USENIX Security 2024

Election Eligibility with OpenID: Turning Authentication into Transferable Proof of Eligibility

Véronique Cortier, Alexandre Debant, **Anselme Goetschmann**, Lucca Hirschi

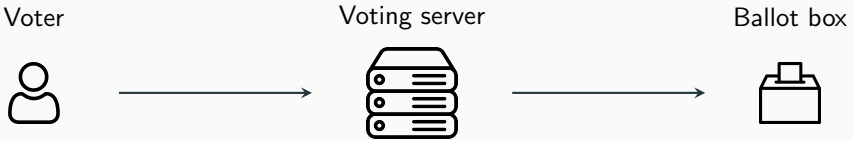
Thursday 15th August, 2024

Université de Lorraine, CNRS, Inria, LORIA, France

Context: online voting systems

Desired properties:

- vote privacy: "no one learns my vote"
- result integrity: result is the combination of cast ballots



Context: online voting systems

Desired properties:

- vote privacy: "no one learns my vote"
- result integrity: result is the combination of cast ballots

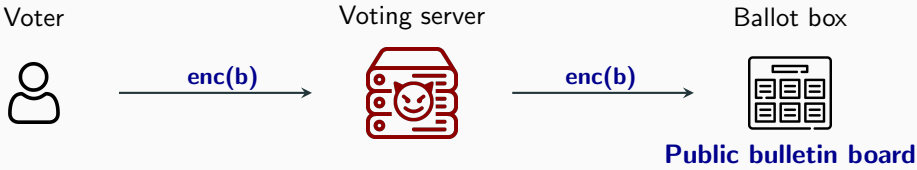
even against dishonest voting server

Context: online voting systems

Desired properties:

- vote privacy: "no one learns my vote"
- result integrity: result is the combination of cast ballots

} **even against dishonest voting server**

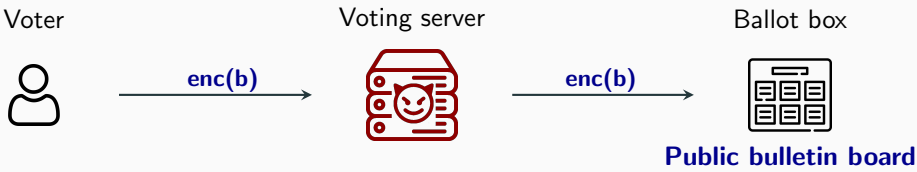


Context: online voting systems

Desired properties:

- vote privacy: "no one learns my vote"
- result integrity: result is the combination of cast ballots

even against dishonest voting server



Studied in academic research and used in real elections:

- Helios, Select, Belenios, ...
- Estonia, Australia, Switzerland, ...

Attack: ballot stuffing

Attack: ballot stuffing

Attack: ballot stuffing

What is missing: verifiable eligibility of ballots

Attack: ballot stuffing

What is missing: verifiable eligibility of ballots

Existing solutions:

- trust the voting server not to add ballots (Helios, France, Australia) **✗ not a solution**

Attack: ballot stuffing

What is missing: verifiable eligibility of ballots

Existing solutions:

- trust the voting server not to add ballots (Helios, France, Australia) ✗ **not a solution**
- sign ballots

Attack: ballot stuffing

What is missing: verifiable eligibility of ballots

Existing solutions:

- trust the voting server not to add ballots (Helios, France, Australia) **✗ not a solution**
- sign ballots
  - PKI (Estonia) **✗ rarely available**

Attack: ballot stuffing

What is missing: verifiable eligibility of ballots

Existing solutions:

- trust the voting server not to add ballots (Helios, France, Australia) **✗ not a solution**
- sign ballots
  - PKI (Estonia) **✗ rarely available**
  - ad-hoc: dedicated authority managing and distributing keys (Belenios, SwissPost)
    ✗ need extra entity and secure channel

Attack: ballot stuffing

What is missing: verifiable eligibility of ballots

Existing solutions:

- trust the voting server not to add ballots (Helios, France, Australia) **✗ not a solution**
- sign ballots
  - PKI (Estonia) **✗ rarely available**
  - ad-hoc: dedicated authority managing and distributing keys (Belenios, SwissPost)
    ✗ need extra entity and secure channel

  ✗ breaks everlasting privacy

Attack: ballot stuffing

What is missing: verifiable eligibility of ballots

Existing solutions:

- trust the voting server not to add ballots (Helios, France, Australia) **✗ not a solution**
- sign ballots
  - PKI (Estonia) **✗ rarely available**
  - ad-hoc: dedicated authority managing and distributing keys (Belenios, SwissPost)
    ✗ need extra entity and secure channel

