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.
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.
Introduction

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.
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?
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

\[
\begin{align*}
s_1 &= 1 & s_2 &= 10 & s_3 &= 011 \\
t_1 &= 101 & t_2 &= 00 & t_3 &= 11
\end{align*}
\]

Has a solution: \(s_1s_3s_2s_3 = 101110011 = t_1t_3t_2t_3\).
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**

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

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 $\langle c_1, \langle c_2, \langle \ldots, \langle c_n, Y \rangle \rangle \rangle$.
- Note: $\langle \cdot, \cdot \rangle$ is not associative.
- The goal is that the intruder never obtains $secret$. 
Reducing PCP to Protocol Security: Example

Example

\[
\begin{align*}
  s_1 &= 1 & s_2 &= 10 & s_3 &= 011 \\
  t_1 &= 101 & t_2 &= 00 & t_3 &= 11
\end{align*}
\]

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

Attack

- The intruder "guesses" the solution 1323.
- He sends \( \langle\langle 3, \langle 2, \langle 3, \langle 1, $\rangle\rangle\rangle, \langle$, $\rangle\rangle \) to \( R_1 \),
- \( R_1 \) replies: \( \langle\langle 3, \langle 2, \langle 3, \langle 1, $\rangle\rangle\rangle, \langle$, $\rangle\rangle|_k \)
- Using the \( R_2^j \) we get in several steps: \( \langle\langle $, \langle t, t\rangle\rangle\rangle|_k \) for \( t = \langle 1, \langle 0, \langle 1, \langle 1, \langle 0, \langle 0, \langle 1, \langle 1, $\rangle\rangle\rangle\rangle\rangle\rangle\rangle \)
- 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 . . .
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 \textit{send}, \textit{receive}, or \textit{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.
Introduction

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 \( \text{rcv}(\{N_1, N_2, N_3\}_{pk(a)}) \); \( N_i : \text{nonce} \).
- The intruder knows nonces \( n_1, \ldots, n_5 \) and \( pk(a) \).
- 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

$\text{DY}(M)$ is infinite, but $t \notin \text{DY}(M)$ 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)**

\[
\frac{m \in \mathcal{DY}_C(M)}{m \in \mathcal{DY}_C(M)} \quad \text{Axiom (} m \in M \text{)}
\]

\[
t_1 \in \mathcal{DY}_C(M) \quad \ldots \quad t_n \in \mathcal{DY}_C(M)
\]

\[
f(t_1, \ldots, t_n) \in \mathcal{DY}_C(M)
\]

Composition (\( f \in \Sigma_p \))

**Definition (Decomposing Intruder)**

\[
\frac{m \in \mathcal{DY}_D(M)}{m \in \mathcal{DY}_D(M)} \quad \text{Axiom (} m \in M \text{)}
\]

\[
\frac{\langle m_1, m_2 \rangle \in \mathcal{DY}_D(M)}{m_i \in \mathcal{DY}_D(M)} \quad \text{Proj}_i
\]

\[
\frac{\{m\}_k \in \mathcal{DY}_D(M)}{k \in \mathcal{DY}_C(\mathcal{DY}_D(M))} \quad \text{DecSym}
\]

\[
\frac{\{m\}_k \in \mathcal{DY}_D(M)}{\text{inv}(k) \in \mathcal{DY}_C(\mathcal{DY}_D(M))} \quad \text{DecAsym}
\]

\[
\frac{\{m\}_{\text{inv}(k)} \in \mathcal{DY}_D(M)}{m \in \mathcal{DY}_D(M)} \quad \text{OpenSig}
\]

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.:
    \[
    \prod_{m \in \mathcal{DY}(M)} \prod_{k \in \mathcal{DY}(M)} m \; \langle m \rangle_k \rightarrow \prod_{m \in \mathcal{DY}(M)} m
    \]
  - 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)**

\[
M = \{ |k| h(n_1, n_2), \{ n_1 \} \text{pk}(i), \{ n_2 \} \text{inv(pk}(a)) \}, \text{pk}(a), \text{pk}(i), \text{inv(pk}(i)), \{| \text{secret} | \}_{k} \}
\]

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

Thus

1. \( \text{secret} \in \mathcal{DY}(M) \) since \( \text{secret} \in M' \)
2. \( t = \{ \text{secret} \}_{\text{inv(pk}(a))} \notin \mathcal{DY}(M) \) since \( t \notin \mathcal{DY}_C(M') \).
3. \( t = \{| n_1 |\} h(k, \text{secret}) \in \mathcal{DY}(M) \) 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 $\text{rcv}(t)$ where $t$ has variables, find substitutions $\sigma$ 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:

\[
\text{from}(\{X\}; M).
\]

- 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 a pair \((C, \sigma)\) of a set \(C\) of constraints of the form \(\text{from}(T; M)\) where \(T\) and \(M\) are sets of messages, and \(\sigma\) is a substitution.

