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

Published at Security&Privacy'16

Lucca Hirschi, David Baelde and Stéphanie Delaune

8th December, 2016

école
normale
supérieure
paris—saclay

Introduction

$\rightsquigarrow$ 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

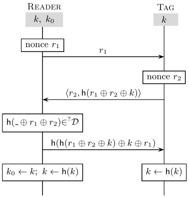
- Unlinkability (=untraceability) [ISO/IEC 15408]:

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

- Anonymity [ISO/IEC 15408]:

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

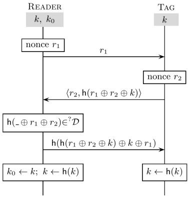
Big Picture

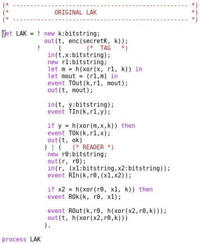


Protocol's specification

Security goal
(*e.g.*, Secrecy)

Big Picture





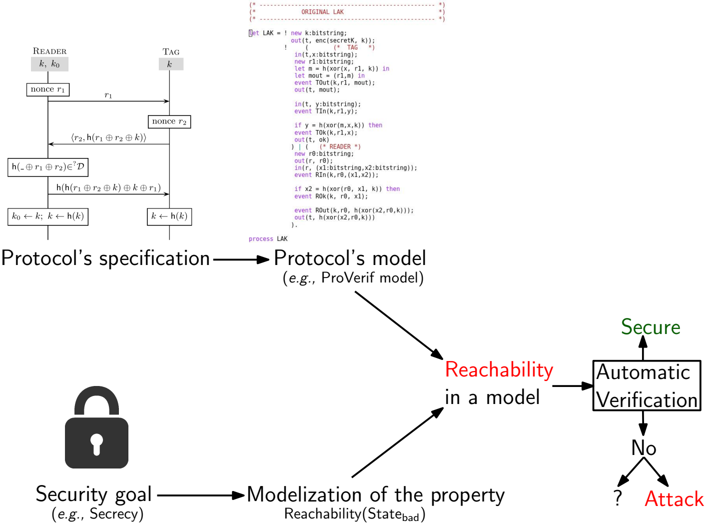
```
(* ------------------------------------------------- *)
(*              ORIGINAL LAK                          *)
(* ------------------------------------------------- *)

let LAK = ! new k:bitstring;
            out(t, enc(secretK, k));
          !    (        (*  TAG   *)
             in(t,x:bitstring);
             new r1:bitstring;
             let m = h(xor(x, r1, k)) in
             let mout = (r1,m) in
             event TOut(k,r1, mout);
             out(t, mout);

             in(t, y:bitstring);
             event TIn(k,r1,y);

             if y = h(xor(m,x,k)) then
             event TOk(k,r1,x);
             out(t, ok)
            ) | (   (* READER *)
             new r0:bitstring;
             out(r, r0);
             in(r, (x1:bitstring,x2:bitstring));
             event RIn(k,r0,(x1,x2));

             if x2 = h(xor(r0, x1, k)) then
             event ROk(k, r0, x1);

             event ROut(k,r0, h(xor(x2,r0,k)));
             out(t, h(xor(x2,r0,k)))
            ).

process LAK
```

Protocol's specification → Protocol's model
(*e.g.*, ProVerif model)

Reachability
in a model

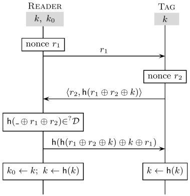
Security goal → Modelization of the property
(*e.g.*, Secrecy) Reachability($\mathsf{State}_{\mathsf{bad}}$)

Big Picture



READER
k, k0
TAG
k
nonce r1
r1
nonce r2
⟨r2, h(r1 ⊕ r2 ⊕ k)⟩
h(_ ⊕ r1 ⊕ r2) ∈? D
h(h(r1 ⊕ r2 ⊕ k) ⊕ k ⊕ r1)
k0 ← k; k ← h(k)
k ← h(k)
(* --- *)
(* ORIGINAL LAK *)
(* --- *)
let LAK = ! new k:bitstring;
out(t, enc(secretK, k));
! ((* TAG *)
in(t,x:bitstring);
new r1:bitstring;
let m = h(xor(x, r1, k)) in
let mout = (r1,m) in
event TOut(k,r1, mout);
out(t, mout);
in(t, y:bitstring);
event TIn(k,r1,y);
if y = h(xor(m,x,k)) then
event TOk(k,r1,x);
out(t, ok)
) | ((* READER *)
new r0:bitstring;
out(r, r0);
in(r, (x1:bitstring,x2:bitstring));
event RIn(k,r0,(x1,x2));
if x2 = h(xor(r0, x1, k)) then
event ROk(k, r0, x1);
event ROut(k,r0, h(xor(x2,r0,k)));
out(t, h(xor(x2,r0,k)))
).
process LAK
Protocol's specification
Protocol's model
(e.g., ProVerif model)
Reachability
in a model
Automatic
Verification
Secure
No
?
Attack
Security goal
(e.g., Secrecy)
Modelization of the property
Reachability(State_bad)

