Election Verifiability with ProVerif
(published at CSF’23)

Vincent Cheval\textsuperscript{1}, Véronique Cortier\textsuperscript{2}, Alexandre Debant\textsuperscript{2}

\textsuperscript{1}University of Oxford
\textsuperscript{2}Université de Lorraine, Inria, CNRS, Nancy, France

Journée du PEPR cybersécurité SVP
06 février 2024 - Paris, France
Security properties

**Vote secrecy**

“No one should know who I voted for”

**Verifiability**

“No one can modify the outcome of the election”
Security properties

Vote secrecy

“No one should know who I voted for”

Verifiability

“No one can modify the outcome of the election”
E-voting protocol
- overview -

**Setup phase**

- login
- login, pwd

**Voting phase**

- ballot
- (ok, file) / (ko, −)

**Verification phase**

- yes/no

**Tally phase**

- result
- (b₁, ..., bₙ)
Definition - An evoting protocol satisfies E2E verifiability if for any execution,

\[
\text{result} = V_{HV} \sqcup V_{HNV} \sqcup V_D
\]

where:

- \( V_{HV} \) is the multiset of votes of honest voters who verify
- \( V_{HNV} \) is a submultiset of the multiset of votes of honest voters who do not verify
- \( V_D \) contains at most one vote per dishonest voter
E2E verifiability [Cortier et al - ESORICS’14]

**Definition** - An evoting protocol satisfies E2E verifiability if for any execution,

\[
\text{result} = \begin{cases} 
V_{HV} & \text{is the multiset of votes of honest voters who verify} \\
V_{HNV} & \text{is a submultiset of the multiset of votes of honest voters who do not verify} \\
V_D & \text{contains at most one vote per dishonest voter}
\end{cases}
\]

Cannot be checked directly with existing tools…
E2E verifiability  [Cortier et al - ESORICS’14]

**Definition** - An evoting protocol satisfies E2E verifiability if for any execution,

\[
\text{result} = V_{HV} \cup V_{HNV} \cup V_D
\]

where:

- \( V_{HV} \) is the multiset of votes of honest voters who verify
- \( V_{HNV} \) is a submultiset of the multiset of votes of honest voters who do not verify
- \( V_D \) contains at most one vote per dishonest voter

**Approaches based on sub-properties**  e.g, [Cortier et al - CSF’19], [Baloglu et al - CSF’21]

- **Eligibility**: each vote has been cast by a legitimate voter
- **Individual verifiability**
  - **Cast-as-intended**: the voter’s ballot contains their intended vote
  - **Recorded-as-cast**: the counted ballot corresponds to the cast one
- **Universal verifiability**: the result corresponds to the content of the ballot-box
- **No clash attacks**: two voters cannot agree on the same ballot
E2E verifiability

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

\[
\text{result} = V_{HV} \cup V_{HNV} \cup V_D
\]

where:

- \( V_{HV} \) is the multiset of votes of honest voters who verify
- \( V_{HNV} \) is a submultiset of the multiset of votes of honest voters who do not verify
- \( V_D \) contains at most one vote per dishonest voter

Approaches based on sub-properties:

- **Eligibility**: each vote has been cast by a legitimate voter
- **Individual verifiability**
  - **Cast-as-intended**: the voter’s ballot contains their intended vote
  - **Recorded-as-cast**: the counted ballot corresponds to the cast one
- **Universal verifiability**: the result corresponds to the content of the ballot box
- **No clash attacks**: two voters cannot agree on the same ballot

Cannot be checked directly with existing tools…

These are only sufficient conditions…

e.g., [Cortier et al - ESORICS'14], [Baloglu et al - CSF'19], [Cortier et al - CSF'19]
Contributions

1. Exact characterization of E2E verifiability

**Theorem** - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2
Contributions

1. Exact characterization of E2E verifiability

   **Theorem** - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

2. A ProVerif framework to analyze evoting protocols

   **Applied to several protocols:** Helios, Belenios, Swiss Post, CHVote
What is ProVerif?

- is an automatic prover for symbolic analysis
  - messages abstracted with terms
  - Dolev-Yao attacker model (intercept/inject/modify)
- can model an unbounded number of sessions
- handles trace-based properties
- handles equivalence-based properties
- has already been used to analyse voting protocols, e.g., Helios, Belenios, Swiss Post, CHVote, etc

