Decidability of trace equivalence for protocols with nonces

Rémy Chrétien[1], Véronique Cortier[2] and Stéphanie Delaune[1]

[1]LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

[2]LORIA - CNRS

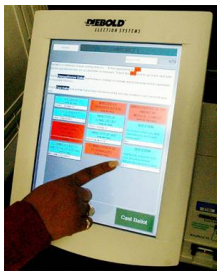
25th February 2015

Cryptographic protocols everywhere

PayPal™

Cryptographic protocols

- small programs designed to secure communication (*e.g.* secrecy)
- use cryptographic primitives (*e.g.* encryption, signature, ...)



Protocols and Security

A difficult design:

- Needham-Shroeder protocol (1978), correction and attack by Lowe (1995): an attacker could pretend to be an honest agent.
- Google Single-Sign-On protocol (2008): an attacker can log in to the Google services of a user.
- French e-passport (2010): an attacker can trace a particular user.

Motivation

- Many security properties are equivalence properties: strong secrecy, anonymity, unlinkability...
- Trace equivalence is undecidable in general (and for large subclasses: one variable and choice is enough).

Where does undecidability come from?

Undecidability encodings rely on two key aspects:

- the ability for the protocol to securely forward messages (with honest encryption)
- the ability for the protocol to loop, *i.e.* re-use messages from the end of a session into a new one.

We need to restrict these properties while keeping our class practical...

Approach

How to find a decidable class of protocols?

1. CONCUR'14 to focus on well-typed attacks only,
2. dependancy graphs to isolate potential causal dependancies between actions in a well-typed execution,
3. prove each well-typed execution is compatible with the dependancy graph,
4. consider protocols with acyclic dependancy graph to bound the length of attack traces.

The result

Decidability of trace equivalence

Let P and Q be two simple protocols type-compliant w.r.t. some structure-preserving typing systems $(\mathcal{T}_P, \delta_P)$ and $(\mathcal{T}_Q, \delta_Q)$, and with acyclic dependency graphs. The problem of deciding whether P and Q are in trace equivalence (i.e. $P \approx Q$) is decidable.

The result

Decidability of trace equivalence

Let P and Q be two simple protocols type-compliant w.r.t. some structure-preserving typing systems $(\mathcal{T}_P, \delta_P)$ and $(\mathcal{T}_Q, \delta_Q)$, and with acyclic dependency graphs. The problem of deciding whether P and Q are in trace equivalence (i.e. $P \approx Q$) is decidable.

Some intuition:

- simple : protocols with explicit execution flow,
- type-compliant w.r.t. structure-preserving typing sytems : tagged protocols,
- acyclic dependancy graph : no loop for the attacker to abuse.

The protocols

- Our primitives: pairs and symmetric encryption.
- We only allow encryption with atomic keys.
- Our grammar:

$$P, Q := \quad 0 \mid \alpha : \mathsf{in}(c, u).P \mid \alpha : \mathsf{out}(c, u).P \mid (P \mid Q) \mid !P \mid \\ \mathsf{new}\ n.P \mid \mathsf{new}\ c'.\mathsf{out}(c, c').P$$

Semantics

$$(\alpha : \mathsf{in}(c, u).P \cup \mathcal{P}; \phi) \xrightarrow{\mathsf{in}(c,R)} (P\sigma \cup \mathcal{P}; \phi) \quad \text{where } R \text{ is a recipe such that } R\phi\downarrow$$
$$\text{is a message and } R\phi\downarrow = u\sigma \text{ for some } \sigma \text{ with } dom(\sigma) = vars(u)$$

Semantics

$$(\alpha : \mathsf{out}(c, u).P \cup \mathcal{P}; \phi) \xrightarrow{\mathsf{out}(c, \mathsf{w}_{i+1})} (P \cup \mathcal{P}; \phi \cup \{\mathsf{w}_{i+1} \triangleright u\})$$

where u is a message and i is the number of elements in ϕ

Semantics

$$(\text{new } c'.\text{out}(c, c').P \cup \mathcal{P}; \phi) \xrightarrow{\text{out}(c, ch_i)} (P\{^{ch_i}/_{c'}\} \cup \mathcal{P}; \phi)$$

where ch_i is the "next" fresh channel name available in $\mathcal{C}h^{\text{fresh}}$