Big Picture




```
(* ---------------------------------------------------- *)
(*              ORIGINAL LAK                            *)
(* ---------------------------------------------------- *)

let LAK = ! new k:bitstring;
            out(t, enc(secretK, k));
          !     (        (*  TAG   *)
             in(t,x:bitstring);
             new r1:bitstring;
             let m = h(xor(x, r1, k)) in
             let mout = (r1,m) in
             event TOut(k,r1, mout);
             out(t, mout);

             in(t, y:bitstring);
             event TIn(k,r1,y);

             if y = h(xor(m,x,k)) then
             event TOk(k,r1,x);
             out(t, ok)
            ) | (    (* READER *)
             new r0:bitstring;
             out(r, r0);
             in(r, (x1:bitstring,x2:bitstring));
             event RIn(k,r0,(x1,x2));

             if x2 = h(xor(r0, x1, k)) then
             event ROk(k, r0, x1);

             event ROut(k,r0, h(xor(x2,r0,k)));
             out(t, h(xor(x2,r0,k)))
            ).

process LAK
```

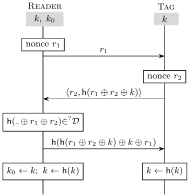
Protocol's specification → Protocol's model
(*e.g.*, ProVerif model)

Equivalence
of two models

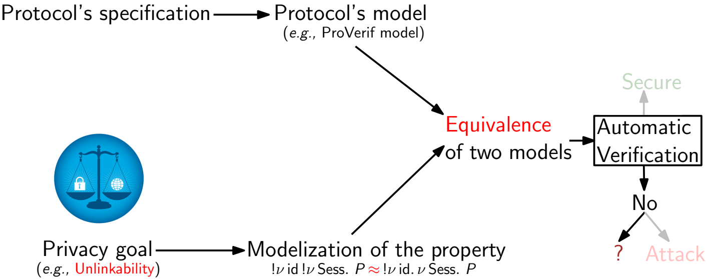
Privacy goal → Modelization of the property
(*e.g.*, Unlinkability)

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

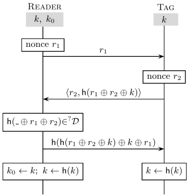
Big Picture




```
(* -------------------------------------------------- *)
(*             ORIGINAL LAK                           *)
(* -------------------------------------------------- *)

let LAK = ! new k:bitstring;
            out(t, enc(secretK, k));
           !    (       (*  TAG   *)
             in(t,x:bitstring);
             new r1:bitstring;
             let m = h(xor(x, r1, k)) in
             let mout = (r1,m) in
             event TOut(k,r1, mout);
             out(t, mout);

             in(t, y:bitstring);
             event TIn(k,r1,y);

             if y = h(xor(m,x,k)) then
             event TOk(k,r1,x);
             out(t, ok)
            ) | (   (* READER *)
             new r0:bitstring;
             out(r, r0);
             in(r, (x1:bitstring,x2:bitstring));
             event RIn(k,r0,(x1,x2));

             if x2 = h(xor(r0, x1, k)) then
             event ROk(k, r0, x1);

             event ROut(k,r0, h(xor(x2,r0,k)));
             out(t, h(xor(x2,r0,k)))
            ).

process LAK
```

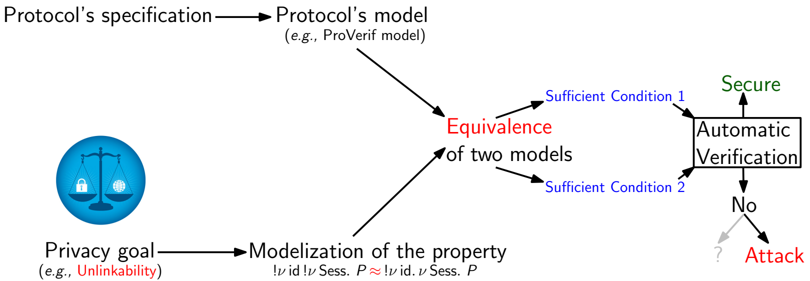


Big Picture




```
(* ------------------------------------------------ *)
(*              ORIGINAL LAK                         *)
(* ------------------------------------------------ *)

let LAK = ! new k:bitstring;
            out(t, enc(secretK, k));
          !    (       (*  TAG   *)
             in(t,x:bitstring);
             new r1:bitstring;
             let m = h(xor(x, r1, k)) in
             let mout = (r1,m) in
             event TOut(k,r1, mout);
             out(t, mout);

             in(t, y:bitstring);
             event TIn(k,r1,y);

             if y = h(xor(m,x,k)) then
             event TOk(k,r1,x);
             out(t, ok)
            ) | (   (* READER *)
             new r0:bitstring;
             out(r, r0);
             in(r, (x1:bitstring,x2:bitstring));
             event RIn(k,r0,(x1,x2));

             if x2 = h(xor(r0, x1, k)) then
             event ROk(k, r0, x1);

             event ROut(k,r0, h(xor(x2,r0,k)));
             out(t, h(xor(x2,r0,k)))
            ).

process LAK
```

