Election Verifiability with ProVerif

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

\textsuperscript{1}Inria Paris, France
\textsuperscript{2}Université de Lorraine, Inria, CNRS, Nancy, France

CSF 2023
Dubrovnik, Croatia
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

Voting phase

Verification phase
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
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**

Cannot be check 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

Cannot be check directly with existing tools...
Definition - An evoting protocol satisfies E2E verifiability if for any execution,

\[ \text{result} = \mathbb{V}_H \cup \mathbb{V}_{HNV} \cup \mathbb{V}_D \]

where:

- \( \mathbb{V}_H \) is the multiset of votes of honest voters who verify
- \( \mathbb{V}_{HNV} \) is a submultiset of the multiset of votes of honest voters who do not verify
- \( \mathbb{V}_D \) contains at most one vote per dishonest voter

Approaches based on sub-properties: e.g., [Cortier et al - CSF’19], [Balaglu 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

Cannot be checked directly with existing tools…

These are only sufficient conditions…
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 be 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 $\text{event}(e(u_1, \ldots, u_n))$. 

$$P, Q := 0$$
$$\quad | \text{new } n; P$$
$$\quad | \text{let } x = v \text{ in } P \text{ else } Q;$$
$$\quad | \text{in}(c, x); P$$
$$\quad | \text{out}(c, u); P$$
$$\quad | (P \mid Q)$$
$$\quad | \neg P$$
$$\quad | \text{event } e(u_1, \ldots, u_n); P$$
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 \( tr_{\tau} = \text{event}(E(u_1, \ldots, u_n)) \).
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

$$\bigwedge_{k=1}^{p} F_k(v_1, \ldots, v_{l_k}) \Rightarrow \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n_i} E_{i,j}(u_1^{i,j}, \ldots, u_{l_{i,j}}^{i,j})$$

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 $(tr, \tau_{i,j}) \vdash E_{i,j}(u_1^{i,j}, \ldots, u_{l_{i,j}}^{i,j})\sigma'$ 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

\[
\text{inj} - F_0(v_0, \ldots, v_{l_0}) \land \bigwedge_{k=1}^{p} F_k(v_1, \ldots, v_{l_k}) \Rightarrow \bigvee_{i=1}^{m} \text{inj} - E_{i,0}(u_1^{i,0}, \ldots, u_{l_0}^{i,0}) \land \bigwedge_{j=1}^{n_i} E_{i,j}(u_1^{i,j}, \ldots, u_{l_j}^{i,j})
\]

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 \( (tr, \tau_{i,j}) \vdash E_{i,j}(u_1^{i,j}, \ldots, u_{l_j}^{i,j})\sigma' \) 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 \( f : \mathcal{F}_0(tr) \rightarrow \mathcal{E}_0(tr) \) such that if \( (tr, \alpha) \vdash F_0(v_1, \ldots, v_{l_0})\sigma \) then

\( (tr, f(\alpha)) \vdash E_{i,0}(u_1^{i,0}, \ldots, u_{l_0}^{i,0})\sigma' \).

\( \mathcal{F}_0(tr), \mathcal{E}_0(tr) \subseteq \{ 1, \ldots, n \} \) are the sets of indices matching respectively \( F_0(v_0, \ldots, v_{l_0}) \) and \( E_{i,0}(u_1^{i,0}, \ldots, u_{l_0}^{i,0}) \).
Injective queries

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

Example: $\rho = \text{inj} - F_0(x) \Rightarrow \text{inj} - E_0(x)$

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

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

$$\sigma_k(tr_1, \tau_k) \vdash F_k(v_1, \ldots, v_{l_k})$$

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

$$\sigma_\tau(tr, f(\alpha)) \vdash E_{i,0}(u_{i,0}^1, \ldots, u_{i,0}^l)$$

$$\mathcal{F}_0(tr), \mathcal{E}_0(tr) \subseteq \{1, \ldots, n\}$$

$F_0(v_1, \ldots, v_{l_0})$ $E_{i,0}(u_{i,0}^1, \ldots, u_{i,0}^l)$

**Example:** $\rho = inj - F_0(x) \Rightarrow inj - E_0(x)$

$$\vee\ inj - E_1(x)$$

$tr_1 = \text{event}(E_0(a)).\text{event}(E_1(a)).\text{event}(F_0(a)).\text{event}(F_0(a))$
Injective queries

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

$$\sigma_{k}(tr, \tau_k) \vdash F_k(v_1, \ldots, v_l)$$

Moreover, there exists an injective function $f$ such that if

$$f(\alpha)\text{satisfies }\rho$$

then

$$\sigma_{\tau_k}F_k(v_1, \ldots, v_l) = F_k(v_1, \ldots, v_l)$$

are the sets of indices matching respectively $tr = tr_1 \ldots tr_n$.