A constraint store \((C, \sigma)\) is called simple iff \(T \subseteq \mathcal{V}\) for all \(\text{from}(T; M) \in C\).

The semantics \(\llbracket(C, \sigma)\rrbracket\) 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 \(x \in \text{dom}(\tau)\)
- \(T\tau \subseteq \mathcal{D}\mathcal{Y}(M\tau)\) for each \(\text{from}(T; M) \in C\).

A constraint store \((C, \sigma)\) is called satisfiable iff \(\llbracket C, \sigma \rrbracket \neq \emptyset\).

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

Example

\[ M = \{ \{ n_1, a \}_k, \{ n_2 \}_k, \{ n_2 \}_k, k_2 \} \]

For an agent expecting \( \{ X \}_k \) we get the constraint store:

\[
\left( \{ \text{from}(\{ X \}_k; M) \}, [ \ ] \right)
\]

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

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

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

\[
\mathcal{L}(C, \sigma) = \mathcal{L}(C_1, \sigma_1) \cup \ldots \cup \mathcal{L}(C_n, \sigma_n)
\]

If \((C, \sigma)\) is unsatisfiable, then \(n = 0\).
Lazy Intruder Reduction

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

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

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

\textbf{Theorem}

\textit{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:

\[
\begin{align*}
(0, \text{snd}(\{na_0, a\}_{pk(i)})) & \quad IK_1 = IK_0 \cup \{na_0, a\}_{pk(i)} \\
(1, \text{rcv}(\{NA, A\}_{pk(b)})) & \quad \text{from}(\{NA, A\}_{pk(b)}; IK_1) \\
(1, \text{snd}(\{NA, nb_1\}_{pk(A)})) & \quad IK_2 = IK_1 \cup \{NA, nb_1\}_{pk(A)} \\
(0, \text{rcv}(\{na_0, NB\}_{pk(a)})) & \quad \text{from}(\{na_0, NB\}_{pk(a)}; IK_2) \\
(0, \text{sig}(\text{secret}, a, i, \langle na_0, NB \rangle)) & \quad IK_3 = IK_2 \cup \{NB\}_{pk(i)} \\
(0, \text{snd}(\{NB\}_{pk(i)})) & \quad \text{from}(\{nb_1\}_{pk(b)}; IK_3) \\
(1, \text{rcv}(\{nb_1\}_{pk(b)})) & \quad \text{from}(\langle NA, nb_1 \rangle; IK_3) \\
(1, \text{sig}(\text{secret}, b, a, \langle NA, nb_1 \rangle)) \\
\end{align*}
\]

*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 $pk(i)$:

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

from($\{ NA, A \}_{pk(b)}; IK_1$)

$$IK_2 = IK_1 \cup \{ NA, nb_1 \}_{pk(A)}$$

from($\{ na_0, NB \}_{pk(a)}; IK_2$)

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

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

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

Let us try the composition rule on the the first message

$$\left( \{ \text{from}(\{ t_1, \ldots, t_n \} \cup T; M) \} \cup C, \sigma \right)$$

Compose ($t = f(t_1, \ldots, t_n), f \in \Sigma_p$)

$$\left( \{ \text{from}(\{ t \} \cup T; M) \} \cup C, \sigma \right)$$

First with $f = \{ \cdot \}$, and then $f = \langle \cdot, \cdot \rangle \ldots$
Example (The NSPK attack—lazily)

\[ \text{IK}_1 = \text{IK}_0 \cup \{ \text{na}_0, a \}_{\text{pk}(i)}, \text{na}_0 \]

from(\(\text{pk}(b), \text{NA}, A; \text{IK}_1\))

\[ \text{IK}_2 = \text{IK}_1 \cup \{ \text{NA}, \text{nb}_1 \}_{\text{pk}(A)} \]

from(\(\{ \text{na}_0, \text{NB} \}_{\text{pk}(a)}; \text{IK}_2\))

\[ \text{IK}_3 = \text{IK}_2 \cup \{ \text{NB} \}_{\text{pk}(i)}, \text{NB} \]

from(\(\{ \text{nb}_1 \}_{\text{pk}(b)}; \text{IK}_3\))

from(\(\langle \text{NA}, \text{nb}_1 \rangle; \text{IK}_3\))