Semantics

$$(\text{new } n.P \cup \mathcal{P}; \phi) \xrightarrow{\tau} (P\{^{n'}/_{n}\} \cup \mathcal{P}; \phi) \quad \text{where } n' \text{ is a fresh name in } \mathcal{N}$$

Semantics

$$(!P \cup \mathcal{P}; \phi) \xrightarrow{\tau} (P \cup !P \cup \mathcal{P}; \phi)$$

Equivalences

Static equivalence

ϕ_1 and ϕ_2 are statically equivalent, $\phi_1 \sim \phi_2$, when $dom(\phi_1) = dom(\phi_2)$ and:

- for any recipe R, $R\phi_1$ is a message iff $R\phi_2\downarrow$ is a message;
- for all recipes R_1 and R_2 such that $R_1\phi_1\downarrow, R_2\phi_1\downarrow$ are messages, we have that $R_1\phi_1\downarrow = R_2\phi_1\downarrow$ iff $R_1\phi_2\downarrow = R_2\phi_2\downarrow$.

Equivalences

Static equivalence

ϕ_1 and ϕ_2 are statically equivalent, $\phi_1 \sim \phi_2$, when $dom(\phi_1) = dom(\phi_2)$ and:

- for any recipe R, $R\phi_1$ is a message iff $R\phi_2\downarrow$ is a message;
- for all recipes R_1 and R_2 such that $R_1\phi_1\downarrow, R_2\phi_1\downarrow$ are messages, we have that $R_1\phi_1\downarrow = R_2\phi_1\downarrow$ iff $R_1\phi_2\downarrow = R_2\phi_2\downarrow$.

Trace equivalence

A protocol P is trace included in a protocol Q, written $P \sqsubseteq Q$, if for every $(\mathrm{tr}, \phi) \in \mathrm{trace}(P)$, there exists $(\mathrm{tr}', \phi') \in \mathrm{trace}(Q)$ such that $\mathrm{tr} = \mathrm{tr}'$ and $\phi \sim \phi'$.
The protocols P and Q are trace equivalent, written $P \approx Q$, if $P \sqsubseteq Q$ and $Q \sqsubseteq P$.

Simple protocols

A simple protocol P is a protocol of the form

$$!\text{new } c'_1.\text{out}(c_1, c'_1).B_1 \mid \ldots \mid !\text{new } c'_m.\text{out}(c_m, c'_m).B_m \mid B_{m+1} \mid \ldots \mid B_{m+n}$$

where each B_i is a ground process on channel c'_i (resp. c_i) built using the following grammar:

$$B := 0 \mid \alpha : \text{in}(c'_i, u).B \mid \alpha : \text{out}(c'_i, u).B \mid \text{new } n.B$$

Moreover, we assume that $c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}$ are pairwise distinct.

Structure-preserving typing systems

Typing system

A structure-preserving typing system is a pair $(\mathcal{T}_0, \delta_0)$ where $\mathcal{T}_0$ is a set of elements called *atomic types*, and δ_0 is a function mapping atomic terms in $\Sigma_0 \cup \mathcal{N} \cup \mathcal{X}$ to types τ generated using the following grammar:

$$\tau, \tau_1, \tau_2 = \tau_0 \mid \langle \tau_1, \tau_2 \rangle \mid \mathsf{enc}(\tau_1, \tau_2) \text{ with } \tau_0 \in \mathcal{T}_0.$$

Then, δ_0 is extended to constructor terms as follows:

$$\delta_0(\mathsf{f}(t_1, \ldots, t_n)) = \mathsf{f}(\delta_0(t_1), \ldots, \delta_0(t_n)) \text{ with } \mathsf{f} \in \Sigma_c.$$

