A Logical Approach for Automated Reasoning about Privacy in Security Protocols

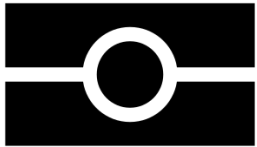
Laouen Fernet[1]

Supervisors: Sebastian Mödersheim[1]
Luca Viganò[2]

[1]DTU Compute, Technical University of Denmark, Denmark

[2]Department of Informatics, King's College London, United Kingdom

February 21, 2025



Take-aways

1. We should formally verify privacy in many applications
2. We can define privacy goals in a declarative and intuitive way using (α, β)-privacy
3. Automated verification of (α, β)-privacy is practical

(α, β)-privacy

Automated verification

Type-flaw resistance

Composability

Conclusion

(α, β)-privacy

Automated verification

Type-flaw resistance

Composability

Conclusion

(α, β)-privacy

Formula α = payload, over alphabet $\Sigma_0 \subset \Sigma$

Formula β = technical information, over alphabet Σ

Violation of privacy = β excludes some models of α

Example: unlinkability

T_1

T_2

$\alpha = T_1, T_2 \in \mathsf{Tags}$

T_1, T_2 *privacy variables* representing some concrete tag names

Example of violations:

- $\beta \models T_1 = T_2$
- $\beta \models T_1 = \mathsf{a}$
- $\beta \models T_2 \neq \mathsf{b}$
- $\ldots$

Intruder knowledge

The intruder knows:

- the protocol
- which transaction is being executed
- the payload α of information intentionally disclosed
- the concrete messages sent over the network

Example cryptographic operators

$$\begin{aligned}
\mathsf{dscrypt}(k, \mathsf{scrypt}(k, m, r)) &\approx m \\
\mathsf{dcrypt}(\mathsf{inv}(k), \mathsf{crypt}(k, m, r)) &\approx m \\
\mathsf{open}(k, \mathsf{sign}(\mathsf{inv}(k), m)) &\approx m \\
\mathsf{pubk}(\mathsf{inv}(k)) &\approx k \\
\mathsf{proj}_1(\mathsf{pair}(m_1, m_2)) &\approx m_1 \\
\mathsf{proj}_2(\mathsf{pair}(m_1, m_2)) &\approx m_2
\end{aligned}$$

We assume failure of decryption can be detected

(α, β)-privacy

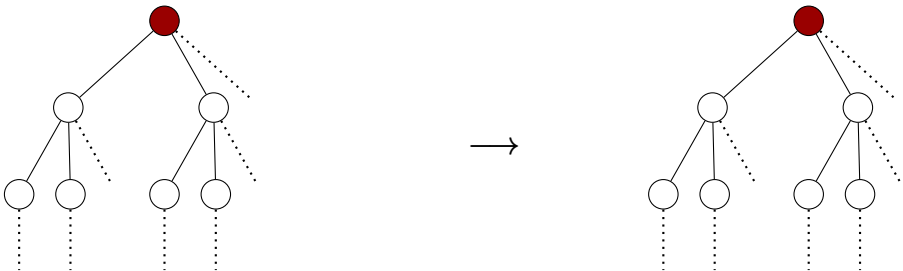
Automated verification

Type-flaw resistance

Composability

Conclusion

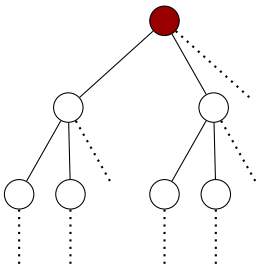
Symbolic representation

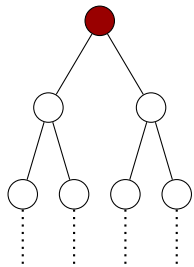


Infinite branching: intruder choices of recipes

Infinite depth: another transaction can always be executed

Symbolic representation



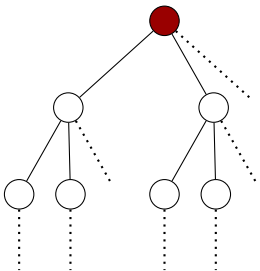


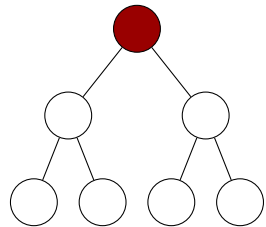
Infinite branching: intruder choices of recipes

