A Method for Verifying Privacy-Type Properties: The Unbounded Case

HotSpot 2016

Lucca Hirschi

April 3rd, 2016

joint work with David Baelde, LSV *and* Stéphanie Delaune, LSV

Introduction

$\rightsquigarrow$ we need formal verification of crypto protocols covering privacy

Introduction

⇝ we need formal verification of crypto protocols covering privacy

Goal:

- checking unlinkability and anonymity
- in the symbolic model (Dolev-Yao)
- for unbounded sessions

Introduction

⇝ we need formal verification of crypto protocols covering privacy

Goal:

- checking unlinkability and anonymity
- in the symbolic model (Dolev-Yao)
- for unbounded sessions

- *Unlinkability (=untraceability)* [ISO/IEC 15408]:

  Ensuring that a user may make multiple uses of a service or resource without others being able to link these uses together.

- *Anonymity* [ISO/IEC 15408]:

  Ensuring that a user may use a service or resource without disclosing the user's identity. [...]

Context

Strong unlinkability [Arapinis, Chothia, Ritter, Ryan CSF'10]:

$$\underbrace{!\ \nu\,\vec{k}\,(!\ \nu\vec{n}(T \mid R))}_{\mathcal{M}} \approx \underbrace{!\ \nu\,\vec{k}.\nu\vec{n}(T \mid R)}_{\mathcal{S}}$$

- $\mathcal{M}$: ∞ many different $T - R$ playing ∞ many sessions
- $\mathcal{S}$: ∞ many different $T - R$ playing at moste one session
- $\approx$: observational equivalence (trace equivalence)

Context

Strong unlinkability [Arapinis, Chothia, Ritter, Ryan CSF'10]:

$$\underbrace{!\,\nu\,\vec{k}\,(!\,\nu\vec{n}(T \mid R))}_{\mathcal{M}} \approx \underbrace{!\,\nu\,\vec{k}.\nu\vec{n}(T \mid R)}_{\mathcal{S}}$$

- $\mathcal{M}$: ∞ many different $T - R$ playing ∞ many sessions
- $\mathcal{S}$: ∞ many different $T - R$ playing at moste one session
- $\approx$: observational equivalence (trace equivalence)

- Checking this is undecidable (because of replication)

Existing approaches:

- manual: need to exhib huge bisimulations
- automatic (ProVerif/Maude-NPA/Tamarin):
  rely on abstraction (diff-equivalence) not precise enough
  $\rightsquigarrow$ always fail to prove unlinkability

Context

Strong unlinkability [Arapinis, Chothia, Ritter, Ryan CSF'10]:

$$\underbrace{!\,\nu\,\vec{k}\,(!\,\nu\vec{n}(T \mid R))}_{\mathcal{M}} \approx \underbrace{!\,\nu\,\vec{k}.\nu\vec{n}(T \mid R)}_{\mathcal{S}}$$

- $\mathcal{M}$: ∞ many different $T - R$ playing ∞ many sessions
- $\mathcal{S}$: ∞ many different $T - R$ playing at moste one session
- $\approx$: observational equivalence (trace equivalence)

- Checking this is undecidable (because of replication)

Existing approaches:

- manual: need to exhib huge bisimulations
- automatic (ProVerif/Maude-NPA/Tamarin):
  rely on abstraction (diff-equivalence) not precise enough
  $\rightsquigarrow$ always fail to prove unlinkability

$\rightsquigarrow$ there is a need for dedicated abstraction targeting unlinkability

Contribution

We identify:

- 2 conditions implying unlinkability and anonymity
- for a class of 2-agents protocols including our target case studies

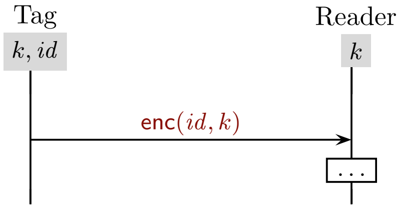
We make sure:

- our conditions can be checked automatically using ProVerif
- they correspond to good design practices

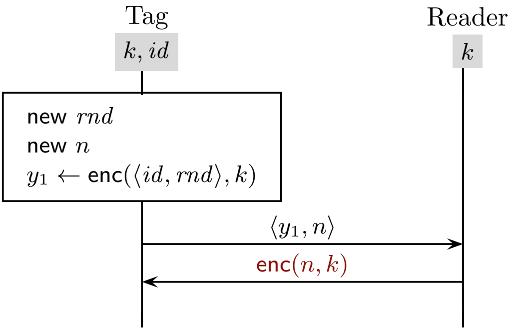
$\rightsquigarrow$ sound approach to check automatically privacy properties
working well in practice

I : What could go wrong ?

