# Election Verifiability with ProVerif

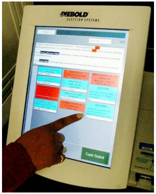
Vincent Cheval, Véronique Cortier, Alexandre Debant

GT-MFS, Oléron, March, 5th 2024

presented at CSF'23

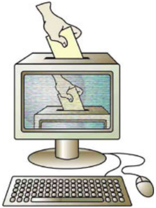
# Two main families for electronic voting

## Voting machines




- Voters attend a polling station;
- Standard authentication (id cards, etc.)

## Internet Voting



- Voters vote from home;
- Using their own computer (or phone, tablet, ...)

# Confidentiality of the votes

Vote privacy

*"No one should know how I voted"*

# Confidentiality of the votes

Vote privacy

*"No one should know how I voted"*

Better: Receipt-free / Coercion-resistant

*"No one should know how I voted,*
*even if I am willing to tell my vote!* "

# Confidentiality of the votes

Vote privacy

*"No one should know how I voted"*

Better: Receipt-free / Coercion-resistant

*"No one should know how I voted,*
*even if I am willing to tell my vote!* "

- vote buying
- coercion

Everlasting privacy: no one should know my vote, even when the cryptographic keys will be eventually broken.

# Verifiability

Individual Verifiability: a voter can check that

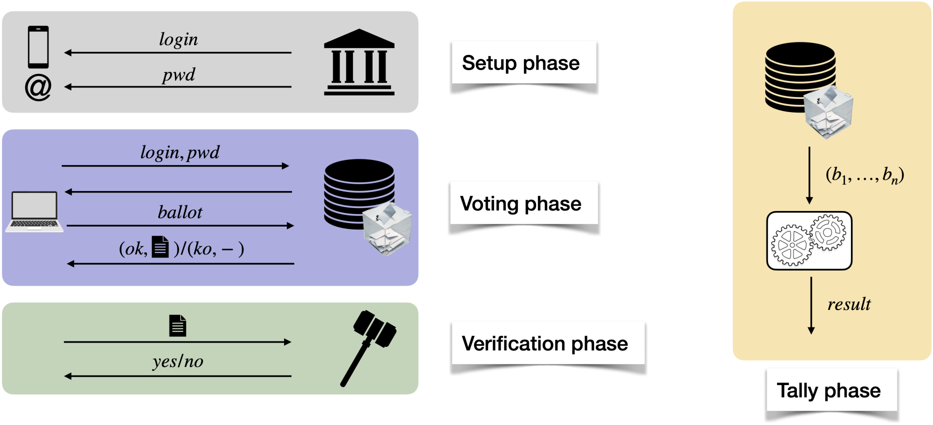
- cast as intended: their ballot contains their intended vote
- recorded as cast: their ballot is in the ballot box.

Universal Verifiability: everyone can check that

- tallied as recorded: the result corresponds to the ballot box.
- eligibility: ballots have been casted by legitimate voters.

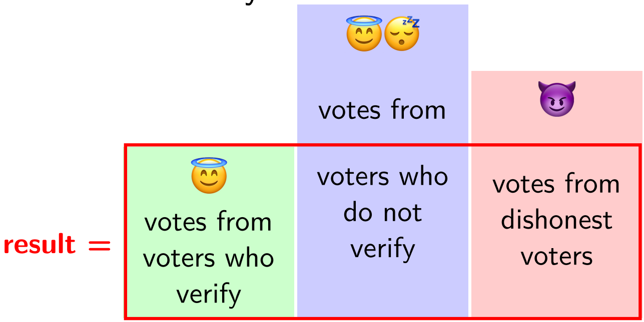
You should verify the election,
not the system.

# Voting protocol - overview

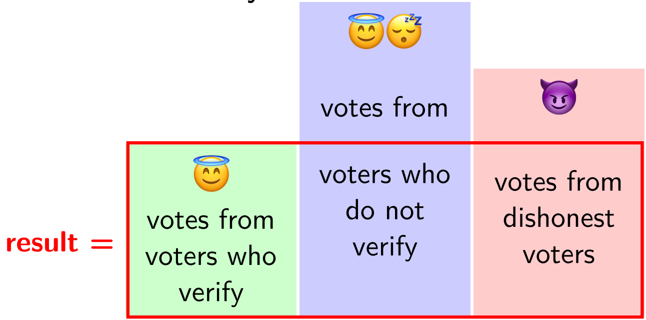



[slide borrowed from Alexandre Debant.]

