Modelling and verifying privacy-type properties of electroning voting protocols

Stéphanie Delaune

School of Computer Science, University of Birmingham, UK
Electronic voting

Advantages:

- Convenient,
- Efficient facilities for tallying votes.

Drawbacks:

- Risk of large-scale and undetectable fraud,
- Such protocols are extremely error-prone.

"A 15-year-old in a garage could manufacture smart cards and sell them on the Internet that would allow for multiple votes"

Avi Rubin

Possible issue: formal methods
abstract analysis of the protocol against formally-stated properties
Electronic voting

Advantages:

- Convenient,
- Efficient facilities for tallying votes.

Drawbacks:

- Risk of large-scale and undetectable fraud,
- Such protocols are extremely error-prone.

"A 15-year-old in a garage could manufacture smart cards and sell them on the Internet that would allow for multiple votes"

Avi Rubin

Possible issue: formal methods
abstract analysis of the protocol against formally-stated properties
Example: Fujioka *et al.* protocol (1992)

First Phase:
the voter gets a “token” from the administrator.

1. $V \rightarrow A : V, \text{sign}(\text{blind}(\text{commit}(\text{vote}, r), b), V)$
2. $A \rightarrow V : \text{sign}(\text{blind}(\text{commit}(\text{vote}, r), b), A)$

→ to ensure privacy, blind signatures are used

Voting phase:

3. $V \rightarrow C : \text{sign}(\text{commit}(\text{vote}, r), A)$
4. $C \rightarrow : l, \text{sign}(\text{commit}(\text{vote}, r), A)$

Counting phase:

5. $V \rightarrow C : l, r$
6. $C$ publishes the outcome of the vote

→ to ensure privacy, anonymous channel are used at step 3 and 5
Example: Fujioka et al. protocol (1992)

First Phase:
the voter gets a “token” from the administrator.

1. \( V \rightarrow A : V, \text{sign}(\text{blind}(\text{commit}(\text{vote}, r), b), V) \)
2. \( A \rightarrow V : \text{sign}(\text{blind}(\text{commit}(\text{vote}, r), b), A) \)

→ to ensure privacy, \text{blind} signatures are used

Voting phase:

3. \( V \rightarrow C : \text{sign}(\text{commit}(\text{vote}, r), A) \)
4. \( C \rightarrow : l, \text{sign}(\text{commit}(\text{vote}, r), A) \)

Counting phase:

5. \( V \rightarrow C : l, r \)
6. \( C \) publishes the outcome of the vote

→ to ensure privacy, anonymous channel are used at step 3 and 5
Example: Fujioka et al. protocol (1992)

First Phase:
the voter gets a “token” from the administrator.

1. \( V \rightarrow A : \ V, \text{sign}(\text{blind}(\text{commit}(\text{vote}, r), b), V) \)
2. \( A \rightarrow V : \ \text{sign}(\text{blind}(\text{commit}(\text{vote}, r), b), A) \)

→ to ensure privacy, blind signatures are used

Voting phase:

3. \( V \rightarrow C : \ \text{sign}(\text{commit}(\text{vote}, r), A) \)
4. \( C \rightarrow : \ l, \text{sign}(\text{commit}(\text{vote}, r), A) \)

Counting phase:

5. \( V \rightarrow C : \ l, r \)
6. \( C \) publishes the outcome of the vote

→ to ensure privacy, anonymous channel are used at step 3 and 5
Privacy-type security properties

**Privacy:** the fact that a particular voted in a particular way is not revealed to anyone

**Receipt-freeness:** a voter cannot prove that she voted in a certain way (this is important to protect voters from coercion)

**Coercion-resistance:** same as receipt-freeness, but the coercer interacts with the voter during the protocol, (*e.g.* by preparing messages)
Summary

Observations:

- Definitions of security properties are often **insufficiently precise**
- No clear distinction between receipt-freeness and coercion-resistance

Goal:

1. Propose “formal methods” definitions of privacy-type properties,
2. Design automatic procedures to verify them.

Difficulties:

- equivalence based-security properties are harder than reachability properties (*e.g.* secrecy, authentication),
- electronic voting protocols are often more complex than authentication protocols,
- less classical cryptographic primitives (*e.g.* blind signature).
Observations:
- Definitions of security properties are often **insufficiently precise**
- No clear distinction between receipt-freeness and coercion-resistance

Goal:
1. Propose “**formal methods**” definitions of privacy-type properties,
2. Design **automatic** procedures to verify them.

Difficulties:
- equivalence based-security properties are harder than reachability properties (e.g. secrecy, authentication),
- electronic voting protocols are often **more complex** than authentication protocols,
- less classical cryptographic primitives (e.g. blind signature).
Summary

Observations:
- Definitions of security properties are often insufficiently precise
- No clear distinction between receipt-freeness and coercion-resistance

Goal:
1. Propose “formal methods” definitions of privacy-type properties,
2. Design automatic procedures to verify them.