Theorem: two conditions
$\Rightarrow$ Unlinkability & Anonymity



Outline

I Model & Problem

II Sufficient Conditions

III Mechanization & Applications

IV Conclusion

I : <u>Model</u> & 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

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

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 arbitrary term algebra

Applied-π - Syntax

- Process: $P, Q \quad := \quad 0$ null
  $\quad\quad | \quad \mathtt{in}(c, x).P$ input
  $\quad\quad | \quad \mathtt{out}(c, u).P$ output
  $\quad\quad | \quad$ if Test then P else Q conditional
  $\quad\quad | \quad P \mid Q$ parallel

Applied-π - Syntax

- Process: P, Q

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

Applied-π - Syntax

- Process: P, Q

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

- Frame (ϕ): the set of messages revelead to the
  $\leadsto$ intuition: attacker's ( ) knowledge

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

Applied-π - Syntax

- Process: P, Q

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

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

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

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

Applied-π - Semantics

- Recipes: are terms built using handles

  e.g., $R = \mathrm{dec}(w_1, w_2)$ $\quad$ for $\phi = \{w_1 \mapsto \mathrm{enc}(m,k), w_2 \mapsto k\}$
  $R\phi \Downarrow m$

  "How  builds messages from its knowledge"

Applied-π - Semantics

- Recipes: are terms built using handles

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

"How builds messages from its knowledge"

- Semantics of configurations:
  - Protocol's output:

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

learns outputted message

Applied-π - Semantics

- Recipes: are terms built using handles

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

  "How builds messages from its knowledge"

- Semantics of configurations:
  - Protocol's output:

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

    learns outputted message

  - Protocol's input:

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

    injects any message he can builds

Applied-π - Semantics

- Recipes: are terms built using handles

  e.g., $R = \mathrm{dec}(w_1, w_2)$ $R\phi \Downarrow m$ for $\phi = \{w_1 \mapsto \mathrm{enc}(m,k), w_2 \mapsto k\}$

  "How builds messages from its knowledge"

- Semantics of configurations:
  - Protocol's output:

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

    learns outputted message

  - Protocol's input:

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

    injects any message he can builds

  - $+$ expected rules for conditional and other constructs

$\rightsquigarrow$ controls all the network

Applied-π - Trace Equivalence

Unlinkability and Anonymity rely on trace equivalence

Applied-π - Trace Equivalence

Unlinkability and Anonymity rely on 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

Applied-π - Trace Equivalence

Unlinkability and Anonymity rely on 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

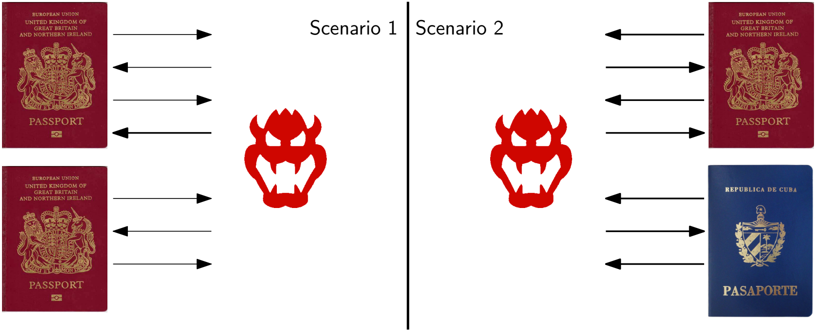
$A \sqsubseteq B$: for any $A \xrightarrow{\text{tr}} A'$ there exists $B \xrightarrow{\text{tr}} B'$ such that $\Phi(A') \sim \Phi(B')$.

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

- Intuition of $A \sqsubseteq B$:
  $\forall$ and behaviour of $(A \|)$ producing observable $\mathcal{D}$
  $\Rightarrow \exists$ behaviour of $(B \|)$ producing observable $\mathcal{D}' \sim \mathcal{D}$

I : Model & 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)

Unlinkability

- Infinitely many users
- Each playing infinitely many sessions

Unlinkability

Scenario 1

id_1 id_2 id_3

...

