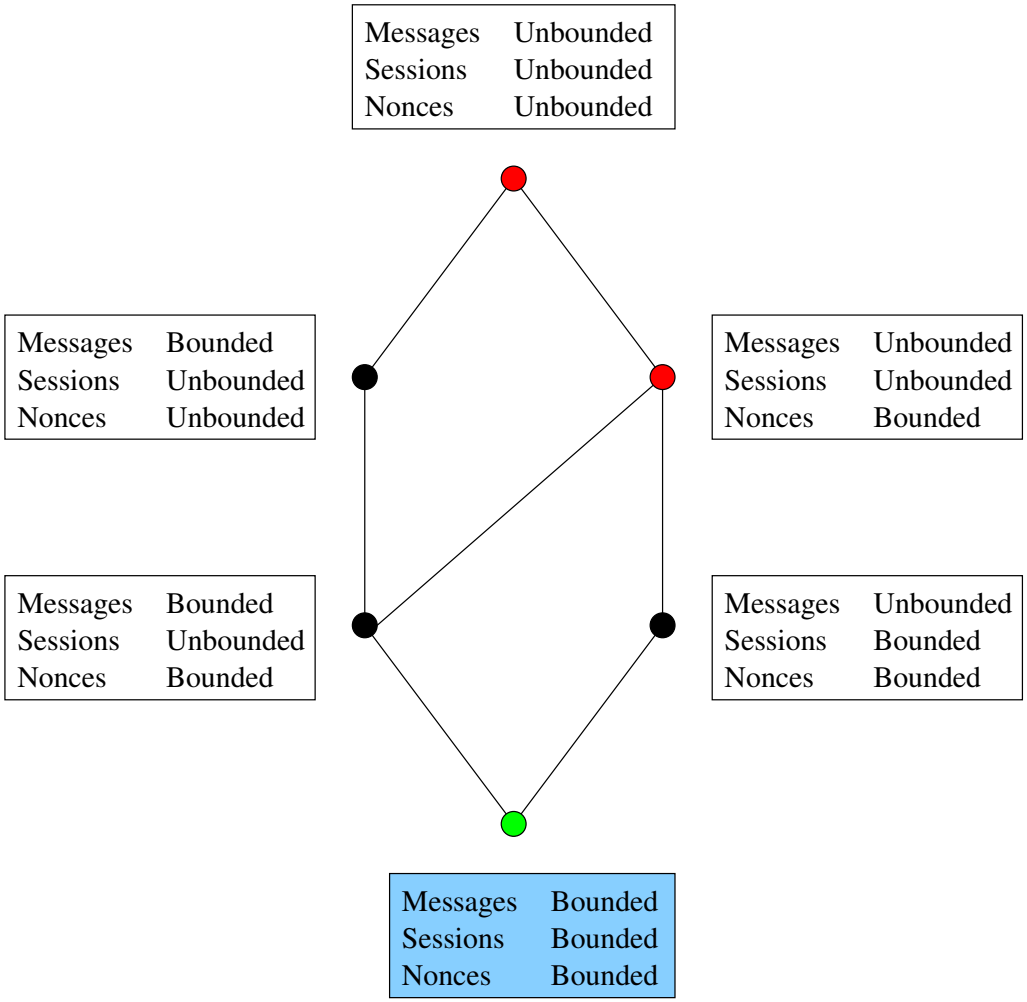
We can consider a search tree, where

- each node is a state,
- the root node is the initial state,
- node n is a child of node m iff state n can be reached from state m by one transition with the rules *send*, *receive*, or *signal*.
- we can check for each state/node whether it violates our secrecy or authentication goals.
- we can use the standard search techniques to browse that tree, e.g. depth first, breadth first, iterative deepening.

Exercise: formalize and prove:

- When bounding everything, this gives us a decision procedure.
- Otherwise, we can give a semi-decision procedure, i.e., one that is guaranteed to terminate with an attack if there is one.

Roadmap


Messages Unbounded
Sessions Unbounded
Nonces Unbounded
Messages Bounded
Sessions Unbounded
Nonces Unbounded
Messages Unbounded
Sessions Unbounded
Nonces Bounded
Messages Bounded
Sessions Unbounded
Nonces Bounded
Messages Unbounded
Sessions Bounded
Nonces Bounded
Messages Bounded
Sessions Bounded
Nonces Bounded

The Dolev-Yao Intruder is Prolific

The intruder can compose messages arbitrarily from his knowledge and send them to any agent at any time.
Thus, most nodes in the search tree have infinitely many successors.
Even bounding messages, this can give an enormous search tree:

Example

- Thread t that starts with $\mathrm{rcv}(\{N_1, N_2, N_3\}_{\mathrm{pk}(a)})$; $N_i : nonce$.
- The intruder knows nonces $n_1, \ldots, n_5$ and $\mathrm{pk}(a)$.
- Then there are 5^3 well-typed messages he can send to t (not to talk of ill-typed ones).
- Thus, every state that contains t has at least 5^3 successors.
- Also: how to compute the set of terms that the intruder can generate to match a receiver-term?

Deciding Dolev-Yao

Lemma

$\mathcal{DY}(M)$ is infinite, but $t \overset{?}{\in} \mathcal{DY}(M)$ is decidable.

Proof idea:

- Split the intruder into two parts: composition and decomposition.
- The composition part is simple to decide.
- Note: for decomposition, you may have to compose a key first!
- The decomposition part can be defined as a finite closure of M: those subterms of M that can be obtained by decomposition.

An Intruder Divided ...

