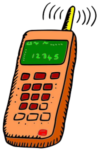
A Method for Verifying Privacy-Type Properties: The Unbounded Case

Security & Privacy 2016

Lucca Hirschi, David Baelde and Stéphanie Delaune

Security & Privacy 2016

Introduction



⇝ we need formal verification of crypto protocols covering privacy

Introduction

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

Goal:

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

Introduction

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

Goal:

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

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.*

Symbolic Model

Symbolic attacker () controls all the network:

Symbolic Model

[$\{n\}_k$: symmetric encryption]

Symbolic attacker ( ) controls all the network:

- eavesdrops messages

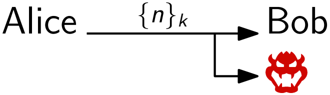
$$\text{Alice} \xrightarrow{\{n\}_k} \text{Bob}$$

Symbolic Model

[$\{n\}_k$: symmetric encryption]

Symbolic attacker ( ) controls all the network:

- eavesdrops messages



Symbolic Model

Symbolic attacker () controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives

$$\left(\text{ knows } \{n\}_k \text{ and } k\right) \Rightarrow \left(\text{ knows } n\right)$$

Symbolic Model

Symbolic attacker ( ) controls all the network:

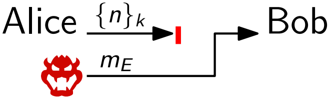
- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

$$\text{Alice} \xrightarrow{\{n\}_k} \text{Bob}$$

Symbolic Model

Symbolic attacker () controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages



Symbolic Model

Symbolic attacker () controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But cannot break crypto primitives.

Symbolic Model

Symbolic attacker ( ) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But cannot break crypto primitives.

Symbolic model, pros & cons:

$\ominus$ less precise than computational model
$\oplus$ allows for automation

Symbolic Model

Symbolic attacker () controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But  cannot break crypto primitives.

Symbolic model, pros & cons:

⊖ less precise than computational model
⊕ allows for automation

Ingredients for modeling:

- messages: term algebra with equational theory

Symbolic Model

Symbolic attacker ( ) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But cannot break crypto primitives.

Symbolic model, pros & cons:

$\ominus$ less precise than computational model

$\oplus$ allows for automation

Ingredients for modeling:

- messages: term algebra with equational theory
- protocols & attacker: process algebra (*e.g.*, applied π-calculus)

Symbolic Model

Symbolic attacker () controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But cannot break crypto primitives.

Symbolic model, pros & cons:

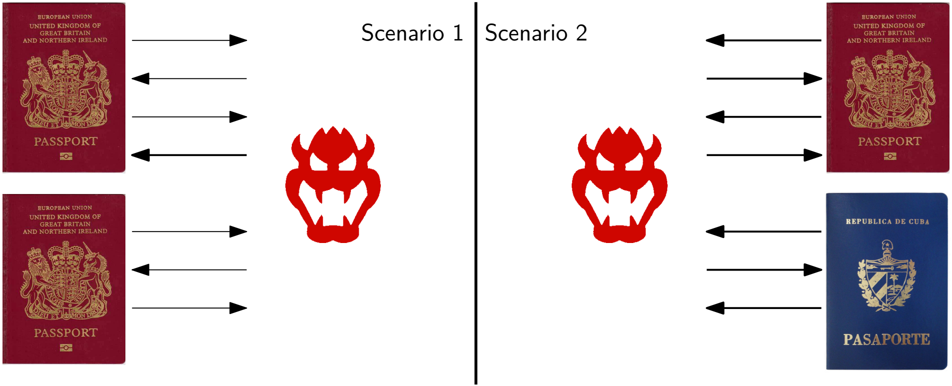
⊖ less precise than computational model
⊕ allows for automation

Ingredients for modeling:

- messages: term algebra with equational theory
- protocols & attacker: process algebra (*e.g.*, applied π-calculus)
- security properties: reachability & observational equivalence

I : Problem

Unlinkability



"Real" usage of the protocol

"Ideal" usage of the protocol

$\forall$ ,  cannot observe any difference

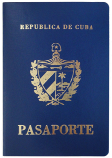
Unlinkability

Scenario 1

"Real" usage of the protocol

$\approx$

Scenario 2



"Ideal" usage of the protocol

$\approx$: trace equivalence

(observational equivalence between processes)

Unlinkability

Scenario 1

• • •

Session 1	Session 1	Session 1
Session 2	Session 2	Session 2
Session 3	Session 3	Session 3
• • •	• • •	• • •

$\approx$

Scenario 2

• • •

Session 1	Session 1	Session 1

- Infinitely many users
- Each playing infinitely many sessions

Unlinkability

Scenario 1

id_1	id_2	id_3	

Session 1	Session 1	Session 1	…
Session 2	Session 2	Session 2	
Session 3	Session 3	Session 3	
…	…	…	