Type-compliant protocols

Type-compliant protocols, ex: tagged protocols

Type-compliant protocols

P is type-compliant w.r.t. $(\mathcal{T}, \delta)$ if for every $t, t' \in ESt(\text{unfold}^2(P))$,

$$t \text{ and } t \text{ unifiable} \Rightarrow \delta(t) = \delta(t')$$

Example

Denning-Sacco protocol

1. $A \to S : A, B$
2. $S \to A : \{B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \to B : \{K_{ab}, A\}_{K_{bs}}$

The formal specification

$$P = \; !\,\text{new } c_1.\text{out}(c_A, c_1).P_A \mid !\,\text{new } c_2.\text{out}(c_B, c_2).P_B \mid !\,\text{new } c_3.\text{out}(c_S, c_3).P_S$$

$$\begin{aligned}
P_A &= \alpha_1 : \text{out}(c_1, \langle \text{a}, \text{b} \rangle). \\
&\quad\; \alpha_2 : \text{in}(c_1, \text{enc}(\langle \text{b}, x_{AB}, x_B \rangle, k_{as})). \\
&\quad\; \alpha_3 : \text{out}(c_1, x_B) \\
P_B &= \beta_1 : \text{in}(c_2, \text{enc}(\langle y_{AB}, \text{a} \rangle, k_{bs})). \\
&\quad\; \beta_2 : \text{out}(c_2, \text{enc}(m_1, y_{AB})) \\
P_S &= \gamma_1 : \text{in}(c_3, \langle \text{a}, \text{b} \rangle).\ \text{new } k_{ab}. \\
&\quad\; \gamma_2 : \text{out}(c_3, \text{enc}(\langle \text{b}, k_{ab}, \text{enc}(\langle k_{ab}, \text{a} \rangle, k_{bs}) \rangle, k_{as}))
\end{aligned}$$

Example

$$
\begin{array}{rcl}
P_A & = & \alpha_1 : \mathsf{out}(c_1, \langle \mathsf{a}, \mathsf{b} \rangle). \\
 & & \alpha_2 : \mathsf{in}(c_1, \mathsf{enc}(\langle \mathsf{b}, x_{AB}, x_B \rangle, k_{as})). \\
 & & \alpha_3 : \mathsf{out}(c_1, x_B) \\
P_B & = & \beta_1 : \mathsf{in}(c_2, \mathsf{enc}(\langle y_{AB}, \mathsf{a} \rangle, k_{bs})). \\
 & & \beta_2 : \mathsf{out}(c_2, \mathsf{enc}(m_1, y_{AB})) \\
P_S & = & \gamma_1 : \mathsf{in}(c_3, \langle \mathsf{a}, \mathsf{b} \rangle). \text{ new } k_{ab}. \\
 & & \gamma_2 : \mathsf{out}(c_3, \mathsf{enc}(\langle \mathsf{b}, k_{ab}, \mathsf{enc}(\langle k_{ab}, \mathsf{a} \rangle, k_{bs}) \rangle, k_{as}))
\end{array}
$$

the typing function δ

$$
\begin{array}{lll}
\delta(a) = \tau_a & \delta(b) = \tau_b & \delta(m_1) = \tau_m \\
\delta(k_{AB}) = \tau_{kab} & \delta(k_{AS}) = \tau_{kas} & \delta(k_{BS}) = \tau_{kbs} \\
\delta(x_{AB}) = \tau_{kab} & \delta(y_{AB}) = \tau_{kab} &
\end{array}
$$

$$
\delta(x_B) = \mathsf{enc}(\langle \tau_{kab}, \tau_a \rangle, \tau_{kbs})
$$

Dependancy graph

What is the dependancy graph of P made of?