Difficulties:
- equivalence based-security properties are harder than reachability properties (e.g. secrecy, authentication),
- electronic voting protocols are often more complex than authentication protocols,
- less classical cryptographic primitives (e.g. blind signature).
Results and Work in Progress

Modelling:

- **Formalisation** of privacy, receipt-freeness and coercion-resistance as some kind of observational **equivalence** in the **applied pi-calculus**,
- Coercion-Resistance $\Rightarrow$ Receipt-Freeness $\Rightarrow$ Privacy,

Case Studies:

- Fujioka *et al.* '92 – commitment and blind signature,
- Okamoto’96 – trap-door bit commitment and blind signature,
- Lee *et al.*’03 – re-encryption and designated verifier proof of re-encryption.

Verification: How to check such privacy-type properties?

- by using an existing tool (*e.g.* ProVerif)
- by developing **new techniques** (symbolic bisimulation)
Results and Work in Progress

Modelling:
- **Formalisation** of privacy, receipt-freeness and coercion-resistance as some kind of observational *equivalence* in the *applied pi-calculus*,
- Coercion-Resistance $\Rightarrow$ Receipt-Freeness $\Rightarrow$ Privacy,

Case Studies:
- Fujioka *et al.*'92 – commitment and blind signature,
- Okamoto’96 – trap-door bit commitment and blind signature,
- Lee *et al.*’03 – re-encryption and designated verifier proof of re-encryption.

Verification: How to check such privacy-type properties?
- by using an existing tool (*e.g.* ProVerif)
- by developing **new techniques** (symbolic bisimulation)
Results and Work in Progress

Modelling:
- **Formalisation** of privacy, receipt-freeness and coercion-resistance as some kind of observational equivalence in the applied pi-calculus,
- Coercion-Resistance $\Rightarrow$ Receipt-Freeness $\Rightarrow$ Privacy,

Case Studies:
- Fujioka *et al.*’92 – commitment and blind signature,
- Okamoto’96 – trap-door bit commitment and blind signature,
- Lee *et al.*’03 – re-encryption and designated verifier proof of re-encryption.

Verification: How to check such privacy-type properties?
- by using an existing tool (*e.g.* ProVerif)
- by developing **new techniques** (symbolic bisimulation)
Outline of the talk

1. Introduction

2. Applied $\pi$-calculus

3. Formalisation of Privacy-type Properties (Privacy, Receipt-Freeness)

4. Verification of privacy-type properties

5. Conclusion and Future Works
Outline of the talk

1. Introduction

2. Applied $\pi$-calculus

3. Formalisation of Privacy-type Properties (Privacy, Receipt-Freeness)

4. Verification of privacy-type properties

5. Conclusion and Future Works
Motivation for using the applied $\pi$-calculus

**Applied pi-calculus:** [Abadi & Fournet, 01]

Basic programming language with constructs for concurrency and communication

- based on the $\pi$-calculus [Milner et al., 92]
- in some ways similar to the spi-calculus [Abadi & Gordon, 98]

**Advantages:**

- allows us to model less classical cryptographic primitives
- both reachability and equivalence-based specification of properties
- automated proofs using ProVerif tool [Blanchet]
- powerful proof techniques for hand proofs
- successfully used to analyze a variety of security protocols
Motivation for using the applied $\pi$-calculus

Applied pi-calculus: [Abadi & Fournet, 01]

- basic programming language with constructs for concurrency and communication
  - based on the $\pi$-calculus [Milner et al., 92]
  - in some ways similar to the spi-calculus [Abadi & Gordon, 98]

Advantages:

- allows us to model less classical cryptographic primitives
- both reachability and equivalence-based specification of properties
- automated proofs using ProVerif tool [Blanchet]
- powerful proof techniques for hand proofs
- successfully used to analyze a variety of security protocols
The applied $\pi$-calculus on an example

Syntax:

- **Equational theory**: $\text{dec}(\text{enc}(x, y), y) = x$
- **Process**:

  $$P = \nu s, k. (\text{out}(c_1, \text{enc}(s, k)) \mid \text{in}(c_1, y). \text{out}(c_2, \text{dec}(y, k))).$$

Semantics:

- **Operational semantics** $\rightarrow$: closed by structural equivalence ($\equiv$) and application of evaluation contexts such that

  Comm: $\text{out}(a, x). P \mid \text{in}(a, x). Q \rightarrow P \mid Q$

  Then: if $M = M$ then $P$ else $Q \rightarrow P$

  Else: if $M = N$ then $P$ else $Q \rightarrow Q$ $(M \not\equiv N)$

  Example: $P \rightarrow \nu s, k. \text{out}(c_2, s)$

- **Labeled operational semantics** $\alpha$
The applied $\pi$-calculus on an example

Syntax:

- **Equational theory**: $\text{dec}(\text{enc}(x, y), y) = x$
- **Process**: $P = \nu s, k.(\text{out}(c_1, \text{enc}(s, k)) | \text{in}(c_1, y).\text{out}(c_2, \text{dec}(y, k)))$.

