Equivalence Properties by Typing in Cryptographic Protocols

Véronique Cortier, Niklas Grimm, Joseph Lallemand, Matteo Maffei

March 18, 2019
Introduction: cryptographic protocols

Cryptographic protocols

- Programs aiming to secure communications
  - payment, web, phones, voting...
- Use cryptographic primitives
  - encryption, signature, hashing...
- Provide security properties
  - secrecy of private data, authentication, anonymity, vote secrecy, ...

Example: SSL/TLS, AKA, ...

⇒ Importance of formal analysis of security protocols
Trace properties

Trace properties = satisfied by all traces of a protocol

Example: reachability properties:

Can the attacker learn a given message?

⇒ 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:

\[
Alice(0) \mid Bob(1) \approx Alice(1) \mid Bob(0)
\]
Proving equivalence

Procedures and tools exist to prove equivalence such as ProVerif, Tamarin, Akiss, Apte, Spec, SAT-Equiv, Deepsec.

- ProVerif, Tamarin handle unbounded numbers of sessions
- Tamarin is interactive, ProVerif is fully automated but more restrictive
- Deepsec, Akiss, Apte, Spec, SAT-Equiv prove equivalence for a bounded number of sessions

- mostly work by analysing all possible traces
Type systems

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

- Type systems: already applied to trace properties

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

- Now: for equivalence

\[ \vdash P \sim Q \implies P \approx Q \]

- 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

\[ \vdash P \sim Q \implies P \approx Q \]

- Efficient (though incomplete) procedures
- Modularity

Problem:

- Usually: typing \(\to\) overapproximate the set of traces.
- Sound for trace properties, but not equivalence
  \(\to\) 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:** \( \text{check}(C) \)
  ensure there are no leaks in the messages sent
  \( \rightarrow \) checking for repetitions

**Example:**

\[
C = \{ \text{enc}(x, k) \sim \text{enc}(a, k), \text{enc}(y, k) \sim \text{enc}(b, k) \}
\]

If in some execution we can have \( x = y \), equivalence is broken.
Main result: Soundness

Theorem (Soundness)

\[ \text{If } \Gamma \vdash P \sim Q : C \text{ and } \forall \theta. \ C\theta \text{ does not leak information, then} \]

\[ P \approx Q \]

Theorem (Procedure to check constraints)

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

Hypotheses:

- atomic keys only
- fixed cryptographic primitives: symmetric and asymmetric encryption, signature, hash, concatenation
- 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)

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

Hypotheses:

- atomic keys only
- fixed cryptographic primitives: symmetric and asymmetric encryption, signature, hash, concatenation
- 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)

\[ \Gamma \vdash P \sim Q : C \quad \implies \quad \Gamma \vdash !P \sim !Q : !C \]

How to check that \(!C\) does not leak information?

\[ \implies \text{It is sufficient to check two copies of } C:\]

Theorem (informal)

\[ \text{check}(C \cup C') \implies \text{check}(!C) \]
Symbolic model

- Messages are **terms** constructed using **abstract cryptographic primitives**,

- **Symbolic attacker** with abilities defined by **deduction rules**

```
\[
\text{enc}(x, y) \quad y \quad x \quad y
\]
\[
\text{enc}(x, y) \quad \langle x, y \rangle
\]
```
Symbolic model

Process algebra similar to the applied pi-calculus

\[ P, Q ::= \]
\[ 0 \]
\[ \text{new } n.P \]
\[ \text{out}(M).P \]
\[ \text{in}(x).P \]
\[ P \mid Q \]
\[ \text{let } x = d(y) \text{ in } P \text{ else } Q \]
\[ \text{if } M = N \text{ then } P \text{ else } Q \]
\[ !P \]
Static equivalence

Frames are sequences of messages modelling the attacker’s knowledge

\[ \phi = \{ x_1 \mapsto k, \ x_2 \mapsto a, \ x_3 \mapsto \text{enc}(b,k) \} \]

Static equivalence = indistinguishability of frames

\[ \phi \approx \phi' \iff \forall R, S. R\phi = S\phi \iff R\phi' = S\phi' \]