Definition (Composing Intruder)

$$\frac{}{m \in \mathcal{DY}_C(M)} \text{ Axiom } (m \in M)$$

$$\frac{t_1 \in \mathcal{DY}_C(M) \quad \ldots \quad t_n \in \mathcal{DY}_C(M)}{f(t_1, \ldots, t_n) \in \mathcal{DY}_C(M)} \text{ Composition } (f \in \Sigma_p)$$

Definition (Decomposing Intruder)

$$\frac{}{m \in \mathcal{DY}_D(M)} \text{ Axiom } (m \in M)$$

$$\frac{\langle m_1, m_2 \rangle \in \mathcal{DY}_D(M)}{m_i \in \mathcal{DY}_D(M)} \text{ Proj}_i \qquad \frac{\{\!|m|\!\}_k \in \mathcal{DY}_D(M) \quad k \in \mathcal{DY}_C(\mathcal{DY}_D(M))}{m \in \mathcal{DY}_D(M)} \text{ DecSym}$$

$$\frac{\{m\}_k \in \mathcal{DY}_D(M) \quad \text{inv}(k) \in \mathcal{DY}_C(\mathcal{DY}_D(M))}{m \in \mathcal{DY}_D(M)} \text{ DecAsym} \qquad \frac{\{m\}_{\text{inv}(k)} \in \mathcal{DY}_D(M)}{m \in \mathcal{DY}_D(M)} \text{ OpenSig}$$

The decomposing intruder uses composition for key-derivation!

Deciding Dolev-Yao

Decision procedure for $t \in \mathcal{DY}(M)$

- First compute $M' = \mathcal{DY}_D(M)$: the analyzed intruder knowledge.
- Then check for a given term whether $t \in \mathcal{DY}_C(M')$.

- $t \in \mathcal{DY}_C(M)$ is decidable:
  - ★ Check $t \in M$
  - ★ If not, and if $t = f(t_1, \ldots, t_n)$ for $f \in \Sigma_p$, check recursively $t_1, \ldots, t_n \in \mathcal{DY}_C(M)$.
- $\mathcal{DY}_D(M)$ is finite: can contain only subterms of M.
- $\mathcal{DY}_C(\mathcal{DY}_D(M)) = \mathcal{DY}(M)$.
  - ★ Consider an arbitrary derivation $t \in \mathcal{DY}(M)$.
  - ★ We normalize the proof to eliminate trivial composition-decomposition pairs, e.g.:

$$\dfrac{\dfrac{\dfrac{\Pi}{m \in \mathcal{DY}(M)} \quad k \in \mathcal{DY}(M)}{\{\!|m|\!\}_k \in \mathcal{DY}(M)} \quad k \in \mathcal{DY}(M)}{m} \quad \mapsto \quad \dfrac{\Pi}{m \in \mathcal{DY}(M)}$$

  - ★ Inductively show that every result of a decomposition step is in $\mathcal{DY}_D(M)$ and the results of all steps are in $\mathcal{DY}_C(\mathcal{DY}_D(M))$.

Deciding Dolev-Yao: Example

Example (Cf. Exercise assignment 2.2)

$$M = \{ \{\!|k|\!\}_{h(n_1,n_2)}, \{n_1\}_{\mathsf{pk}(i)}, \{n_2\}_{\mathsf{inv}(\mathsf{pk}(a))},$$

$$\mathsf{pk}(a), \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), \{\!|secret|\!\}_k \}$$

$$M' = \mathcal{DY}_D(M) = M \cup \{n_1, n_2, k, secret\}$$

Thus

1. $secret \in \mathcal{DY}(M)$ since $secret \in M'$
2. $t = \{secret\}_{\mathsf{inv}(\mathsf{pk}(a))} \notin \mathcal{DY}(M)$ since $t \notin \mathcal{DY}_C(M')$.
3. $t = \{\!|n_1|\!\}_{h(k,secret)} \in \mathcal{DY}(M)$ since $t \in \mathcal{DY}_C(M')$.

Towards a Lazy Intruder

For transitions, we have a slightly different problem:

Given a thread that starts with rcv(t) where t has variables, find substitutions σ such that $t\sigma \in \mathcal{DY}(M)$.

Can we modify the $\mathcal{DY}$ decision procedure to deal with variables?

Example

$M = \{\{n_1, a\}_k, \{n_2\}_k, \{n_2\}_{k_2}, k_2\}$

- $t = \{X\}_k$: t can be matched with some terms in M:
  $\sigma = [X \mapsto \langle n_1, a\rangle]$ and $\sigma = [X \mapsto n_2]$
- $t = \{X\}_{k_2}$:
  - ★ A match is possible: $\sigma = [X \mapsto n_2]$.
  - ★ The intruder can also compose t, since k_2 is known: he can then take any term $m \in \mathcal{DY}(M)$ for X.

The Lazy Intruder

Key Idea

Whenever the intruder can freely choose a value from his knowledge M for a variable X, then do not instantiate X and simply store the constraint:

$$from(\{X\}; M) \ .$$

- What X precisely is does not matter at this point, but may matter later when honest agents work with value X.
- Instead of eagerly exploring choices for X, we choose X in a demand-driven, lazy way!

Lazy Intruder Constraints

Definition

A constraint store is pair (C, σ) of a set C of constraints of the form $from(T; M)$ where T and M are sets of messages, and σ is a substitution.
A constraint store (C, σ) is called simple iff $T \subseteq \mathcal{V}$ for all $from(T; M) \in C$.
The semantics $[\![(C, \sigma)]\!]$ of a constraint store is the set of all substitutions $\tau \succeq \sigma$ such that

- $C\tau$ is ground
- $x\tau$ is ground for all $x \in dom(\tau)$
- $T\tau \subseteq \mathcal{DY}(M\tau)$ for each $from(T; M) \in C$.

