A Decision Procedure for Alpha-Beta Privacy for a Bounded Number of Transitions

Laouen Fernet[1,*] Sebastian Mödersheim[1] Luca Viganò[2]

[1]DTU Compute, Kongens Lyngby, Danmark [*]`lpkf@dtu.dk`

[2]KCL Informatics, London, United Kingdom

July 9, 2024

Motivation

Declarative specification of privacy goals based on (α, β)-privacy semantics

Automation of verification with bounded number of transactions and unbounded intruder

(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:

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

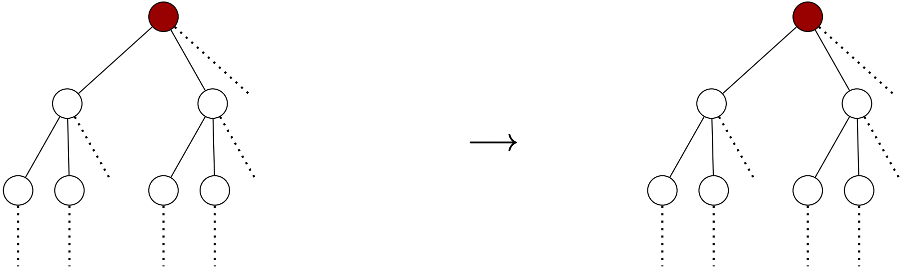
Intruder knowledge

The intruder knows:

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

α is specified as part of the protocol, while β is defined through a *simulation* of all possible protocol executions: we keep track of all the possibilities that could have happened and the intruder knows the concrete execution is one of these possibilities

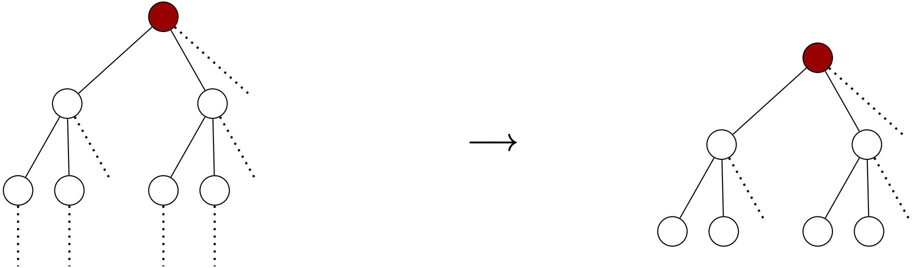
Symbolic representation



Infinite depth: another transaction can always be executed

Infinite branching: intruder choices of recipes

Symbolic representation

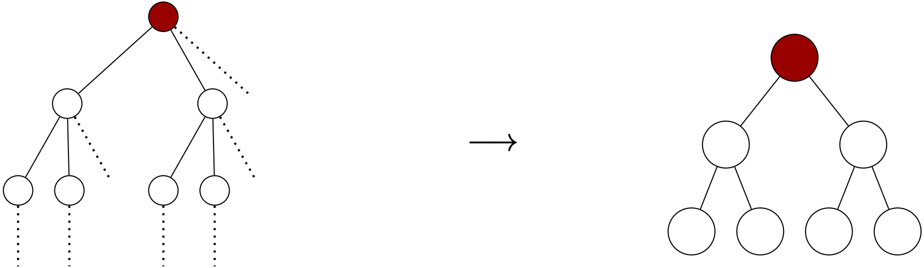


Infinite depth: another transaction can always be executed

Infinite branching: intruder choices of recipes

Finite depth: bound on the number of transactions executed

Symbolic representation



Infinite depth: another transaction can always be executed

Infinite branching: intruder choices of recipes

Finite depth: bound on the number of transactions executed

Finite branching: constraint solving (lazy intruder)