Example:

\[ \{ \text{enc}(a,k) \} \approx \{ \text{enc}(b,k) \} \]

but

\[ \{ \text{enc}(a,k), \text{enc}(a,k) \} \not\approx \{ \text{enc}(a,k), \text{enc}(b,k) \} \]

and

\[ \{ k, \text{enc}(a,k) \} \not\approx \{ k, \text{enc}(b,k) \} \]
Trace equivalence

A trace \((tr, \phi)\) 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)

\[
\forall (tr, \phi) \in \text{trace}(P). \exists (tr, \phi') \in \text{trace}(Q). \phi \approx \phi'
\]

and

\[
\forall (tr, \phi) \in \text{trace}(Q). \exists (tr, \phi') \in \text{trace}(P). \phi \approx \phi'
\]
Typing messages

Types for messages:

\[
I ::= \text{LL} \mid \text{HL} \mid \text{HH} \\
T ::= I \\
| \text{key}^I(T) \\
| T \ast T \\
| T \lor T \\
| \ldots
\]

- labels = levels of **confidentiality** and **integrity**
  - LL for public messages
  - HH for secret values

- key types \(\text{key}^I(T)\)

**Example:**

\[\text{key}^{HH}(\text{LL} \ast \text{HH})\]
Typing messages

\[
\Gamma \vdash M \sim N : T \quad \Gamma(k) = \text{key}^{HH}(T) \\
\Gamma \vdash \text{enc}(M, k) \sim \text{enc}(N, k) : LL
\]

Ensure the messages sent are safe to output:

\[ \rightarrow \text{similar structure} \]

\[
\langle a, b \rangle \not\sim a
\]

\[
\text{enc}(\langle a, b \rangle, k) \sim \text{enc}(a, k) \quad \text{only if } k \text{ is secret}
\]
Typing messages

\[ \Gamma \vdash M \sim N : T \rightarrow c \quad \Gamma(k) = \text{key}^{\text{HH}}(T) \]
\[ \Gamma \vdash \text{enc}(M, k) \sim \text{enc}(N, k) : \text{LL} \rightarrow c \cup \{ \text{enc}(M, k) \sim \text{enc}(N, k) \} \]

- 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 \( \text{enc}(M, k) \sim \text{enc}(N, k) \) to the constraint
Typing processes

- All output messages must be of type $LL$
- Their constraints are collected

$$
\Gamma \vdash M \sim N : LL \rightarrow c \quad \Gamma \vdash P \sim Q : C
\quad
\Gamma \vdash \text{out}(M).P \sim \text{out}(N).Q : C \cup c
$$

- All input messages are considered to be of type $LL$

$$
\Gamma, x : LL \vdash P \sim Q : C
\quad
\Gamma \vdash \text{in}(x).P \sim \text{in}(x).Q : C
$$
Typing processes (2)

Processes have to progress the same way:

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

Example: applying destructors

\[
\begin{align*}
\Gamma(x) &= LL, \quad \Gamma(k) = \text{key}^{HH}(T) \\
\Gamma, y : T &\vdash P \sim Q : C, \quad \Gamma \vdash P' \sim Q' : C' \\
\Gamma &\vdash \text{let } y = \text{dec}(x, k) \text{ in } P \text{ else } P' \sim \text{let } y = \text{dec}(x, k) \text{ in } Q \text{ else } Q' : C \cup C'
\end{align*}
\]
Constraints

Why do we need constraints?
Constraints

Why do we need constraints?

→ Local checks on the messages are not sufficient for equivalence
Constraints

Why do we need constraints?

Local checks on the messages are not sufficient for equivalence

Example: If $k$ is a secret key

\[
\text{out}(\text{enc}(a, k)) \sim \text{out}(\text{enc}(b, k)) \quad \text{is fine}
\]
\[
\text{out}(\text{enc}(a, k)) \sim \text{out}(\text{enc}(a, k)) \quad \text{is fine}
\]

but not both together

\[
\text{out}(\text{enc}(a, k)) \mid \text{out}(\text{enc}(a, k)) \neq \text{out}(\text{enc}(b, k)) \mid \text{out}(\text{enc}(a, k))
\]
Constraints

Why do we need constraints?