A constraint store (C, σ) is called satisfiable iff $[\![C, \sigma]\!] \neq \emptyset$.

Every simple constraint store is satisfiable: for each variable, the intruder can choose an arbitrary message that he can construct.

Lazy Intruder Constraints

Example

$M = \{\{n_1, a\}_k,\ \{n_2\}_k,\ \{n_2\}_{k_2},\ k_2\}$
For an agent expecting $\{X\}_{k_2}$ we get the constraint store:

$$(\{\ \mathit{from}(\{\ \ \{X\}_{k_2}\ \ \};M)\ \},[\ ])$$

This store has two solutions in the form of simple constraints:

- $(\emptyset, [X \mapsto n_2])$
- $(\{\ \mathit{from}(\{X\};M)\ \},[\ ])$

To do: reduction procedure that transforms a given constraint store (C, σ) into a an equivalent set $\{(C_1, \sigma_1), \ldots, (C_n, \sigma_n)\}$ of simple constraint stores, i.e. such that:

$$[\![(C, \sigma)]\!] = [\![(C_1, \sigma_1)]\!] \cup \ldots \cup [\![(C_n, \sigma_n)]\!]$$

If (C, σ) is unsatisfiable, then $n = 0$.

Lazy Intruder Reduction

$$\frac{((\{from(T;M)\} \cup C)\tau, \sigma\tau)}{(\{from(\{t\} \cup T;M)\} \cup C, \sigma)} \text{ Unify } (t \notin \mathcal{V}, s \in M, \tau = \text{mgu}(s,t))$$

$$\frac{(\{from(\{t_1,\ldots,t_n\} \cup T;M)\} \cup C, \sigma)}{(\{from(\{t\} \cup T;M)\} \cup C, \sigma)} \text{ Compose } (t = f(t_1,\ldots,t_n), f \in \Sigma_p)$$

- These rules are read "bottom-up": this is a backward search.
- mgu denotes the most general unifier.
- The condition $t \notin \mathcal{V}$ of the Unify rule reflects the laziness.
- There are similar rules for the decomposition of messages.

Theorem

With the lazy intruder deduction rules, one can transform any constraint store (that can arise in our protocol model) into an equivalent set of simple constraint stores.

Example (The NSPK attack—lazily)

Take the steps from the NSPK attack without instantiating any variables of the roles:

$$
\begin{array}{ll}
(0, \mathrm{snd}(\{na_0, a\}_{\mathrm{pk}(i)})) & IK_1 = IK_0 \cup \{na_0, a\}_{\mathrm{pk}(i)} \\
(1, \mathrm{rcv}(\{NA, A\}_{\mathrm{pk}(b)})) & from(\{NA, A\}_{\mathrm{pk}(b)}; IK_1) \\
(1, \mathrm{snd}(\{NA, nb_1\}_{\mathrm{pk}(A)})) & IK_2 = IK_1 \cup \{NA, nb_1\}_{\mathrm{pk}(A)} \\
(0, \mathrm{rcv}(\{na_0, NB\}_{\mathrm{pk}(a)})) & from(\{na_0, NB\}_{\mathrm{pk}(a)}; IK_2) \\
(0, \mathrm{sig}(secret, a, i, \langle na_0, NB \rangle) & \\
(0, \mathrm{snd}(\{NB\}_{\mathrm{pk}(i)})) & IK_3 = IK_2 \cup \{NB\}_{\mathrm{pk}(i)} \\
(1, \mathrm{rcv}(\{nb_1\}_{\mathrm{pk}(b)})) & from(\{nb_1\}_{\mathrm{pk}(b)}; IK_3) \\
(1, \mathrm{sig}(secret, b, a, \langle NA, nb_1 \rangle)) & from(\langle NA, nb_1 \rangle; IK_3)
\end{array}
$$

Last constraint represents the check whether the intruder can generate the secret.
If this constraint store is satisfiable, then there is an attack.

Example (The NSPK attack—lazily)

Using decomposition rules (that we do not discuss), the intruder can read all messages encrypted with his public key pk(i):

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathrm{pk}(i)}, na_0$$
$$from(\{NA, A\}_{\mathrm{pk}(b)}; IK_1)$$
$$IK_2 = IK_1 \cup \{NA, nb_1\}_{\mathrm{pk}(A)}$$
$$from(\{na_0, NB\}_{\mathrm{pk}(a)}; IK_2)$$
$$IK_3 = IK_2 \cup \{NB\}_{\mathrm{pk}(i)}, NB$$
$$from(\{nb_1\}_{\mathrm{pk}(b)}; IK_3)$$
$$from(\langle NA, nb_1\rangle; IK_3)$$

Let us try the composition rule on the the first message

$$\frac{(\{from(\{t_1, \ldots, t_n\} \cup T; M)\} \cup C, \sigma)}{(\{from(\{t\} \cup T; M)\} \cup C, \sigma)} \text{ Compose } (t = f(t_1, \ldots, t_n), f \in \Sigma_p)$$

First with $f = \{\cdot\}$. and then $f = \langle \cdot, \cdot \rangle$ ...

