Equivalence Properties by Typing in Cryptographic Branching Protocols

Joseph Lallemand (Loria)

joint work with Véronique Cortier, Niklas Grimm, Matteo Maffei

presented at CCS'17, POST'18

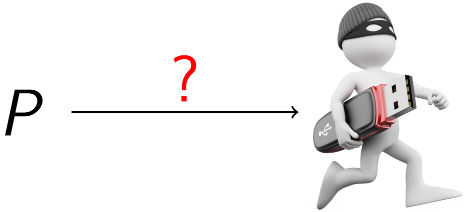
March 14, 2018

Trace properties

Trace properties = satisfied by all traces of a protocol

Example: reachability properties:

Can the attacker learn a given message ?

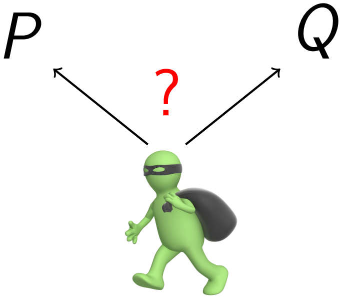


$\Longrightarrow$ secrecy, authentication, . . .

Equivalence

Some properties require the notion of equivalence:

Are two protocols indistinguishable for an attacker?

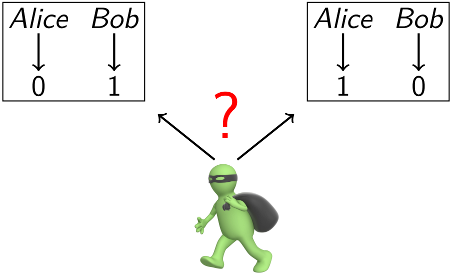


Example:

vote privacy, strong flavours of secrecy, anonymity, unlinkability, . . .

Example: vote privacy

Example: Privacy of the vote in voting protocols



Alice and Bob vote for either 0 or 1.

The values of the votes = 0 and 1 are not secret

The votes are secret if:

$$Alice(0) \mid Bob(1) \approx Alice(1) \mid Bob(0)$$

Type systems

Idea: design a type system that ensures protocols satisfy security properties

- Type systems: already applied to trace properties

$$M : \mathit{Secret} \vdash P \implies M \text{ is not deducible in } P$$

- Now: for equivalence

$$\vdash P \sim Q \quad \implies \quad P \approx Q$$

- Efficient (though incomplete) procedures
- Modularity

Type systems

Idea: design a type system that ensures protocols satisfy security properties

- Type systems: already applied to trace properties

$$M : Secret \vdash P \implies M \text{ is not deducible in } P$$

- Now: for equivalence

$$\vdash P \sim Q \quad \implies \quad P \approx Q$$

- Efficient (though incomplete) procedures
- Modularity

Problem:

- Usually: typing $\rightarrow$ overapproximate the set of traces.
- Sound for trace properties, but not equivalence
  $\rightarrow$ might miss that some traces are only possible for P and not Q

Main idea

- Step 1: $\vdash P \sim Q : C$
  typing to ensure no leaks in behaviours
  collect all symbolic messages sent on the network into a *constraint*

- Step 2: $check(C)$
  ensure there are no leaks in the messages sent
  $\longrightarrow$ checking for repetitions

  Example:

  $$C = \{\mathtt{enc}(x, k) \sim \mathtt{enc}(a, k),\ \mathtt{enc}(y, k) \sim \mathtt{enc}(b, k)\}$$

  If in some execution we can have $x = y$, equivalence is broken.

Main result: Soundness

Theorem (Soundness)

If $\Gamma \vdash P \sim Q : C$ *and* $\forall\theta.\ C\theta$ *does not leak information, then*

$$P \approx Q$$

Theorem (Procedure to check constraints)

$$check(C) \Rightarrow \forall\theta.\ C\theta \textit{ does not leak information.}$$

Hypotheses:

- atomic keys only
- fixed cryptographic primitives: symmetric and asymmetric encryption, signature, hash, pairing
- no replication (bounded number of sessions only)

Main result: Soundness

Theorem (Soundness)