Semantics:

- **Operational semantics** $\rightarrow$: closed by structural equivalence ($\equiv$) and application of evaluation contexts such that
  
  - **Comm** $\text{out}(a, x).P | \text{in}(a, x).Q \rightarrow P | Q$
  - **Then** if $M = M$ then $P$ else $Q \rightarrow P$
  - **Else** if $M = N$ then $P$ else $Q \rightarrow Q$ ($M \not\equiv E N$)

  **Example**: $P \rightarrow \nu s, k.\text{out}(c_2, s)$

- **Labeled operational semantics** $\alpha$
The applied $\pi$-calculus on an example

Syntax:

- **Equational theory**: $\text{dec}(\text{enc}(x, y), y) = x$
- **Process**:

  $$P = \nu s, k. (\text{out}(c_1, \text{enc}(s, k)) \mid \text{in}(c_1, y). \text{out}(c_2, \text{dec}(y, k))).$$

Semantics:

- **Operational semantics** $\rightarrow$: closed by structural equivalence ($\equiv$) and application of evaluation contexts such that

  - **Comm**: $\text{out}(a, x).P \mid \text{in}(a, x).Q \rightarrow P \mid Q$
  - **Then**: if $M = M$ then $P$ else $Q \rightarrow P$
  - **Else**: if $M = N$ then $P$ else $Q \rightarrow Q$ ($M \not\equiv E N$)

- **Example**: $P \rightarrow \nu s, k. \text{out}(c_2, s)$

- **Labeled operational semantics** $\alpha$
The applied $\pi$-calculus on an example

Syntax:
- Equational theory: $\text{dec}(\text{enc}(x, y), y) = x$
- Process:
  $$P = \nu s, k.(\text{out}(c_1, \text{enc}(s, k)) \mid \text{in}(c_1, y).\text{out}(c_2, \text{dec}(y, k))).$$

Semantics:
- Operational semantics $\rightarrow$: closed by structural equivalence ($\equiv$) and application of evaluation contexts such that
  - Comm: $\text{out}(a, x).P \mid \text{in}(a, x).Q \rightarrow P \mid Q$
  - Then: if $M = M$ then $P$ else $Q \rightarrow P$
  - Else: if $M = N$ then $P$ else $Q \rightarrow Q \quad (M \not\equiv N)$

  Example: $P \rightarrow \nu s, k.\text{out}(c_2, s)$

- Labeled operational semantics $\overset{\alpha}{\rightarrow}$


**Observational equivalence ($\approx$)**

The largest symmetric relation $\mathcal{R}$ on processes such that $A \mathcal{R} B$ implies

1. if $A \Downarrow a$, then $B \Downarrow a$,
2. if $A \xrightarrow{\ast} A'$, then $B \xrightarrow{\ast} B'$ and $A' \mathcal{R} B'$ for some $B'$,

**Labeled bisimilarity ($\approx_\ell$)**

The largest symmetric relation $\mathcal{R}$ on processes, such that $A \mathcal{R} B$ implies

1. $\phi(A) \approx_s \phi(B)$ (static equivalence)
2. if $A \xrightarrow{\alpha} A'$, then $B \xrightarrow{\ast} B'$ and $A' \mathcal{R} B'$ for some $B'$,
3. if $A \xrightarrow{\alpha} A'$, then $B \xrightarrow{\ast \alpha \ast} B'$ and $A' \mathcal{R} B'$ for some $B'$. 

---

S. Delaune (Sch. of Computer Science)  
Electronic Voting  
11 / 28
Equivalences on processes

**Observational equivalence** ($\approx$)

The largest symmetric relation $\mathcal{R}$ on processes such that $A \mathcal{R} B$ implies

1. if $A \Downarrow a$, then $B \Downarrow a$,
2. if $A \xrightarrow{*} A'$, then $B \xrightarrow{*} B'$ and $A' \mathcal{R} B'$ for some $B'$,

**Labeled bisimilarity** ($\approx_\ell$)

The largest symmetric relation $\mathcal{R}$ on processes, such that $A \mathcal{R} B$ implies

1. $\phi(A) \approx_s \phi(B)$ (static equivalence)
2. if $A \xrightarrow{\alpha} A'$, then $B \xrightarrow{*} B'$ and $A' \mathcal{R} B'$ for some $B'$,
3. if $A \xrightarrow{\alpha} A'$, then $B \xrightarrow{* \alpha} \xrightarrow{*} B'$ and $A' \mathcal{R} B'$ for some $B'$. 
Voting protocols in the applied $\pi$-calculus

**Definition (Voting process)**

\[
VP \equiv \nu \tilde{n}. (V \sigma_1 | \cdots | V \sigma_n | A_1 | \cdots | A_m)
\]

- $V \sigma_i$: voter processes and $v \in \text{dom}(\sigma_i)$ refers to the value of the vote.
- $A_j$: election authorities which are required to be honest.
- $\tilde{n}$: channel names.