A trace \( tr \) is a finite sequence of \( \text{in}, \text{out}, \) or \( \text{event}(e(u_1, \ldots, u_n)) \).
Event satisfaction - A trace \( tr = tr_1 \ldots tr_n \) executes event \( E(u_1, \ldots, u_n) \) at time \( \tau \in \{ 1, \ldots, n \} \), noted \( (tr, \tau) \vdash E(u_1, \ldots, u_n) \), if \( tr_\tau = \text{event}(E(u_1, \ldots, u_n)) \).
Queries

**Event satisfaction** - A trace \( tr = tr_1 \ldots tr_n \) executes event \( E(u_1, \ldots, u_n) \) at time \( \tau \in \{1,\ldots,n\} \), noted \( (tr, \tau) \vdash E(u_1, \ldots, u_n) \), if \( tr_\tau = \text{event}(E(u_1, \ldots, u_n)) \)

**Query formula** - A trace \( tr = tr_1 \ldots tr_n \) satisfies a query of the form

\[
\bigwedge_{k=1}^{p} F_k(v_1, \ldots, v_{l_k}) \Rightarrow \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n_i} E_{i,j}(u_{1,i}^{j}, \ldots, u_{l_{i,j}}^{j})
\]

if for all substitution \( \sigma \) such that for all \( k \), \( (tr, \tau_k) \vdash F_k(v_1, \ldots, v_{l_k})\sigma \) for some \( \tau_k \), there exists \( \sigma' \) and \( i \) such that for all \( j \), there exists \( \tau_{i,j} \) such that \( (tr, \tau_{i,j}) \vdash E_{i,j}(u_{1,i}^{j}, \ldots, u_{l_{i,j}}^{j})\sigma' \) and \( F_k(v_1, \ldots, v_{l_k})\sigma = F_k(v_1, \ldots, v_{l_k})\sigma' \)
Injective queries

**Injective query** - A trace $tr = tr_1 \ldots tr_n$ satisfies an injective query of the form

$$\text{inj} = F_0(v_0, \ldots, v_l_0) \land \bigwedge_{k=1}^{p} F_k(v_1, \ldots, v_l_k) \Rightarrow \bigvee_{i=1}^{m} \text{inj} = E_{i,0}(u_{i,1}^{i,0}, \ldots, u_{i,l_i}^{i,0}) \land \bigwedge_{j=1}^{n_i} E_{i,j}(u_{i,1}^{i,j}, \ldots, u_{i,l_i}^{i,j})$$

if for all substitution $\sigma$ such that for all $k$, $(tr, \tau_k) \vdash F_k(v_1, \ldots, v_l_k)\sigma$ for some $\tau_k$, there exists $\sigma'$ and $i$ such that for all $j$, there exists $\tau_{i,j}$ such that $(tr, \tau_{i,j}) \vdash E_{i,j}(u_{i,1}^{i,j}, \ldots, u_{i,l_i}^{i,j})\sigma'$ and $F_k(v_1, \ldots, v_l_k)\sigma = F_k(v_1, \ldots, v_l_k)\sigma'$.

Moreover, there exists an injective function $f : \mathcal{F}_0(tr) \to \mathcal{E}_0(tr)$ such that if $(tr, \alpha) \vdash F_0(v_1, \ldots, v_l_0)\sigma$ then $(tr, f(\alpha)) \vdash E_{i,0}(u_{i,1}^{i,0}, \ldots, u_{i,l_i}^{i,0})\sigma'$.

$\mathcal{F}_0(tr), \mathcal{E}_0(tr) \subseteq \{1, \ldots, n\}$ are the sets of indices matching respectively $F_0(v_0, \ldots, v_l_0)$ and $E_{i,0}(u_{i,1}^{i,0}, \ldots, u_{i,l_i}^{i,0})$. 
Injective queries

**Injective query** - A trace $tr = tr_1 \ldots tr_n$ satisfies an injective query of the form

$$\forall \sigma_k \left(\forall t \in tr, \exists \tau_k \left(\forall i, j, \sigma_{\tau_k i, j} (tr, \sigma_k (tr, f(\alpha))) \Rightarrow \exists F_k (v_1, \ldots, v_l) \right) \right)$$