Scenario 2

id_1	id_2	id_3	

Session 1	Session 1	Session 1	…

$$!\nu\,\mathrm{id}\;!\nu\,\mathrm{Sess}.\;P \quad \approx \quad !\nu\,\mathrm{id}.\nu\,\mathrm{Sess}.\;P$$

∞ users

∞ sessions

∞ users

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

The Problem & Existing Approaches

Goal

- automatic verification of

$$!\,\nu\,\mathrm{id}.\,(!\,\nu\,\mathrm{Sess}.P) \approx\ !\,\nu\,\mathrm{id}.\,(\nu\,\mathrm{Sess}.P)$$

for a large class of 2-party protocols (think of $P = \mathrm{Tag} \mid \mathrm{Reader}$)

The Problem & Existing Approaches

Goal

- automatic verification of

$$!\,\nu\,\mathrm{id}.\,(!\,\nu\,\mathrm{Sess}.P) \approx\ !\,\nu\,\mathrm{id}.\,(\nu\,\mathrm{Sess}.P)$$

for a large class of 2-party protocols (think of $P = \mathrm{Tag} \mid \mathrm{Reader}$)

Existing approaches:

- manual: long, difficult, and highly error prone
- automatic (only ProVerif/Maude-NPA/Tamarin):
  - rely on too imprecise approximation of $\approx$
  - $\rightsquigarrow$ always fail to prove unlinkability

Contributions

Theory:

- 2 reasonable conditions implying unlinkability (& anonymity)
- for a large class of 2-party protocols

Contributions

Theory:

- 2 reasonable conditions implying unlinkability (& anonymity)
- for a large class of 2-party protocols

Practice:

- our conditions can be checked automatically using existing tools
- we provide tool support for that (UKano)

Contributions

Theory:

- 2 reasonable conditions implying unlinkability (& anonymity)
- for a large class of 2-party protocols

Practice:

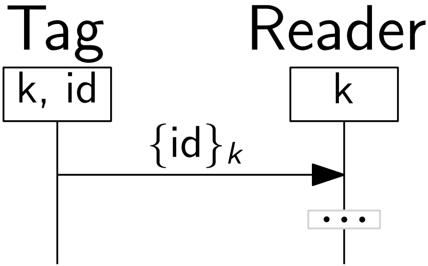
- our conditions can be checked automatically using existing tools
- we provide tool support for that (UKano)

Applications:

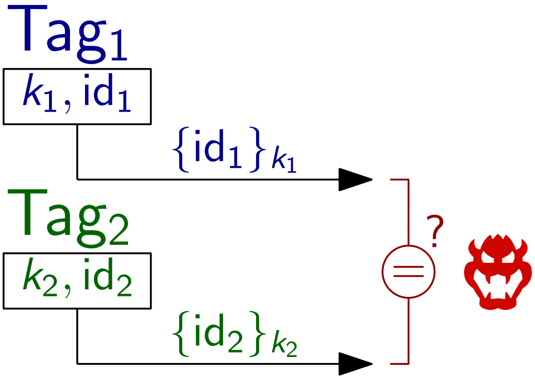
- new proofs & attacks on RFID protocols

II : Two Generic Classes of Attacks
Two Conditions to Avoid them

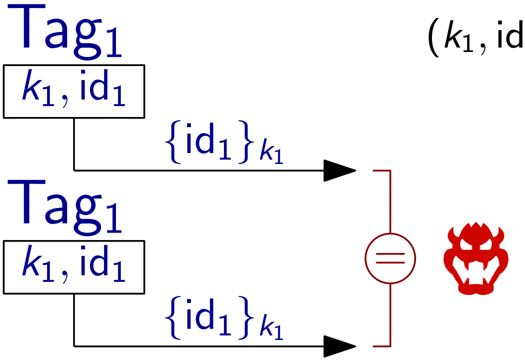
1^{st} Class: Leaks through Relations over Messages



1^{st} Class: Leaks through Relations over Messages

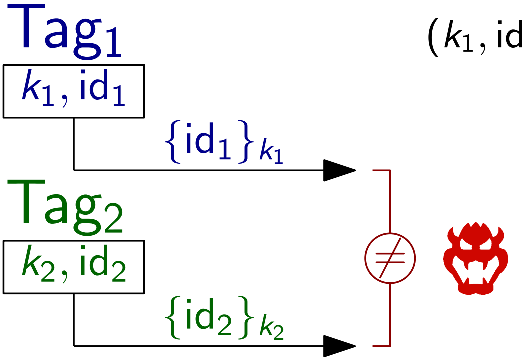


1^{st} Class: Leaks through Relations over Messages



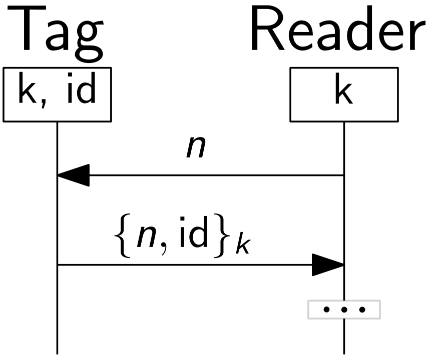
$$(k_1, \mathrm{id}_1) = (k_2, \mathrm{id}_2)$$