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

...

id_1	id_2	id_3
Session 1	Session 1	Session 1

$$\underbrace{!\nu}_{\infty \text{ users}}\, \mathrm{id}\; \underbrace{!\nu}_{\infty \text{ sessions}}\, \mathrm{Sess}.\; P \quad \approx \quad \underbrace{!\nu}_{\infty \text{ users}}\, \mathrm{id}.\underbrace{\nu}_{\text{1 session}}\, \mathrm{Sess}.\; P$$

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

The Problem

Goal

Automatic verification of

$$\underbrace{!\,\nu\,\mathsf{id}.\,(!\,\nu\,\mathsf{Sess}.P)}_{\mathcal{M}} \approx \underbrace{!\,\nu\,\mathsf{id}.\,(\nu\,\mathsf{Sess}.P)}_{\mathcal{S}}$$

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

The Problem

Goal

Automatic verification of

$$\underbrace{!\,\nu\,\mathrm{id}.\,(!\,\nu\,\mathrm{Sess}.P)}_{\mathcal{M}} \approx \underbrace{!\,\nu\,\mathrm{id}.\,(\nu\,\mathrm{Sess}.P)}_{\mathcal{S}}$$

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

Our class of protocols

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

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

The Problem

Goal

Automatic verification of

$$\underbrace{!\,\nu\,\mathrm{id}.\,(!\,\nu\,\mathrm{Sess}.P)}_{\mathcal{M}} \approx \underbrace{!\,\nu\,\mathrm{id}.\,(\nu\,\mathrm{Sess}.P)}_{\mathcal{S}}$$

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

Our class of protocols

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

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

- 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)

Existing Approaches

Goal

Automatic verification of

$$\underbrace{!\,\nu\,\mathsf{id}.\,(!\,\nu\,\mathsf{Sess}.P)}_{\mathcal{M}} \approx \underbrace{!\,\nu\,\mathsf{id}.\,(\nu\,\mathsf{Sess}.P)}_{\mathcal{S}}$$

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

Existing approaches:

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

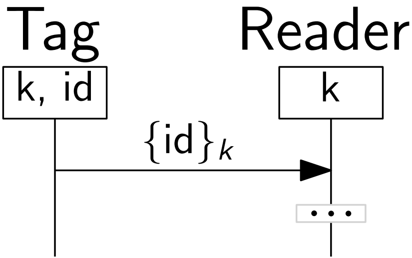
Applications:

- new proofs & attacks on RFID protocols

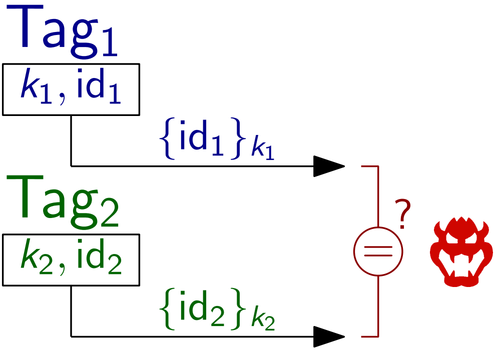
Outline

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

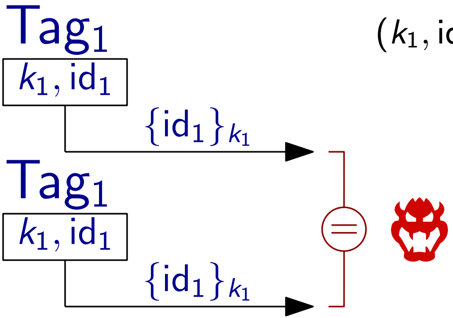
1st Class: Leaks through Relations over Messages



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

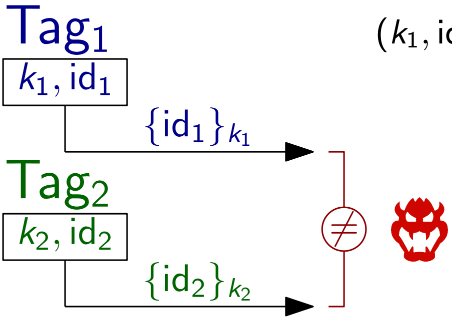


1^{st} Class: Leaks through Relations over Messages



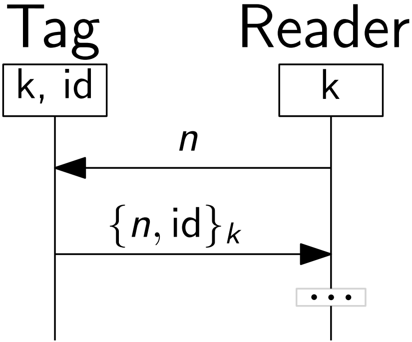
$$(k_1, \mathsf{id}_1) = (k_2, \mathsf{id}_2)$$

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

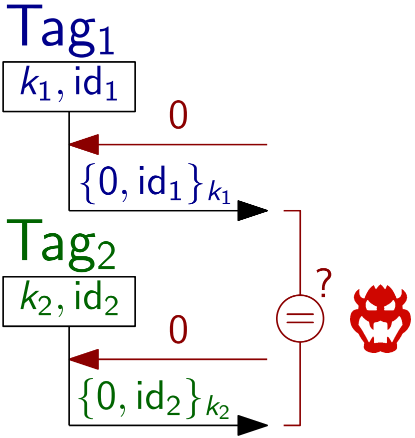


$(k_1, \mathrm{id}_1) \neq (k_2, \mathrm{id}_2)$

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



1st Class: Leaks through Relations over Messages



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

Problem

For some 's behaviors, relations over messages leak info about involved agents.