Moreover, there exists an injective function $f: \mathcal{F}_0 (tr) \rightarrow \mathcal{E}_0 (tr)$ such that if

$$\rho = \text{inj} - F(x) \Rightarrow \text{inj} - E_0(x) \lor \text{inj} - E_1(x)$$

Example: $\rho = \text{inj} - F(x) \Rightarrow \text{inj} - E_0(x) \lor \text{inj} - E_1(x)$
Injective queries

Injective query - A trace $tr = tr_1 \ldots tr_n$ satisfies an injective query of the form

\[ \sigma_{k}(tr, \tau_k) \vdash F_k(v_1, \ldots, v_{l_k}) \]

Moreover, there exists an injective function $\mathcal{f}$ such that if

\[ \mathcal{f}: \mathcal{F}_0(tr) \rightarrow \mathcal{E}_0(tr)(tr, \alpha) \vdash F_0(v_1, \ldots, v_{l_0}) \]

\[ if \quad tr \vdash E_{i,0}(u_{i,0}1, \ldots, u_{i,0}l_i) \sigma' \]

\[ \mathcal{f}_0(tr), \mathcal{E}_0(tr) \subseteq \{1, \ldots, n\} \]

\[ F_0(v_1, \ldots, v_{l_0}) \]

\[ \sigma' \mathcal{f}_0(tr) \]

\[ E_{i,0}(u_{i,0}1, \ldots, u_{i,0}l_i) \sigma' \]

\[ F_k(v_1, \ldots, v_{l_k}) \mathcal{f} \]

Example: \( \rho = \text{inj} - F(x) \Rightarrow \text{inj} - E_0(x) \)

\[ \lor \text{ inj} - E_1(x) \]

\[ tr_1 = \text{event}(E_0(a)).\text{event}(E_1(a)).\text{event}(F(a)).\text{event}(F(a)) \]
Injective queries

**Example:** \( \rho = \text{inj} - F(x) \Rightarrow \text{inj} - E_0(x) \vee \text{inj} - E_1(x) \)

\( tr_1 = \text{event}(E_0(a)) \cdot \text{event}(E_1(a)) \cdot \text{event}(F(a)) \cdot \text{event}(F(a)) \)

\( tr_1 \) satisfies \( \rho \) ✅
Injective queries

A trace $tr = tr_1 \ldots tr_n$ satisfies an injective query of the form

$$\forall \tau_{1:n} \in \mathcal{T}_{1:n}.$$ 

Moreover, there exists an injective function $f: \mathcal{F}_0(tr) \to \mathcal{E}_0(tr)$ such that if

$$tr_1 = \text{event}(E_0(a)) \cdot \text{event}(E_1(a)) \cdot \text{event}(F(a)) \cdot \text{event}(F(a))$$

then $tr_1$ satisfies $\rho$.

Example: $\rho = \text{inj} - F(x) \Rightarrow \text{inj} - E_0(x)$

$\lor \text{inj} - E_1(x)$

- $tr_1 = \text{event}(E_0(a)) \cdot \text{event}(E_1(a)) \cdot \text{event}(F(a)) \cdot \text{event}(F(a))$

$tr_1$ satisfies $\rho$.

- $tr_2 = \text{event}(E_0(a)) \cdot \text{event}(F(a)) \cdot \text{event}(F(a))$
Injective queries

Example: \( \rho = \text{inj} - F(x) \Rightarrow \text{inj} - E_0(x) \) 
\[ \vee \text{ inj} - E_1(x) \]

\( tr_1 = \text{event}(E_0(a)).\text{event}(E_1(a)).\text{event}(F(a)).\text{event}(F(a)) \)

\( tr_2 = \text{event}(E_0(a)).\text{event}(F(a)).\text{event}(F(a)) \)

\( f \)

\( tr_1 \) satisfies \( \rho \)

\( tr_2 \) does not satisfy \( \rho \)
E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:
- hv(id), an honest voter who verifies
- hnv(id), an honest voter who does not verify
- corrupt(id), a dishonest voter

Protocol steps
- voted(id, v), voter id has cast a vote v
- verified(id, v), voter id has cast a vote v and verified
- counted(v), a vote for v has been counted during the tally
- finish, the tally has been completed
E-voting protocol
- overview -

