Reducing the number of agents in equivalence properties.

Antoine Dallon

July 8\textsuperscript{th}, 2015
Introduction

- Security properties should hold for an unbounded number of agents using the protocol.
- We often forget about it.
- I will show that with some hypothesis, it is sufficient to verify a protocol with a small number of agents.
Introduction

- Security properties should hold for an unbounded number of agents using the protocol.
- We often forget about it.
- I will show that with some hypothesis, it is sufficient to verify a protocol with a small number of agents.
Introduction

- Security properties should hold for an unbounded number of agents using the protocol.
- We often forget about it.
- I will show that with some hypothesis, it is sufficient to verify a protocol with a small number of agents.
Goal

Theorem (Informal)

There exists a class $C$ of protocols (containing at least some interesting examples) such that for every $P, Q \in C$, if there exists an attack against $P \approx Q$, then there is an attack against $P \approx Q$ involving a small finite and calculable number of agents.
Table of contents

1. Agents and nonces
2. Extension
3. Negative results
## Table of contents

1. **Agents and nonces**
2. **Extension**
3. **Negative results**
Bounding the number of agents implies controlling the keys.

- So we assume that there are private unary function symbols $pk$, $sk$, $hon$, $dis$.
- $pk$ represents the public key of an agent, $sk$ the private key.
- $hon$ and $dis$ are predicates that represent the fact that an agent is honest or dishonest.
- We can also add a binary function symbol $shk$ that will represent shared keys.
Generalities

- Bounding the number of agents implies controlling the keys.
- So we assume that there are private unary function symbols $pk$, $sk$, $hon$, $dis$.
  - $pk$ represents the public key of an agent, $sk$ the private key.
  - $hon$ and $dis$ are predicates that represent the fact that an agent is honest or dishonest.
  - We can also add a binary function symbol $shk$ that will represent shared keys.
Generalities

- Bounding the number of agents implies controlling the keys.
- So we assume that there are private unary function symbols \( pk, \ sk, \ hon, \ dis \).
- \( pk \) represents the public key of an agent, \( sk \) the private key.
- \( hon \) and \( dis \) are predicates that represent the fact that an agent is honest or dishonest.
- We can also add a binary function symbol \( shk \) that will represent shared keys.
Generalities

- Bounding the number of agents implies controlling the keys.
- So we assume that there are private unary function symbols \( pk, sk, hon, dis \).
- \( pk \) represents the public key of an agent, \( sk \) the private key.
- \( hon \) and \( dis \) are predicates that represent the fact that an agent is honest or dishonest.
- We can also add a binary function symbol \( shk \) that will represent shared keys.
Generalities

- Bounding the number of agents implies controlling the keys.
- So we assume that there are private unary function symbols \( pk, sk, hon, dis \).
- \( pk \) represents the public key of an agent, \( sk \) the private key.
- \( hon \) and \( dis \) are predicates that represent the fact that an agent is honest or dishonest.
- We can also add a binary function symbol shk that will represent shared keys.
\[ P_{Header} = \]

\begin{align*}
! \text{new } c & . \text{out}(c_H, c). \text{new } ag_H . \text{out}(c, < ag_H, \text{pk}(ag_H), \text{hon}(ag_H) >) \\
| ! \text{new } c & . \text{out}(c_D, c). \text{new } ag_D . \text{out}(c, < ag_D, \text{pk}(ag_D), \text{dis}(ag_D), \text{sk}(ag_D) >) \\
| K
\end{align*}

where \( K \), defined as follows, is used to do key distribution:

\[ K = ! \text{new } c . \text{out}(c_K, c). \text{in}(\text{dis}(x), \text{pk}(y)). \text{out}(\text{shk}(x, y)) \]
Example: Needham-Schroeder-Lowe
The protocol

1. $A \rightarrow B : \{N_a, A\}_{pk(B)}$
2. $B \rightarrow A : \{N_a, N_b, B\}_{pk(A)}$
3. $A \rightarrow B : \{N_b\}_{pk(B)}$
4. $P : B \rightarrow ? : \{m_1\}_{N_b}$ or $Q : B \rightarrow ? : \{m_1\}_k$ where $k$ is a fresh key.
Example: Needham-Schroeder-Lowe