If $\Gamma \vdash P \sim Q : C$ *and* $\forall \theta.\ C\theta$ *does not leak information, then*

$$P \approx Q$$

Theorem (Procedure to check constraints)

$$check(C) \Rightarrow \forall \theta.\ C\theta \text{ does not leak information.}$$

Hypotheses:

- atomic keys only
- fixed cryptographic primitives: symmetric and asymmetric encryption, signature, hash, pairing
- no replication (bounded number of sessions only)

From two to unbounded number of sessions

If one session typechecks, then any number of sessions typecheck:

Theorem (informal)

$$\Gamma \vdash P \sim Q : C \implies \Gamma \vdash !P \sim !Q : !C$$

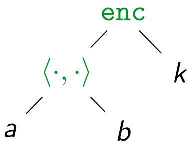
How to check that $!C$ does not leak information?
$\longrightarrow$ It is sufficient to check two copies of C:

Theorem (informal)

$$\mathit{check}(C \cup C') \implies \mathit{check}(!C)$$

Symbolic model

- Messages are terms
  constructed using abstract cryptographic primitives,



- Symbolic attacker
  with abilities defined by deduction rules

$$\frac{\mathtt{enc}(x, y) \qquad y}{x} \qquad\qquad \frac{x \qquad y}{\langle x, y \rangle}$$

Symbolic model

Process algebra similar to the applied pi-calculus

$$
\begin{array}{rcl}
P, Q & ::= & \\
 & & 0 \\
 & \mid & \mathtt{new}\ n.P \\
 & \mid & \mathtt{out}(M).P \\
 & \mid & \mathtt{in}(x).P \\
 & \mid & P \mid Q \\
 & \mid & \mathtt{let}\ x = d(y)\ \mathtt{in}\ P\ \mathtt{else}\ Q \\
 & \mid & \mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q \\
 & \mid & !P
\end{array}
$$

Static equivalence

Frames are sequences of messages modelling the attacker's knowledge

$$\phi = \{x_1 \mapsto k,\ x_2 \mapsto a,\ x_3 \mapsto \mathtt{enc}(b, k)\}$$

Static equivalence = indistinguishability of frames

$$\phi \approx \phi' \quad \iff \quad \forall R, S.\ R\phi = S\phi \Leftrightarrow R\phi' = S\phi'$$

Example:

$$\{\mathtt{enc}(a, k)\} \quad \approx \quad \{\mathtt{enc}(b, k)\}$$

but

$$\{\mathtt{enc}(a, k), \mathtt{enc}(a, k)\} \quad \not\approx \quad \{\mathtt{enc}(a, k), \mathtt{enc}(b, k)\}$$

and

$$\{k, \mathtt{enc}(a, k)\} \quad \not\approx \quad \{k, \mathtt{enc}(b, k)\}$$

Trace equivalence

A trace (tr, ϕ) is a sequence of observable actions
$+$ a frame of messages sent on the network

Definition (Trace equivalence)

P and Q are trace equivalent if any trace of P can be mimicked by a trace of Q (and conversely)

i.e.

$$\forall (tr, \phi) \in \mathrm{trace}(P).\ \exists (tr, \phi') \in \mathrm{trace}(Q).\ \phi \approx \phi'$$

and

$$\forall (tr, \phi) \in \mathrm{trace}(Q).\ \exists (tr, \phi') \in \mathrm{trace}(P).\ \phi \approx \phi'$$

Typing messages

Types for messages :

$$
\begin{array}{lcl}
l & ::= & \mathtt{LL} \mid \mathtt{HL} \mid \mathtt{HH} \\
T & ::= & l \\
 & \mid & \text{key}^{l}(T) \\
 & \mid & T * T \\
 & \mid & T \vee T \\
 & \mid & \dots
\end{array}
$$

- labels = levels of confidentiality and integrity
  - $\mathtt{LL}$ for public messages
  - $\mathtt{HH}$ for secret values
- key types $\text{key}^{l}(T)$

  Example:

$$\text{key}^{\mathtt{HH}}(\mathtt{LL} * \mathtt{HH})$$

Typing messages

$$\frac{\Gamma \vdash M \sim N : T \qquad \Gamma(k) = key^{\mathtt{HH}}(T)}{\Gamma \vdash \mathtt{enc}(M, k) \sim \mathtt{enc}(N, k) : \mathtt{LL}}$$