R1: Messing up with messages



R1: Messing up with messages



Practical examples (RFID protocols): HB$^+$, DM, KCL, LBV, LD, ...

R1: Messing up with messages

Problem

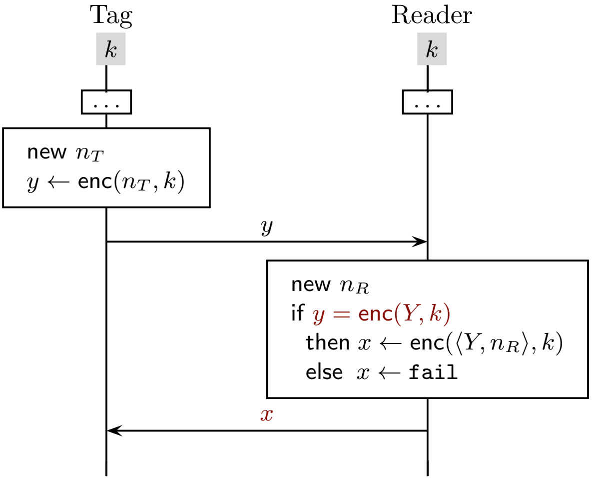
For some malicious beahvior, relations over messages leak info about involved agents.

Main idea to avoid that:

- outputs are (statically) indistiguishable from $\neq$ nonces

$\rightsquigarrow$ Condition 1: Frame Opacity (FO)

R2: Messing up with conditionals



Practical examples: BAC (ePassport), some versions of PACE (new version of ePassport), LAK, CH

R2: Messing up with conditionals

Problem

For some malicious behavior, outcome of conditionals leak info about involved agents

Main idea to avoid that:

- conditional true $\iff$ attacker did not interfer

$\rightsquigarrow$ Condition 2: Well-Authentication (WA)

II : Big picture

	Equivalence?	Active Attacker?
UK/ANO	☑	☑

	Equivalence?	Active Attacker?
UK/ANO	☑	☑

⇑ **Theorem: implies** ⇑

FO	"Messages are without relations"
WA	"Conditionals hold only for honest interactions"

	Equivalence?	Active Attacker?
UK/ANO	☑	☑

⇑ **Theorem: implies** ⇑

	Equivalence?	Active Attacker?
FO	☑	☐
WA	☐	☑

	Equivalence?	Active Attacker?
UK/ANO	☑	☑

$\Uparrow$ **Theorem: implies** $\Uparrow$

	Equivalence?	Active Attacker?
FO	☑	☐
WA	☐	☑

$\Uparrow$ can be **checked** $\Uparrow$

- FO: automatic check of diff-equivalence using Proverif
- WA: automatic check of correspondence prop. using Proverif

	Equivalence?	Active Attacker?
UK/ANO	☑	☑

⇑ **Theorem: implies** ⇑

	Equivalence?	Active Attacker?
FO	☑	☐
WA	☐	☑

⇑ can be **checked** ⇑

- FO: automatic check of diff-equivalence using Proverif
- WA: automatic check of correspondence prop. using Proverif

Tight enough to conclude on our case studies:

(BAC, LAK, Hash-Lock, EKE, SPKE)

III : Model and Problem

Applied-π - Terms

Any Σ-algebra $+$ equational theory E $+$ reduction rules (*à la Proverif*)