- Vertices : the labels of P
- Edges : of 3 kinds
  1. sequential dependancy: if two actions follow each other in $P\delta$,
  2. data dependancy: if a deducible subterm of an input in $P\delta$ appears as a deducible subterm of an output in $P\delta$,
  3. key dependancy: if a key in an output in $P\delta$ can be deduced with the aid of another key, deducible in another ouput in $P\delta$.

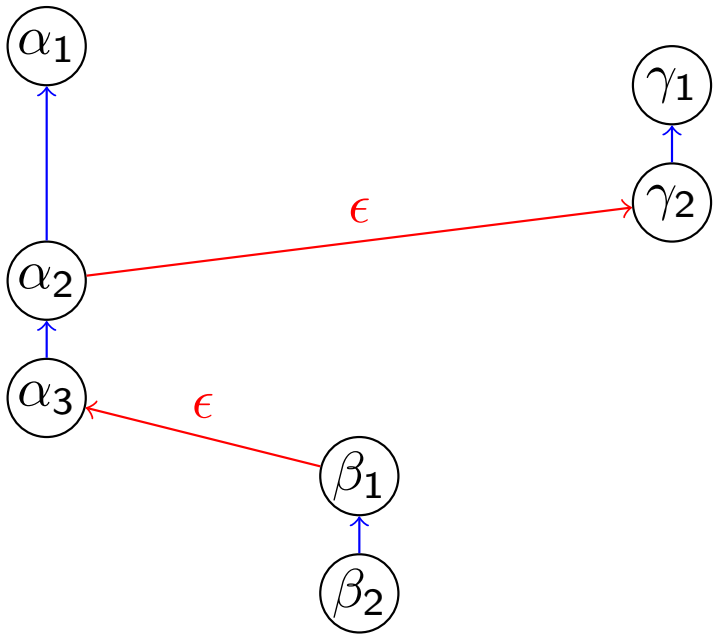
Dependancy graph

$P_A\delta$, $P_B\delta$ and $P_S\delta$

$\alpha_1 : \mathsf{out}(c_1, \langle \tau_a, \tau_b \rangle).$
$\alpha_2 : \mathsf{in}(c_1, \mathsf{enc}(\langle \tau_b, \tau_{kab},$
$\mathsf{enc}(\langle \tau_{kab}, \tau_a \rangle, \tau_{kbs}) \rangle, \tau_{kas})).$
$\alpha_3 : \mathsf{out}(c_1, \mathsf{enc}(\langle \tau_{kab}, \tau_a \rangle, \tau_{kbs}))$

$\beta_1 : \mathsf{in}(c_2, \mathsf{enc}(\langle \tau_{kab}, \tau_a \rangle, \tau_{kbs}))$