Ensure the messages sent are safe to output:
$\longrightarrow$ similar structure

$$\langle a, b \rangle \quad \not\sim \quad a$$

$$\mathtt{enc}(\langle a, b \rangle, k) \quad \sim \quad \mathtt{enc}(a, k) \quad \text{only if } k \text{ is secret}$$

Typing messages

$$\frac{\Gamma \vdash M \sim N : T \to c \qquad \Gamma(k) = key^{\mathtt{HH}}(T)}{\Gamma \vdash \mathtt{enc}(M, k) \sim \mathtt{enc}(N, k) : \mathtt{LL} \to c \cup \{\mathtt{enc}(M, k) \sim \mathtt{enc}(N, k)\}}$$

- Establish invariants regarding the types of keys
  If k is secret, the type of M, N must match the type of k
- Collect constraints
  Here we add the couple $\mathtt{enc}(M, k) \sim \mathtt{enc}(N, k)$ to the constraint

Typing processes

- All output messages must be of type $\mathtt{LL}$
- Their constraints are collected

$$\frac{\Gamma \vdash M \sim N : \mathtt{LL} \to c \qquad \Gamma \vdash P \sim Q : C}{\Gamma \vdash \mathtt{out}(M).P \sim \mathtt{out}(N).Q : C \cup c}$$

- All input messages are considered to be of type $\mathtt{LL}$:

$$\frac{\Gamma, x : \mathtt{LL} \vdash P \sim Q : C}{\Gamma \vdash \mathtt{in}(x).P \sim \mathtt{in}(x).Q : C}$$

Typing processes (2)

$\longrightarrow$ Processes have to progress the same way:

accept inputs/outputs at the same time,
follow (typably) equivalent branches

Example: applying destructors

$$\frac{\Gamma(x) = \mathtt{LL} \qquad \Gamma(k) = \mathsf{key}^{\mathtt{HH}}(T) \qquad \Gamma, y : T \vdash P \sim Q : C \qquad \Gamma \vdash P' \sim Q' : C'}{\Gamma \vdash \mathtt{let}\ y = \mathtt{dec}(x, k)\ \mathtt{in}\ P\ \mathtt{else}\ P' \sim \mathtt{let}\ y = \mathtt{dec}(x, k)\ \mathtt{in}\ Q\ \mathtt{else}\ Q' : C \cup C'}$$

Constraints

Why do we need constraints?

Constraints

Why do we need constraints?

$\longrightarrow$ Local checks on the messages are not sufficient for equivalence

Constraints

Why do we need constraints?

$\longrightarrow$ Local checks on the messages are not sufficient for equivalence

Example: If k is a secret key

$$\begin{array}{lcll} \mathtt{out}(\mathtt{enc}(a, k)) & \sim & \mathtt{out}(\mathtt{enc}(b, k)) & \text{is fine} \\ \mathtt{out}(\mathtt{enc}(a, k)) & \sim & \mathtt{out}(\mathtt{enc}(a, k)) & \text{is fine} \end{array}$$

but not both together

$$\mathtt{out}(\mathtt{enc}(a, k)) \mid \mathtt{out}(\mathtt{enc}(a, k)) \not\sim \mathtt{out}(\mathtt{enc}(b, k)) \mid \mathtt{out}(\mathtt{enc}(a, k))$$

Constraints

Why do we need constraints?