Infinite depth: another transaction can always be executed

Finite branching: constraint solving (lazy intruder)

Symbolic representation





Infinite branching: intruder choices of recipes

Infinite depth: another transaction can always be executed

Finite branching: constraint solving (lazy intruder)

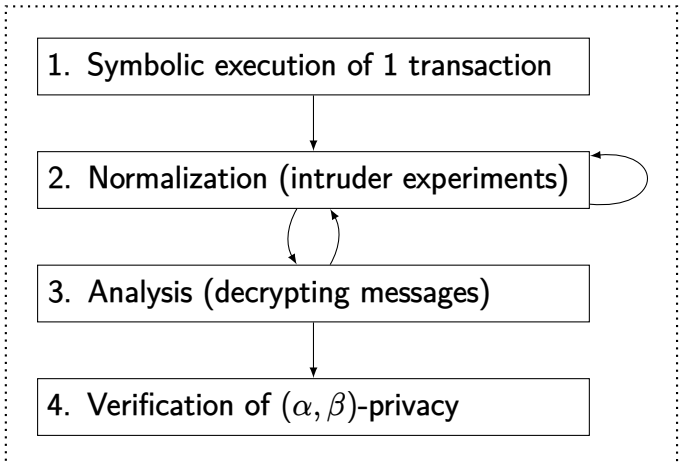
Finite depth: bound on the number of transactions executed

Summary of the procedure

Until the bound is reached:

1. Symbolic execution of a given transaction
2. Normalization via intruder experiments
3. Analysis of messages in intruder knowledge
4. Verification of (α, β)-privacy

Summary of the procedure



Counts as 1 towards the bound

1. Execution: Runex

Transaction process

$\star\ x \in \mathsf{Agent}.$
$\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.$
$\mathsf{rcv}(M).$
$\mathsf{try}\ N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in}$
$\mathsf{if}\ y = \mathsf{yes}\ \mathsf{then}$
$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), R'))$
else
$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, R'))$

State

$\alpha = \mathsf{true}$
$\mathcal{P} = \{(\star\ x \in \mathsf{Agent} \ldots, \mathsf{true}, \mathcal{A})\}$

$\mathcal{A} = -l_1 \mapsto \mathsf{inv}(\mathsf{pk}(\mathsf{i})). \cdots$

1. Execution: Runex

Transaction process

- $\star$ $x \in \text{Agent}$.
- $\star$ $y \in \{\text{yes}, \text{no}\}$.

$\text{rcv}(M)$.
try $N = \text{dcrypt}(\text{inv}(\text{pk}(\text{s})), M)$ in
if $y = \text{yes}$ then
  $\nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{pair}(\text{yes}, N), R'))$
else
  $\nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{no}, R'))$

State

$\alpha = x \in \text{Agent} \wedge y \in \{\text{yes}, \text{no}\}$
$\mathcal{P} = \{(\text{rcv}\ldots, \text{true}, \mathcal{A})\}$

$\mathcal{A} = -l_1 \mapsto \text{inv}(\text{pk}(\text{i})).\cdots$

1. Execution: Runex

Transaction process

$\star$ $x \in \text{Agent}$.
$\star$ $y \in \{\text{yes}, \text{no}\}$.
$\text{rcv}(M)$.
try $N = \text{dcrypt}(\text{inv}(\text{pk}(\text{s})), M)$ in
if $y = \text{yes}$ then
  $\nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{pair}(\text{yes}, N), R'))$
else
  $\nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{no}, R'))$

State

$\alpha = x \in \text{Agent} \wedge y \in \{\text{yes}, \text{no}\}$
$\mathcal{P} = \{(\text{try}\ldots, \text{true}, \mathcal{A})\}$

$\mathcal{A} = \cdots .+R \mapsto M$

1. Execution: Runex

Transaction process

$\star\ x \in \mathsf{Agent}.$
$\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.$
$\mathsf{rcv}(M).$
$\mathsf{try}\ N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in}$
$\mathsf{if}\ y = \mathsf{yes}\ \mathsf{then}$
$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), R'))$
else
$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, R'))$

State

$\alpha = x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\}$
$\mathcal{P} = \{(\mathsf{try} \ldots, \mathsf{true}, \mathcal{A})\}$