The role of $A$

\[ P_A = Q_A = \]
\[ ! \text{new } c. \text{out}(c_A, c). \]
\[ \text{in}(c, < \text{pk}(x_a), \text{pk}(x_b)>). \]
\[ \text{new } n_a. \]
\[ \text{out}(c, \text{aenc}(< n_a, x_a >, \text{pk}(x_b))). \]
\[ \text{in}(c, \text{aenc}(< n_a, x_nb, x_b >, \text{pk}(x_a))). \]
\[ \text{out}(c, \text{aenc}(x_nb, \text{pk}(x_b))). \]
Example: Needham-Schroeder-Lowe
The honest role of B - P side

\[ P_B^H = \]
\[ \text{! new } c. \text{ out}(c_{BH}, c). \]
\[ \text{in}(c, < \text{hon}(z_a), \text{hon}(z_b)>). \]
\[ \text{in}(c, \text{aenc}(< z_{na}, z_a>, \text{pk}(z_b))). \]
\[ \text{new } n_b. \]
\[ \text{out}(c, \text{aenc}(< z_{na}, n_b, z_b>, \text{pk}(z_a))). \]
\[ \text{in}(c, \text{aenc}(n_b, \text{pk}(z_b))). \]
\[ \text{out}(c, \text{senc}(m_1, n_b)) \]
Example: Needham-Schroeder-Lowe
The honest role of B - P side

\[ P_B^H = \]
! new c. out(\(c_{BH}, c\)).
in(c, \langle \text{hon}(z_a), \text{hon}(z_b) \rangle).
in(c, \text{aenc}(\langle z_{na}, z_a \rangle, \text{pk}(z_b))).
new n_b.
out(c, \text{aenc}(\langle z_{na}, n_b, z_b \rangle, \text{pk}(z_a))).
in(c, \text{aenc}(n_b, \text{pk}(z_b))).
out(c, \text{senc}(m_1, n_b))
Example: Needham-Schroeder-Lowe
The honest role of B - Q side

\[ Q_B^H = \]

! new c. out \((c_{BH}, c)\).

in \((c, < \text{hon}(z_a), \text{hon}(z_b) >)\).

in \((c, \text{aenc}(< z_{na}, z_a >, \text{pk}(z_b)))\).

new \(n_b\).

out \((c, \text{aenc}(< z_{na}, n_b, z_b >, \text{pk}(z_a)))\).

in \((c, \text{aenc}(z_{nb}, \text{pk}(z_b)))\).

new \(k\). out \((c, \text{senc}(m_2, k))\)
Example: Needham-Schroeder-Lowe
The honest role of B - Q side

\[ Q^H_B = \]
\[
! \text{new } c. \text{ out}(c_{BH}, c).
\]
\[
\text{in}(c, < \text{hon}(z_a), \text{hon}(z_b) >).
\]
\[
\text{in}(c, \text{aenc}(< z_{na}, z_a >, \text{pk}(z_b))).
\]
\[
\text{new } n_b.
\]
\[
\text{out}(c, \text{aenc}(< z_{na}, n_b, z_b >, \text{pk}(z_a))).
\]
\[
\text{in}(c, \text{aenc}(z_{nb}, \text{pk}(z_b))).
\]
\[
\text{new } k. \text{ out}(c, \text{senc}(m_2, k))
\]
Example: Needham-Schroeder-Lowe
The dishonest role of B

\[ P_B^D = Q_B^D = \]
\[ ! \text{new}\ c.\ \text{out}(c_{BD}, c). \]
\[ \text{in}(c, <\ pk(y_a), pk(y_b) >). \]
\[ \text{in}(c, aenc(< y_{na}, y_a >, pk(y_b))). \]
\[ \text{new}\ n_b. \]
\[ \text{out}(c, aenc(< y_{na}, n_b, y_b >, pk(y_a))). \]
\[ \text{in}(c, aenc(y_{nb}, pk(y_b))). \]

We get:

\[ P = P_{Header} | P_A | P_B^D | P_B^H \quad \text{and} \quad Q = P_{Header} | Q_A | Q_B^D | Q_B^H \]
Example: Needham-Schroeder-Lowe
The dishonest role of B

\[ P_B^D = Q_B^D = \]
\[ ! \text{new } c. \text{ out}(c_{BD}, c). \]
\[ \text{in}(c, < \text{pk}(y_a), \text{pk}(y_b)>). \]
\[ \text{in}(c, \text{aenc}(< y_{na}, y_a>, \text{pk}(y_b))). \]
\[ \text{new } n_b. \]
\[ \text{out}(c, \text{aenc}(< y_{na}, n_b, y_b>, \text{pk}(y_a))). \]
\[ \text{in}(c, \text{aenc}(y_{nb}, \text{pk}(y_b))) \]

We get:

\[ P = P_{Header} \| P_A \| P_B^D \| P_B^H \] and \[ Q = P_{Header} \| Q_A \| Q_B^D \| Q_B^H \]
Rémy’s Transformation
The transformation

Let $P$ be a simple protocol. Let $N$ be a set of nonces.