$\longrightarrow$ Local checks on the messages are not sufficient for equivalence

Example: If k is a secret key

$$\begin{array}{lcll} \mathtt{out}(\mathtt{enc}(a,k)) & \sim & \mathtt{out}(\mathtt{enc}(b,k)) & \text{is fine} \\ \mathtt{out}(\mathtt{enc}(a,k)) & \sim & \mathtt{out}(\mathtt{enc}(a,k)) & \text{is fine} \end{array}$$

but not both together

$$\mathtt{out}(\mathtt{enc}(a,k)) \mid \mathtt{out}(\mathtt{enc}(a,k)) \not\sim \mathtt{out}(\mathtt{enc}(b,k)) \mid \mathtt{out}(\mathtt{enc}(a,k))$$

$$\mathcal{C} = \{\mathtt{enc}(a,k) \sim \mathtt{enc}(b,k), \mathtt{enc}(a,k) \sim \mathtt{enc}(a,k)\}$$

Constraints

Collect symbolic messages in a constraint C while typing
and check that it is consistent

i.e. for any possible instantiation, C instantiated does not leak anything:

$$C = \{u_1 \sim v_1, \ldots, u_n \sim v_n\}$$

must satisfy

$$\forall \theta, \theta'. \quad \{u_1\theta, \ldots, u_n\theta\} \approx \{v_1\theta', \ldots, v_n\theta'\}$$

Constraints: Checking consistency

- Open messages as much as possible:

$$\begin{array}{rcll} \langle M, N \rangle & \longrightarrow & M, N & \\ \mathtt{enc}(M, k) & \longrightarrow & M & \text{if } k \text{ has type } \mathrm{key}^{\mathtt{LL}}(\cdot) \\ \ldots & & & \end{array}$$

- Check that both sides of the opened constraint satisfy the same equalities once instantiated (unification)

$$M \sim N, M' \sim N' \in C$$

$$\forall \theta, \theta'.\ M\theta = M'\theta \iff N\theta' = N'\theta'$$

- Actually only consider well-typed θ, θ'
i.e.

$$\forall x.\ \vdash \theta(x) \sim \theta'(x) : \Gamma(x)$$

The case of different keys

In the rules shown before, the keys were the same on both sides

$$\frac{\Gamma \vdash M \sim N : T \to c \qquad \Gamma(k) = \text{key}^{\text{HH}}(T)}{\Gamma \vdash \texttt{enc}(M, k) \sim \texttt{enc}(N, k) : \texttt{LL} \to c \cup \{\texttt{enc}(M, k) \sim \texttt{enc}(N, k)\}}$$

$\longrightarrow$ How to handle more complex cases where different keys are used?

Example: anonymity, unlinkability

Example: Private Authentication

$\longrightarrow$ Authenticating B to A anonymously to others

$$A \to B : \quad \mathtt{aenc}(\langle N_a, \mathtt{pk}(k_a)\rangle, \mathtt{pk}(k_b))$$

$$B \to A : \quad \begin{cases} \mathtt{aenc}(\langle N_a, \langle N_b, \mathtt{pk}(k_b)\rangle\rangle, \mathtt{pk}(k_a)) & \text{if } B \text{ accepts } A\text{'s request} \\ \mathtt{aenc}(N_b, \mathtt{pk}(k)) & \text{if } B \text{ declines } A\text{'s request} \end{cases}$$

$\mathtt{pk}(k) =$ decoy key. No one has the secret key k.

Example: Private Authentication

$\longrightarrow$ Authenticating B to A anonymously to others

$$A \rightarrow B: \quad \mathtt{aenc}(\langle N_a, \mathtt{pk}(k_a)\rangle, \mathtt{pk}(k_b))$$

$$B \rightarrow A: \quad \begin{cases} \mathtt{aenc}(\langle N_a, \langle N_b, \mathtt{pk}(k_b)\rangle\rangle, \mathtt{pk}(k_a)) & \text{if } B \text{ accepts } A\text{'s request} \\ \mathtt{aenc}(N_b, \mathtt{pk}(k)) & \text{if } B \text{ declines } A\text{'s request} \end{cases}$$

$\mathtt{pk}(k) =$ decoy key. No one has the secret key k.

Anonymity: an attacker cannot learn whether B is willing to talk to A or not

$$Alice \mid Bob(pk_{Alice}) \approx Alice \mid Bob(pk_{Charlie})$$

Example: Private Authentication

$\longrightarrow$ Authenticating B to A anonymously to others

$$A \rightarrow B : \quad \mathtt{aenc}(\langle N_a, \mathtt{pk}(k_a)\rangle, \mathtt{pk}(k_b))$$

$$B \rightarrow A : \quad \begin{cases} \mathtt{aenc}(\langle N_a, \langle N_b, \mathtt{pk}(k_b)\rangle\rangle, \mathtt{pk}(k_a)) & \text{if } B \text{ accepts } A\text{'s request} \\ \mathtt{aenc}(N_b, \mathtt{pk}(k)) & \text{if } B \text{ declines } A\text{'s request} \end{cases}$$

$\mathtt{pk}(k) =$ decoy key. No one has the secret key k.