# End2End verifiability



# End2End verifiability

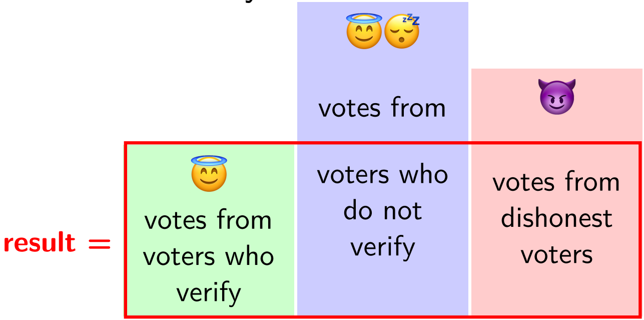



⚠ Hard to verify for tools
→ check subproperties instead

**Theorem** ([Cortier *et al* CSF'19, Baloglu *et al* CSF'21])

*eligibility* + *cast-as-intended* + *recorded-as-cast* + *tallied-as-recorded* + *no clash* ⇒ *E2E Verifibility*

# End2End verifiability



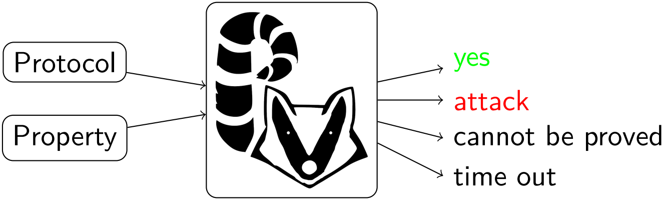

⚠ Hard to verify for tools
→ check subproperties instead

**Theorem** ([Cortier *et al* CSF'19, Baloglu *et al* CSF'21])

*eligibility* + *cast-as-intended* + *recorded-as-cast* + *tallied-as-recorded* + *no clash* ⇒ *E2E Verifibility*

⚠ sufficient (but not tight) conditions

# Goal: verifiability in ProVerif




- Works on most of existing protocols in the literature
- Is also used on industrial protocols (e.g. TLS, Signal, ...)
- used to pass Swiss requirements on voting
  - Neuchâtel/Scytl protocol [C., Galindo, Turuani 2018]
  - CHVote protocol [Bernhard, C., Gaudry, Turuani, Warinschi 2019]

# ProVerif: protocols

The grammar of processes is as follows:

$$P, Q, R :=$$
$$0$$
$$\texttt{if } M_1 = M_2 \texttt{ then } P \texttt{ else } Q$$
$$\text{let } x = M \text{ in } P$$
$$\text{in}(c, x); P$$
$$\text{out}(c, N); P$$
$$\text{new } n; P$$
$$P \mid Q$$
$$!P$$
$$\text{event} E; P$$

# ProVerif: properties

$e_i, e_j$: events

simplified fragment: $\wedge \vee e_i \Longrightarrow \wedge \vee e'_j$

Example 1: $\text{Paid}(Alice, x) \Rightarrow \text{Received}(Bob, x)$

# ProVerif: properties

$e_i, e_j$: events

$$\text{simplified fragment:} \qquad \wedge \vee e_i \Longrightarrow \wedge \vee e'_j$$

Example 1: $\text{Paid}(Alice, x) \Rightarrow \text{Received}(Bob, x)$

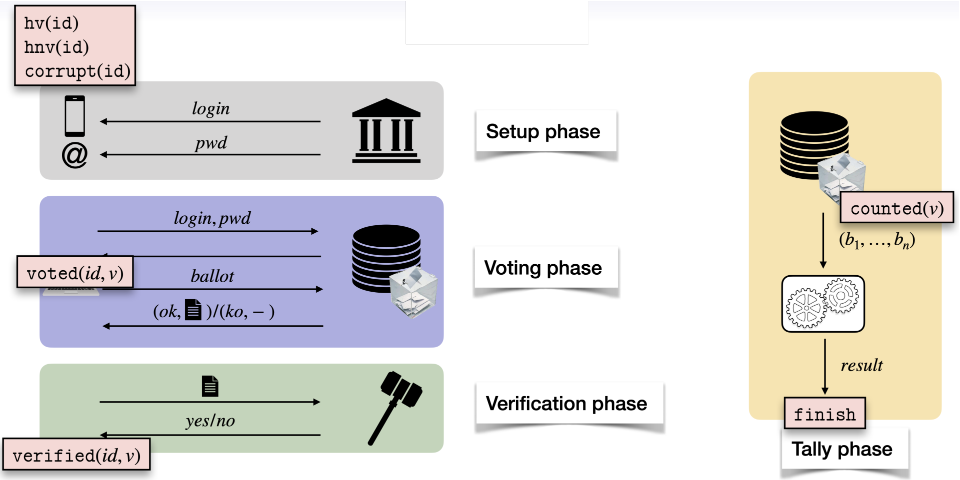
Example 2: injective events

$$\texttt{inj}-A(x) \Rightarrow \texttt{inj}-B(x)$$

✓ $\text{tr}_1 : B(0).B(0).A(0).A(0)$

✗ $\text{tr}_2 : B(0).A(0).A(0)$

# Voting protocol - overview




[slide borrowed from Alexandre Debant.]