- We denote by $P^N$ the protocol $P$ where we have removed every instruction new $n$ with $n \in N$.
- We denote by $B(c)$ the process occurring in $P$ on channel $c$, and by $B^*(c^*)$ the process obtained from $B(c)$ renaming each new $n$ into new $n^*$.

**Definition**

$$P^N,c = P^N | B^*_c$$
Rémy’s Transformation
The transformation

Let $P$ be a simple protocol. Let $N$ be a set of nonces.

- We denote by $\overline{P}^N$ the protocol $P$ where we have removed every instruction `new n` with $n \in N$.
- We denote by $B(c)$ the process occurring in $P$ on channel $c$, and by $B^*(c^*)$ the process obtained from $B(c)$ renaming each new $n$ into new $n^*$.

**Definition**

$$\overline{P}^{N,c} = \overline{P}^N | B^*_c$$
Let $P$ be a simple protocol. Let $N$ be a set of nonces.

- We denote by $\overline{P}^N$ the protocol $P$ where we have removed every instruction new $n$ with $n \in N$.
- We denote by $B(c)$ the process occurring in $P$ on channel $c$, and by $B^*(c^*)$ the process obtained from $B(c)$ renaming each new $n$ into new $n^*$.

**Definition**

\[
\overline{P}^{N,c} = \overline{P}^N | B^*_c
\]
Rémy’s Transformation
The transformation

Let $P$ be a simple protocol. Let $N$ be a set of nonces.

- We denote by $\overline{P}^N$ the protocol $P$ where we have removed every instruction new $n$ with $n \in N$.
- We denote by $B(c)$ the process occurring in $P$ on channel $c$, and by $B^*(c^*)$ the process obtained from $B(c)$ renaming each new $n$ into new $n^*$.

**Definition**

$$\overline{P}^{N,c} = \overline{P}^N | B^*_c$$
Rémy’s Transformation
Example: the header

\[
P_{\text{Header}} =
!\text{new } c . \text{out}(c_H, c). \text{new } ag_H . \text{out}(c, < ag_H, \text{pk}(ag_H), \text{hon}(ag_H) >)
|!\text{new } c . \text{out}(c_D, c). \text{new } ag_D . \text{out}(c, < ag_D, \text{pk}(ag_D), \text{dis}(ag_D), \text{sk}(ag_D) >)
| K
\]

We take \( N = \{ ag_H, ag_D \} \) and \( c = c_H \).
Rémý’s Transformation
Example: the header

\[
P_{\text{Header}} = \!
\text{! new } c . \text{ out}(c_H, c). \text{ new } ag_H . \text{ out}(c, < ag_H, pk(ag_H), hon(ag_H) >) \\
| \!
\text{! new } c . \text{ out}(c_D, c). \text{ new } ag_D . \text{ out}(c, < ag_D, pk(ag_D), dis(ag_D), sk(ag_D) >) \\
| K
\]

We take \( N = \{ ag_H, ag_D \} \) and \( c = c_H \).
Rémy’s Transformation

Example

\[
\begin{align*}
\overline{P}_{\text{Header}}^{N,c_H} &= \\
! \text{new } c \cdot \text{out}(c_H, c) . \text{out}(c, < ag_H, pk(ag_H), hon(ag_H) >) & \\
| \text{new } ag_H^* \cdot \text{out}(c_H^*, < ag_H^*, pk(ag_H^*), hon(ag_H^*) >) & \\
|! \text{new } c \cdot \text{out}(c_D, c) . \text{out}(c, < ag_D, pk(ag_D), dis(ag_D), sk(ag_D) >) & \\
| K
\end{align*}
\]
Rémy’s Theorem

Given a process $P$, we denote by $Ch(P)$ the set of public channels names occurring under a replication in $P$.

**Theorem (Rémy)**

Let $P$ and $Q$ be two simple protocols such that $Ch(P) = Ch(Q)$, and $N$ be a set of names. We have that:

$$\forall c \in Ch(P).\overline{P}^{N,c} \approx \overline{Q}^{N,c} \Rightarrow P \approx Q$$
Rémy’s Theorem

Given a process $P$, we denote by $Ch(P)$ the set of public channels names occurring under a replication in $P$.