Example

- $\Sigma_c = \{\mathsf{dh}/2, \langle \_, \_ \rangle/2, \mathsf{enc}/2, \mathsf{ok}/0, \mathsf{no}/0\}$
- $\Sigma_d = \{\pi_1/1, \pi_2/1, \mathsf{dec}/2\}$
- $\mathsf{E} = \{(\mathsf{dh}(\mathsf{dh}(x, y), z) = \mathsf{dh}(\mathsf{dh}(x, z), y))\}$
- $\mathsf{def}_\Sigma(\mathsf{dec}) = \{\mathsf{dec}(\mathsf{enc}(x, y), y) \to x\}$
- $\mathsf{def}_\Sigma(\pi_i) = \{\pi_i(\langle x_1, x_2 \rangle) \to x_i\}$

induce

- a congruence $=_\mathsf{E}$ — *e.g.*, $g^{x^{y^z}} =_\mathsf{E} g^{z^{y^x}}$
- a "computation" relation $\Downarrow$ — *e.g.*, $\mathsf{dec}(\mathsf{enc}(n, g^{a^b}), g^{b^a}) \Downarrow n$

Applied-π - Terms

Any Σ-algebra $+$ equational theory E $+$ reduction rules (*à la Proverif*)

Example

- $\Sigma_c = \{\mathsf{dh}/2, \langle\_,\_\rangle/2, \mathsf{enc}/2, \mathsf{ok}/0, \mathsf{no}/0\}$
- $\Sigma_d = \{\pi_1/1, \pi_2/1, \mathsf{dec}/2\}$
- $\mathsf{E} = \{(\mathsf{dh}(\mathsf{dh}(x,y),z) = \mathsf{dh}(\mathsf{dh}(x,z),y))\}$
- $\mathsf{def}_\Sigma(\mathsf{dec}) = \{\mathsf{dec}(\mathsf{enc}(x,y),y) \to x\}$
- $\mathsf{def}_\Sigma(\pi_i) = \{\pi_i(\langle x_1, x_2\rangle) \to x_i\}$

induce

- a congruence $=_\mathsf{E}$ — *e.g.,* $g^{x^{y^z}} =_\mathsf{E} g^{z^{y^x}}$
- a "computation" relation $\Downarrow$ — *e.g.,* $\mathsf{dec}(\mathsf{enc}(n, g^{a^b}), g^{b^a}) \Downarrow n$

$\rightsquigarrow$ We deal with an arbitrary theory.

Applied-π - Syntax

- Process:

$$
\begin{array}{rcll}
P, Q & := & 0 & \text{null} \\
& | & \mathtt{in}(c, x).P & \text{input} \\
& | & \mathtt{out}(c, u).P & \text{output} \\
& | & \text{if Test then } P \text{ else } Q & \text{conditional} \\
& | & P \mid Q & \text{parallel} \\
& | & !P & \text{replication} \\
& | & \nu\, n.P & \text{restriction}
\end{array}
$$

Applied-π - Syntax

- Process:

$$
\begin{array}{rclr}
P, Q & := & 0 & \text{null} \\
 & | & \mathtt{in}(c, x).P & \text{input} \\
 & | & \mathtt{out}(c, u).P & \text{output} \\
 & | & \text{if Test then } P \text{ else } Q & \text{conditional} \\
 & | & P \mid Q & \text{parallel} \\
 & | & !P & \text{replication} \\
 & | & \nu\, n.P & \text{restriction}
\end{array}
$$

- Frame (ϕ): the set of messages revelead to the network
  $\rightsquigarrow$ intuition: intruder's knowledge

$$\phi = \{\underbrace{w_1}_{\text{handle}} \mapsto \underbrace{\mathrm{enc}(m, k)}_{\text{out. message}}; w_2 \mapsto k\}$$

Applied-π - Syntax

- Process:

$$
\begin{array}{rcll}
P, Q & := & 0 & \text{null} \\
 & | & \mathtt{in}(c, x).P & \text{input} \\
 & | & \mathtt{out}(c, u).P & \text{output} \\
 & | & \text{if Test then } P \text{ else } Q & \text{conditional} \\
 & | & P \mid Q & \text{parallel} \\
 & | & !P & \text{replication} \\
 & | & \nu\, n.P & \text{restriction}
\end{array}
$$