Example cryptographic operators

$$
\begin{array}{ll}
\mathsf{dcrypt}(s_1, s_2) \approx t & \text{if } s_1 \approx \mathsf{inv}(k) \text{ and } s_2 \approx \mathsf{crypt}(k, t, r) \\
\mathsf{pubk}(s) \approx k & \text{if } s \approx \mathsf{inv}(k) \\
\mathsf{proj}_1(s) \approx t_1 & \text{if } s \approx \mathsf{pair}(t_1, t_2) \\
\mathsf{proj}_2(s) \approx t_2 & \text{if } s \approx \mathsf{pair}(t_1, t_2) \\
\text{and } \ldots \approx \mathsf{fail} & \text{otherwise}
\end{array}
$$

Outline of the procedure

Until the bound is reached:

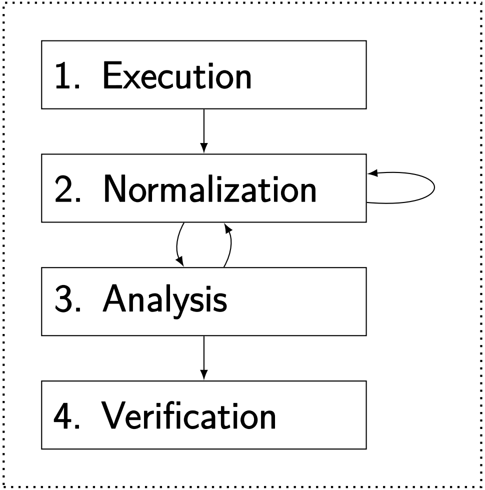
1. Execution of some transaction

2. Normalization via intruder experiments

3. Analysis of messages in intruder knowledge

4. Verification of (α, β)-privacy

Outline of the procedure



Counts as 1 towards the bound

Outline of the procedure

Until the bound is reached:

1. Execution of some transaction
2. Normalization via intruder experiments
3. Analysis of messages in intruder knowledge
4. Verification of (α, β)-privacy

1. Execution: Runex

Transaction process

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

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

$\mathrm{rcv}(M).$

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

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

$\quad \nu \mathrm{r}.\ \mathrm{snd}(\mathrm{crypt}(\mathrm{pk}(x), \mathrm{pair}(\mathrm{yes}, N), \mathrm{r}))$

else

$\quad \nu \mathrm{r}.\ \mathrm{snd}(\mathrm{crypt}(\mathrm{pk}(x), \mathrm{no}, \mathrm{r}))$

State

$\alpha = \mathrm{true}$

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

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

1. Execution: Runex

Transaction process

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

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

$\text{rcv}(M).$

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

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

$\quad \nu\text{r}.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{pair}(\text{yes}, N), \text{r}))$

else

$\quad \nu\text{r}.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{no}, \text{r}))$

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

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

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

$\mathsf{rcv}(M).$

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

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

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

else

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

State

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

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

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

$\mathcal{A}$ is a *FLIC*: a frame where we also record the recipes and messages sent by the intruder

1. Execution: Runex

Transaction process

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

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

$\mathsf{rcv}(M).$

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

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

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

else

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

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 $M \stackrel{?}{=} \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), M_1, M_2)$

1. Execution: Runex

The constraint of the example is expressed with the FLIC:

$$\mathcal{A} = \cdots .+R \mapsto \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), M_1, M_2)$$

Lazy intruder constraint solving: one solution is to *compose* the message with recipe $R = \mathsf{crypt}(\mathsf{pk}(\mathsf{s}), R_1, R_2)$
At this point it does not matter what R_1, R_2 (resp. M_1, M_2) are so they are left as variables.

The solutions are computed in a single FLIC but the solutions are applied to every possibility

1. Execution: Runex

Transaction process

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

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

$\mathsf{rcv}(M)$.