Setup phase

Voting phase

Verification phase

Tally phase

hv(id)
hnv(id)
corrupt(id)

login

pwd

login, pwd

voted(id, v)
ballet

(ok, ✓)/(ko, −)

verifed(id, v)

counted(v)

(b1, ..., bn)

result

 finish
E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:
- $hv(id)$, an honest voter who verifies
- $hnv(id)$, an honest voter who does not verify
- $corrupt(id)$, a dishonest voter

Protocol steps
- $voted(id, v)$, voter $id$ has cast a vote $v$
- $verified(id, v)$, voter $id$ has cast a vote $v$ and verified
- $counted(v)$, a vote for $v$ has been counted during the tally
- $finish$, the tally has been completed

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$result = V_{HV} \cup V_{HNV} \cup V_D$$

where:
- $result = \{ v \mid (tr, \tau) \vdash counted(v) \}$
- $V_{HV} = \{ v \mid (tr, \tau) \vdash verified(id, v) \text{ and } (tr, \tau) \vdash hv(id) \}$
- $V_{HNV} \subseteq_m \{ v \mid (tr, \tau) \vdash voted(id, v) \text{ and } (tr, \tau) \vdash hnv(id) \}$
- $|V_D| \leq |D|$, where $D = \{ id \mid (tr, \tau) \vdash corrupt(id) \}$
Exact characterization of E2E verifiability

**Theorem** - An evoting protocol satisfies E2E verifiability if and only if it all its traces \(tr\) satisfy:

- (Query 1) \(\text{finish} \land \text{inj} \rightarrow \text{counted}(x) \Rightarrow \text{inj} \rightarrow \text{hv}(z) \land \text{verified}(z, x)\)
  - \(\lor \text{inj} \rightarrow \text{hnv}(z) \land \text{voted}(z, x)\)
  - \(\lor \text{inj} \rightarrow \text{corrupt}(z)\)
Theorem - An evoting protocol satisfies E2E verifiability if and only if it all its traces \( tr \) satisfy:

- (Query 1) \[ \text{finish} \land \text{inj} - \text{counted}(x) \Rightarrow \text{inj} - \text{hv}(z) \land \text{verified}(z, x) \]
  \[ \lor \text{inj} - \text{hvn}(z) \land \text{voted}(z, x) \]
  \[ \lor \text{inj} - \text{corrupt}(z) \]
Theorem - An evoting protocol satisfies E2E verifiability if and only if it all its traces \( tr \) satisfy:

- **(Query 1)** \( finish \land inj - counted(x) \Rightarrow inj - hv(z) \land verified(z, x) \land inj - hnv(z) \land voted(z, x) \land inj - corrupt(z) \)

- **(Query 2)** \( finish \land inj - verified(z, x) \Rightarrow inj - counted(x) \)

Strong notion of eligibility
Exact characterization of E2E verifiability

**Theorem** - An evoting protocol satisfies E2E verifiability if and only if it all its traces $tr$ satisfy:

- **(Query 1)** $\text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land \text{hv}(z) \land \text{verified}(z,x) \lor \text{inj} \land \text{hnv}(z) \land \text{voted}(z,x) \lor \text{inj} \land \text{corrupt}(z)$

- **(Query 2)** $\text{finish} \land \text{inj} \land \text{verified}(z,x) \Rightarrow \text{inj} \land \text{counted}(x)$
Theorem - An evoting protocol satisfies E2E verifiability if and only if it all its traces \( tr \) satisfy:

- **(Query 1)** \( \text{finish} \land \text{inj} - \text{counted}(x) \Rightarrow \text{inj} - \text{hv}(z) \land \text{verified}(z, x) \)
  
  \[ \lor \text{inj} - \text{hnv}(z) \land \text{voted}(z, x) \]
  
  \[ \lor \text{inj} - \text{corrupt}(z) \]

- **(Query 2)** \( \text{finish} \land \text{inj} - \text{verified}(z, x) \Rightarrow \text{inj} - \text{counted}(x) \)

Ideas of the proof

\[ \Rightarrow \] “easy”, we can straightforwardly verify the queries
Exact characterization of E2E verifiability