$\leftarrow S$ is a context which is as $VP$ but has a hole instead of two of the $V \sigma_i$. 

\[\]
Example: Fujioka et al. (1992)

Main Process

process

(* private channels *)
ν. privCh; ν. pkaCh1; ν. pkaCh2; ν. skaCh;
ν. skvaCh; ν. skvbCh;
(* administrators *)
(processK | processA | processA | processC | processC |
(* voters *)
(let skvCh = skvaCh in let v = a in processV) |
(let skvCh = skvbCh in let v = b in processV) )
Example: Fujioka et al. (1992)

let processV =
  (* his private key *)
in(skvCh,skv); let hostv = host(pk(skv)) in
  (* public keys of the administrator *)
in(pkaCh1,pubka);
\nu. blinder; \nu. r; let committedvote = commit(v,r) in
let blindedcommittedvote=blind(committedvote,blinder) in
out(ch,(hostv,sign(blindedcommittedvote,skv)));
in(ch,m2);
let result = checksign(m2,pubka) in
if result = blindedcommittedvote then
  let signedcommittedvote=unblind(m2,blinder) in
  phase 1;
out(ch,(committedvote,signedcommittedvote));
in(ch,(l,=committedvote,=signedcommittedvote));
  phase 2;
out(ch,(l,r)).
Example: Fujioka et al. (1992)

```plaintext
let processV =
(* his private key *)
in(skvCh, skv); let hostv = host(pk(skv)) in
(* public keys of the administrator *)
in(pkaCh1, pubka);
ν. blinder; ν. r; let committedvote = commit(v, r) in
let blindedcommittedvote = blind(committedvote, blinder) in
out(ch, (hostv, sign(blindedcommittedvote, skv)));
in(ch, m2);
let result = checksign(m2, pubka) in
if result = blindedcommittedvote then
let signedcommittedvote = unblind(m2, blinder) in
phase 1;
out(ch, (committedvote, signedcommittedvote));
in(ch, (1, committedvote, =signedcommittedvote));
phase 2;
out(ch, (1, r)).
```
Example: Fujioka et al. (1992)

```prolog
let processV =
  (* his private key *)
in(skvCh,skv); let hostv = host(pk(skv)) in
  (* public keys of the administrator *)
in(pkaCh1,pubka);
ν. blinder; ν. r; let committedvote = commit(v,r) in
let blindedcommittedvote=blind(committedvote,blinder) in
out(ch,(hostv,sign(blindedcommittedvote,skv)));
in(ch,m2);
let result = checksign(m2,pubka) in
if result = blindedcommittedvote then
let signedcommittedvote=unblind(m2,blinder) in
  phase 1;
out(ch,(committedvote,signedcommittedvote));
in(ch,(l,=committedvote,=signedcommittedvote));
  phase 2;
out(ch,(l,r)).
```
Example: Fujioka et al. (1992)

```
let processV =
  (* his private key *)
  in(skvCh,skv); let hostv = host(pk(skv)) in
  (* public keys of the administrator *)
  in(pkaCh1,pubka);
ν.blinder; ν.r; let committedvote = commit(v,r) in
  let blindedcommittedvote=blind(committedvote,blinder) in
  out(ch,(hostv,sign(blindedcommittedvote,skv)));
  in(ch,m2);
  let result = checksign(m2,pubka) in
  if result = blindedcommittedvote then
    let signedcommittedvote=unblind(m2,blinder) in
    phase 1;
    out(ch,(committedvote,signedcommittedvote));
    in(ch,(l,=committedvote,=signedcommittedvote));
    phase 2;
    out(ch,(l,r)).
```
Outline of the talk

1. Introduction

2. Applied $\pi$-calculus

3. Formalisation of Privacy-type Properties (Privacy, Receipt-Freeness)

4. Verification of privacy-type properties

5. Conclusion and Future Works
Formalisation of privacy

Classically modeled as observational equivalences between two slightly different processes $P_1$ and $P_2$, but

- changing the identity does not work, as identities are revealed
- changing the vote does not work, as the votes are revealed at the end

Solution:
$
\leftrightarrow$ consider 2 honest voters and swap their votes

A voting protocol respects privacy if

$$S[V_A^{a/v} | V_B^{b/v}] \approx S[V_A^{b/v} | V_B^{a/v}]$$
Formalisation of privacy