# First contribution: exact characterization

**Theorem**

*E2E-verifiability* $\Leftrightarrow$ **query1** *and* **query2**

**query1** $\text{finish} \wedge \texttt{inj}-\text{verified}(z, x) \Rightarrow \texttt{inj}-\text{counted}(x)$

Intuition: *individual verifability*

# First contribution: exact characterization

**Theorem**

*E2E-verifiability* $\Leftrightarrow$ **query1** *and* **query2**

**query1** $\text{finish} \wedge \texttt{inj}-\text{verified}(z, x) \Rightarrow \texttt{inj}-\text{counted}(x)$

Intuition: *individual verifability*

**query2**
$$\begin{aligned} \text{finish} \wedge \texttt{inj}-\text{counted}(x) \quad \Rightarrow \quad & \texttt{inj}-\text{hv}(z) \wedge \text{verified}(z, x) \\ \vee \quad & \texttt{inj}-\text{hnv}(z) \wedge \text{voted}(z, x) \\ \vee \quad & \texttt{inj}-\text{corrupt}(z) \end{aligned}$$

Intuition: *extended universal verifiability*

# First contribution: exact characterization

**Theorem**

*E2E-verifiability* $\Leftrightarrow$ **query1** *and* **query2**

**query1** $\text{finish} \wedge \texttt{inj}-\text{verified}(z, x) \Rightarrow \texttt{inj}-\text{counted}(x)$

Intuition: *individual verifability*

**query2**
$$\begin{aligned} \text{finish} \wedge \texttt{inj}-\text{counted}(x) \quad \Rightarrow & \quad \texttt{inj}-\text{hv}(z) \wedge \text{verified}(z, x) \\ \vee & \quad \texttt{inj}-\text{hnv}(z) \wedge \text{voted}(z, x) \\ \vee & \quad \texttt{inj}-\text{corrupt}(z) \end{aligned}$$

Intuition: *extended universal verifiability*

Issue: make sure finish is executed only once <u>all</u> ballots are counted

# Second contribution: make it work in ProVerif

Protocol
Property

yes
attack
cannot be proved
time out

**Framework** for e-voting protocols

- 12 protocol specific processes
- 8 generic processes

**Library** for verifiability

- GSVerif-like axioms [CSF'18]
- 8 well-crafted lemmas

# Generic processes

```
let Tally(e_id) =
  in(cell_tally(e_id),i);
  event Tally_Read(e_id,i)
  if i = 0 then event finish(e_id)
  else
    get public_identifier_id(=e_id,=i,ident) in
    in(cell_tally_last_vote(e_id,ident),x);
    if x = empty_ballot then out(cell_tally(e_id),i-1)
    else
      Decrypt_Ballot(e_id,i,ident,x) |
      in(res_decrypt(e_id,i),v);
      event Counted(e_id,v);
      event CountedExtended(e_id,v,i,ident);
      out(c_pub,v); out(cell_tally(e_id),i-1).
```

```
let Voter(e_id) =
  in(c_pub,v);
  if is_valid(v) then
  get voting_data(e_id,v_idx,v_data) in
  in(cell_voter(e_id,v_idx),nb_vote);
  Voting(e_id,v_idx,nb_vote+1,v,v_data) |
  in(res_voting(e_id,v_idx),res_data);
  in(c_pub, is_last);
  if is_last then
    Final_Check(e_id,v_idx,v,v_data,res_data,nb_vote+1)
  else
    out(cell_voter(e_id,v_index),nb_vote+1).
```

# Protocol specific processes

```
let Voting(e_id,v_idx,nb_vote,v,voting_data) =
  get election_key(=e_id,_,pkE) in
  let (pseudo,c_auth) = voting_data in
  new r_ctxt;
  let ctxt = aenc(pkE,v,r_ctxt) in
  event voted(e_id,v_idx,v);
  let b = (pseudo,ctxt) in
  out(c_pub, b); out(c_auth, b);

  out(res_voting(e_id,v_idx,nb_vote),b).
```

# A library for verifiability

Axioms (correction guaranteed!)

- counter intervals
- term freshness

Generic lemmas

- to help with termination
- proved each time in ProVerif
- some using induction

| | Voter | Registrar (setup) | Server (1 CCR/M) | E2E Verifiability |
|---|---|---|---|---|
| Helios (toy ex.) | 😇 | – | 😇 | ✓ 16s |
| **Belenios tally** | 😇 | 😇 / 😈 | 😈 / 😇 | ✓ 24s |
| **Belenios last** | 😇 | 😇 | 😇 | ✗ 5s |
| **Belenios-counter last** | 😇 | 😇 | 😇 | ✗ 8s |
| Belenios-hash last | 😇 | 😇 / 😈 | 😈 / 😇 | ✓ 62s |
| **Swiss Post** | 😇 | 😇 | 😇 | ✓ 58s |
| **CHVote** | 😇 | 😇 | 😇 | ✓ 17s |

**in bold:** we used existing ProVerif files

# Conclusion

- Tally phase finally modeled!
- rather "plug and play" for existing ProVerif files
- make use of recent features of ProVerif (counters, lemmas, ...)

Future work

- extend to vote privacy
- could it apply to other tools such as Tamarin?