**Theorem** - An evoting protocol satisfies E2E verifiability if and only if it all its traces $tr$ satisfy:

- (Query 1) $\text{finish} \land \text{inj} - \text{counted}(x) \Rightarrow \text{inj} - \text{hv}(z) \land \text{verified}(z, x)$
  \hspace{1cm} \lor \hspace{1cm} \text{inj} - \text{hvn}(z) \land \text{voted}(z, x)$
  \hspace{1cm} \lor \hspace{1cm} \text{inj} - \text{corrupt}(z)$

- (Query 2) $\text{finish} \land \text{inj} - \text{verified}(z, x) \Rightarrow \text{inj} - \text{counted}(x)$

**Ideas of the proof**

⇒ “easy”, we can straightforwardly verify the queries

⇐ “more difficult”…
Assumptions - for all traces \( tr \), Query 1 and Query 2 are satisfied.

- (Query 1) \( \text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land \text{hv}(z) \land \text{verified}(z, x) \)
  \[ \lor \text{inj} \land \text{hnv}(z) \land \text{voted}(z, x) \]
  \[ \lor \text{inj} \land \text{corrupt}(z) \]

- (Query 2) \( \text{finish} \land \text{inj} \land \text{verified}(z, x) \Rightarrow \text{inj} \land \text{counted}(x) \)
Goal: define an injective function

$h : \text{result} \rightarrow HV \cup HNV \cup D$ that is surjective over $HV$

Assumptions - for all traces $tr$, Query 1 and Query 2 are satisfied.

- (Query 1) $\text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land \text{hv}(z) \land \text{verified}(z, x)$
  $\lor \text{inj} \land \text{h宛}(z) \land \text{voted}(z, x)$
  $\lor \text{inj} \land \text{corrupt}(z)$

- (Query 2) $\text{finish} \land \text{inj} \land \text{verified}(z, x) \Rightarrow \text{inj} \land \text{counted}(x)$

idea of the proof
**Goal:** define an injective function 

\[ h : \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D} \] 

that is surjective over HV

**Assumptions** - for all traces \( tr \), Query 1 and Query 2 are satisfied.

- **(Query 1)** \( \text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land \text{hv}(z) \land \text{verified}(z, x) \)
  \[ \lor \text{inj} \land \text{hvn}(z) \land \text{voted}(z, x) \]
  \[ \lor \text{inj} \land \text{corrupt}(z) \]

- **(Query 2)** \( \text{finish} \land \text{inj} \land \text{verified}(z, x) \Rightarrow \text{inj} \land \text{counted}(x) \)
**Goal:** define an injective function $h : \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D}$ that is surjective over HV.

**Assumptions** - for all traces $tr$, Query 1 and Query 2 are satisfied.

- (Query 1) $\text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land \text{hv}(z) \land \text{verified}(z, x)$
  $\lor \text{inj} \land \text{hnv}(z) \land \text{voted}(z, x)$
  $\lor \text{inj} \land \text{corrupt}(z)$

- (Query 2) $\text{finish} \land \text{inj} \land \text{verified}(z, x) \Rightarrow \text{inj} \land \text{counted}(x)$
Goal: define an injective function $h : \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D}$ that is surjective over HV.
**Goal:** define an injective function $h : \text{result} \to \text{HV} \cup \text{HNV} \cup \text{D}$ that is surjective over HV

\[
h(x) = \begin{cases} 
g^{-1}(x) & \text{if } x \in g(\text{HV}) 
\end{cases}
\]
Goal: define an injective function 

\[
h : \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D}
\]

that is surjective over HV

\[
h(x) = \begin{cases} 
  g^{-1}(x) & \text{if } x \in g(\text{HV}) \\
  (f \circ g)^n \circ f(x) & \text{if } x \notin g(\text{HV}) \text{ and } f(x) \in \text{HV} \\
  & \text{where } n = \min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \text{HV}\}
\end{cases}
\]
Assumptions - for all traces $tr$, Query 1 and Query 2 are satisfied.

- (Query 1) $\text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land h(x) \land \text{verified}(z,x)$
  $\lor \text{inj} \land h_n(z) \land \text{voted}(z,x)$
  $\lor \text{inj} \land \text{corrupt}(z)$

- (Query 2) $\text{finish} \land \text{inj} \land \text{verified}(z,x) \Rightarrow \text{inj} \land \text{counted}(x)$

Goal: define an injective function $h: \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D}$ that is surjective over HV.