**Theorem (Rémy)**

Let $P$ and $Q$ be two simple protocols such that $Ch(P) = Ch(Q)$, and $N$ be a set of names. We have that:

\[ \forall c \in Ch(P). P^N_c \approx Q^N_c \Rightarrow P \approx Q \]
From nonces to agents

- We apply this theorem on our model with $N = \{ag_H, ag_D\}$.
- It says that if there is an attack, then there is an attack with 3 agents.
- We need 4 agents (two honests and two dishonests).
- Agents are allowed to talk to themselves.
- The other limits are those of Rémy’s theorem.
From nonces to agents

- We apply this theorem on our model with $N = \{ag_H, ag_D\}$.
- It says that if there is an attack, then there is an attack with 3 agents.
- We need 4 agents (two honests and two dishonests)
- Agents are allowed to talk to themselves.
- The other limits are those of Rémy’s theorem.
From nonces to agents

- We apply this theorem on our model with $N = \{ag_H, ag_D\}$.
- It says that if there is an attack, then there is an attack with 3 agents.
- We need 4 agents (two honests and two dishonests)
  - Agents are allowed to talk to themselves.
- The other limits are those of Rémy’s theorem.
We apply this theorem on our model with $N = \{ag_H, ag_D\}$. It says that if there is an attack, then there is an attack with 3 agents. We need 4 agents (two honests and two dishonests). Agents are allowed to talk to themselves. The other limits are those of Rémy’s theorem.
From nonces to agents

- We apply this theorem on our model with $N = \{ag_H, ag_D\}$.
- It says that if there is an attack, then there is an attack with 3 agents.
- We need 4 agents (two honests and two dishonests).
- Agents are allowed to talk to themselves.
- The other limits are those of Rémy’s theorem.
Table of contents

1. Agents and nonces
2. Extension
3. Negative results
New result:

- Not only for simple protocols.
- More conditionals (disjunctions allowed).
- Errors outputed when tests fail.
New result:

- Not only for simple protocols.
- More conditionals (disjunctions allowed).
- Errors outputted when tests fail.
New result:

- Not only for simple protocols.
- More conditionals (disjunctions allowed).
- Errors outputed when tests fail.
Let $\Sigma = \Sigma_c \cup \Sigma_d$ be a signature.
Let $\mathcal{X}, \mathcal{W}$ be two sets of variables, and $\mathcal{N}$ be a set of names.

**Messages**

Let $M_\Sigma$ be a set of ground constructor terms that stable by renaming (that is $M_\Sigma \rho \subset M_\Sigma$ for any renaming $\rho$). We call messages the element of $M_\Sigma$.

**Constructors, destructors**

The set of rewriting rules $\mathcal{R}$ is a set of rules of the form $d(c_1, \ldots, c_n)$ where $d \in \Sigma_d$ and $c_1, \ldots, c_n \in T(\Sigma_c, \mathcal{X})$. 
Let $\Sigma = \Sigma_c \cup \Sigma_d$ be a signature.
Let $\mathcal{X}, \mathcal{W}$ be two sets of variables, and $\mathcal{N}$ be a set of names.

**Messages**

Let $\mathcal{M}_\Sigma$ be a set of ground constructor terms that stable by renaming (that is $\mathcal{M}_\Sigma \rho \subset \mathcal{M}_\Sigma$ for any renaming $\rho$). We call messages the element of $\mathcal{M}_\Sigma$.

**Constructors, destructors**

The set of rewriting rules $\mathcal{R}$ is a set of rules of the form $d(c_1, \ldots, c_n)$ where $d \in \Sigma_d$ and $c_1, \ldots, c_n \in T(\Sigma_c, \mathcal{X})$. 
Let $\Sigma = \Sigma_c \cup \Sigma_d$ be a signature.

Let $\mathcal{X}, \mathcal{W}$ be two sets of variables, and $\mathcal{N}$ be a set of names.

### Messages

Let $\mathcal{M}_\Sigma$ be a set of ground constructor terms that stable by renaming (that is $\mathcal{M}_\Sigma \rho \subset \mathcal{M}_\Sigma$ for any renaming $\rho$). We call messages the element of $\mathcal{M}_\Sigma$.

### Constructors, destructors

The set of rewriting rules $\mathcal{R}$ is a set of rules of the form $d(c_1, \ldots, c_n)$ where $d \in \Sigma_d$ and $c_1, \ldots, c_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. 
Let $Ch_0$ and $Ch^{fresh}$ be two disjoint sets of public channel names.