Example (The NSPK attack—lazily)

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathrm{pk}(i)}, na_0$$
$$from(\mathrm{pk}(b), NA, A; IK_1)$$
$$IK_2 = IK_1 \cup \{NA, nb_1\}_{\mathrm{pk}(A)}$$
$$from(\{na_0, NB\}_{\mathrm{pk}(a)}; IK_2)$$
$$IK_3 = IK_2 \cup \{NB\}_{\mathrm{pk}(i)}, NB$$
$$from(\{nb_1\}_{\mathrm{pk}(b)}; IK_3)$$
$$from(\langle NA, nb_1\rangle; IK_3)$$

$\mathrm{pk}(b) \in IK_0$, thus we can eliminate that using the unify rule ($\tau = [\,]$):

$$\frac{((\{from(T; M)\} \cup C)\tau, \sigma\tau)}{(\{from(\{t\} \cup T; M)\} \cup C, \sigma)} \text{ Unify } (t \notin \mathcal{V}, s \in M, \tau = \mathrm{mgu}(s, t))$$

Example (The NSPK attack—lazily)

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathrm{pk}(i)}, na_0$$
$$from(NA, A; IK_1)$$
$$IK_2 = IK_1 \cup \{NA, nb_1\}_{\mathrm{pk}(A)}$$
$$from(\{na_0, NB\}_{\mathrm{pk}(a)}; IK_2)$$
$$IK_3 = IK_2 \cup \{NB\}_{\mathrm{pk}(i)}, NB$$
$$from(\{nb_1\}_{\mathrm{pk}(b)}; IK_3)$$
$$from(\langle NA, nb_1\rangle; IK_3)$$

NA and A are variables, that we leave for now, since we are lazy.

Example (The NSPK attack—lazily)

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathrm{pk}(i)}, na_0$$
$$from(NA, A; IK_1)$$
$$IK_2 = IK_1 \cup \{NA, nb_1\}_{\mathrm{pk}(A)}$$
$$from(\{na_0, NB\}_{\mathrm{pk}(a)}; IK_2)$$
$$IK_3 = IK_2 \cup \{NB\}_{\mathrm{pk}(i)}, NB$$
$$from(\{nb_1\}_{\mathrm{pk}(b)}; IK_3)$$
$$from(\langle NA, nb_1\rangle; IK_3)$$

To solve the next constraint, let us try unification with a term in IK_2:

$$\frac{((\{from(T; M)\} \cup C)\tau, \sigma\tau)}{(\{from(\{t\} \cup T; M)\} \cup C, \sigma)} \text{ Unify } (t \notin \mathcal{V}, s \in M, \tau = \mathrm{mgu}(s, t))$$

Thus $\tau = [NA \mapsto na_0, NB \mapsto nb_1, A \mapsto a]$.

Example (The NSPK attack—lazily)

$\tau = [NA \mapsto na_0, NB \mapsto nb_1, A \mapsto a]$.

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathsf{pk}(i)}, na_0$$
$$from(na_0, a; IK_1)$$
$$IK_2 = IK_1 \cup \{na_0, nb_1\}_{\mathsf{pk}(a)}$$
$$from(\emptyset; IK_2)$$
$$IK_3 = IK_2 \cup \{nb_1\}_{\mathsf{pk}(i)}, nb_1$$
$$from(\{nb_1\}_{\mathsf{pk}(b)}; IK_3)$$
$$from(\langle na_0, nb_1\rangle; IK_3)$$

Example (The NSPK attack—lazily)

There are some terms that can be generated trivially now:

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathrm{pk}(i)}, na_0$$

$$from(na_0, a; IK_1)$$

$$IK_2 = IK_1 \cup \{na_0, nb_1\}_{\mathrm{pk}(a)}$$

$$IK_3 = IK_2 \cup \{nb_1\}_{\mathrm{pk}(i)}, nb_1$$

$$from(\{nb_1\}_{\mathrm{pk}(b)}; IK_3)$$

$$from(\langle na_0, nb_1 \rangle; IK_3)$$

Example (The NSPK attack—lazily)

For the remaining constraints we can only use the composition rule (because there is no unifying term in the IKs):

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathsf{pk}(i)}, na_0$$
$$IK_2 = IK_1 \cup \{na_0, nb_1\}_{\mathsf{pk}(a)}$$
$$IK_3 = IK_2 \cup \{nb_1\}_{\mathsf{pk}(i)}, nb_1$$
$$from(\{nb_1\}_{\mathsf{pk}(b)}; IK_3)$$
$$from(\langle na_0, nb_1 \rangle; IK_3)$$

Example (The NSPK attack—lazily)

And these are all again in the respective IKs:

$$IK_1 = IK_0 \cup \{na_0, a\}_{\mathrm{pk}(i)}, na_0$$
$$IK_2 = IK_1 \cup \{na_0, nb_1\}_{\mathrm{pk}(a)}$$
$$IK_3 = IK_2 \cup \{nb_1\}_{\mathrm{pk}(i)}, nb_1$$
$$from(\mathrm{pk}(b), nb_1; IK_3)$$
$$from(na_0, nb_1; IK_3)$$

So we have found a solution and thus an attack!

Lazy Intruder: Integration into Search

So far, we have:

Transition Rules

$$State = Trace \times IntruderKnowledge \times Threads$$

where $Trace, IntruderKnowledge$ ground and $Threads$ closed.

$$\frac{th(tid) = \mathsf{snd}(t) \cdot tl}{(tr, IK, th) \rightarrow (tr \cdot (tid, \mathsf{snd}(t)), IK \cup \{t\}, th[tid \mapsto tl])} \text{ snd}$$

$$\frac{th(tid) = \mathsf{rcv}(t) \cdot tl \quad dom(\sigma) = var(t) \quad t\sigma \in \mathcal{DY}(IK)}{(tr, IK, th) \rightarrow (tr \cdot (tid, \mathsf{rcv}(t\sigma)), IK, th[tid \mapsto tl\sigma])} \text{ rcv}$$

Let us assume from now on that the bound variables are disjoint for all threads in the initial state.