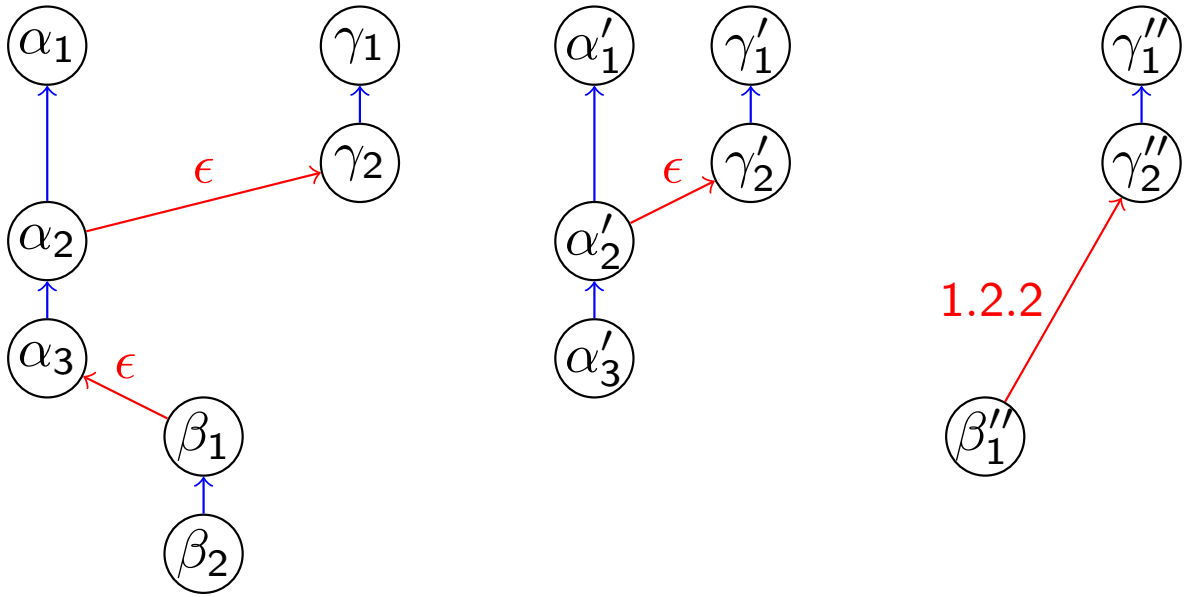
$\gamma_1 : \mathsf{in}(c_3, \langle \tau_a, \tau_b \rangle).\ \mathsf{new}\ \tau_{kab}.$
$\gamma_2 : \mathsf{out}(c_3, \mathsf{enc}(\langle \tau_b, \tau_{kab},$
$\mathsf{enc}(\langle \tau_{kab}, \tau_a \rangle, \tau_{kbs}) \rangle, \tau_{kas}))$



τ_a, τ_b are public types; τ_{kbs} and τ_{kas} are honest types.

A more complex scenario

What happens when A and S communicate with C dishonest (α'_i and γ'_i) and when B and S talk with C (β''_i and γ''_i)?



$$
\begin{array}{rcl}
P''_B & = & \beta''_1 : \mathsf{in}(c_2, \mathsf{enc}(\langle y_{CB}, c\rangle, k_{bs})).\\
P''_S & = & \gamma_1 : \mathsf{in}(c_3, \langle c, \mathsf{b}\rangle).\ \mathsf{new}\ k_{cb}.\\
 & & \gamma''_2 : \mathsf{out}(c_3, \mathsf{enc}(\langle \mathsf{b}, k_{cb}, \mathsf{enc}(\langle k_{cb}, c\rangle, k_{bs})\rangle, k_{cs}))
\end{array}
$$