Classically modeled as observational equivalences between two slightly different processes $P_1$ and $P_2$, but

- changing the identity does not work, as identities are revealed
- changing the vote does not work, as the votes are revealed at the end

Solution:

→ consider 2 honest voters and swap their votes

A voting protocol respects privacy if

$$S[V_A^{a/v} | V_B^{b/v}] \approx S[V_A^{b/v} | V_B^{a/v}]$$
Some Examples

\[ S[V_A \{^a_v\} \mid V_B \{^b_v\}] \approx S[V_A \{^b_v\} \mid V_B \{^a_v\}] \]

Naive vote protocol (version 1)

\[ V \rightarrow S : \{v\}_{pub(S)} \]

What about privacy?
Some Examples

\[ S[V_A\{a/v\} \mid V_B\{b/v\}] \approx S[V_A\{b/v\} \mid V_B\{a/v\}] \]

Naive vote protocol (version 1)

\[ V \rightarrow S : \{v\}_{\text{pub}(S)} \]

What about privacy? OK
Some Examples

\[ S[V_{A\{a\}/v\}} \| V_{B\{b\}/v\}] \approx S[V_{A\{b\}/v\}} \| V_{B\{a\}/v\}] \]

**Naive vote protocol (version 1)**

\[ V \rightarrow S : \{v\}_{pub(S)} \]

What about privacy? **OK**

**Naive vote protocol (version 2)**

\[ V \rightarrow S : Id, \{v\}_{pub(S)} \]

What about privacy?
Some Examples

\[ S[V_A^{a/v} \mid V_B^{b/v}] \approx S[V_A^{b/v} \mid V_B^{a/v}] \]

Naive vote protocol (version 1)

\[ V \rightarrow S : \{v\}_{\text{pub}(S)} \]

What about privacy?  OK

Naive vote protocol (version 2)

\[ V \rightarrow S : Id, \{v\}_{\text{pub}(S)} \]

What about privacy?

- deterministic encryption: NOT OK
- probabilistic encryption: OK
Leaking secrets to the coercer

To model receipt-freeness we need to specify that a coerced voter cooperates with the coercer by leaking secrets on a channel $ch$.

We denote by $V^{ch}$ the process built from the process $V$ as follows:

- $0^{ch} \equiv 0$,
- $(P \parallel Q)^{ch} \equiv P^{ch} \parallel Q^{ch}$,
- $(\nu n. P)^{ch} \equiv \nu n. \text{out}(ch, n). P^{ch}$,
- $(\text{in}(u, x). P)^{ch} \equiv \text{in}(u, x). \text{out}(ch, x). P^{ch}$,
- $(\text{out}(u, M). P)^{ch} \equiv \text{out}(u, M). P^{ch}$,
- $\ldots$

We denote by $V^{out(ch, \cdot)} \equiv \nu ch.( V \parallel \text{in}(ch, x))$. 
Receipt-freeness

Definition (Receipt-freeness)

A voting protocol is receipt-free if there exists a process $V'$, satisfying

\begin{itemize}
\item $V' \setminus \mathsf{out}(\text{chc}, \cdot) \simeq V_A^{a/v}$,
\item $S[V_A^{c/v} \text{chc} \mid V_B^{a/v}] \simeq S[V' \mid V_B^{c/v}]$.
\end{itemize}

Intuitively, there exists a process $V'$ which

\begin{itemize}
\item does vote $a$,
\item leaks (possibly fake) secrets to the coercer,
\item and makes the coercer believe he voted $c$.
\end{itemize}
Summary

Coersion-Resistance is defined in a similar way (the voter has to used the outputs provided by the coercer)

Lemma

Let $VP$ be a voting protocol. We have formally shown that: $VP$ is coercion-resistant $\implies VP$ is receipt-free $\implies VP$ respects privacy.

Case Study (1): Fujioka et al.

- We have established privacy
  $\iff$ holds even if the authorities are corrupt
- This protocol is not receipt-free
  $\iff$ the random numbers for blinding and commitment can be used as a receipt
Outline of the talk

1. Introduction
2. Applied \(\pi\)-calculus
3. Formalisation of Privacy-type Properties (Privacy, Receipt-Freeness)
4. Verification of privacy-type properties
5. Conclusion and Future Works
An existing tool (ProVerif)

### Labeled bisimilarity ($\approx_\ell$)

The largest symmetric relation $\mathcal{R}$ on processes, such that $A \mathcal{R} B$ implies

1. $\phi(A) \approx_s \phi(B)$ (depends on $E$),
2. if $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some $B'$,
3. if $A \overset{\alpha}{\rightarrow} A'$, then $B \rightarrow^* \overset{\alpha}{\rightarrow}^* B'$ and $A' \mathcal{R} B'$ for some $B'$.

This relation is in general undecidable. Why?