$h(x) = \begin{cases} 
  g^{-1}(x) & \text{if } x \in g(\text{HV}) \\
  (f \circ g)^n \circ f(x) & \text{if } x \notin g(\text{HV}) \text{ and } f(x) \in \text{HV} \\
\end{cases}$
where $n = \min \{i > 0 \mid (f \circ g)^i \circ f(x) \notin \text{HV}\}$
Assumptions - for all traces $tr$, Query 1 and Query 2 are satisfied.

- (Query 1) $\text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land \text{hv}(z) \land \text{verified}(z,x)$
  \[ \lor \text{inj} \land \text{hvv}(z) \land \text{voted}(z,x) \]
  \[ \lor \text{inj} \land \text{corrupt}(z) \]

- (Query 2) $\text{finish} \land \text{inj} \land \text{verified}(z,x) \Rightarrow \text{inj} \land \text{counted}(x)$

Goal: define an injective function $h : \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D}$ that is surjective over HV

$h(x) =
\begin{cases}
  g^{-1}(x) & \text{if } x \in g(\text{HV}) \\
  (f \circ g)^n \circ f(x) & \text{if } x \notin g(\text{HV}) \text{ and } f(x) \in \text{HV}
\end{cases}

\text{where } n = \min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \text{HV}\}$
Assumptions - for all traces \( tr \), Query 1 and Query 2 are satisfied.

- (Query 1) \( \text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land h(v(z)) \land \text{verified}(z, x) \)
  \[ \lor \text{inj} \land h_n(v(z)) \land \text{voted}(z, x) \]
  \[ \lor \text{inj} \land \text{corrupt}(z) \]
- (Query 2) \( \text{finish} \land \text{inj} \land \text{verified}(z, x) \Rightarrow \text{inj} \land \text{counted}(x) \)

Goal: define an injective function
\[ h : \text{result} \rightarrow HV \cup HNV \cup \Delta \] that is surjective over HV.

\[ h(x) = \begin{cases} 
  g^{-1}(x) & \text{if } x \in g(HV) \\
  (f \circ g)^n \circ f(x) & \text{if } x \not\in g(HV) \text{ and } f(x) \in HV
\end{cases} \]

where \( n = \min \{ i > 0 \mid (f \circ g)^i \circ f(x) \not\in HV \} \)
Goal: define an injective function $h: \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D}$ that is surjective over HV

$$h(x) = \begin{cases} g^{-1}(x) & \text{if } x \in g(\text{HV}) \\ (f \circ g)^n \circ f(x) & \text{if } x \notin g(\text{HV}) \text{ and } f(x) \in \text{HV} \\ \end{cases}$$

where $n = \min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \text{HV}\}$
**Goal:** define an injective function

\[ h : \text{result} \rightarrow HV \cup HNV \cup D \text{ that is surjective over } HV \]

\[ h(x) = \begin{cases} 
  g^{-1}(x) & \text{if } x \in g(HV) \\
  (f \circ g)^n \circ f(x) & \text{if } x \notin g(HV) \text{ and } f(x) \in HV \\
  f(x) & \text{otherwise}
\end{cases} \]

where \( n = \min \{ i > 0 \mid (f \circ g)^i \circ f(x) \notin HV \} \)
Contributions

1. Exact characterization of E2E verifiability

   Theorem - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

2. A ProVerif framework to analyze evoting protocols

   Applied to several protocols: Helios, Belenios, Swiss Post, CHVote
Our framework

Description of each role (in isolation)

Generic process
- linking the different roles
- defining accurate and meaningful scenarios

ProVerif

✅

❓

❌
Our framework

Description of each role (in isolation)

Generic process
- linking the different roles
- defining accurate and meaningful scenarios

ProVerif

GSVerif-like ProVerif library

[Cheval et al - CSF'18]
Our framework

Description of each role (in isolation)

Generic process
- linking the different roles
- defining accurate and meaningful scenarios

ProVerif

GSVerif-like ProVerif library

Generic lemmas

[Cheval et al - CSF’18]
Which scenarios?

A scenario as general as possible

- An arbitrary number of voters…
- who can freely revote…
- with a private bulletin board…
- and a counting function equivalent to the multiset of votes.
Which scenarios?