The result

Decidability of trace equivalence

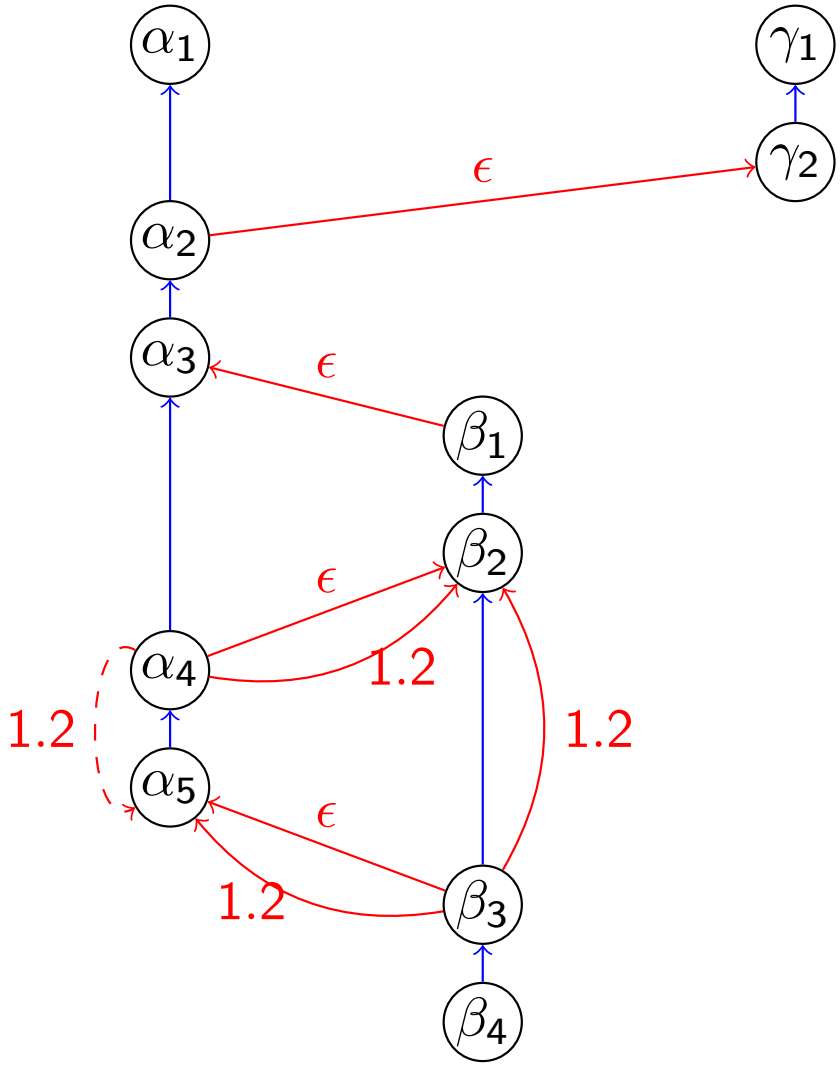
Let P and Q be two simple protocols type-compliant w.r.t. some structure-preserving typing systems $(\mathcal{T}_P, \delta_P)$ and $(\mathcal{T}_Q, \delta_Q)$, and with acyclic dependency graphs. The problem of deciding whether P and Q are in trace equivalence (i.e. $P \approx Q$) is decidable.

Denning-Sacco and Wide-Mouthed Frog fall into this class.

Refined dependancy graph

Needham-Schroeder protocol

1. $A \to S : \ A, B, N_a$
2. $S \to A : \ \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \to B : \ \{K_{ab}, A\}_{K_{bs}}$
4. $B \to A : \ \{\mathsf{req}, N_b\}_{K_{ab}}$
5. $A \to B : \ \{\mathsf{rep}, N_b\}_{K_{ab}}$



Refined dependancy graph

How to deal with these acyclic graphs?

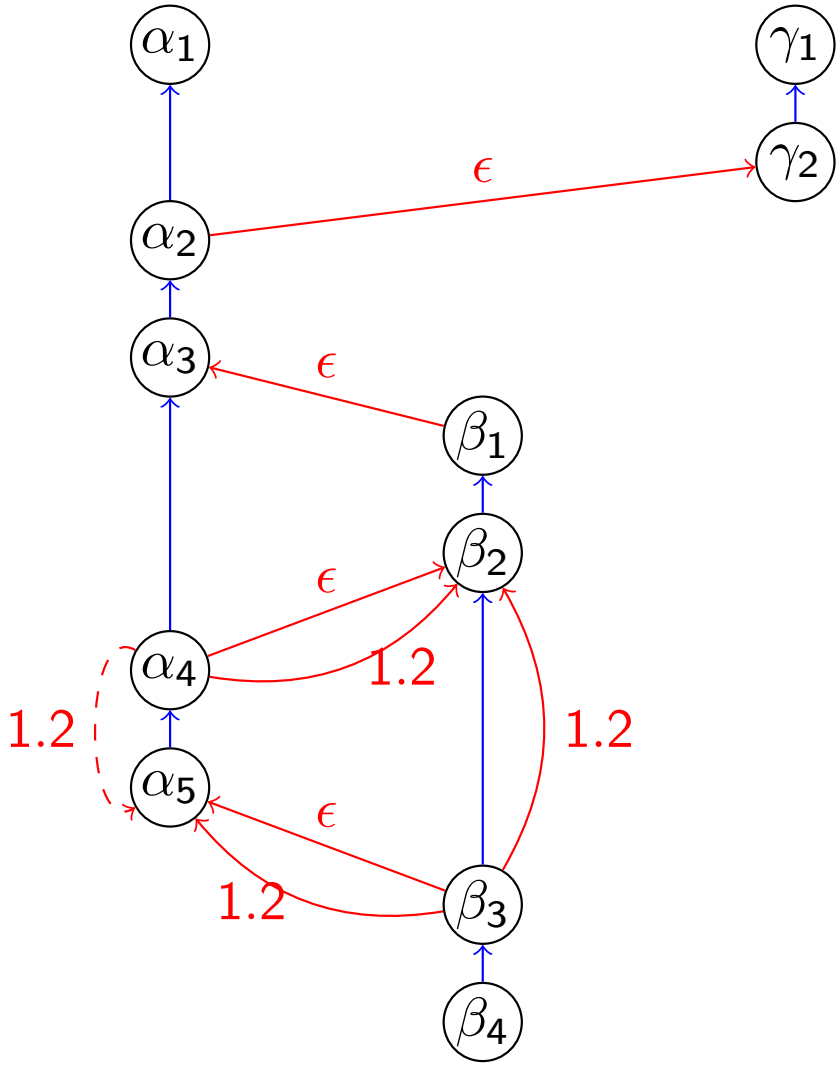
- introduce a (semantic) notion of marking to pinpoint terms which are useless to the attacker,
- propose a (syntactic) criterion to generate such markings in practice,
- and use a refined notion of the dependancy graph.