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

Problem

For some 's behaviors, relations over messages leak info about involved agents.

Main idea to avoid that:

- no relation at all
- (roughly) $\rightsquigarrow$ outputs are indistinguishable from fresh nonces

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

Problem

For some 's behaviors, relations over messages leak info about involved agents.

Main idea to avoid that:

- no relation at all
- (roughly) $\rightsquigarrow$ outputs are indistinguishable from fresh nonces

1^{st} Condition: Frame Opacity (informal)

For all $\mathcal{M} \xrightarrow{t} (\mathcal{P}; \Phi)$, we have that $\underbrace{\Phi}_{\text{"real" frame}} \sim \underbrace{[\Phi]^{\text{nonce}}}_{\text{"ideal" frame}}$.

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

Problem

For some 's behaviors, relations over messages leak info about involved agents.

Main idea to avoid that:

- no relation at all
- (roughly) $\rightsquigarrow$ outputs are indistinguishable from fresh nonces

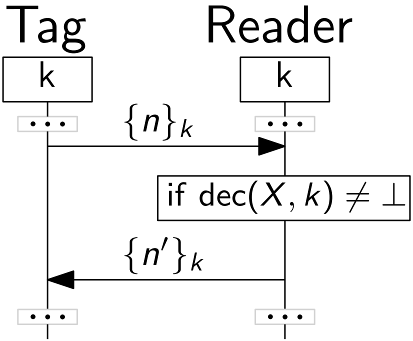
1^{st} Condition: Frame Opacity (informal)

For all $\mathcal{M} \xrightarrow{t} (\mathcal{P}; \Phi)$, we have that $\underbrace{\Phi}_{\text{"real" frame}} \sim \underbrace{[\Phi]^{\text{nonce}}}_{\text{"ideal" frame}}$.

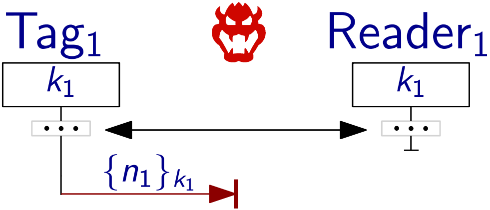
- $\Phi = \{w \mapsto \langle \mathsf{enc}(n_1, k), \mathsf{enc}(n_2, k)\rangle\}$
- $[\Phi]^{\text{nonce}} = \{w \mapsto \langle n, n' \rangle\}$ and $\Phi \sim [\Phi]^{\text{nonce}}$

$[\Phi]^{\text{nonce}}$ based on a notion of transparent function symbols

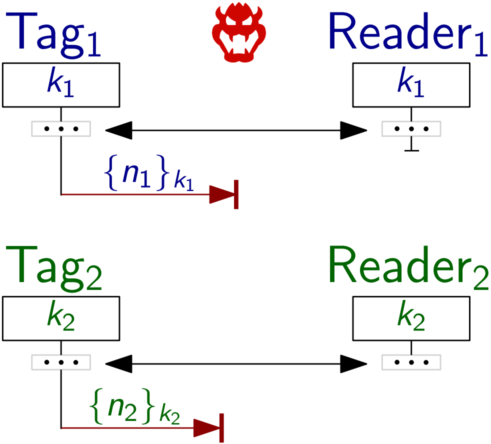
2^{nd} Class: Leaks through Conditionals' Outcomes



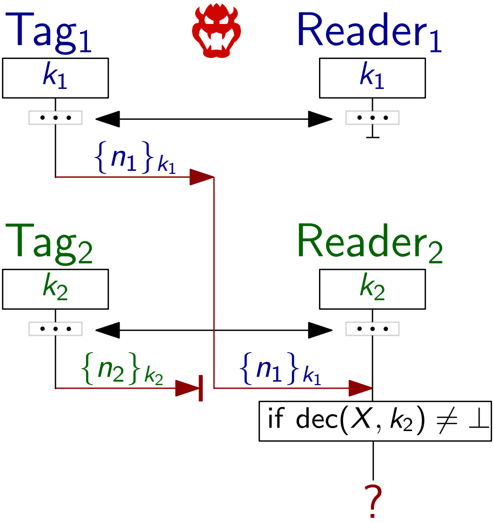
2^{nd} Class: Leaks through Conditionals' Outcomes