Lazy Intruder: Integration into Search

Symbolic States and Transition Rules

$$SymbolicState = Trace \times IntruderKnowledge \times Threads \times \mathbb{P}(Constraint)$$

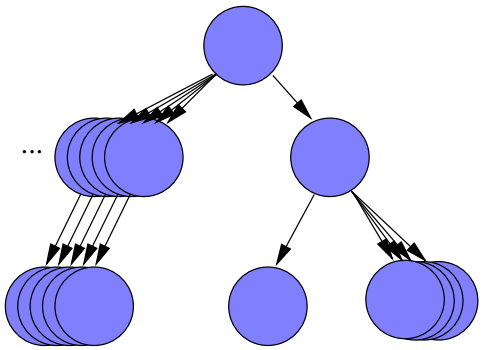
where $Trace, IntruderKnowledge, Threads$
may have free variables that occur in the constraints.

$$\frac{th(tid) = \mathsf{snd}(t) \cdot tl}{(tr, IK, th, C) \rightarrow (tr \cdot (tid, \mathsf{snd}(t)), IK \cup \{t\}, th[tid \mapsto tl], C)} \text{ snd}$$

$$\frac{th(tid) = \mathsf{rcv}(t) \cdot tl}{(tr, IK, th, C) \rightarrow (tr \cdot (tid, \mathsf{rcv}(t)), IK, th[tid \mapsto tl], C \cup \{from(\{t\}; IK)\})} \text{ rcv}$$

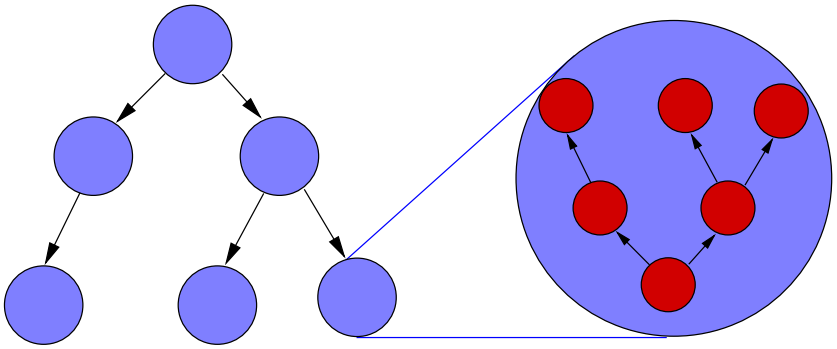
- A symbolic state is satisfiable iff a simple constraint store can be derived from $(C, [\,])$. Unsatisfiable states can be removed from the search tree: all successors are also unsatisfiable.
- Attack states: as before + the symbolic state must be satisfiable and conditions on the intruder knowledge (e.g. secrecy) are translated into appropriate constraints.

Lazy Intruder: Summary



- With the naive approach, most states of our search tree have infinitely many successors, because for rcv(t) there are usually infinitely many σ with $t\sigma \in \mathcal{DY}(M)$.
- We avoid the enumeration by using symbolic states with constraints. This gives us at most one successor per thread.
- 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)

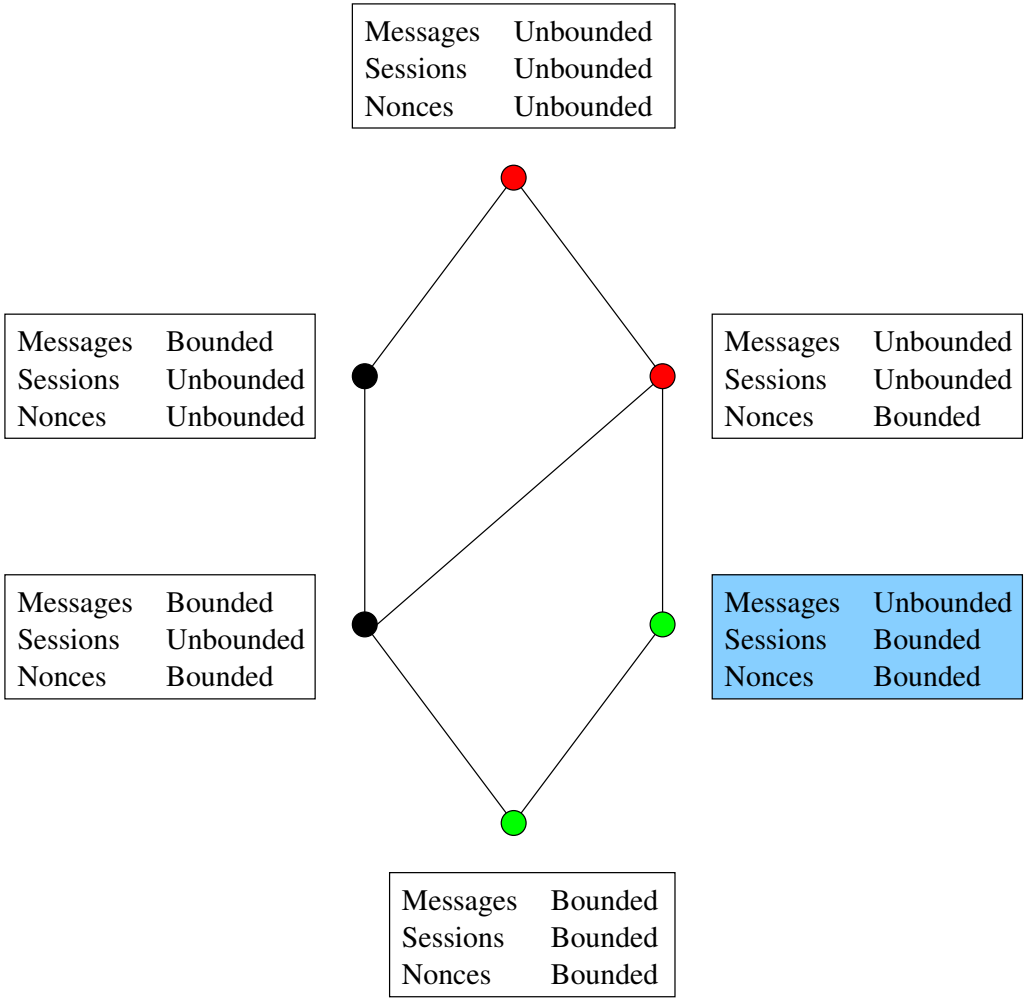
Protocol insecurity for a bounded number of sessions is NP-complete.