Refined dependancy graph

Needham-Schroeder protocol

1. $A \to S : \ A, B, N_a$
2. $S \to A : \ \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \to B : \ \{K_{ab}, A\}_{K_{bs}}$
4. $B \to A : \ \{\mathsf{req}, N_b\}_{K_{ab}}$
5. $A \to B : \ \{\mathsf{rep}, N_b\}_{K_{ab}}$

$\to$ we can pinpoint position 1.2 in α_5.



The (refined) result

Decidability of trace equivalence

Let P and Q be two simple protocols type-compliant w.r.t. some structure-preserving typing systems $(\mathcal{T}_P, \delta_P)$ and $(\mathcal{T}_Q, \delta_Q)$, and with acyclic refined dependency graphs. The problem of deciding whether P and Q are in trace equivalence (i.e. $P \approx Q$) is decidable.

	Dependency graph		In our class
	Normal	Refined	
Denning-Sacco	✓	✓	yes
Needham-Schroeder		✓	yes
Otway-Rees		✓	yes
Yahalom (Paulson)		✓	yes
Wide-Mouthed-Frog	✓	✓	yes
Yahalom			no
Kao-Chow (modified)		✓	yes

Figure : A ✓ means that the corresponding dependency graph is acyclic.

Proof outline

Our proof can be summarised as follows:

1. We first rely on our type-compliance assumption. We show that we can restrict our attention to witnesses that are well-typed and we further show that each message occurring in such a trace can be computed as soon as possible.
2. Then, we show that all the dependencies occurring in such a well-typed and asap trace comply with the dependency graph. Hence, we bound the *width* as well as the *depth* of such a witness exploiting the acyclicity of our dependency graph.
3. Lastly, we explain how to bound the length of a minimal witness:

$$2(1 + \|\mathsf{out}_P\|)^{\mathsf{depth}(G_P)+1}(1 + \|\mathsf{in}_P\|(1 + \|\mathsf{out}_P\|)^{\mathsf{depth}(G_P)+1})^{\mathsf{depth}(G_P)+1}.$$

Conclusion

We obtained:

- a decidability result for equivalence of simple acyclic type-compliant protocols,
- which extends existing results for reachability;
- along with syntactic/semantic criterion to easily obtain acyclic protocols,
- and most of the studied examples fall into this class.

Future work

What remains to be done:

- extend our signature to asymmetric cryptography,
- relax the hypothesis of simple protocols (to action-determinate)
- implement a tool to automatically compute et verify the acyclicity of any protocol.

Thank you for your attention.