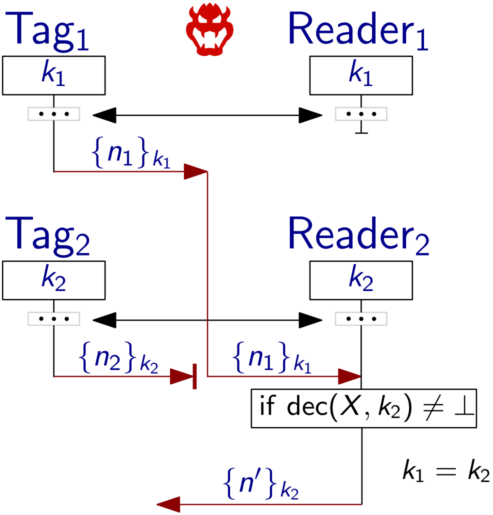
2^{nd} Class: Leaks through Conditionals' Outcomes



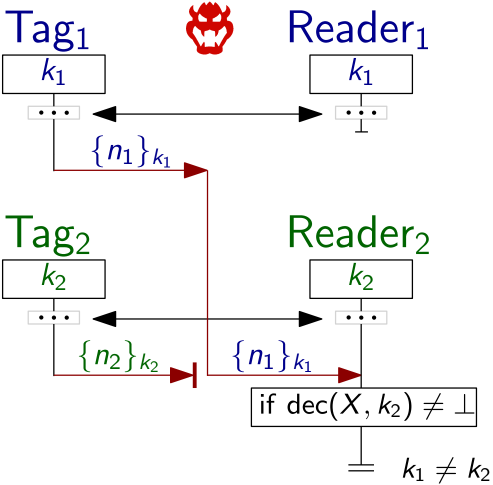
2nd Class: Leaks through Conditionals' Outcomes



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



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



2nd Class: Leaks through Conditionals' Outcomes

Problem

For some 's behaviors, conditionals' outcomes leak info about involved agents.

Main idea to avoid that:

- when plays such an attack $\Rightarrow$ conditional evaluates negatively
- $\rightsquigarrow$ conditional evaluates positively $\iff$ did not interfer

2nd Class: Leaks through Conditionals' Outcomes

Problem

For some 's behaviors, conditionals' outcomes leak info about involved agents.

Main idea to avoid that:

- when plays such an attack $\Rightarrow$ conditional evaluates negatively
- $\leadsto$ conditional evaluates positively $\iff$ did not interfer

2nd Condition: Well-Authentication (informal)

$$\forall \mathcal{M} \xrightarrow{t.\texttt{test-ok}[T(\mathrm{id},\mathrm{sess})]} (\mathcal{P};\Phi)$$

there must be a $R(\mathrm{id},\mathrm{sess}')$ such that $T(\mathrm{id},\mathrm{sess})$ and $R(\mathrm{id},\mathrm{sess}')$ were having an honest interaction.

Main Result

Theorem

For any protocol in our class:

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

III : Mechanization & Applications

Mechanization

Both conditions can be automatically verified using ProVerif:

- **Well Authentication:** $\rightsquigarrow$ just reachability properties
  - no longer equivalence property

Mechanization

Both conditions can be automatically verified using ProVerif:

- **Well Authentication:** $\rightsquigarrow$ just reachability properties
  - no longer equivalence property
- **Frame Opacity:** $\rightsquigarrow$ equivalence between messages
  - checkable with good precision via diff-equivalence and encodings

Mechanization

Both conditions can be automatically verified using ProVerif:

- **Well Authentication:** $\rightsquigarrow$ just reachability properties
  - no longer equivalence property
- **Frame Opacity:** $\rightsquigarrow$ equivalence between messages
  - checkable with good precision via diff-equivalence and encodings

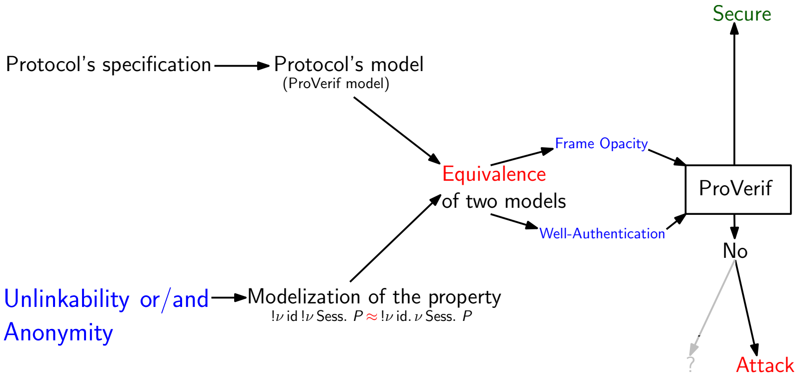
Tool: UKano

Built on top of ProVerif that automatically checks our conditions.