$\mathcal{A} = \cdots .+ R \mapsto M$

Solve $M \stackrel{?}{=} \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), M_1, M_2)$
with the lazy intruder

Lazy intruder

Constraint-solving technique: variables are instantiated on demand, only when it is actually needed

Intruder model: the intruder can send their own messages and reuse observed messages (also make guessing attacks, for privacy variables)

1. Execution: Runex

Transaction process

$\star\ x \in \mathsf{Agent}.$

$\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.$

$\mathsf{rcv}(M).$

try $N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)$ in

if $y = \mathsf{yes}$ then

$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N\ ), R'))$

else

$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, R'))$

State

$\alpha = x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\}$

$\mathcal{P} = \{(\mathsf{try} \ldots, \mathsf{true}, \mathcal{A})\}$

$\mathcal{A} = \cdots .+ R \mapsto M$

$R \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), R_1, R_2)$

$M \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), M_1, M_2)$

$N \mapsto M_1$

1. Execution: Runex

Transaction process

$\star\ x \in \text{Agent}.$
$\star\ y \in \{\text{yes}, \text{no}\}.$
$\text{rcv}(M).$
$\text{try } N = \text{dcrypt}(\text{inv}(\text{pk}(\text{s})), M) \text{ in}$
$\text{if } y = \text{yes then}$
$\quad \nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{pair}(\text{yes}, M_1), R'))$
else
$\quad \nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{no}, R'))$

State

$\alpha = x \in \text{Agent} \wedge y \in \{\text{yes}, \text{no}\}$
$\mathcal{P} = \{(\text{if} \ldots, \text{true}, \mathcal{A})\}$

$\mathcal{A} = \cdots .+R_1 \mapsto M_1 .+R_2 \mapsto M_2$

1. Execution: Runex

Transaction process

- $\star$ $x \in \text{Agent}$.
- $\star$ $y \in \{\text{yes}, \text{no}\}$.

$\text{rcv}(M)$.
try $N = \text{dcrypt}(\text{inv}(\text{pk}(\text{s})), M)$ in
if $y = \text{yes}$ then
  $\nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{pair}(\text{yes}, M_1), R'))$
else
  $\nu R'.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{no}, R'))$

State

$\alpha = x \in \text{Agent} \wedge y \in \{\text{yes}, \text{no}\}$
$\mathcal{P} = \{(\text{snd}\ldots, y = \text{yes}, \mathcal{A}),$
  $(\text{snd}\ldots, y \neq \text{yes}, \mathcal{A})\}$
$\mathcal{A} = \cdots .+R_1 \mapsto M_1.+R_2 \mapsto M_2$

1. Execution: Runex

Transaction process

- $\star$ $x \in \mathsf{Agent}$.
- $\star$ $y \in \{\mathsf{yes}, \mathsf{no}\}$.

$\mathsf{rcv}(M)$.
try $N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)$ in
if $y = \mathsf{yes}$ then
  $\nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, M_1), R'))$
else
  $\nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, R'))$

State

$\alpha = x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\}$
$\mathcal{P} = \{(0, y = \mathsf{yes}, \mathcal{A}_1),$
  $(0, y \neq \mathsf{yes}, \mathcal{A}_2)\}$
$\mathcal{A}_1 = \cdots .-l_2 \mapsto \mathsf{crypt}(\ldots \mathsf{yes} \ldots)$
$\mathcal{A}_2 = \cdots .-l_2 \mapsto \mathsf{crypt}(\ldots \mathsf{no} \ldots)$

All processes are nil, we are in a *reachable* state

2. Normalization via intruder experiments

Suppose encryption is not randomized:

$$\begin{aligned}
\alpha &= x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\} \\
\mathcal{P} &= \{(0, y = \mathsf{yes}, \mathcal{A}_1), (0, y \neq \mathsf{yes}, \mathcal{A}_2)\} \\
\mathcal{A}_1 &= \cdots .{-}I_2 \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, M_1)) \\
\mathcal{A}_2 &= \cdots .{-}I_2 \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no})
\end{aligned}$$

The intruder can make a guessing attack:

$I_2 \stackrel{?}{\simeq} \mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no}) \quad I_2 \stackrel{?}{\simeq} \mathsf{crypt}(\mathsf{pk}(\mathsf{b}), \mathsf{no}) \quad \ldots$