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. □

Roadmap



The remaining two for unbounded sessions: next week.

Interleaving

- By lazy intruder: for each node at the most one successor per thread.
- Still a lot: search tree with branching degree around k for k threads.
- Classic problem: several programs running independently in parallel can give rise to an exponential number of interleavings —even without an intruder.

$process_a$	$process_b$
a_1	
a_2	
	b_1
a_3	
	b_2
	b_3

Interleaving Problem

Standard Solutions in Model-Checking:

Pretending Atomicity

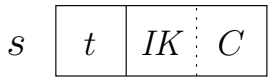
- Consider the thread $\mathrm{rcv}(m) \cdot \mathrm{snd}(m') \cdot tl$.
- It is not a restriction to consider receiver and send here as atomic: when executing the receive, we directly proceed with the following send without any other transitions in between.
- Atomicity reduces interleavings, but we must ensure that we do not exclude attacks here.

Partial-Order Reduction

- Consider two processes a and b that run in parallel and that have *nothing to do with each other*.
- It may not make much of a difference whether
  - ★ first a makes a step and then b
  - ★ first b and then a
- In some cases, it is thus possible to eliminate one of the two interleavings.

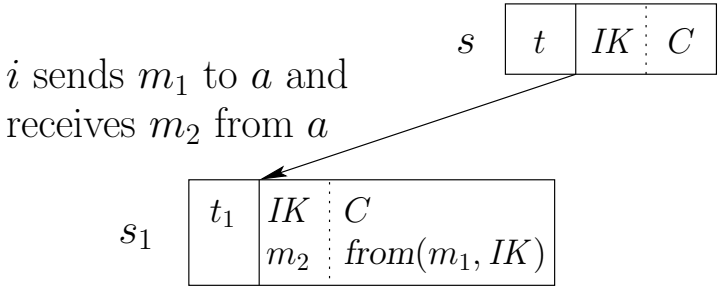
Constraint Differentiation: Idea

Symbolic states are depicted as (t, IK, C) where t is the set of threads, IK the intruder knowledge, C the set of constraints.



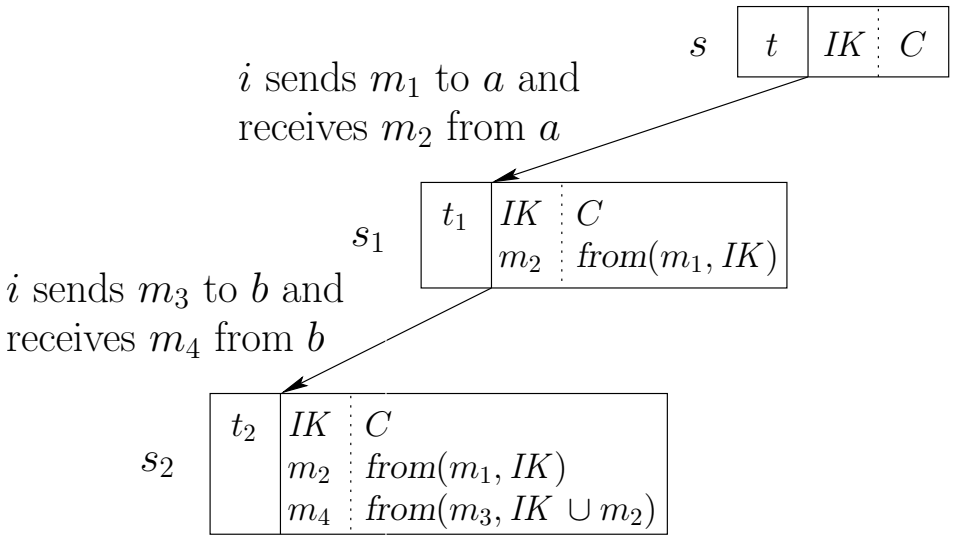
Constraint Differentiation: Idea

Symbolic states are depicted as (t, IK, C) where t is the set of threads, IK the intruder knowledge, C the set of constraints.