Sources of UKano at

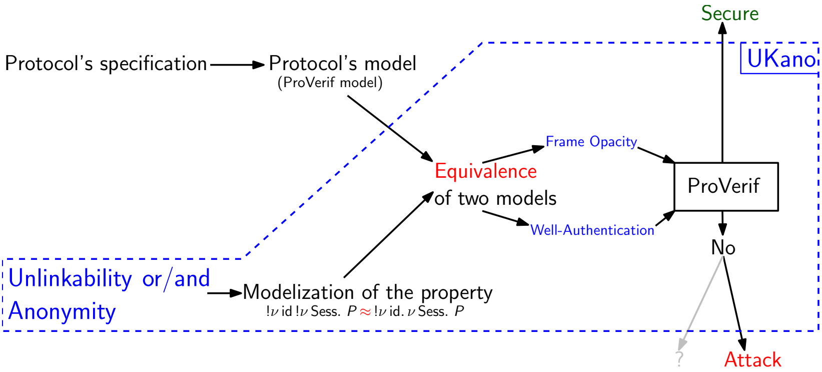
`http://projects.lsv.ens-cachan.fr/ukano/`

UKano



Secure
Protocol's specification
Protocol's model
(ProVerif model)
Frame Opacity
Equivalence
of two models
ProVerif
Well-Authentication
No
Unlinkability or/and
Anonymity
Modelization of the property
$!\nu\,\mathrm{id}\,!\nu\,\mathrm{Sess}.\ P \approx\ !\nu\,\mathrm{id}.\,\nu\,\mathrm{Sess}.\ P$
?
Attack

UKano


Secure
UKano
Protocol's specification
Protocol's model
(ProVerif model)
Frame Opacity
Equivalence
of two models
ProVerif
Well-Authentication
No
Unlinkability or/and
Anonymity
Modelization of the property
$!\nu\,\mathrm{id}\,!\nu\,\mathrm{Sess}.\ P \approx\ !\nu\,\mathrm{id}.\,\nu\,\mathrm{Sess}.\ P$
?
Attack

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

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

Conclusion

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

~~Future~~ **Current Work**:

- more precise notion of **frame opacity** (for *e.g.,* signature, ZK)
- **extend** the **class** of protocols $+$ unlinkability **scenarios**
  - $\mapsto$ DAA, ABCDH: new case studies
- for standard crypto, conditions checkable **without equivalence**

Conclusion

Future Work

Improve the method:

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

Conclusion

Future Work

Improve the method:

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

Reusing core ideas:

- reuse **methodology** for other contextes/privacy properties
  - e-voting: done for ballot secrecy with Cas Cremers this summer
  - (?) attribute-based credentials, TPM, blockchain technologies, transparent certificate authorities, . . .
- extract **guidelines** for privacy from our conditions

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

Thank you !

More : Extensions

Extensions

A more precise notion of **frame opacity** (for *e.g.,* signature, ZK)

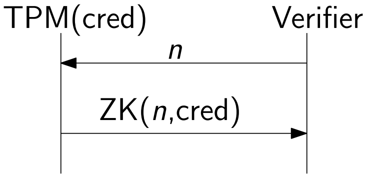
B **extend** the **class** of protocols $+$ unlinkability **scenarios**

- $\mapsto$ DAA, ABCDH: new case studies

C for standard crypto, conditions checkable **without equivalence**

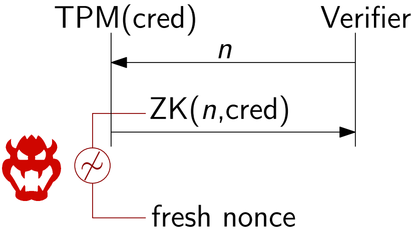
A: Improving Preciseness of Frame Opacity

Example DAA Sign:



A: Improving Preciseness of Frame Opacity

Example DAA Sign:



A: Improving Preciseness of Frame Opacity

Before

- No relation at all
- $\rightsquigarrow$ Outputs are indistinguishable from fresh nonces

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

A: Improving Preciseness of Frame Opacity

Before

- No relation at all
- $\rightsquigarrow$ Outputs are indistinguishable from fresh nonces

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

Now

- Relations must only depend on what is already observable

For any execution $\mathcal{M} \xrightarrow{t} (\mathcal{P}; \Phi)$, we have that $\Phi \sim \mathrm{ideal}(t)$.

B: Variations of Scenarios

Unlinkability

$$! \, \nu \, \mathrm{id}. \, ! \, \nu \, \mathrm{Sess}.(T \mid R) \approx \, ! \, \nu \, \mathrm{id}. \, (\nu \, \mathrm{Sess}.(T \mid R))$$

Well-Authentication

$$\forall \, (\mathcal{M}; \emptyset) \xrightarrow{t.\mathtt{then}[T(\mathrm{id},\mathrm{sess})]} (\mathcal{P}; \Phi)$$

there must be a $R(\mathrm{id}, \mathrm{sess}')$ such that $T(\mathrm{id}, \mathrm{sess})$ and $R(\mathrm{id}, \mathrm{sess}')$ were having an honest interaction.