Anonymity: an attacker cannot learn whether B is willing to talk to A or not

$$Alice \mid Bob(pk_{Alice}) \approx Alice \mid Bob(pk_{Charlie})$$

Problems: different keys and non uniform branching

Bikeys

$\longrightarrow$ We introduce bikeys: pairs of keys with a type

Example:

$$\Gamma(k_1, k_2) = \mathsf{key}^{\mathtt{HH}}(\mathtt{LL} * \mathtt{HH})$$

There may be multiple bindings for the same key :

$$\begin{array}{rcl} \Gamma(k_1, k_2) & = & \mathsf{key}^{\mathtt{HH}}(\mathtt{LL} * \mathtt{HH}) \\ \Gamma(k_1, k_3) & = & \mathsf{key}^{\mathtt{HH}}(\mathtt{HH} * \mathtt{LL}) \end{array}$$

We also add a type specifying that the keys are actually the same:

$$\Gamma(k, k) = \mathsf{eqkey}^{\mathtt{HH}}(\mathtt{HH})$$

Bikeys: encrypting

$\longrightarrow$ The rules for encrypting go as expected:
allow any pair of keys that is valid in Γ

$$\frac{\Gamma \vdash M \sim N : T \to c \qquad \Gamma(k_1, k_2) = \mathsf{key}^{\mathtt{HH}}(T)}{\Gamma \vdash \mathtt{enc}(M, k_1) \sim \mathtt{enc}(N, k_2) : \mathtt{LL} \to c \cup \{\mathtt{enc}(M, k_1) \sim \mathtt{enc}(N, k_2)\}}$$

$\longrightarrow$ Similarly for asymmetric encryption and signature

Bikeys: decrypting

Previously:

$$\frac{\Gamma(x) = \mathtt{LL} \qquad \Gamma(k) = \mathsf{key}^{\mathtt{HH}}(T) \qquad \Gamma, x : T \vdash P \sim Q : C \qquad \Gamma \vdash P' \sim Q' : C'}{\Gamma \vdash \mathtt{let}\ y = \mathtt{dec}(x, k)\ \mathtt{in}\ P\ \mathtt{else}\ P' \sim \mathtt{let}\ y = \mathtt{dec}(x, k)\ \mathtt{in}\ Q\ \mathtt{else}\ Q' : C \cup C'}$$

Bikeys: decrypting

With different keys ?

$$\frac{\Gamma(x) = \texttt{LL} \qquad \Gamma(k_1, k_2) = \mathsf{key}^{\texttt{HH}}(T) \qquad \Gamma, x : T \vdash P \sim Q : C \qquad \Gamma \vdash P' \sim Q' : C'}{\Gamma \vdash \texttt{let } y = \texttt{dec}(x, k_1) \texttt{ in } P \texttt{ else } P' \sim \texttt{let } y = \texttt{dec}(x, k_2) \texttt{ in } Q \texttt{ else } Q' : C \cup C'}$$

Bikeys: decrypting

With different keys ?

$$\frac{\Gamma(x) = \mathtt{LL} \qquad \Gamma(k_1, k_2) = \mathsf{key}^{\mathtt{HH}}(T) \qquad \Gamma, x : T \vdash P \sim Q : C \qquad \Gamma \vdash P' \sim Q' : C'}{\Gamma \vdash \mathtt{let}\ y = \mathtt{dec}(x, k_1)\ \mathtt{in}\ P\ \mathtt{else}\ P' \sim \mathtt{let}\ y = \mathtt{dec}(x, k_2)\ \mathtt{in}\ Q\ \mathtt{else}\ Q' : C \cup C'}$$

Problem: There may be several bindings for k_1 in Γ
x may be encrypted with k_1 on the left, $k_3 \neq k_2$ on the right

$\longrightarrow$ We do not know that decryption succeeds or fails equally

$=$ the processes may branch non uniformly *i.e.* follow different branches

The problem of non-uniform branching

How to handle cases where the processes follow different branches?