A scenario as general as possible

- An arbitrary number of voters…
- who can freely revote…
- with a private bulletin board…
- and a counting function equivalent to the multiset of votes.

How to extend/adapt this scenario?

- Bound the number of voters ➔ define a restriction
- Prevent revote ➔ define a restriction
- Model a public bulletin board ➔ add an output on a public channel
- …
Details

- **12 processes**
- **Setup phase:** 4 processes (how voting data are generated, how they are received by voters, what are their initial knowledge, what is a valid vote)
- **Voting phase:**
  - **Voter:** 2 processes (how a voter casts a vote, how they verify)
  - **Bulletin board:** 5 processes (how to update the bulletin board, what is a valid ballot, how voters are publicly identified)
- **Tally:** 1 process (how to open a ballot)
- **Other components:** everything else (e.g. voting server, control component, etc)
Details

- **12 processes**

  - **Setup phase:** 4 processes (how voting data are generated, how they are received by voters, what are their initial knowledge, what is a valid vote)

  - **Voting phase:**
    - **Voter:** 2 processes (how a voter casts a vote, how they verify)
    - **Bulletin board:** 5 processes (how to update the bulletin board, what is a valid ballot, how voters are publicly identified)
  
  - **Tally:** 1 process (how to open a ballot)

- **Other components:** everything else (e.g. voting server, control component, etc)

---

- **Protocol specific processes**

- **Generic processes and libraries**

  - **8 processes defining the generic scenario** (voter registration, voting process, tally, main system...)

  - GSVerif-like axioms to manipulate cells, counters, etc
    - 2 new axioms for nested counters and emphasize term freshness

  - 8 well-crafted lemmas (27 queries) to improve termination and accuracy
## Applications

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Origin of the files</th>
<th>Voter</th>
<th>Registrar (setup)</th>
<th>Server (1 CCR/M)</th>
<th>E2E verifiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helios (toy)</td>
<td>(new files)</td>
<td>😊</td>
<td>—</td>
<td>😊</td>
<td>✔️ 16s</td>
</tr>
<tr>
<td>Belenios (tally)</td>
<td>(existing personal files)</td>
<td>😊</td>
<td>😎</td>
<td>😎</td>
<td>✔️ 24s</td>
</tr>
<tr>
<td>Belenios (last)</td>
<td>(existing personal files)</td>
<td>😊</td>
<td>😎</td>
<td>😎</td>
<td>✗ 5s</td>
</tr>
<tr>
<td>Belenios-counter (last)</td>
<td>(existing personal files)</td>
<td>😊</td>
<td>😎</td>
<td>😎</td>
<td>✗ 8s</td>
</tr>
<tr>
<td>Belenios-hash(^1) (last)</td>
<td>(new files)</td>
<td>😊</td>
<td>😎</td>
<td>😎</td>
<td>✔️ 62s</td>
</tr>
<tr>
<td>Swiss Post</td>
<td>(Swiss Post gitlab(^2))</td>
<td>😊</td>
<td>😎</td>
<td>😎</td>
<td>✔️ 58s</td>
</tr>
<tr>
<td>CHVote</td>
<td>[Bernhard et al - 2018]</td>
<td>😊</td>
<td>😎</td>
<td>😎</td>
<td>✔️ 17s</td>
</tr>
</tbody>
</table>

\(^1\)inspired by [Baloglu et al - EVoteID 2021]

\(^2\)https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/Symbolic-models
Conclusion

1. Exact characterization of E2E verifiability

**Theorem** - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

2. A ProVerif framework to analyze evoting protocols

**Applied to several protocols**: Helios, Belenios, Swiss Post, CHVote
Conclusion

1. Exact characterization of E2E verifiability

**Theorem** - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

2. A ProVerif framework to analyze evoting protocols

**Applied to several protocols:** Helios, Belenios, Swiss Post, CHVote

**Future work**

- Extend the framework to analyze vote secrecy
- Extend GSVerif with the new invariants introduced in this work
- Improve the modeling of the tally:
  - consider counting functions different from the multiset of votes (e.g., Condorcet, Single Transferable Vote, d’Hondt method)
  - provide a more accurate model of the homomorphic or mixnet tally