\(\text{pk}(b) \in \text{IK}_0\), thus we can eliminate that using the unify rule (\(\tau = [ ]\)):

\[
\frac{((\{ \text{from}(T; M) \} \cup C)_{\tau}, \sigma_{\tau})}{(\{ \text{from}(\{ t \} \cup T; M) \} \cup C, \sigma)} \quad \text{Unify (} t \not\in \mathcal{V}, s \in M, \tau = \text{mgu}(s, t))
\]
Example (The NSPK attack—lazily)

\[ IK_1 = IK_0 \cup \{ na_0, a \}_{pk(i)}, na_0 \]

from \((NA, A; IK_1)\)

\[ IK_2 = IK_1 \cup \{ NA, nb_1 \}_{pk(A)} \]

from \((\{ na_0, NB \}_{pk(a)}; IK_2)\)

\[ IK_3 = IK_2 \cup \{ NB \}_{pk(i)}, NB \]

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

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)

\[ \text{IK}_1 = \text{IK}_0 \cup \{na_0, a\}_{pk(i)}; na_0 \]

from (NA, A; IK_1)

\[ \text{IK}_2 = \text{IK}_1 \cup \{NA, nb_1\}_{pk(A)} \]

from (\{na_0, NB\}_{pk(a)}; IK_2)

\[ \text{IK}_3 = \text{IK}_2 \cup \{NB\}_{pk(i)}, NB \]

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

from (⟨NA, nb_1⟩; IK_3)

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

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

Thus \(\tau = [NA \mapsto na_0, NB \mapsto nb_1, A \mapsto a]\).
Example (The NSPK attack—lazily)

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

\[ IK_1 = IK_0 \cup \{na_0, a\}_{pk(i)}, na_0 \]

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

\[ IK_2 = IK_1 \cup \{na_0, nb_1\}_{pk(a)} \]

from \((\emptyset; IK_2)\)

\[ IK_3 = IK_2 \cup \{nb_1\}_{pk(i)}, nb_1 \]

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

from \((\langle na_0, nb_1 \rangle; IK_3)\)
Example (The NSPK attack—lazily)

There are some terms that can be generated trivially now:

\[ IK_1 = IK_0 \cup \{ na_0, a \}_{pk(i)}, na_0 \]
\[ from(na_0, a; IK_1) \]
\[ IK_2 = IK_1 \cup \{ na_0, nb_1 \}_{pk(a)} \]
\[ IK_3 = IK_2 \cup \{ nb_1 \}_{pk(i)}, nb_1 \]
\[ from(\{ nb_1 \}_{pk(b)}; IK_3) \]
\[ from(\langle na_0, nb_1 \rangle; IK_3) \]
Example (The NSPK attack—lazily)

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

\[
IK_1 = IK_0 \cup \{na_0, a\}_{pk(i)}, na_0
\]

\[
IK_2 = IK_1 \cup \{na_0, nb_1\}_{pk(a)}
\]

\[
IK_3 = IK_2 \cup \{nb_1\}_{pk(i)}, nb_1
\]

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

from ($\langle na_0, nb_1 \rangle; IK_3$)
Example (The NSPK attack—lazily)

And these are all again in the respective $IK$s:

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

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

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

from $(pk(b), nb_1; IK_3)$

from $(na_0, nb_1; IK_3)$

So we have found a solution and thus an attack!
So far, we have:

**Transition Rules**

\[ State = Trace \times IntruderKnowledge \times Threads \]
where \( Trace, IntruderKnowledge \) ground and \( Threads \) closed.

\[
\begin{align*}
th(tid) &= snd(t) \cdot tl \\
(tr, IK, th) \rightarrow (tr \cdot (tid, snd(t)), IK \cup \{t\}, th[tid \mapsto tl]) & \text{snd} \\
tr(tid) &= rcv(t) \cdot tl \\
dom(\sigma) &= var(t) \\
& t\sigma \in DY(IK) \text{ } & \text{rcv} \\
(tr, IK, th) \rightarrow (tr \cdot (tid, rcv(t\sigma)), IK, th[tid \mapsto tl\sigma])
\end{align*}
\]

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

\[ \text{SymbolicState} = \text{Trace} \times \text{IntruderKnowledge} \times \text{Threads} \times \mathcal{P}(\text{Constraint}) \]

where \( \text{Trace}, \text{IntruderKnowledge}, \text{Threads} \)

may have free variables that occur in the constraints.

\[ th(tid) = \text{snd}(t) \cdot tl \]

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

\[ th(tid) = \text{rcv}(t) \cdot tl \]

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

- 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 \( \text{rcv}(t) \) there are usually infinitely many \( \sigma \) 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.
The remaining two for unbounded sessions: next week.
Introduction

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.

\[
\begin{array}{c|c|c}
\text{process}_a & \text{process}_b \\
\hline
a_1 & \hphantom{a_2} \\
\hphantom{a_1} & a_2 \\
\hphantom{a_1} & a_3 \\
\hphantom{a_2} & b_1 \\
\hphantom{a_3} & b_2 \\
\hphantom{a_2} & b_3 \\
\end{array}
\]
Interleaving Problem

Standard Solutions in Model-Checking:

**Pretending Atomicity**

- Consider the thread $\text{rcv}(m) \cdot \text{snd}(m') \cdot tl$.
- 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
  - $\star$ first $a$ makes a step and then $b$
  - $\star$ 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.