**Processes**

Let $Ch$ be an infinite set of channels. We consider processes built using the following grammar:

\[
P, Q ::= 0 \mid !c_iP \mid (P|Q) \mid \text{new } n.P \mid \text{in}(c, u).P \\
\mid \text{out}(c, u).P \mid \text{let } y = v \text{ in } P \text{ else } E \mid \text{if } \text{cond} \text{ then } P \text{ else } E
\]

where $u \in T(\Sigma_c, N \cup X), v \in T(\Sigma, N \cup X), n \in N, e \in E, E$ disjoint from $C$ and $\Sigma_0$ and $c, c' \in Ch$, and:

\[E = 0 | \text{out}(c, e)\]
A configuration is a pair \((\mathcal{P}, \phi)\) where \(\mathcal{P}\) is a multiset of ground processes and \(\phi\) is a substitution \(\phi = \{w_1 \triangleright m_1; \ldots; w_n \triangleright m_n\}\) where \(w_1, \ldots, w_n \in \mathcal{W}\) and \(m_1, \ldots, m_n\) are messages, that is terms in \(\mathcal{M}_\Sigma\).

Sometimes, we may denote \(\{\mathcal{P}\}\) by \(\mathcal{P}\) and \((\mathcal{P}, \emptyset)\) by \(\mathcal{P}\).
The semantics is given by the following rules:

- \((0 \cup P, \phi) \rightarrow^\tau (P, \phi)\)
- \((!c_c P \cup P, \phi) \rightarrow^{\text{sess}(c, ch)} (P\{ch_i / c'_i\} \cup !c, P \cup P, \phi)\) with \(ch_i\) a new fresh channel name.
- \(((P|Q) \cup P, \phi) \rightarrow^\tau (P \cup Q \cup P, \phi)\)
- \((\text{new } n.P, \phi) \rightarrow^\tau (P\{n'/n\} \cup P, \phi)\) with \(n'\) fresh.
- \((\text{in}(c, u).P, \phi) \rightarrow^{\text{in}(c,R)} (P\sigma \cup P, \phi)\) when \(R\phi \downarrow\) and \(u\) are unifiable and where \(\sigma\) is the most general unifier of \(R\phi \downarrow\) and \(u\).
- \((\text{out}(c, u).P, \phi) \rightarrow^{\text{out}(c,w)} (P \cup P, \phi \cup \{w \triangleright u\phi \downarrow\})\)
(let \( x = v \) in \( P \) else \( E \cup P, \phi \)) \( \rightarrow^\tau \) \( (P \{ v \downarrow / x \} \cup P, \phi) \) when \( v \downarrow \in M_\Sigma \).

(let \( x = v \) in \( P \) else \( E \cup P, \phi \)) \( \rightarrow^\tau \) \( (E \cup P, \phi) \) when \( v \downarrow \notin M_\Sigma \).

(if \( t \) then \( P \) else \( E \cup P, \phi \)) \( \rightarrow^\tau \) \( (P \cup P, \phi) \) where \( t = b_1 \lor \cdots \lor b_n \) evaluates as true.

(if \( t \) then \( P \) else \( E \cup P, \phi \)) \( \rightarrow^{\tau, \text{out}(c,e)} \) \( (P, \phi) \) where \( t = b_1 \lor \cdots \lor b_n \) evaluates as false.
Definition

Let $B$ be a process and $c$ a channel name. We say that $B$ is a basic process built on $c$ if it is written in the following grammar:

$$B := 0 \mid \text{in}(c, u).B \mid \text{out}(c, u).B \mid \text{let } x = v \text{ in } B \text{ else } E$$

$$\mid \text{if } t \text{ then } P \text{ else } E \mid \text{new } n.B$$
Simple protocols

Definition

Let \( P \) be a protocol. We say that \( P \) is simple if:

\[
P = !\text{new } c'_1. \text{out}(c_1, c'_1).B_1 | \ldots | !\text{new } c'_m. \text{out}(c_m, c'_m).B_m
\]

\[
| B_{m+1} | \ldots | B_p
\]

where each \( B_i \) for \( 1 \leq i \leq m \) is a basic process built on channel \( c'_i \), and each \( B_i \) for \( m + 1 \leq i \leq p \) is a basic process built on channels \( c_i \).
**Hypothesis**

**Definition (Action-Determinism)**

Let $P$ be a protocol. We say that $P$ is action-deterministic if for every trace $tr$ such that $P \Rightarrow ^{tr} (\mathcal{P}, \phi)$, if there are $Q_1, Q_2 \in \mathcal{P}$, when the visible actions act, act’ pass respectively in $Q_1$ and $Q_2$ then act $\neq$ act’, or they occur on channels $c \neq c'$.