3. Analysis of messages in intruder knowledge

$$\alpha = x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\}$$
$$\mathcal{P} = \{(0, y = \mathsf{yes}, \mathcal{A}_1), (0, y \neq \mathsf{yes}, \mathcal{A}_2)\}$$
$$\mathcal{A}_1 = -l_1 \mapsto \mathsf{inv}(\mathsf{pk}(\mathsf{i})). \cdots .-l_2 \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, M_1), \mathsf{r})$$
$$\mathcal{A}_2 = -l_1 \mapsto \mathsf{inv}(\mathsf{pk}(\mathsf{i})). \cdots .-l_2 \mapsto \mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, \mathsf{r})$$

$$\mathsf{dcrypt}(l_1, l_2) \stackrel{?}{\simeq} \mathsf{fail}$$

2. Normalization via intruder experiments

If $\mathsf{dcrypt}(l_1, l_2) \not\approx \mathsf{fail}$, i.e., $x = \mathsf{i}$:

$$\begin{aligned}
\alpha &= x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\} \\
\mathcal{P} &= \{(0, y = \mathsf{yes} \wedge x = \mathsf{i}, \mathcal{A}_1), (0, y \neq \mathsf{yes} \wedge x = \mathsf{i}, \mathcal{A}_2)\} \\
\mathcal{A}_1 &= \cdots .-l_3 \mapsto \mathsf{pair}(\mathsf{yes}, M_1).-l_4 \mapsto \mathsf{inv}(\mathsf{pk}(x)) \\
\mathcal{A}_2 &= \cdots .-l_3 \mapsto \mathsf{no}.-l_4 \mapsto \mathsf{inv}(\mathsf{pk}(x))
\end{aligned}$$

New experiment: $l_3 \stackrel{?}{\simeq} \mathsf{no}$

4. Verification of (α, β)-privacy

If $l_3 \simeq \mathsf{no}$:

$$\alpha = x \in \mathsf{Agent} \wedge y \in \{\mathsf{yes}, \mathsf{no}\}$$
$$\mathcal{P} = \{(0, y \neq \mathsf{yes} \wedge x = \mathsf{i}, \mathcal{A}_2)\}$$
$$\mathcal{A}_2 = \cdots .-l_3 \mapsto \mathsf{no}.-l_4 \mapsto \mathsf{inv}(\mathsf{pk}(x)).-l_5 \mapsto \mathsf{pk}(x)$$

Intruder learns $y \neq \mathsf{yes} \wedge x = \mathsf{i}$ (more than allowed by α)
$\rightarrow$ violation of privacy

Release: selective disclosure of information

$\star\ x \in \mathsf{Agent}.$

$\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.$

$\mathsf{rcv}(M).$

$\mathsf{try}\ N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in}$

$\mathsf{if}\ x \in \mathsf{Dishonest}\ \mathsf{then}\ \star\ x = \gamma(x) \wedge y = \gamma(y)$

$\mathsf{else}\ \star\ x \in \mathsf{Honest}.$

$\mathsf{if}\ y = \mathsf{yes}\ \mathsf{then}$

$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), R')).$

else

$\quad \nu R'.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, R'))$

Now (α, β)-privacy holds

Case studies

- Basic Hash: provides unlinkability but not forward privacy
- OSK: has linkability flaws
- BAC: has linkability flaws in some implementations
- Private Authentication: several variants with privacy goals beyond unlinkability

(α, β)-privacy

Automated verification

Type-flaw resistance

Composability

Conclusion

Type-flaws

A participant in the protocol expects a message of some type, but receives a message of a different type

E.g., expect a nonce but receive an encrypted message

Type annotations

$\star\ x \in \mathsf{Agent}.$

$\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.$

$\mathsf{rcv}(M : \mathsf{crypt}(\mathsf{pk}(\mathsf{agent}), \mathsf{nonce}, \mathsf{nonce})).$

$\mathsf{try}\ N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in}$

$\mathsf{if}\ y = \mathsf{yes}\ \mathsf{then}$

$\quad \nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), R'))$

else

$\quad \nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, R'))$

Making sure the outgoing messages have the same type

$\star\ x \in \mathsf{Agent}.$

$\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.$

$\mathsf{rcv}(M : \mathsf{crypt}(\mathsf{pk}(\mathsf{agent}), \mathsf{nonce}, \mathsf{nonce})).$

$\mathsf{try}\ N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in}$

$\mathsf{if}\ y = \mathsf{yes}\ \mathsf{then}$

$\quad \nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), R'))$

else

$\quad \nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{no}, \mathsf{n}_0), R'))$

From explicit destructors to pattern matching

$\star$ $x \in \mathsf{Agent}$.