- Frame (ϕ): the set of messages revelead to the network
  $\rightsquigarrow$ intuition: intruder's knowledge

$$\phi = \{\underbrace{w_1}_{\text{handle}} \mapsto \underbrace{\text{enc}(m, k)}_{\text{out. message}}; w_2 \mapsto k\}$$

- Configuration: $A = (\mathcal{P}; \phi)$

Applied-π - Semantics

- Recipes: are terms built using handles

$$e.g.,\quad \begin{array}{l} R = \mathrm{dec}(w_1, w_2) \\ R\phi \Downarrow m \end{array} \qquad \text{for } \phi = \{w_1 \mapsto \mathrm{enc}(m, k);\, w_2 \mapsto k\}$$

$\rightsquigarrow$ intuition: how the environment builds messages from its knowledge

Applied-π - Semantics

- Recipes: are terms built using handles

$$e.g.,\quad \begin{array}{l} R = \mathrm{dec}(w_1, w_2) \\ R\phi \Downarrow m \end{array} \qquad \text{for } \phi = \{w_1 \mapsto \mathrm{enc}(m, k); w_2 \mapsto k\}$$

$\leadsto$ intuition: how the environment builds messages from its knowledge

- Semantics of configurations:

$$(\mathtt{in}(c, x).P \cup \mathcal{P}; \phi) \xrightarrow{\mathtt{in}(c,R)} (P\{x \mapsto u\} \cup \mathcal{P}; \phi) \qquad \text{if } R\phi \Downarrow u$$

$$(\mathtt{out}(c, u).P \cup \mathcal{P}; \phi) \xrightarrow{\mathtt{out}(c,w)} (P \cup \mathcal{P}; \phi \cup \{w \mapsto u\}) \quad \text{if } w \text{ fresh}$$

Applied-π - Semantics

- Recipes: are terms built using handles

$$e.g.,\quad \begin{array}{l} R = \mathrm{dec}(w_1, w_2) \\ R\phi \Downarrow m \end{array} \qquad \text{for } \phi = \{w_1 \mapsto \mathrm{enc}(m, k); w_2 \mapsto k\}$$

$\rightsquigarrow$ intuition: how the environment builds messages from its knowledge

- Semantics of configurations:

$$(\mathtt{in}(c, x).P \cup \mathcal{P}; \phi) \xrightarrow{\mathtt{in}(c,R)} (P\{x \mapsto u\} \cup \mathcal{P}; \phi) \qquad \text{if } R\phi \Downarrow u$$

$$(\mathtt{out}(c, u).P \cup \mathcal{P}; \phi) \xrightarrow{\mathtt{out}(c,w)} (P \cup \mathcal{P}; \phi \cup \{w \mapsto u\}) \quad \text{if } w \text{ fresh}$$

$+$ expected rules for conditional and other constructs

Applied-π - Trace Equivalence

Static Equivalence (intuitively)

$\Phi \sim \Psi$ when

- $\mathrm{dom}(\Phi) = \mathrm{dom}(\Psi)$ and
- for all tests, it holds on $\phi \iff$ it holds on ψ

Trace Equivalence

$A \sqsubseteq B$ when, for any $A \xrightarrow{\mathrm{tr}} A'$ there exists $B \xrightarrow{\mathrm{tr}} B'$ such that $\Phi(A') \sim \Phi(B')$.

$$A \approx B, \text{ when } A \sqsubseteq B \text{ and } B \sqsubseteq A.$$

Our class of protocols & our problem

Our class

- Intuitively, a party P is a process of the form:

$$\begin{array}{rcl} P & ::= & 0 \mid \texttt{in}(c,y).\ \text{if Test then } \texttt{out}(c,u).P_R \text{ else } P_{\text{else}} \\ P_{\text{else}} & ::= & 0 \mid \texttt{out}(c',u') \end{array}$$

Our class of protocols & our problem

Our class

- Intuitively, a party P is a process of the form:

$$P \quad ::= \quad 0 \mid \mathtt{in}(c,y).\ \text{if Test then } \mathtt{out}(c,u).P_R \text{ else } P_{\text{else}}$$
$$P_{\text{else}} \quad ::= \quad 0 \mid \mathtt{out}(c',u')$$