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

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

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

else

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

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

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

State

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

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

1. Execution: Runex

Transaction process

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

$\text{rcv}(M).$

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

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

$\quad \nu\text{r. snd}(\text{crypt}(\text{pk}(x), \text{pair}(\text{yes}, M_1), \text{r}))$

else

$\quad \nu\text{r. snd}(\text{crypt}(\text{pk}(x), \text{no}, \text{r}))$

State

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

$\mathcal{P} = \{(\text{snd} \ldots, y = \text{yes}, \mathcal{A}),$
$\quad (\text{snd} \ldots, y \neq \text{yes}, \mathcal{A})\}$

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

1. Execution: Runex

Transaction process

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

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

$\text{rcv}(M).$

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

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

$\quad \nu r.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{pair}(\text{yes}, M_1), \text{r}))$

else

$\quad \nu r.\ \text{snd}(\text{crypt}(\text{pk}(x), \text{no}, \text{r}))$

State

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

$\mathcal{P} = \{(0, y = \text{yes}, \mathcal{A}_1),$
$\quad (0, y \neq \text{yes}, \mathcal{A}_2)\}$

$\mathcal{A}_1 = \cdots .{-}l_2 \mapsto \text{crypt}(\ldots \text{yes} \ldots)$

$\mathcal{A}_2 = \cdots .{-}l_2 \mapsto \text{crypt}(\ldots \text{no} \ldots)$

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

Outline of the procedure

Until the bound is reached:

1. Execution of some transaction
2. Normalization via intruder experiments
3. Analysis of messages in intruder knowledge
4. Verification of (α, β)-privacy

2. Normalization via intruder experiments

If encryption is not randomized:

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

Intruder can compare recipes: *guessing attack!*
l_2 vs $\mathsf{crypt}(\mathsf{pk}(\mathsf{a}), \mathsf{no}, \mathsf{r}_{pub})$ l_2 vs $\mathsf{crypt}(\mathsf{pk}(\mathsf{b}), \mathsf{no}, \mathsf{r}_{pub})$ ...

The intruder learns either the true value of x (and then also $y = \mathsf{no}$) or $y = \mathsf{yes}$

Outline of the procedure

Until the bound is reached:

1. Execution of some transaction

2. Normalization via intruder experiments

3. Analysis of messages in intruder knowledge

4. Verification of (α, β)-privacy

3. Analysis of messages in intruder knowledge

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

Intruder can try $\mathsf{dcrypt}(l_1, l_2)$, failure or success is observable

Decryption succeeds iff $x = \mathsf{i}$

2. Normalization via intruder experiments

Case where decryption really succeeded:

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

Intruder can compare l_3 vs no and learn which possibility is the concrete one

Outline of the procedure

Until the bound is reached:

1. Execution of some transaction
2. Normalization via intruder experiments
3. Analysis of messages in intruder knowledge
4. Verification of (α, β)-privacy

4. Verification of (α, β)-privacy

Case where l_3 and no are equivalent:

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

Intruder knows that $y \neq \mathsf{yes} \wedge x = \mathsf{i}$, which is more than what is allowed

I.e., β excludes some models of α (e.g., $\mathcal{I} = [x \mapsto \mathsf{i}, y \mapsto \mathsf{yes}]$)

Release

Selective disclosure of information by augmenting α

$$
\begin{array}{l}
\star\ \ x \in \mathsf{Agent}. \\
\star\ \ y \in \{\mathsf{yes}, \mathsf{no}\}. \\
\mathsf{rcv}(M). \\
\mathsf{try}\ \ N = \mathsf{dcrypt}(\mathsf{inv}(\mathsf{pk}(\mathsf{s})), M)\ \mathsf{in} \\
\mathsf{if}\ x \in \mathsf{Dishonest}\ \mathsf{then}\ \ \star\ \ x = \gamma(x) \wedge y = \gamma(y). \\
\mathsf{if}\ \ y = \mathsf{yes}\ \mathsf{then} \\
\quad \nu \mathsf{r}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{pair}(\mathsf{yes}, N), \mathsf{r})). \\
\mathsf{else} \\
\quad \nu \mathsf{r}.\ \mathsf{snd}(\mathsf{crypt}(\mathsf{pk}(x), \mathsf{no}, \mathsf{r}))
\end{array}
$$

Conclusion

1. We have contributed a decision procedure for (α, β)-privacy for a bounded number of transitions, with proofs of soundness and completeness in extended version
2. We have implemented the procedure in a tool called *noname*[1]
   - Library of models for several existing protocols
   - Automatic or manual exploration of the transition system
   - Explainable privacy violations

[1] `https://people.compute.dtu.dk/lpkf`