- when decrypting with bikeys
- conditional branching where uniform execution cannot be ensured

$\longrightarrow$ We have to take all cases into account:

$$\frac{\begin{array}{c}\Gamma(y) = \mathtt{LL} \qquad \Gamma(k_1, k_2) = \mathsf{key}^{\mathtt{HH}}(T) \\ \Gamma, x : T \vdash P \sim Q \to C \qquad \Gamma \vdash P' \sim Q' \to C' \\ (\forall T'.\forall k_3 \neq k_2.\ \Gamma(k_1, k_3) = \mathsf{key}^{\mathtt{HH}}(T') \Rightarrow \Gamma, x : T' \vdash P \sim Q' \to C_{k_3}) \\ (\forall T'.\forall k_3 \neq k_1.\ \Gamma(k_3, k_2) = \mathsf{key}^{\mathtt{HH}}(T') \Rightarrow \Gamma, x : T' \vdash P' \sim Q \to C'_{k_3})\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{let}\ x = \mathtt{dec}(y, k_1)\ \mathtt{in}\ P\ \mathtt{else}\ P' \sim \mathtt{let}\ x = \mathtt{dec}(y, k_2)\ \mathtt{in}\ Q\ \mathtt{else}\ Q' \\ \to C \cup C' \cup (\bigcup_{k_3} C_{k_3}) \cup (\bigcup_{k_3} {C'}_{k_3})\end{array}}$$

Note: the simple rule still applies when keys have type $\mathsf{eqkey}^{l}(T)$

Back to Private Authentication

$$A \to B: \quad \mathtt{aenc}(\langle N_a, \mathtt{pk}(k_a)\rangle, \mathtt{pk}(k_b))$$

$$B \to A: \quad \begin{cases} \mathtt{aenc}(\langle N_a, \langle N_b, \mathtt{pk}(k_b)\rangle\rangle, \mathtt{pk}(k_a)) & \text{if } B \text{ accepts } A\text{'s request} \\ \mathtt{aenc}(N_b, \mathtt{pk}(k)) & \text{if } B \text{ declines } A\text{'s request} \end{cases}$$

$$Alice \mid Bob(pk_a) \approx Alice \mid Bob(pk_c)$$

We can typecheck Bob's response by having bindings in Γ for all cases

- (k_a, k) authentication succeeds on the left, fails on the right
- (k, k_c) authentication succeeds on the right, fails on the left
- (k_a, k_c) authentication succeeds on both sides
- (k, k) authentication fails on both sides

Done?

Done? Not. Yet.

The case of dynamic keys

In the rules shown before, the keys were all fixed, long-term keys

$\longrightarrow$ We also want to consider key distribution mechanisms, where keys are

- generated (session keys)
- received from the network and then used to encrypt, decrypt, sign

The case of dynamic keys (2)

$\longrightarrow$ A new type for session keys

$$\text{seskey}^l(T)$$

Processes can

- generate session keys (must specify a type annotation)

$$\frac{\Gamma, (k, k) : \text{seskey}^l(T) \vdash P \sim Q : C}{\Gamma \vdash \texttt{new}\ k : \text{seskey}^l(T).\ P \sim \texttt{new}\ k : \text{seskey}^l(T).\ Q : C}$$

- receive and store session keys in variables of type $\text{seskey}^l(T)$
- use these variables as keys to encrypt, decrypt, ...