- A protocol Π is a tuple $(\vec{k}, \vec{n}_T, \vec{n}_R, T, R)$ where:
  - T and R are parties
  - $\vec{k}$: identity names and $\vec{n}_T/\vec{n}_R$: session names
  - $fn(T) \subseteq \vec{k} \sqcup \vec{n}_T$ (resp. for R)

Our class of protocols & our problem

Our class

- Intuitively, a party P is a process of the form:

$$\begin{array}{rcl} P & ::= & 0 \mid \mathtt{in}(c, y).\ \text{if Test then } \mathtt{out}(c, u).P_R \text{ else } P_{\text{else}} \\ P_{\text{else}} & ::= & 0 \mid \mathtt{out}(c', u') \end{array}$$

- A protocol Π is a tuple $(\vec{k}, \vec{n}_T, \vec{n}_R, T, R)$ where:
  - T and R are parties
  - $\vec{k}$: identity names and $\vec{n}_T / \vec{n}_R$: session names
  - $fn(T) \subseteq \vec{k} \sqcup \vec{n}_T$ (resp. for R)

Unlinkability

$$\underbrace{!\ \nu\ \vec{k}\ (!\ (\nu \vec{n}_T T \mid \nu \vec{n}_R R))}_{\mathcal{M}} \approx \underbrace{!\ \nu\ \vec{k}.(\nu \vec{n}_T T \mid \nu \vec{n}_R R)}_{\mathcal{S}}$$

IV : Sufficient conditions

Frame opacity

Frame opacity

For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^{\mathrm{nonce}}$.

Frame opacity

Frame opacity

For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^{\mathsf{nonce}}$.

Require that all outputs are $\sim$ from nonces is too strong:

- $\Phi = \{w \mapsto \langle \mathsf{enc}(n_1, k), \mathsf{enc}(n_2, k)\rangle\}$
- if $[\Phi]^{\mathsf{nonce}} = \{w \mapsto n\}$ then $\Phi \not\sim [\Phi]^{\mathsf{nonce}}$
- if $[\Phi]^{\mathsf{nonce}} = \{w \mapsto \langle n, n'\rangle\}$ then $\Phi \sim [\Phi]^{\mathsf{nonce}}$

Frame opacity

Frame opacity

For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^{\mathrm{nonce}}$.

Require that all outputs are $\sim$ from nonces is too strong:

- $\Phi = \{w \mapsto \langle \mathrm{enc}(n_1, k), \mathrm{enc}(n_2, k) \rangle\}$
- if $[\Phi]^{\mathrm{nonce}} = \{w \mapsto n\}$ then $\Phi \not\sim [\Phi]^{\mathrm{nonce}}$
- if $[\Phi]^{\mathrm{nonce}} = \{w \mapsto \langle n, n' \rangle\}$ then $\Phi \sim [\Phi]^{\mathrm{nonce}}$

Transparent function symbols

$f \in \Sigma_c$ is *transparent* if:

- attacker can extract its arguments and
- does not appear in E.

Frame opacity

Frame opacity

For any execution $\mathcal{M} \xrightarrow{t} B$, we have that $\Phi(B) \sim [\Phi(B)]^{\mathsf{nonce}}$.

- $\Phi = \{w \mapsto \langle \mathsf{enc}(n_1, k), \mathsf{enc}(n_2, k) \rangle\}$
- $[\Phi]^{\mathsf{nonce}} = \{w \mapsto \langle n, n' \rangle\}$ and $\Phi \sim [\Phi]^{\mathsf{nonce}}$

Transparent function symbols

$f \in \Sigma_c$ is *transparent* if:

- attacker can extract its arguments and
- does not appear in E.

Idealization

There exists a function $[\cdot]^{\mathsf{ideal}} : \mathcal{T}(\Sigma_c, \mathcal{N}) \to \mathcal{T}(\Sigma_t, \{\Box\})$ such that:

- $[u]^{\mathsf{ideal}} = \mathsf{f}([u_1]^{\mathsf{ideal}}, \ldots)$ if $u =_{\mathsf{E}} \mathsf{f}(u_1, \ldots)$ for some $\mathsf{f} \in \Sigma_t$,
- and $[u]^{\mathsf{ideal}} = \Box$ otherwise.