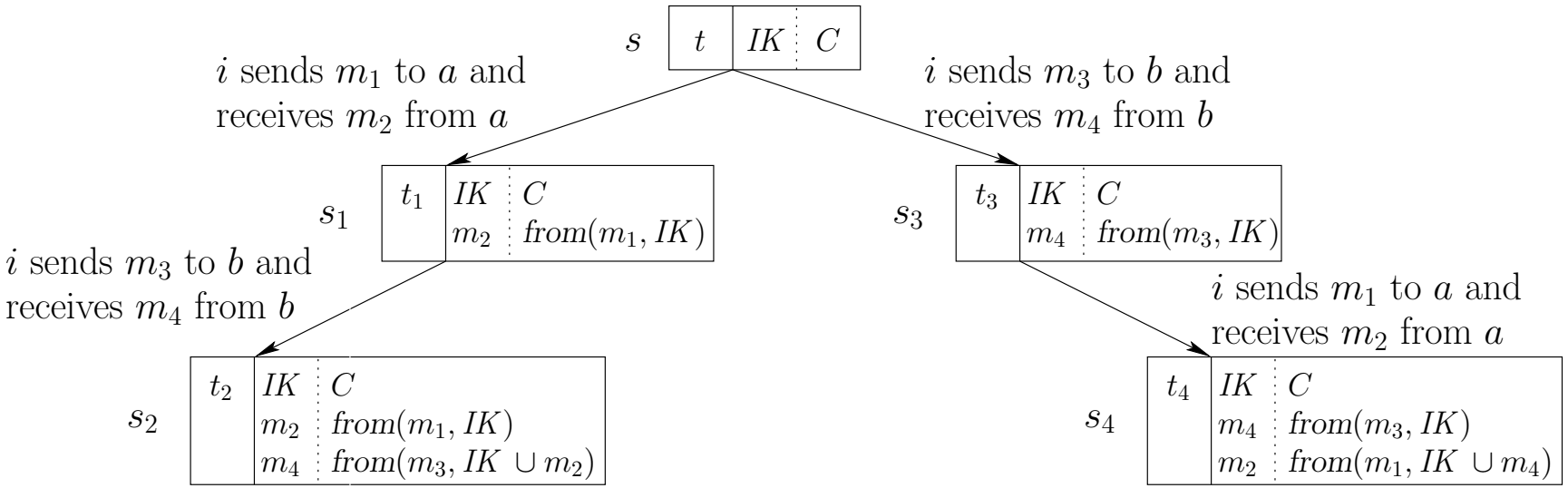
Constraint Differentiation: Idea

Symbolic states are depicted as (t, IK, C) where t is the set of threads, IK the intruder knowledge, C the set of constraints.



Constraint Differentiation: Idea

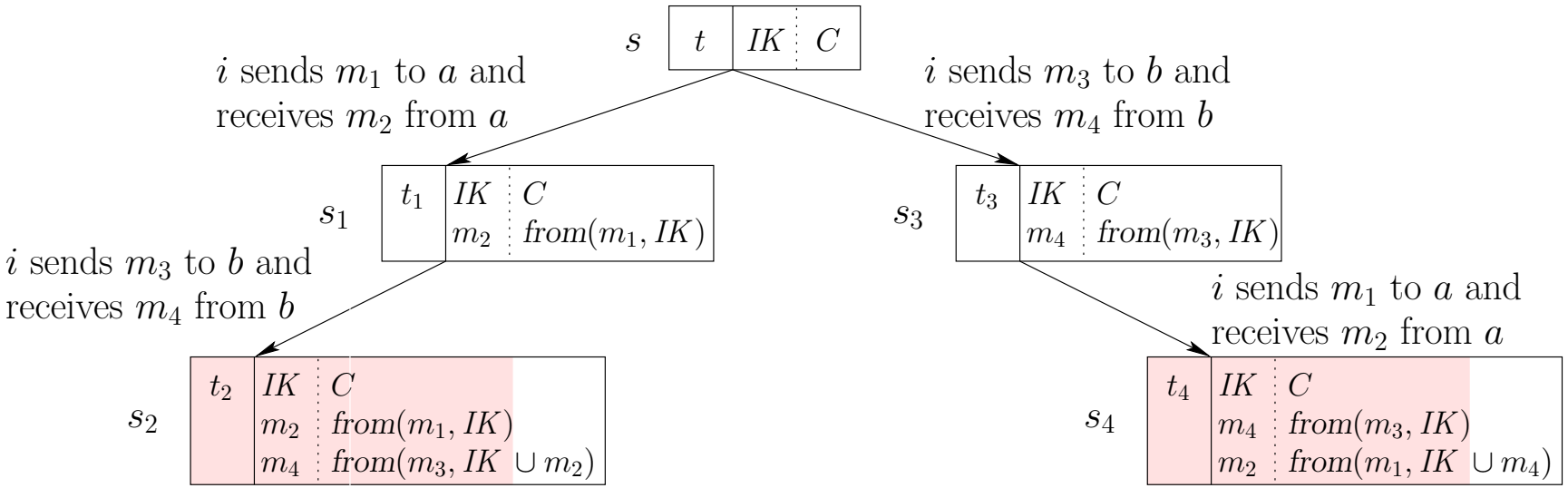
Symbolic states are depicted as (t, IK, C) where t is the set of threads, IK the intruder knowledge, C the set of constraints.



where $t_2 = t_4$. Note that the corresponding traces are also equal up to reordering of events.