The case of dynamic keys (3)

$\longrightarrow$ Tricky point: consistency of the constraints

The case of dynamic keys (3)

$\longrightarrow$ Tricky point: consistency of the constraints

Example: If $x : \mathtt{LL}$ (key provided by the attacker), typechecking

$$\mathtt{out}(\mathtt{enc}(a, x)) \sim \mathtt{out}(\mathtt{enc}(a, x))$$

yields the constraint

$$\mathtt{enc}(a, x) \sim \mathtt{enc}(a, x)$$

The case of dynamic keys (3)

$\longrightarrow$ Tricky point: consistency of the constraints

Example: If $x : \mathtt{LL}$ (key provided by the attacker), typechecking

$$\mathtt{out}(\mathtt{enc}(a, x)) \sim \mathtt{out}(\mathtt{enc}(a, x))$$

yields the constraint

$$\mathtt{enc}(a, x) \sim \mathtt{enc}(a, x)$$

If we proceed as before and open the messages we get

$$x \sim x$$

which typically renders the constraint inconsistent

The case of dynamic keys (4)

Indeed: as soon as C contains

$$x \sim x \text{ and } M \sim N$$

if we choose $\theta(x) = M$ and $\theta'(x) \neq N$,

C instantiated with θ, θ' is not statically equivalent

The case of dynamic keys (4)

Indeed: as soon as C contains

$$x \sim x \text{ and } M \sim N$$

if we choose $\theta(x) = M$ and $\theta'(x) \neq N$,

C instantiated with θ, θ' is not statically equivalent

$\longrightarrow$ We need to further restrict the θ we consider

The case of dynamic keys (4)

Indeed: as soon as C contains

$$x \sim x \text{ and } M \sim N$$

if we choose $\theta(x) = M$ and $\theta'(x) \neq N$,

C instantiated with θ, θ' is not statically equivalent

$\longrightarrow$ We need to further restrict the θ we consider

$\longrightarrow$ Invariant: variables of type $\mathtt{LL}$ only contain messages
the attacker can construct from the remainder of the constraint

The case of dynamic keys (4)

Indeed: as soon as C contains

$$x \sim x \text{ and } M \sim N$$

if we choose $\theta(x) = M$ and $\theta'(x) \neq N$,

C instantiated with θ, θ' is not statically equivalent

$\longrightarrow$ We need to further restrict the θ we consider

$\longrightarrow$ Invariant: variables of type $\mathtt{LL}$ only contain messages the attacker can construct from the remainder of the constraint

$\longrightarrow$ Prevents the previous θ, θ' and solves the problem

Experimental results

- Prototype implementation for our type system
- We implement a typechecker,
  together with the procedure for constraints
- Very efficient
- But requires some type annotations

Protocol	Akiss	Apte	Apte-POR	Spec	Sat-Eq	TypeEq
Denning-Sacco	10	6	12	7	>30	>30
Wide Mouth Frog	14	7	12	7	>30	>30
Needham-Schroeder Symmetric Key	10	6	10	6	>30	>30
Yahalom-Lowe	10	6	10	7	>30	>30
Otway-Rees	6	3	6	6	-	>30
Needham-Schroeder-Lowe	8	4	4	4	-	>20

Number of sessions treated when proving secrecy
(bounded case)

Experimental results

Closer look for the Needham-Schroeder symmetric key protocol:

# sessions	Akiss	Apte	Apte-POR	Spec	Sat-Eq	TypeEq
3	0.1s	0.4s	0.02s	52s	0.2s	0.003s
6	20s	TO	4s	MO	0.4s	0.003s
7	2m		8m		1.3s	0.003s
10	SO		TO		2.3s	0.005s
12					4s	0.005s
14					7s	0.007s
30					1m6s	0.01s

Experimental results (unbounded)

We also compare to ProVerif for unbounded numbers of sessions:

Protocols	ProVerif	TypeEq
Helios	x	0.005s
Needham-Schroeder (sym)	0.23s	0.016s
Needham-Schroeder-Lowe	0.08s	0.008s
Yahalom-Lowe	0.48s	0.020s
Private Authentication	0.034s	0.008s
BAC	0.038s	0.005s

- Performances comparable to ProVerif for unbounded numbers of sessions
- First automated proof for Helios with unbounded number of sessions without private channels

Conclusion and future work

- a new approach to automatic proofs of equivalence properties for cryptographic protocols
- based on type systems + constraints
- handle bounded and unbounded number of sessions (CCS'17), dynamic keys, bikeys and non uniform branching (POST'18)
- efficient implementation

Future work:

- type inference
- computational soundness
- composition