$\star$ $y \in \{\mathsf{yes}, \mathsf{no}\}$.

$\mathsf{rcv}(\mathsf{crypt}(\mathsf{pk}(S : \mathsf{agent}), N : \mathsf{nonce}, R : \mathsf{nonce}))$.

if $S = \mathsf{s}$ then

  if $y = \mathsf{yes}$ then

    $\nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), R'))$

  else

    $\nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{no}, \mathsf{n}_0), R'))$

Message patterns = all messages in the transactions after removing destructors

$\mathsf{Agent} \cup \{x, y, \mathsf{yes}, \mathsf{no}, \mathsf{crypt}(\mathsf{pk}(S), N, R), S, \mathsf{s}, R', \mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), R'),$
$\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{no}, \mathsf{n}_0), R')\}$

Type-flaw resistance

SMP = all message patterns in protocol and their well-typed instances (including subterms and key terms)

$$\forall s, t \in SMP \setminus \mathcal{V}.\ (s, t) \text{ unifiable } \longrightarrow type(s) = type(t)$$

Runex does not achieve type-flaw resistance

- $\mathsf{crypt}(\mathsf{pk}(S), N, R) \in \mathit{SMP} \setminus \mathcal{V}$
- $\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{pair}(\mathsf{yes}, \mathsf{n}_2), \mathsf{r}) \in \mathit{SMP} \setminus \mathcal{V}$
- These are unifiable, but
  $\mathit{type}(\mathsf{crypt}(\mathsf{pk}(S), N, R)) \neq \mathit{type}(\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{pair}(\mathsf{yes}, \mathsf{n}_2), \mathsf{r}))$
- Because
  $\mathit{type}(N) = \mathsf{nonce} \neq \mathsf{pair}(\mathit{decision}, \mathsf{nonce}) = \mathit{type}(\mathsf{pair}(\mathsf{yes}, \mathsf{n}_2))$

Achieving type-flaw resistance with formats

$\star\ x \in \mathsf{Agent}.$

$\star\ y \in \{\mathsf{yes}, \mathsf{no}\}.$

$\mathsf{rcv}(\mathsf{crypt}(\mathsf{pk}(\mathsf{s}), \mathsf{f}_1(N : \mathsf{nonce}), R : \mathsf{nonce})).$

$\mathsf{if}\ y = \mathsf{yes}\ \mathsf{then}$

$\quad \nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{f}_2(\mathsf{yes}, N), R'))$

else

$\quad \nu R' : \mathsf{nonce}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{f}_2(\mathsf{no}, \mathsf{n}_0), R'))$

Typing result

For type-flaw resistant protocols: **“if there is an attack, then there is a well-typed one”**

Proof argument: the lazy intruder, intruder experiments and analysis are only doing well-typed instantiations

The result holds without any bound on the number of transitions

Case studies

- Basic Hash: type-flaw resistant
- OSK: out of scope
- BAC and Private Authentication: type-flaw resistant (after some adaptations to the models)

(α, β)-privacy

Automated verification

Type-flaw resistance

Composability

Conclusion

Composition and modular verification

$$Spec = Spec_1 \parallel Spec_2$$

Can we derive the security of $Spec$ by verifying $Spec_1$ and $Spec_2$ in isolation?

Needham-Schroeder-Lowe with lookup for public keys

Role *Initiator* :

$\star$ $x_A \in$ Honest.

$\star$ $x_B \in$ Agent.

$PKB := lookup(x_A, x_B)$

;

νN_A : nonce, R : nonce.

snd(crypt(PKB, $f_1(N_A, x_A)$, R)).

...

Role *Responder* :

rcv(crypt(pk(B : agent),

$f_1(N_A$: nonce, A : agent), _ : nonce)).

if $B \notin$ Honest then

stop

;

$PKA := lookup(B, A)$

...