**Definition (Adequate theories)**

Let $\Sigma = \Sigma_c \cup \Sigma_d$ be a signature, $\mathcal{R}$ be a convergent rewriting system and $\mathcal{M}_\Sigma$ be a set of messages. We say that the theory $\mathcal{M}_\Sigma$ is adequate w.r.t. $\mathcal{M}_\Sigma$ when for any term in normal form, there exist $n_1, n_2$ such that for any renaming $\rho$, $n_1 \rho \neq n_2 \rho$ implies $t\rho \notin \mathcal{M}_\Sigma$. 

Hypothesis

Definition (Action-Determinism)

Let $P$ be a protocol. We say that $P$ is action-deterministic if for every trace $tr$ such that $P \Rightarrow^{tr} \Sigma$, if there are $Q_1, Q_2 \in P$, when the visible actions act, $act, act'$ pass respectively in $Q_1$ and $Q_2$ then $act \neq act'$, or they occur on channels $c \neq c'$.

Definition (Adequate theories)

Let $\Sigma = \Sigma_c \cup \Sigma_d$ be a signature, $\mathcal{R}$ be a convergent rewriting system and $\mathcal{M}_\Sigma$ be a set of messages. We say that the theory $\mathcal{M}_\Sigma$ is adequate w.r.t. $\mathcal{M}_\Sigma$ when for any term in normal form, there exist $n_1, n_2$ such that for any renaming $\rho$, $n_1 \rho \neq n_2 \rho$ implies $t \rho \not\in \mathcal{M}_\Sigma$. 

New transformation

Let \( P \) be a protocol in the form:

\[
P = P_S | P_G
\]

where \( P_S \) is simple.

**Definition**

Let \( C = (c_1, \ldots, c_k) \) be a multiset of channels of \( P \). We define \( \overline{P}^{N,C} \) as:

\[
\overline{P}^{N,C} = \overline{P}^N | B^*_1(c^*_1) | \ldots | B^*_k(c^*_k)
\]

where \( B_i \) is the basic process corresponding to the channel \( c_i \) in \( P_S \).
Let $P$ be a protocol. We denote by $k_P$ the maximal size of a disjonction occurring in $P$.

**Theorem**

Let $P$ and $Q$ be two action-deterministic protocols. If there an attack against $P_{\text{Header}} \mid P \approx P_{\text{Header}} \mid Q$, then there is an attack against $P_{\text{Header}} \mid \overline{P}^{N,C}_{\text{Header}} \approx Q_{\text{Header}} \mid \overline{P}^{N,C}_{\text{Header}}$ for some multiset $C$ built on $\{c_H, c_D\}$ of size less than $k_P + k_Q$. 
Result

Let $P$ be a protocol. We denote by $k_P$ the maximal size of a disjonction occurring in $P$.

**Theorem**

Let $P$ and $Q$ be two action-deterministic protocols. If there is an attack against $P_{\text{Header}^1} P \approx P_{\text{Header}^2} Q$, then there is an attack against $P_{\text{Header}} \overline{P}_{\text{Header}}^{N,C} \approx Q_{\text{Header}} \overline{P}_{\text{Header}}^{N,C}$ for some multiset $C$ built on $\{c_H, c_D\}$ of size less than $k_P + k_Q$. 
How to build a counter-example?

- Take a protocol such that there is an attack iff an instance of PCP has a solution.
- The adversary is allowed to add a tile iff he gives a nonce (or an agent name).
- These names are stocked in a list.
- Find a way to do that the attack is possible iff the elements of the lists are pairwise distinct.
How to build a counter-example?

- Take a protocol such that there is an attack iff an instance of PCP has a solution.
- The adversary is allowed to add a tile iff he gives a nonce (or an agent name).
- These names are stocked in a list.
- Find a way to do that the attack is possible iff the elements of the lists are pairwise distinct.
How to build a counter-example?

- Take a protocol such that there is an attack iff an instance of PCP has a solution.
- The adversary is allowed to add a tile iff he gives a nonce (or an agent name).
- These names are stocked in a list.
- Find a way to do that the attack is possible iff the elements of the lists are pairwise distinct.
How to build a counter-example?

- Take a protocol such that there is an attack iff an instance of PCP has a solution.
- The adversary is allowed to add a tile iff he gives a nonce (or an agent name).
- These names are stocked in a list.
- Find a way to do that the attack is possible iff the elements of the lists are pairwise distinct.
Forbid the agents to speak to themselves.