\(i\) sends \(m_1\) to \(a\) and receives \(m_2\) from \(a\).

\[
\begin{array}{c|c|c|c}
\text{\textbf{s}} & t & IK & C \\
\hline
\text{\textbf{s}_1} & t_1 & IK & C \\
& m_2 & \text{from}(m_1, IK) & \\
\end{array}
\]
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.

\[
\begin{align*}
\text{s} & \quad \begin{array}{c|c|c}
\text{t} & \text{IK} & \text{C} \\
\end{array} \\
\text{s}_1 & \quad \begin{array}{c|c|c}
\text{t}_1 & \text{IK} & \text{C} \\
\text{m}_2 & \text{from}(\text{m}_1, \text{IK}) \\
\end{array} \\
\text{s}_2 & \quad \begin{array}{c|c|c}
\text{t}_2 & \text{IK} & \text{C} \\
\text{m}_2 & \text{from}(\text{m}_1, \text{IK}) \\
\text{m}_4 & \text{from}(\text{m}_3, \text{IK} \cup \text{m}_2) \\
\end{array}
\end{align*}
\]

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

\[
\begin{align*}
\text{\(s_1\)} & \quad \begin{array}{|c|c|}
\hline
\text{t}_1 & \text{IK} \\
\text{\textbf{\(m_2\)}} & \text{\textbf{\textit{from}(m_1, IK)}} \\
\hline
\end{array} \\
i & \text{ sends } m_1 \text{ to } a \text{ and} \\
& \text{ receives } m_2 \text{ from } a
\end{align*}
\]

\[
\begin{align*}
\text{\(i\)} & \quad \text{s} \\
\text{IK} & \text{C} \\
\hline
\end{align*}
\]

\[
\begin{align*}
\text{\(s_2\)} & \quad \begin{array}{|c|c|}
\hline
\text{t}_2 & \text{IK} \\
\text{\textbf{\(m_2\)}} & \text{\textbf{\textit{from}(m_1, IK)}} \\
\text{\textbf{\(m_4\)}} & \text{\textbf{\textit{from}(m_3, IK \cup m_2)}} \\
\hline
\end{array} \\
i & \text{ sends } m_3 \text{ to } b \text{ and} \\
& \text{ receives } m_4 \text{ from } b
\end{align*}
\]

\[
\begin{align*}
\text{\(s_3\)} & \quad \begin{array}{|c|c|}
\hline
\text{t}_3 & \text{IK} \\
\text{\textbf{\(m_4\)}} & \text{\textbf{\textit{from}(m_3, IK)}} \\
\hline
\end{array} \\
i & \text{ sends } m_1 \text{ to } a \text{ and} \\
& \text{ receives } m_2 \text{ from } a
\end{align*}
\]

\[
\begin{align*}
\text{\(s_4\)} & \quad \begin{array}{|c|c|}
\hline
\text{t}_4 & \text{IK} \\
\text{\textbf{\(m_4\)}} & \text{\textbf{\textit{from}(m_3, IK)}} \\
\text{\textbf{\(m_2\)}} & \text{\textbf{\textit{from}(m_1, IK \cup m_4)}} \\
\hline
\end{array} \\
i & \text{ sends } m_1 \text{ to } a \text{ and} \\
& \text{ receives } m_2 \text{ from } a
\end{align*}
\]

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.

\[ \begin{array}{c}
\text{i sends } m_1 \text{ to } a \text{ and receives } m_2 \text{ from } a \\
\text{i sends } m_3 \text{ to } b \text{ and receives } m_4 \text{ from } b \\
\text{i sends } m_3 \text{ to } b \text{ and receives } m_4 \text{ from } b \\
\text{i sends } m_1 \text{ to } a \text{ and receives } m_2 \text{ from } a \\
\end{array} \]

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.
• New kind of constraints: $D$-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 $NIK$.

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

• Variants of reduction rules for $D$-from constraints and corresponding correctness theorem.
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:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$th_1$  $NSPK(A)[A \mapsto a, B \mapsto i]$</td>
</tr>
<tr>
<td>$th_2$  $NSPK(B)[B \mapsto b]$</td>
</tr>
</tbody>
</table>

- 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**

\[\begin{align*}
th_1 & \quad NSPK(A)[A \mapsto A_1, B \mapsto B_1] \\
th_2 & \quad NSPK(B)[B \mapsto B_2]
\end{align*}\]

- \(from(A_1, B_1, B_2; a, b, c, d, \ldots)\).
- 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