- unfolding tree is infinite in depth
- unfolding tree is infinitely branching (because of inputs)
- equational theories may be complex

**Tool:** ProVerif

$\rightarrow$ Obviously, the procedure is not complete.
An existing tool (ProVerif)

Labeled bisimilarity ($\cong_{\ell}$)

The largest symmetric relation $\mathcal{R}$ on processes, such that $A \mathcal{R} B$ implies

1. $\phi(A) \cong_{s} \phi(B)$ (depends on $E$),
2. if $A \rightarrow A'$, then $B \rightarrow^{*} B'$ and $A' \mathcal{R} B'$ for some $B'$,
3. if $A \xrightarrow{\alpha} A'$, then $B \rightarrow^{*} \xrightarrow{\alpha} B'$ and $A' \mathcal{R} B'$ for some $B'$.

This relation is in general undecidable. Why?

- unfolding tree is infinite in depth
- unfolding tree is infinititely branching (because of inputs)
- equational theories may be complex

Tool: Prooverif

$\rightarrow$ Obviously, the procedure is not complete.
An existing tool (ProVerif)

Labeled bisimilarity ($\cong_\ell$)

The largest symmetric relation $\mathcal{R}$ on processes, such that $A \mathcal{R} B$ implies

1. $\phi(A) \cong_s \phi(B)$ (depends on $E$),
2. if $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some $B'$,
3. if $A \xrightarrow{\alpha} A'$, then $B \rightarrow^* \xrightarrow{\alpha}^* B'$ and $A' \mathcal{R} B'$ for some $B'$.

This relation is in general undecidable. Why?

- unfolding tree is infinite in depth
- unfolding tree is infinitely branching (because of inputs)
- equational theories may be complex

Tool: ProVerif

→ Obviously, the procedure is not complete.
Drawbacks of ProVerif

Proverif is not able to establish privacy for the naive vote protocol

\[\{a\}_{\text{pub}(S)} \mid \{b\}_{\text{pub}(S)} \approx \{b\}_{\text{pub}(S)} \mid \{a\}_{\text{pub}(S)}\]

... and more generally for any electronic voting protocols.

Why?

- ProVerif works on biprocesses (processes having the same structure).

\[P \approx Q \iff \begin{array}{l}
\text{let } \text{bool} = \text{choice[true, false]} \text{ in} \\
\quad \text{if } \text{bool} = \text{true} \text{ then } P \text{ else } Q
\end{array}\]

- Technique relies on easily matching up the execution paths of the two processes
Drawbacks of ProVerif

ProVerif is not able to establish privacy for the naive vote protocol

\[ \{a\}_{pub(S)} \parallel \{b\}_{pub(S)} \approx \{b\}_{pub(S)} \parallel \{a\}_{pub(S)} \]

... and more generally for any electronic voting protocols.

Why?

- ProVerif works on **biprocesses** (processes having the same structure).
  
  \[ P \approx Q \iff \text{let bool }= \text{choice[true,false] in} \]
  
  \[ \text{if bool }= \text{true then } P \text{ else } Q \]

- Technique relies on easily matching up the execution paths of the two processes
Drawbacks of ProVerif

ProVerif is not able to establish privacy for the naive vote protocol

\[
\{a\}_{\text{pub}(S)} \mid \{b\}_{\text{pub}(S)} \approx \{b\}_{\text{pub}(S)} \mid \{a\}_{\text{pub}(S)}
\]

... and more generally for any electronic voting protocols.

Why?

- ProVerif works on biprocesses (processes having the same structure).

\[
P \approx Q \iff \text{let bool } = \text{choice[true, false]} \text{ in}
\]
\[
\text{if bool } = \text{true} \text{ then } P \text{ else } Q
\]

- Technique relies on easily matching up the execution paths of the two processes

First Phase

\[
V_A\{^a/\nu\} \mid V_B\{^b/\nu\} \approx V_A\{^b/\nu\} \mid V_B\{^a/\nu\}
\]
Drawbacks of ProVerif

ProVerif is not able to establish privacy for the naive vote protocol

\[
\{a\}_{\text{pub}}(S) \mid \{b\}_{\text{pub}}(S) \approx \{b\}_{\text{pub}}(S) \mid \{a\}_{\text{pub}}(S)
\]

... and more generally for any electronic voting protocols.

Why?

- ProVerif works on biprocesses (processes having the same structure).

\[
P \approx Q \iff \text{let bool = choice[true,false] in if bool = true then P else Q}
\]

- Technique relies on easily matching up the execution paths of the two processes

Second Phase

\[
V_A\{^a/v\} \mid V_B\{^b/v\} \approx V_A\{^b/v\} \mid V_B\{^a/v\}
\]
How can we improve ProVerif?