**Initialisation**

We assume that the protocol is never played with $A = B$.

\[
\begin{align*}
B &\rightarrow S : A, B \\
S &\rightarrow B : \text{enc}(< A, B >, K_{\text{diff}}), \\
&\quad \text{enc}(<< u_{\text{init}}, v_{\text{init}} >, << B, end >>, K_{PCP}) \\
B &\rightarrow S : \text{enc}(<< x, y >, z_\ell >, K_{PCP})
\end{align*}
\]

For a honest execution, the only possibility would be for $B$ to forward the second message, but it is not checked by the server.
Forbid the agents to speak to themselves.

Initialisation

We assume that the protocol is never played with $A = B$.

$$B \rightarrow S : A, B$$
$$S \rightarrow B : \text{enc}(\langle A, B \rangle, K_{\text{diff}}),$$
$$\text{enc}(\langle \langle u_{\text{init}}, v_{\text{init}} \rangle, \langle B, \text{end} \rangle \rangle, K_{\text{PCP}})$$
$$B \rightarrow S : \text{enc}(\langle \langle x, y \rangle, z_\ell \rangle, K_{\text{PCP}})$$

For a honest execution, the only possibility would be for $B$ to forward the second message, but it is not checked by the server.
Forbid the agents to speak to themselves.

**Initialisation**

We assume that the protocol is never played with $A = B$.

\[ B \rightarrow S : A, B \]
\[ S \rightarrow B : \text{enc}(< A, B >, K_{\text{diff}}), \]
\[ \text{enc}(<< u_{\text{init}}, v_{\text{init}} >>, < B, \text{end} >>, K_{\text{PCP}}) \]
\[ B \rightarrow S : \text{enc}(<< x, y >>, z_{\ell} >, K_{\text{PCP}}) \]

For a honest execution, the only possibility would be for $B$ to forward the second message, but it is not checked by the server.
Forbid the agents to speak to themselves.

Add a tile.

Recall the last step:

\[ B \rightarrow S : \text{enc}(\langle\langle x, y \rangle, z_\ell \rangle, K_{PCP}) \]
\[ S \rightarrow A : \text{enc}(\langle\langle xu_1, yv_1 \rangle, \langle A, z_\ell \rangle\rangle, K_{PCP}), \]
\[ \ldots, \]
\[ \text{enc}(\langle\langle xu_n, yv_n \rangle, \langle A, z_\ell \rangle\rangle, K_{PCP}) \]
Forbid the agents to speak to themselves.
Get the encrypted secret.

\[ S \rightarrow A : \]
\[
\text{enc}(\text{enc}(\text{secret}, \text{enc}(< A, z_\ell >, K_{\text{approved}}))), \\
\text{enc}(<< xu_1, xu_1 >, < A, z_\ell >>, K_{\text{PCP}})), \\
\ldots, \\
\text{enc}(\text{enc}(\text{secret}, \text{enc}(< A, z_\ell >, K_{\text{approved}}))), \\
\text{enc}(<< xu_n, xu_n >, < A, z_\ell >>, K_{\text{PCP}}))
\]
Forbid the agents to speak to themselves.
Check the list

? → S : < A, end >
S → ? : enc(< A, end >, K_{approved})
? → S : enc(< x', z'_\ell >, K_{approved}),
enc(< y', z'_\ell >, K_{approved}), enc(< x', y' >, K_{diff})
S → ? : enc(< x', < y', z'_\ell >>, K_{approved})
Forbid the agents to speak to themselves.

Sketch of proof

- We have to get \textit{SECRET}.
- It is encrypted twice: \( \text{enc} (\text{enc}(\text{SECRET}, K_2), K_1) \).
- We can obtain the first key iff we have a solution of PCP:
  \( K_1 = \text{enc}(<< x, x >, z_\ell >, K_{PCP}) \).
- We can obtain the second key iff we have the corresponding list of identities encrypted by \( K_{\text{approved}} \):
  \( K_2 = \text{enc}(z_\ell, K_{\text{approved}}) \).
- This can be done iff the elements of the list are pairwise distinct.
- The length of the list is not always calculable.
Forbid the agents to speak to themselves.

Sketch of proof

- We have to get \textit{SECRET}.
- It is encrypted twice: \texttt{enc(\texttt{enc(SECRET, K_2), K_1})}.
- We can obtain the first key iff we have a solution of PCP: \[ K_1 = \text{enc}(<< x, x >, z_\ell >, K_{PCP}) \].
- We can obtain the second key iff we have the corresponding list of identities encrypted by \[ K_{approved} : K_2 = \text{enc}(z_\ell, K_{approved}) \].
- This can be done iff the elements of the list are pairwise distinct.
- The length of the list is not always calculable.
Forbid the agents to speak to themselves.