→ Local checks on the messages are not sufficient for equivalence

Example: If $k$ is a secret key

$$\text{out}(\text{enc}(a, k)) \sim \text{out}(\text{enc}(b, k)) \text{ is fine}$$
$$\text{out}(\text{enc}(a, k)) \sim \text{out}(\text{enc}(a, k)) \text{ is fine}$$

but not both together

$$\text{out}(\text{enc}(a, k)) | \text{out}(\text{enc}(a, k)) \not\sim \text{out}(\text{enc}(b, k)) | \text{out}(\text{enc}(a, k))$$

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

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

$\rightarrow$ Actually only consider well-typed $\theta, \theta'$

i.e.

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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Akiss</th>
<th>Apte</th>
<th>Apte-POR</th>
<th>Spec</th>
<th>Sat-Eq</th>
<th>Deepsec</th>
<th>TypeEq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denning-Sacco</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>7</td>
<td>&gt;20</td>
<td>&gt;20</td>
<td>&gt;20</td>
</tr>
<tr>
<td>Wide Mouth Frog</td>
<td>14</td>
<td>7</td>
<td>12</td>
<td>7</td>
<td>&gt;20</td>
<td>&gt;20</td>
<td>&gt;20</td>
</tr>
<tr>
<td>Needham-Schroeder Symmetric Key</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>&gt;20</td>
<td>&gt;20</td>
<td>&gt;20</td>
</tr>
<tr>
<td>Yahalom-Lowe</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>7</td>
<td>&gt;20</td>
<td>&gt;20</td>
<td>&gt;20</td>
</tr>
<tr>
<td>Otway-Rees</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>-</td>
<td>&gt;20</td>
<td>&gt;20</td>
</tr>
</tbody>
</table>

Number of sessions treated when proving secrecy (bounded case)
Experimental results

Closer look for the Needham-Schroeder symmetric key protocol:

<table>
<thead>
<tr>
<th># sessions</th>
<th>Akiss</th>
<th>Apte</th>
<th>Apte-POR</th>
<th>Spec</th>
<th>Sat-Eq</th>
<th>Deepsec (on 40 cpu cores)</th>
<th>TypeEq</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.1s</td>
<td>0.4s</td>
<td>0.02s</td>
<td>52s</td>
<td>0.1s</td>
<td>0.16s</td>
<td>0.003s</td>
</tr>
<tr>
<td>6</td>
<td>20s</td>
<td></td>
<td>4s</td>
<td></td>
<td>0.2s</td>
<td>0.17s</td>
<td>0.003s</td>
</tr>
<tr>
<td>7</td>
<td>2m</td>
<td></td>
<td>8m</td>
<td></td>
<td>0.5s</td>
<td>0.17s</td>
<td>0.003s</td>
</tr>
<tr>
<td>10</td>
<td>SO</td>
<td></td>
<td>TO</td>
<td></td>
<td>0.9s</td>
<td>0.18s</td>
<td>0.005s</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.7s</td>
<td>0.2s</td>
<td>0.005s</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2s</td>
<td>0.3s</td>
<td>0.007s</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17s</td>
<td>8s</td>
<td>0.01s</td>
</tr>
</tbody>
</table>
Experimental results (unbounded)

We also compare to ProVerif for unbounded numbers of sessions:

<table>
<thead>
<tr>
<th>Protocols</th>
<th>ProVerif</th>
<th>TypeEq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helios</td>
<td>x</td>
<td>0.005s</td>
</tr>
<tr>
<td>Needham-Schroeder (sym)</td>
<td>0.23s</td>
<td>0.016s</td>
</tr>
<tr>
<td>Needham-Schroeder-Lowe</td>
<td>0.08s</td>
<td>0.008s</td>
</tr>
<tr>
<td>Yahalom-Lowe</td>
<td>0.48s</td>
<td>0.020s</td>
</tr>
<tr>
<td>Private Authentication</td>
<td>0.034s</td>
<td>0.008s</td>
</tr>
<tr>
<td>BAC</td>
<td>0.038s</td>
<td>0.005s</td>
</tr>
</tbody>
</table>

- 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 numbers of sessions (CCS’17), dynamic keys, non uniform branching (POST’18)
- efficient implementation

Future work:

- type inference
- computational soundness
- composition