→ with Mark Ryan and Ben Smith (University of Birmingham)

\[ V_A^{a/v} \mid V_B^{b/v} \approx V_A^{b/v} \mid V_B^{a/v} \]

where \( V_X = V_X^1; \text{phase1}; V_X^2 \)

**Conjecture**

To establish the equivalence, it may be sufficient to show that

- \( V_A^1 \{a/v\} \mid V_B^1 \{b/v\} \approx V_A^1 \{b/v\} \mid V_B^1 \{a/v\}, \) (1st phase)
- for all interleaving \( l_1 \) of \( V_A^1 \{a/v\} \mid V_B^1 \{b/v\} \), there exists an interleaving \( l_2 \) of \( V_A^1 \{b/v\} \mid V_B^1 \{a/v\} \) such that

\[ l_1; \text{phase1}; (V_A^2 \{a/v\} \mid V_B^2 \{b/v\}) \approx l_2; \text{phase1}; (V_B^2 \{a/v\} \mid V_A^2 \{b/v\}) \]

and vice-versa,
- and some additional assumptions.
Can we do better than ProVerif?

→ with Steve Kremer (LSV) and Mark Ryan (University of Birmingham)

Our Goal:
to do better than Proverif in the context of a bounded number of sessions

- Infinite depth:
  → we restrict to consider processes without replication.

- Infinite branching:
  → define a notion of symbolic processes and symbolic bisimulation
Symbolic Bisimulation

Concrete Side:
\[ \nu s, k. (\text{in}(c, x); \ P \mid \{s\}_k \! / \! y) \xrightarrow{\text{in}(c, m_1)} \nu s, k. (P^{m_1}_x \mid \{s\}_k \! / \! y) \]

Symbolic Side:
\[ (\nu s, k. (\text{in}(c, x); \ P \mid \{s\}_k \! / \! y) \ ; \ C) \xrightarrow{\text{in}(c, x)} (\nu s, k. (P \mid \{s\}_k \! / \! y) \ ; \ C \cup \{\nu s, k. \{s\}_k \! / \! y \models x\}) \]

Definition

Symbolic bisimulation \(\approx_{symb}\) is the largest symmetric relation \(\mathcal{R}\) such that
\[ (A; C_A) \mathcal{R} (B; C_B) \]
implies
- \(C_A\) and \(C_B\) are E-equivalent,
- if \((A; C_A) \rightarrow_s (A'; C'_A)\) with \(\text{SolE}(C'_A) \neq \emptyset\) then there exists \((B'; C'_B)\) such that \((B; C_B) \rightarrow^* (B'; C'_B)\) and \((A'; C'_A) \mathcal{R} (B'; C'_B)\)
- if \((A; C_A) \xrightarrow{\alpha} (A'; C'_A)\) ...
Symbolic Bisimulation

Concrete Side:
\[ \nu s, k.(\text{in}(c, x); P \mid \{s\}_k / y)) \xrightarrow{\text{in}(c, m_1)} \nu s, k.(P^{m_1 / x} \mid \{s\}_k / y)) \]

Symbolic Side:
\[ (\nu s, k.(\text{in}(c, x); P \mid \{s\}_k / y)) ; C \xrightarrow{\text{in}(c, x)} (\nu s, k.(P \mid \{s\}_k / y)) ; C \cup \{\nu s, k.\{s\}_k / y \models x\}) \]

Definition

Symbolic bisimulation \( \approx_{symb} \) is the largest symmetric relation \( R \) such that
\( (A ; C_A) R (B ; C_B) \) implies
- \( C_A \) and \( C_B \) are E-equivalent,
- if \( (A ; C_A) \rightarrow_s (A' ; C'_A) \) with \( \text{Sol}_E(C'_A) \neq \emptyset \) then there exists \( (B' ; C'_B) \) such that \( (B ; C_B) \rightarrow^* (B' ; C'_B) \) and \( (A' ; C'_A) R (B' ; C'_B) \)
- if \( (A ; C_A) \xrightarrow{\alpha} (A' ; C'_A) \) ...
Symbolic Bisimulation

Concrete Side:

\[
\nu s, k . (\text{in}(c, x); P \mid \{ s \}^k/y) \xrightarrow{\text{in}(c,m_1)} \nu s, k . (P^{m_1/x} \mid \{ s \}^k/y)
\]

Symbolic Side:

\[
(\nu s, k . (\text{in}(c, x); P \mid \{ s \}^k/y) ; C) \xrightarrow{\text{in}(c,x)} (\nu s, k . (P \mid \{ s \}^k/y) ; C \cup \{ \nu s, k . \{ s \}^k/y \mid - x \})
\]

Definition

Symbolic bisimulation \( \sim_{symb} \) is the largest symmetric relation \( R \) such that