**Example:** $\rho = \text{inj} - F_0(x) \Rightarrow \text{inj} - E_0(x)$

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

- $tr_1 = \text{event}(E_0(a)) \cdot \text{event}(E_1(a)) \cdot \text{event}(F_0(a)) \cdot \text{event}(F_0(a))$

$tr_1$ satisfies $\rho$ ✅
Injective queries

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

\[ \rho = \text{inj} - F_0(x) \implies \text{inj} - E_0(x) \]

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

\[ tr_1 = \text{event}(E_0(a)).\text{event}(E_1(a)).\text{event}(F_0(a)).\text{event}(F_0(a)) \]

\[ \text{tr}_1 \text{ satisfies } \rho \]

\[ tr_2 = \text{event}(E_0(a)).\text{event}(F_0(a)).\text{event}(F_0(a)) \]
Injective queries

Example: \( \rho = \text{inj} - F_0(x) \Rightarrow \text{inj} - E_0(x) \)
\( \lor \text{inj} - E_1(x) \)

- \( tr_1 = \text{event}(E_0(a)) \cdot \text{event}(E_1(a)) \cdot \text{event}(F_0(a)) \cdot \text{event}(F_0(a)) \)
  \( tr_1 \) satisfies \( \rho \)

- \( tr_2 = \text{event}(E_0(a)) \cdot \text{event}(F_0(a)) \cdot \text{event}(F_0(a)) \)
  \( tr_2 \) 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
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:
- \( result = \{ v \mid (tr, \tau) \vdash counted(v) \} \)
- \( V_{HV} = \{ v \mid (tr, \tau) \vdash verified(id, v) \land (tr, \tau') \vdash hv(id) \} \)
- \( V'_{HNV} \subseteq_m V_{HNV} = \{ v \mid (tr, \tau) \vdash voted(id, v) \land (tr, \tau') \vdash hnv(id) \} \)
- \(| V_D | \leq |D| \) where \( D = \{ id \mid (tr, \tau) \vdash corrupt(id) \} \)
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
- \( finish(v) \), 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:
- \( result = \{v \mid (tr, \tau) \vdash counted(v)\} \)
- \( V_{HV} = \{v \mid (tr, \tau) \vdash verified(id, v) \text{ and } (tr, \tau') \vdash hv(id)\} \)
- \( V'_{HNV} \subseteq_m V_{HNV} = \{v \mid (tr, \tau) \vdash voted(id, v) \text{ and } (tr, \tau') \vdash hnv(id)\} \)
- \( |V_D| \leq |D| \text{ where } D = \{id \mid (tr, \tau) \vdash corrupt(id)\} \)
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:
- \( result = \{ v \mid (tr, \tau) \vdash counted(v) \} \)
- \( V_{HV} = \{ v \mid (tr, \tau) \vdash verified(id, v) \text{ and } (tr, \tau') \vdash hv(id) \} \)
- \( V'_{HNV} \subseteq_m V_{HNV} = \{ v \mid (tr, \tau) \vdash voted(id, v) \text{ and } (tr, \tau') \vdash hnv(id) \} \)
- \( |V_D| \leq |D| \) where \( D = \{ id \mid (tr, \tau) \vdash corrupt(id) \} \)
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:
- $result = \{v \mid (tr, \tau) \vdash counted(v)\}$
- $V_{HV} = \{v \mid (tr, \tau) \vdash verified(id, v) \text{ and } (tr, \tau') \vdash hv(id)\}$
- $V'_{HNV} \subseteq m V_{HNV} = \{v \mid (tr, \tau) \vdash voted(id, v) \text{ and } (tr, \tau') \vdash hnv(id)\}$
- $|V_D| \leq |D|$ where $D = \{id \mid (tr, \tau) \vdash corrupt(id)\}$

all the votes of honest voters who verify

all the counted votes

A subset of the votes of honest voters who do not verify
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,

\[
\text{result} = V^\prime_H \cup V''_H \cup V_D
\]

where:

- \(\text{result} = \{v | (tr, \tau) \vdash counted(v)\}\)
- \(V_H = \{v | (tr, \tau) \vdash verified(id, v) \text{ and } (tr, \tau') \vdash hv(id)\}\)
- \(V''_H \subseteq_m V_H = \{v | (tr, \tau) \vdash voted(id, v) \text{ and } (tr, \tau') \vdash hnv(id)\}\)
- \(|V_D| \leq |D|\) where \(D = \{id | (tr, \tau) \vdash corrupt(id)\}\)
Theorem - An evoting protocol satisfies E2E verifiability if and only if it all its traces $tr$ satisfy:

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

- **(Query 2)** $\text{finish} \land \text{inj} \rightarrow \text{verified}(z, x) \Rightarrow \text{inj} \rightarrow \text{counted}(x)$
Exact characterization
of E2E verifiability

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

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

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

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

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

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

- (Query 2) $\text{finish} \land \text{inj} \rightarrow \text{verified}(z,x) \Rightarrow \text{inj} \rightarrow \text{counted}(x)$
idea of the proof

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

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

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

Goal: define an injective function

\( h : \text{result} \rightarrow HV \cup HNV \cup D \) that is

surjective over HV
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) \( \text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land h\text{v}(z) \land \text{verified}(z, x) \)
  \( \lor \) \( \text{inj} \land h\text{v}(z) \land \text{voted}(z, x) \)
  \( \lor \) \( \text{inj} \land \text{corrupt}(z) \)

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

Goal: define an injective function

\[ h : \text{result} \rightarrow HV \cup HNV \cup D \] 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
**idea of the proof**

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

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

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

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

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

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

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

**Diagram:**

- $HV$
- $HNV$
- $D$
- subset of result mapped to $HV$ by $f$
- subset of result mapped to $HNV$ by $f$
- subset of result mapped to $D$ by $f$
Assumptions - for all traces \( tr \), Query 1 and Query 2 are satisfied.

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

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

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

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

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

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

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

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

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

\[
h(x) = \begin{cases} 
  g^{-1}(x) & \text{if } x \in g(HV) \\
  (f \circ g)^n \circ f(x) & \text{if } x \notin g(HV) \text{ and } f(x) \in HV \\
  & \text{where } n = \min \{ i > 0 \mid (f \circ g)^i \circ f(x) \notin HV \} 
\end{cases}
\]
**Goal:** define an injective function

\( h : \text{result} \to HV \cup HNV \cup D \) that is surjective over HV

\[
h(x) = \begin{cases} 
    g^{-1}(x) & \text{if } x \in g(HV) \\
    (f \circ g)^n \circ f(x) & \text{if } x \notin g(HV) \text{ and } f(x) \in HV \\
    & \text{where } n = \min \{i > 0 \mid (f \circ g)^i \circ f(x) \notin HV \}
\end{cases}
\]
**Goal:** define an injective function $h : \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup \text{D}$ that is surjective over HV

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

where $n = \min\{i > 0 \mid (f \circ g)^i \circ f(x) \not\in \text{HV}\}$
Assumptions - for all traces \( tr \), Query 1 and Query 2 are satisfied.

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

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

Goal: define an injective function 
\( h : \text{result} \rightarrow \text{HV} \cup \text{HNV} \cup D \) 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)** \( \text{finish} \land \text{inj} \land \text{counted}(x) \Rightarrow \text{inj} \land \text{hv}(z) \land \text{verified}(z, x) \)
  \[ \lor \text{inj} \land \text{h}(z) \land \text{voted}(z, x) \]
  \[ \lor \text{inj} \land \text{corrupt}(z) \]

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

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

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

where \( n = \min \{ i > 0 \mid (f \circ g)^i \circ f(x) \notin \text{HV} \} \)
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
[Cheval et al - CSF’18]

Generic lemmas

[Cheval et al - CSF’18]
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)
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)

We do model the tally unlike previous approaches
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)

**Protocol specific processes**

**Generic processes and libraries**

- **8 processes** (voter registration, voting process, tally, main system…)
- Unbounded number of elections and voters
- Modeler can define honesty assumptions through restrictions
- 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

We do model the tally unlike previous approaches.
# 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>Belenos (tally)</td>
<td>(existing personal files)</td>
<td>😊</td>
<td>😬</td>
<td>😊</td>
<td>✓ 24s</td>
</tr>
<tr>
<td>Belenos (last)</td>
<td>(existing personal files)</td>
<td>😊</td>
<td>😊</td>
<td>😁</td>
<td>✗ 5s</td>
</tr>
<tr>
<td>Belenos-counter (last)</td>
<td>(existing personal files)</td>
<td>😊</td>
<td>😊</td>
<td>😁</td>
<td>✗ 8s</td>
</tr>
<tr>
<td>Belenos-hash¹ (last)</td>
<td>(new files)</td>
<td>😊</td>
<td>😬</td>
<td>😊</td>
<td>✓ 62s</td>
</tr>
<tr>
<td>Swiss Post</td>
<td>(Swiss Post gitlab²)</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>

¹inspired by [Baloglu et al - EVoteID 2021]
Conclusion

1. Exact characterization of E2E verifiability

\textbf{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

\textbf{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