B: Variations of Scenarios

What if R has no proper identity ?

Unlinkability

$(!\ \nu\,\mathsf{id}_T.\,!\ \nu\,\mathsf{Sess}_T.T) \mid !\nu\,\mathsf{Sess}_R.R \approx (!\ \nu\,\mathsf{id}_T.\,\nu\,\mathsf{Sess}_T.T) \mid !\nu\,\mathsf{Sess}_R.R$

Well-Authentication

$$\forall\, (\mathcal{M};\emptyset) \xrightarrow{t.\mathtt{then}[T(\mathrm{id},\mathrm{sess})]} (\mathcal{P};\Phi)$$

there must be a $R(\mathrm{sess}')$ such that $T(\mathrm{id},\mathrm{sess})$ and $R(\mathrm{sess}')$ were having an honest interaction.

B: Variations of Scenarios

What if T and R never share identity ?

Unlinkability

$$(!\ \nu\, \mathsf{id}_T.\ !\ \nu\, \mathsf{Sess}_T.T) \mid (!\ \nu\, \mathsf{id}_R.\ !\ \nu\, \mathsf{Sess}_R.R)$$
$$\approx$$
$$(!\ \nu\, \mathsf{id}_T.\nu\, \mathsf{Sess}_T.T) \mid (!\ \nu\, \mathsf{id}_R.\nu\, \mathsf{Sess}_R.R)$$

Well-Authentication

$$\forall\, (\mathcal{M};\emptyset) \xrightarrow{t.\mathtt{then}[T(\mathrm{id},\mathrm{sess})]} (\mathcal{P};\Phi)$$

there must be a $R(\mathrm{id}', \mathrm{sess}')$ such that $T(\mathrm{id}, \mathrm{sess})$ and $R(\mathrm{id}', \mathrm{sess}')$ were having an honest interaction.

B: Variations of Scenarios

What if sessions of T (or/and R) cannot be executed concurrently ?

Unlinkability

$$! \, \nu \, \mathsf{id}. \, ¡ \, \nu \, \mathsf{Sess}.(T \mid R) \approx \, ! \, \nu \, \mathsf{id}. \, (\nu \, \mathsf{Sess}.(T \mid R))$$

$$\begin{array}{lcl} !P & \sim & P \mid P \mid P \mid \ldots \\ ¡P & \sim & P; P; P; \ldots \end{array}$$

B: Variations of Scenarios

1. What if R has no proper identity ?
2. What if T and R never share identity ?
3. What if sessions of T (or R) cannot be executed concurrently ?

- Now, we can deal with all combinations of those variations
- They all are over-approximations of strong unlinkability
- new case studies: DAA Join & Sign, attribute-based authentication (abcdh used in IRMA)

Protocol	Frame opacity	Well-auth.	Unlinkability
DAA sign	✓	✓	**safe**
DAA join	✓	✓	**safe**
abcdh (irma)	✓	✓	**safe**

C: UK & Ano $\rightsquigarrow$ Reachability Problem

(More propsective: defs OK but proofs not finished yet.)

For standard crypto (enc, hash, mac, sign, data structures):

- syntactical checks $+$ secrecy checks $\Rightarrow$ Frame Opacity
- $\rightsquigarrow$ bunch of reachability checks $\Rightarrow$ UK & ANO

C: UK & Ano $\rightsquigarrow$ Reachability Problem

(More propsective: defs OK but proofs not finished yet.)

For standard crypto (enc, hash, mac, sign, data structures):

- syntactical checks + secrecy checks $\Rightarrow$ Frame Opacity
- $\rightsquigarrow$ bunch of reachability checks $\Rightarrow$ UK & ANO

Sufficient heuristic: All top-most "crypto" messages:

- (a) cannot be forged by . Checked via secrecy of keys/sub-messages.
- (b) are pairwise distinct. Checked via freshness syntactical conditions.

C: UK & Ano $\rightsquigarrow$ Reachability Problem

(More propsective: defs OK but proofs not finished yet.)

For standard crypto (enc, hash, mac, sign, data structures):

- syntactical checks $+$ secrecy checks $\Rightarrow$ Frame Opacity
- $\rightsquigarrow$ bunch of reachability checks $\Rightarrow$ UK & ANO

Sufficient heuristic: All top-most “crypto” messages:

- (a) cannot be forged by . Checked via secrecy of keys/sub-messages.
- (b) are pairwise distinct. Checked via freshness syntactical conditions.

Those messages are “black-boxed”: without any relation
$\Rightarrow$ Frame Opacity