Constraint Differentiation: Idea

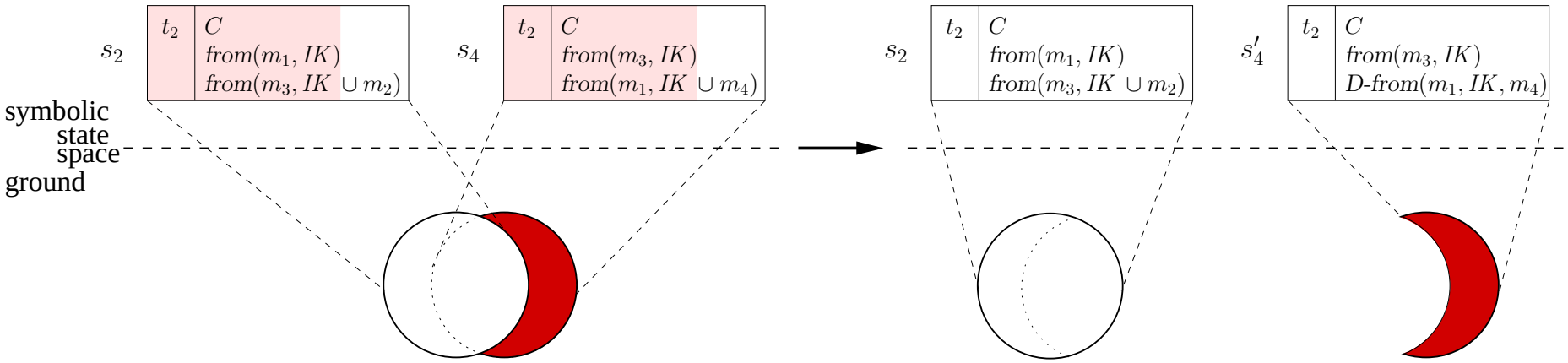
Symbolic states are depicted as (t, IK, C) where t is the set of threads, IK the intruder knowledge, C the set of constraints.



where $t_2 = t_4$. Note that the corresponding traces are also equal up to reordering of events.

- Idea: exploit redundancies in the symbolic states, i.e. reduction exploits overlapping of the sets of ground states.

Constraint Differentiation

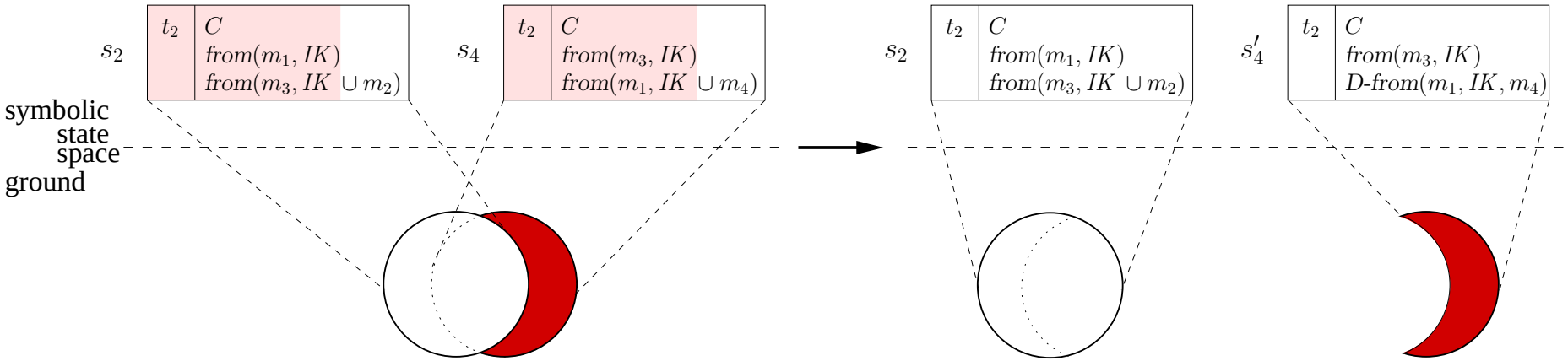


- New kind of constraints: $D\text{-}from(T; IK; NIK)$.
- Intuition:
  - ★ Intruder has just learned some new intruder knowledge NIK.
  - ★ All solutions $[\![from(T; IK \cup NIK)]\!]$ are "correct" but a solution is interesting only if it requires NIK.

$$[\![D\text{-}from(T; IK; NIK)]\!] = [\![from(T; IK \cup NIK)]\!] \setminus [\![from(T; IK)]\!].$$

- Variants of reduction rules for *D-from* constraints and corresponding correctness theorem.

Constraint Differentiation



The constraint-differentiation technique allows to prune many subtrees of the search tree:

- directly when the red part is empty (and the *D-from* constraints thus unsatisfiable).
- indirectly in successor states of a "differentiated" state.

The pruning usually results in a reduction of the average branching degree of the search tree.

The Instantiation Problem

So far, we instantiate all roles with concrete agent names in the initial state:

Example

$th_1 \quad NSPK(A)[A \mapsto a, B \mapsto i]$
$th_2 \quad NSPK(B)[B \mapsto b]$

- Actually, we do not want to do that manually!
- Problem: given n uninstantiated threads, compute the set of all instances of the role names in these threads with agent names.
- Even if there are just two agents (a and i), the number of instances is exponential in the number of threads n.
- Naively enumerating all instances and model checking each of them is inefficient.

Symbolic sessions

Idea: let the intruder choose the instantiation—lazily!

- Assume that all agent names are public: $a, b, c, i, \ldots \in IK_0$.
- Use disjoint variables for agent names in initial threads.
- Have an initial constraint $from(Ag; IK_0)$ where Ag is the set of agent variables in the threads.

Example

- $\begin{array}{ll} th_1 & NSPK(A)[A \mapsto A_1, B \mapsto B_1] \\ th_2 & NSPK(B)[B \mapsto B_2] \end{array}$
- $from(A_1, B_1, B_2; a, b, c, d, \ldots)$.
- Constraint reduction instantiates $B_1 \mapsto i$ during search.
- No further instantiations are necessary: the attack works for any agents A_1 and B_2!

Symbolic sessions use the lazy intruder to do also the instantiation in a demand driven way and thus avoid the complete enumeration.

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.