1st Class: Leaks through Relations over Messages

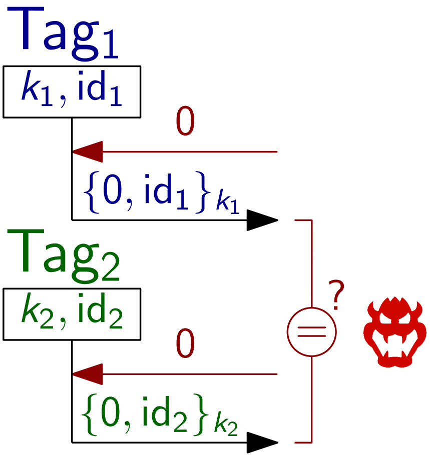


$$(k_1, \mathrm{id}_1) \neq (k_2, \mathrm{id}_2)$$

1st Class: Leaks through Relations over Messages



1^{st} Class: Leaks through Relations over Messages



1^{st} Class: Leaks through Relations over Messages

Problem

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

1^{st} Class: Leaks through Relations over Messages

Problem

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

Main idea to avoid that:

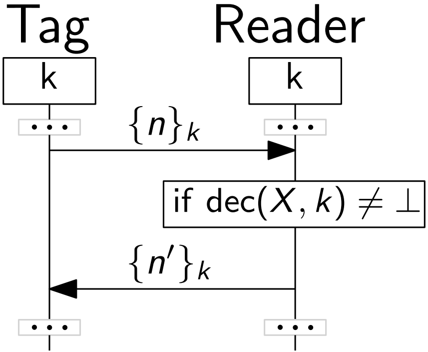
- outputs are indistinguishable from fresh nonces

$$e.g., \langle \mathrm{error}; \{u\}_k \rangle \longrightarrow \langle \mathrm{error}; n \rangle$$

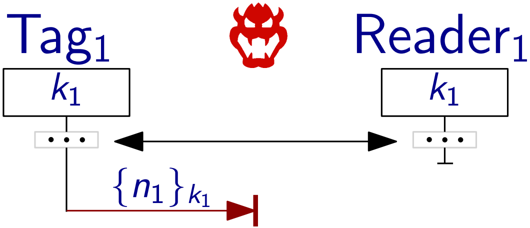
$\rightsquigarrow$ 1^{st} Condition: Frame Opacity (FO)

... formal definition in the paper

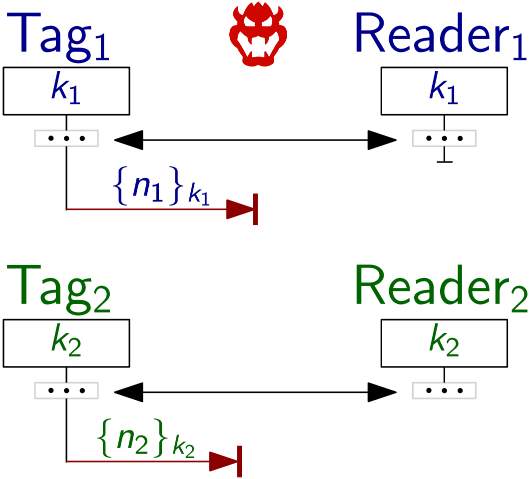
2^{nd} Class: Leaks through Conditionals' Outcomes



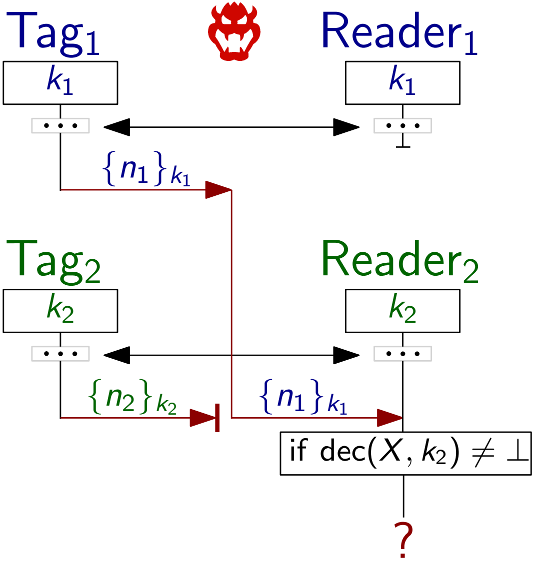
2^{nd} Class: Leaks through Conditionals' Outcomes