\( (A ; C_A) R (B ; C_B) \) implies

- \( C_A \) and \( C_B \) are E-equivalent,
- if \( (A ; C_A) \xrightarrow{s} (A' ; C'_A) \) with \( \text{Sol}_E(C'_A) \neq \emptyset \) then there exists \( (B' ; C'_B) \) such that \( (B ; C_B) \xrightarrow{*} (B' ; C'_B) \) and \( (A' ; C'_A) R (B' ; C'_B) \)
- if \( (A ; C_A) \xrightarrow{\alpha} (A' ; C'_A) \) ...
Main Result

Conjecture

Let $A$ and $B$ be two processes. We have that

$$(A ; \emptyset) \approx_{symb} (B ; \emptyset) \quad \Longrightarrow \quad A \approx_{\ell} B$$

Sources of Incompleteness

$\hookrightarrow$ due to the fact that the instanciation of an input variable is postponed until the moment it is actually used

Example: $P_1 \approx_{\ell} Q_1$ whereas $(P_1 ; \emptyset) \not\approx_{symb} (Q_1 ; \emptyset)$.

$P_1 = \nu c_1 . \text{in}(c_2, x) . (\text{out}(c_1, b) \mid \text{in}(c_1, y) \mid \text{if } x = a \text{ then } \text{in}(c_1, z) . \text{out}(c_2, a))$

$Q_1 = \nu c_1 . \text{in}(c_2, x) . (\text{out}(c_1, b) \mid \text{in}(c_1, y) \mid \text{in}(c_1, z) . \text{if } x = a \text{ then } \text{out}(c_2, a))$

$\hookrightarrow$ but we think that our symbolic bisimulation is complete enough to deal with many interesting cases.
Main Result

Conjecture

Let $A$ and $B$ be two processes. We have that

$$(A ; \emptyset) \approx_{symb} (B ; \emptyset) \implies A \approx_{\ell} B$$

Sources of Incompleteness

$\leftarrow$ due to the fact that the instanciation of an input variable is postponed until the moment it is actually used

Example: $P_1 \approx_{\ell} Q_1$ whereas $(P_1 ; \emptyset) \not\approx_{symb} (Q_1 ; \emptyset)$.

$P_1 = \nu c_1.\mathsf{in}(c_2, x).(\mathsf{out}(c_1, b) \mid \mathsf{in}(c_1, y) \mid \text{if } x = a \text{ then } \mathsf{in}(c_1, z).\mathsf{out}(c_2, a))$

$Q_1 = \nu c_1.\mathsf{in}(c_2, x).(\mathsf{out}(c_1, b) \mid \mathsf{in}(c_1, y) \mid \mathsf{in}(c_1, z).\text{if } x = a \text{ then } \mathsf{out}(c_2, a))$

$\leftarrow$ but we think that our symbolic bisimulation is complete enough to deal with many interesting cases.
Main Result

Conjecture

Let $A$ and $B$ be two processes. We have that

$$(A ; \emptyset) \approx_{symb} (B ; \emptyset) \implies A \approx_{\ell} B$$

Sources of Incompleteness

$\leftarrow$ due to the fact that the instanciation of an input variable is postponed until the moment it is actually used.

Example: $P_1 \approx_{\ell} Q_1$ whereas $(P_1 ; \emptyset) \not\approx_{symb} (Q_1 ; \emptyset)$.

$P_1 = \nu c_1.\text{in}(c_2, x).(\text{out}(c_1, b) | \text{in}(c_1, y) | \text{if } x = a \text{ then } \text{in}(c_1, z).\text{out}(c_2, a))$

$Q_1 = \nu c_1.\text{in}(c_2, x).(\text{out}(c_1, b) | \text{in}(c_1, y) | \text{in}(c_1, z).\text{if } x = a \text{ then } \text{out}(c_2, a))$

$\leftarrow$ but we think that our symbolic bisimulation is complete enough to deal with many interesting cases.
Conclusion and Future Works

Conclusion:

- First **formal definitions** of receipt-freeness and coercion-resistance
- Coercion-Resistance ⇒ Receipt-Freeness ⇒ Privacy,
- 3 Case studies giving interesting insights

Current Works:

- An automatic procedure based on ProVerif
- A symbolic bisimulation for the applied pi calculus

Future Works:

- to design a procedure to solve our constraint systems for a class of equational theory as larger as possible
- to implement a tool based on this approach,
- other properties based on *not being able to prove* (abuse freeness)
Conclusion and Future Works

Conclusion:
- First **formal definitions** of receipt-freeness and coercion-resistance
- Coercion-Resistance $\Rightarrow$ Receipt-Freeness $\Rightarrow$ Privacy,
- 3 Case studies giving interesting insights

Current Works:
- An automatic procedure based on ProVerif
- A symbolic bisimulation for the applied pi calculus

Future Works:
- to **design a procedure** to solve our constraint systems for a class of equational theory as larger as possible
- to implement a **tool** based on this approach,
- **other properties** based on *not being able to prove* (abuse freeness)