Well-Authentication

We assume additional annotations to actions:

$$e.g., (\{T\{\vec{k} \mapsto \vec{k}_0; \vec{n}_T \mapsto \vec{n}_0\}\}; \phi) \xrightarrow{\mathtt{in}(c,x)[T(\vec{k}_0,\vec{n}_0)].\mathtt{then}[T(\vec{k}_0,\vec{n}_0)]} .$$

Well-Authentication

We assume additional annotations to actions:

$$e.g., (\{T\{\vec{k} \mapsto \vec{k}_0; \vec{n}_T \mapsto \vec{n}_0\}\}; \phi) \xrightarrow{\mathtt{in}(c,x)[T(\vec{k}_0,\vec{n}_0)].\mathtt{then}[T(\vec{k}_0,\vec{n}_0)]} .$$

Well-Authentication

$\Pi = (\vec{k}, \vec{n}_T, \vec{n}_R, T, R)$ is *well-authenticating* if, for any execution

$$(\mathcal{M}; \emptyset) \xrightarrow{t.\mathtt{then}[T(\vec{k},\vec{n}_1)]} (\mathcal{P}; \Phi)$$

there must be a $R(\vec{k}, \vec{n}_2)$ such that $T(\vec{k}, \vec{n}_1)$ and $R(\vec{k}, \vec{n}_2)$ were having an honest execution in (t, Φ). $+$ similarly for R

A trace t is *honest* for a frame Φ if

- $\mathtt{else} \notin t$ and
- $\mathrm{obs}(t) = \mathtt{out}(\cdot, w_0).\mathtt{in}(\cdot, M_0).\mathtt{out}(\cdot, w_1)\ldots$ with $M_i\Phi \Downarrow =_E w_i\Phi$.

Main Theorem

If $\Pi = (\vec{k}, \vec{n}_T, \vec{n}_R, T, R)$ is well-authenticating and $\mathcal{M}$ ensures frame opacity, then Π ensures unlinkability.

A similar theorem for both unlinkability and anonymity.

V : Applications

Tool: UKano

We wrote UKano: a tool built on top of ProVerif that automatically checks our two sufficient conditions.

Tool: UKano

We wrote UKano: a tool built on top of ProVerif that automatically checks our two sufficient conditions.

New proofs of Unlinkability & Anonymity for:

- Feldhofer, Hash-Lock and (fixed) LAK (RFID auth.);
- BAC+PA+AA, (fixed) PACE (ePassport);

Tool: UKano

We wrote UKano: a tool built on top of ProVerif that automatically checks our two sufficient conditions.

New proofs of Unlinkability & Anonymity for:

- Feldhofer, Hash-Lock and (fixed) LAK (RFID auth.);
- BAC+PA+AA, (fixed) PACE (ePassport);

When conditions fail to hold: no direct attacks but still...

Flaws/attacks discovered:

- PACE ($\neg$ UK);
- LAK ($\neg$ UK).

Paper, sources of UKano, ProVerif files at
`http://projects.lsv.ens-cachan.fr/ukano/`

VI : Conclusion

	Equivalence?	Active Attacker?
UK/ANO	☑	☑

⇑ **Theorem: implies** ⇑

FO	“Messages are without relations”
WA	“Conditionals hold only for honest interactions”

⇑ can be **checked** ⇑

- FO: automatic check of diff-equivalence using Proverif
- WA: automatic check of correspondence prop. using Proverif

Tight enough to conclude on our case studies:

(BAC, LAK, Hash-Lock, EKE, SPKE)

Future Work

Improve the method

- tackle memory (often used in RFID)
- move to other tools as backends (Tamarin, Maude-NPA)
- allow more flexibility for idealization

Future Work

Improve the method

- tackle memory (often used in RFID)
- move to other tools as backends (Tamarin, Maude-NPA)
- allow more flexibility for idealization

Reusing core ideas

- exploit our conditions to obtain other properties
- extract guidelines from our conditions

Paper, sources of UKano, ProVerif files at
`http://projects.lsv.ens-cachan.fr/ukano/`

Thank you !