Needham-Schroeder-Lowe with lookup for public keys

Procedure $\mathit{lookup}(A : \mathsf{agent}, B : \mathsf{agent}) :$

$\nu N : \mathsf{nonce}, R : \mathsf{nonce}.$

$\mathsf{snd}(\mathsf{scrypt}(\mathsf{sk}(A, \mathsf{s}), \mathsf{req}(B, N), R))$

;

$\mathsf{rcv}(\mathsf{scrypt}(\mathsf{sk}(A, \mathsf{s}), \mathsf{resp}(B, PKB : \mathsf{pk}(\mathsf{agent}), N), \_ : \mathsf{nonce})).$

$\mathsf{assert}(PKB = \mathsf{pk}(B)).$

$\mathsf{return}(\mathsf{pk}(B))$

Needham-Schroeder-Lowe with lookup for public keys

Role *Server* :

$\mathsf{rcv}(\mathsf{scrypt}(\mathsf{sk}(A : \mathsf{agent}, \mathsf{s}), \mathsf{req}(B : \mathsf{agent}, N : \mathsf{nonce}), \_ : \mathsf{nonce})).$

$\nu R : \mathsf{nonce}.$

$\mathsf{snd}(\mathsf{scrypt}(\mathsf{sk}(A, \mathsf{s}), \mathsf{resp}(B, \mathsf{pk}(B), N), R))$

Interface between protocols

Specify what each component must at least know about the other components

- All choices of privacy variables and releases
- Procedure calls between components
- Shared messages

Composability requirements

- Operations on *shared* messages, memory cells and procedures must be part of the interface
- All shared messages must be public or declared as secrets (secrets may be declassified)
- ...

Compositionality result

For composable protocols: **"if each component is secure, then their composition is secure"**

Proof argument: any attack trace on the composition can be mapped to an attack trace on one component

secure $=(\alpha, \beta)$-privacy goals $+$ all assertions hold $+$ no secret leaked

Examples

- NSL with lookup
- Basic model of TLS handshake

(α, β)-privacy

Automated verification

Type-flaw resistance

Composability

Conclusion

Main contributions

1. A decision procedure for (α, β)-privacy for a bounded number of transitions, with proofs of soundness and completeness
2. Prototype tool called *noname* implementing the procedure
3. For type-flaw resistant protocols: "if there is an attack, then there is a well-typed one"
4. For composable protocols: "if each component is secure, then their composition is secure"

References

[1] L. Fernet, S. Mödersheim, and L. Viganò. "A Decision Procedure for Alpha-Beta Privacy for a Bounded Number of Transitions". In: *CSF 2024*. Extended version at https://people.compute.dtu.dk/samo/alphabeta. IEEE, 2024, pp. 159–174. DOI: 10.1109/CSF61375.2024.00011.

[2] L. Fernet and S. Mödersheim. *noname: Formal Verification of (Alpha, Beta)-Privacy in Security Protocols*. Version 0.3. Nov. 2024. DOI: 10.5281/zenodo.14198336. URL: https://doi.org/10.5281/zenodo.14198336.

[3] L. Fernet and S. Mödersheim. "Private Authentication with Alpha-Beta-Privacy". In: *OID 2023*. LNI. GI, 2023. DOI: 10.18420/OID2023_05. URL: https://people.compute.dtu.dk/samo/alphabeta.

[4] L. Fernet, S. Mödersheim, and L. Viganò. "A decision procedure and typing result for alpha-beta privacy". In: *J. Comput. Secur.* (2024). Submitted for review.

[5] L. Fernet, S. Mödersheim, and L. Viganò. *A Compositionality Result for Alpha-Beta Privacy*. Tech. rep. DTU, Denmark; King's, United Kingdom, 2024. URL: https://people.compute.dtu.dk/samo.

Ideas for future work

- Implement checks for type-flaw resistance and composability requirements
- Support more algebraic theories (Diffie-Hellman)
- Procedure for unbounded case?
- Investigate relation to approaches based on process equivalences
- Design a procedure for probabilistic (α, β)-privacy

Take-aways

1. We should formally verify privacy in many applications
2. We can define privacy goals in a declarative and intuitive way using (α, β)-privacy
3. Automated verification of (α, β)-privacy is practical