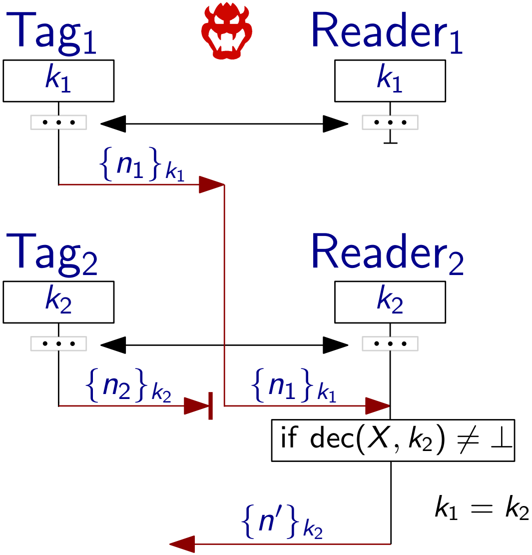
2nd Class: Leaks through Conditionals' Outcomes



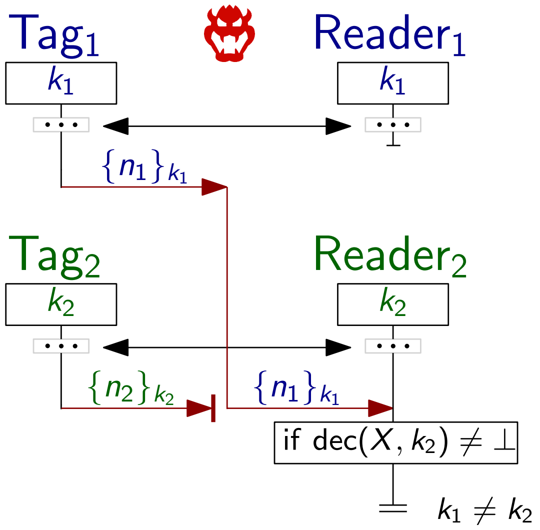
2^{nd} Class: Leaks through Conditionals' Outcomes



2^{nd} Class: Leaks through Conditionals' Outcomes



2^{nd} Class: Leaks through Conditionals' Outcomes



2^{nd} Class: Leaks through Conditionals' Outcomes

Problem

For some malicious behavior, conditionals' outcomes leak info about involved agents.

Main idea to avoid that:

- conditional evaluates positively $\iff$ attacker did not interfer

$\rightsquigarrow$ 2^{nd} Condition: Well-Authentication (WA)

... formal definition in the paper

Main Result

Theorem

For any protocol in our class:

$$\left.\begin{array}{c}\textit{frame opacity}\\ \&\\ \textit{well-authentication}\end{array}\right\} \Rightarrow \left\{\begin{array}{c}\textit{Unlinkability}\\ \&\\ \textit{Anonymity}\end{array}\right.$$

... formal statement and proof in the paper

III : Mechanization & Applications

Mechanization

Both conditions can be automatically verified using ProVerif:

- **Frame Opacity:** $\rightsquigarrow$ equivalence between messages
- **Well Authentication:** $\rightsquigarrow$ just reachability properties

Mechanization

Both conditions can be automatically verified using ProVerif:

- **Frame Opacity:** $\rightsquigarrow$ equivalence between messages
- **Well Authentication:** $\rightsquigarrow$ just reachability properties

Tool: UKano

Built on top of ProVerif that automatically checks our conditions.

Case Studies

RFID auth. protocol	Frame opacity	Well-auth.	Unlinkability
Feldhofer	✓	✓	**safe**
Hash-Lock	✓	✓	**safe**
LAK (stateless)	—	×	
Fixed LAK	✓	✓	**safe**

ePassport protocol	Frame opacity	Well-auth.	Unlinkability
BAC	✓	✓	**safe**
BAC/PA/AA	✓	✓	**safe**
PACE (faillible dec)	—	×	
PACE (missing test)	—	×	
PACE	—	×	
PACE with tags	✓	✓	**safe**

- Found automatically new proofs and new attacks using UKano

IV : Conclusion

Conclusion

Contributions

- **Theory**: 2 conditions $\Rightarrow$ unlinkability & anonymity
- **Practice**: UKano automatically verifies them
- **Applications**: new proofs & attacks on RFID protocols

Conclusion

Contributions

- **Theory**: 2 conditions $\Rightarrow$ unlinkability & anonymity
- **Practice**: UKano automatically verifies them
- **Applications**: new proofs & attacks on RFID protocols

Future Work

- Improve the method (class of protocols, other back-end)
- Seek other types of protocols (*e.g.*, e-Voting)

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