Sketch of proof

- We have to get $SECRET$.

- It is encrypted twice: $\text{enc}(\text{enc}(SECRET, K_2), K_1)$.

- We can obtain the first key iff we have a solution of PCP: $K_1 = \text{enc}(<< x, x >, z_\ell >, K_{PCP})$.

- We can obtain the second key iff we have the corresponding list of identities encrypted by $K_{approved}$: $K_2 = \text{enc}(z_\ell, K_{approved})$.

- This can be done iff the elements of the list are pairwise distinct.

- The length of the list is not always calculable.
Forbid the agents to speak to themselves.

Sketch of proof

- We have to get \textit{SECRET}.
- It is encrypted twice: \(\text{enc} (\text{enc} (\textit{SECRET}, K_2), K_1)\).
- We can obtain the first key iff we have a solution of PCP: \(K_1 = \text{enc}(<<x, x>, z_\ell>, K_{PCP})\).
- We can obtain the second key iff we have the corresponding list of identities encrypted by \(K_{approved}: K_2 = \text{enc}(z_\ell, K_{approved})\).
- This can be done iff the elements of the list are pairwise distinct.
- The length of the list is not always calculable.
Forbid the agents to speak to themselves.

Sketch of proof

- We have to get $SECRET$. 
- It is encrypted twice : $\text{enc(\text{enc}(SECRET, K_2), K_1)}$. 
- We can obtain the first key iff we have a solution of PCP : $K_1 = \text{enc}(<<x, x>, z_\ell>, K_{PCP})$. 
- We can obtain the second key iff we have the corresponding list of identities encrypted by $K_{\text{approved}} : K_2 = \text{enc}(z_\ell, K_{\text{approved}})$. 
- This can be done iff the elements of the list are pairwise distinct. 
- The length of the list is not always calculable.
Forbid the agents to speak to themselves.

Sketch of proof

- We have to get \textit{SECRET}.
- It is encrypted twice: \(\text{enc(\text{enc(SECRET, K}_2, K_1)})\).
- We can obtain the first key iff we have a solution of PCP: \(K_1 = \text{enc(<< x, x >>, z}_\ell, K_{PCP})\).
- We can obtain the second key iff we have the corresponding list of identities encrypted by \(K_{approved}\): \(K_2 = \text{enc(z}_\ell, K_{approved})\).
- This can be done iff the elements of the list are pairwise distinct.
- The length of the list is not always calculable.
Other negative results.

- **A fortiori**: else branches.
- Model with XOR and pair (or only dec, enc with explicit destructors).
- Remove the action-determinism.
Other negative results.

- A fortiori: else branches.
- Model with XOR and pair (or only dec, enc with explicit destructors).
- Remove the action-determinism.
Other negative results.

- A fortiori: else branches.
- Model with XOR and pair (or only dec, enc with explicit destructors).
- Remove the action-determinism.
Model with XOR and pair

- There are two trees that are equals whenever the elements of a list are not pairwise distinct.
- These trees can be built inductively from the list.
- So we can get almost the same counter-example.
There are two trees that are equals whenever the elements of a list are not pairwise distinct.

These trees can be built inductively from the list.

So we can get almost the same counter-example.
There are two trees that are equals whenever the elements of a list are not pairwise distinct.

These trees can be built inductively from the list.

So we can get almost the same counter-example.
- It is also impossible to remove this hypothesis.
- Problem: A lot of different frames
- It is impossible to bound their number, even between equivalent frames.
Remove the action-determinism

- It is also impossible to remove this hypothesis.
- Problem: A lot of different frames
- It is impossible to bound their number, even between equivalent frames.
Remove the action-determinism

- It is also impossible to remove this hypothesis.
- Problem: A lot of different frames
- It is impossible to bound their number, even between equivalent frames.
Conclusion

- Rémy’s theorem give a solution for simple protocols without else branch.
- We can add error branches, tests of disjunctions of equalities, and exchange simple for action-deterministic.
- They are counter-examples for some extensions: else branches, determinacy and model with XOR and pair.
Rémy’s theorem give a solution for simple protocols without else branch.

We can add error branches, tests of disjunctions of equalities, and exchange simple for action-deterministic.

They are counter-examples for some extensions: else branches, determinacy and model with XOR and pair.
Rémy’s theorem gives a solution for simple protocols without else branch.

We can add error branches, tests of disjunctions of equalities, and exchange simple for action-deterministic.

They are counter-examples for some extensions: else branches, determinacy and model with XOR and pair.