  ✗ breaks everlasting privacy

Current problem: no existing *privacy-preserving* and *practical* solution

OpenID for e-voting

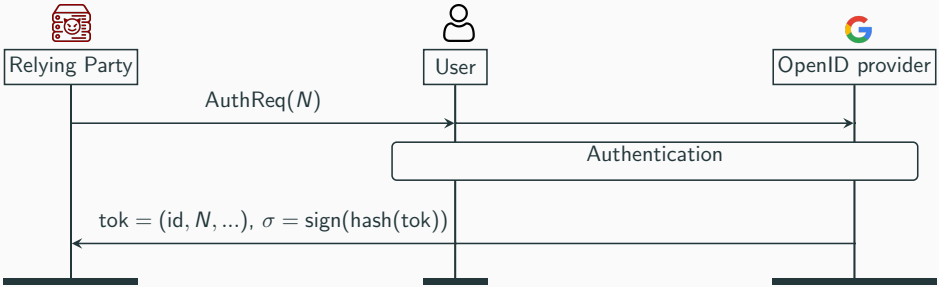
OpenID Connect:

- protocol to delegate authentication to identity provider
- widely deployed   ...

OpenID for e-voting

OpenID Connect:

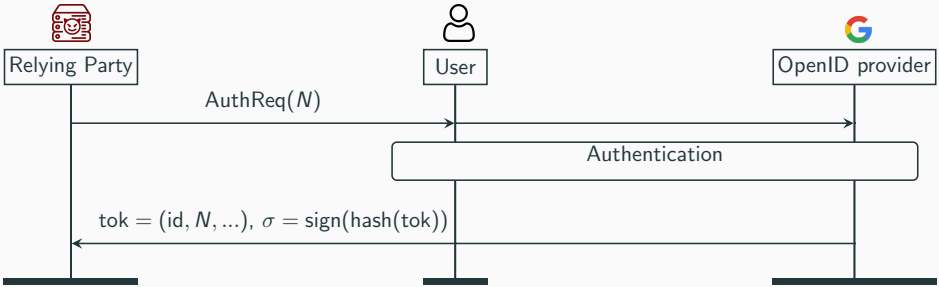
- protocol to delegate authentication to identity provider
- widely deployed · · ·



OpenID for e-voting

OpenID Connect:

- protocol to delegate authentication to identity provider
- widely deployed · · ·



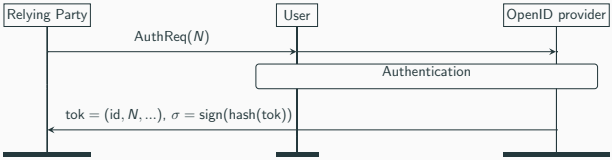
Problem for e-voting: non-transitive authentication

⇒ **not suitable for e-voting out-of-the-box**

Sign ballots with OpenID

Idea:

- set N = ballot
  $\Rightarrow$ **σ is linked to ballot**
- publish σ along ballot

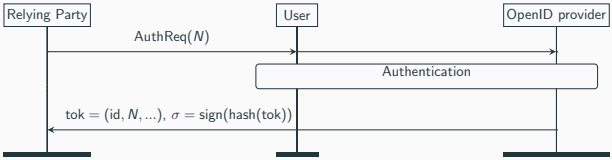


OpenID Connect flow

Sign ballots with OpenID

Idea:

- set N = ballot
  $\Rightarrow$ **σ is linked to ballot**
- publish σ along ballot



OpenID Connect flow

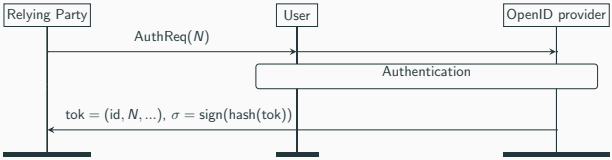
Challenges:

1. voter should control **when** ballot is validated

Sign ballots with OpenID

Idea:

- set N = ballot
  $\Rightarrow$ **σ is linked to ballot**
- publish σ along ballot



OpenID Connect flow

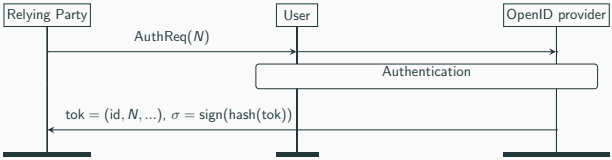
Challenges:

1. voter should control **when** ballot is validated
2. voter should control **which** ballot is signed

Sign ballots with OpenID

Idea:

- set N = ballot
  $\Rightarrow$ $\boldsymbol{\sigma}$ **is linked to ballot**
- publish σ along ballot



OpenID Connect flow

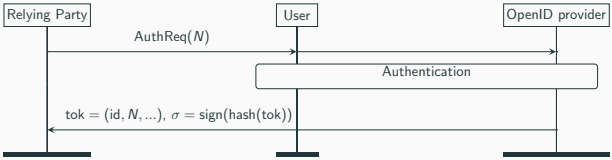
Challenges:

1. voter should control **when** ballot is validated
2. voter should control **which** ballot is signed
3. published data should not **reveal identity of voter**

Sign ballots with OpenID

Idea:

- set N = ballot
  $\Rightarrow$ **σ is linked to ballot**
- publish σ along ballot



OpenID Connect flow

Challenges:

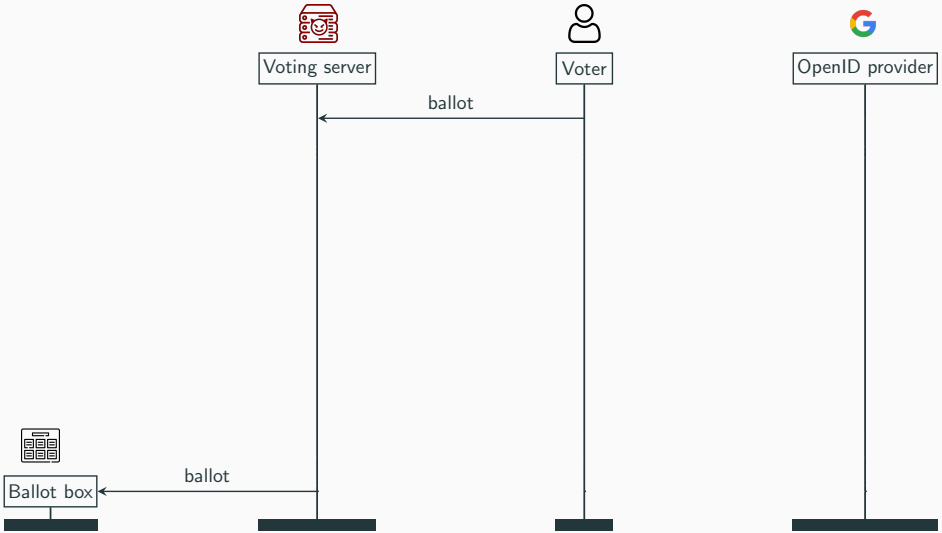
1. voter should control **when** ballot is validated
2. voter should control **which** ballot is signed

} **solved with OIDEli protocol**

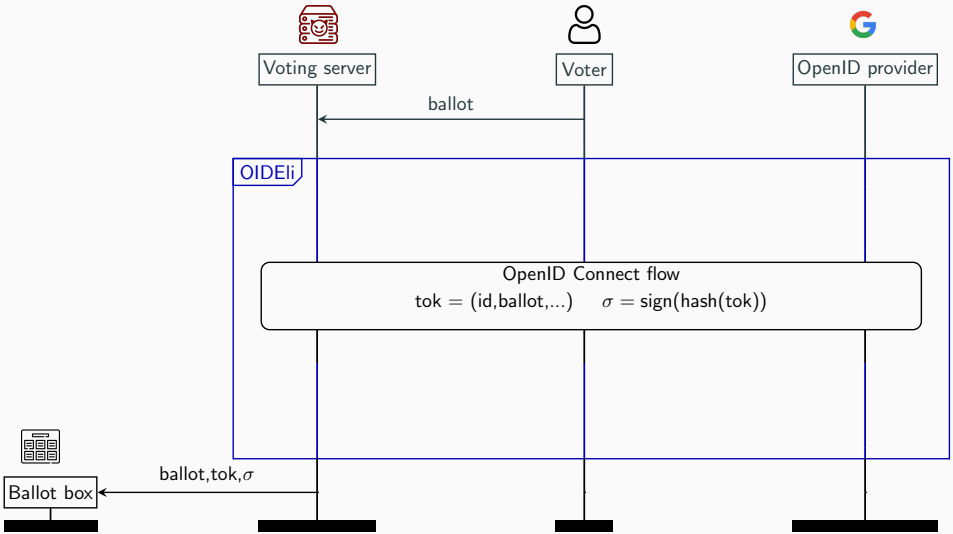
3. published data should not **reveal identity of voter**

} **solved with ZKP**

OIDEli protocol: OpenID for Eligibility

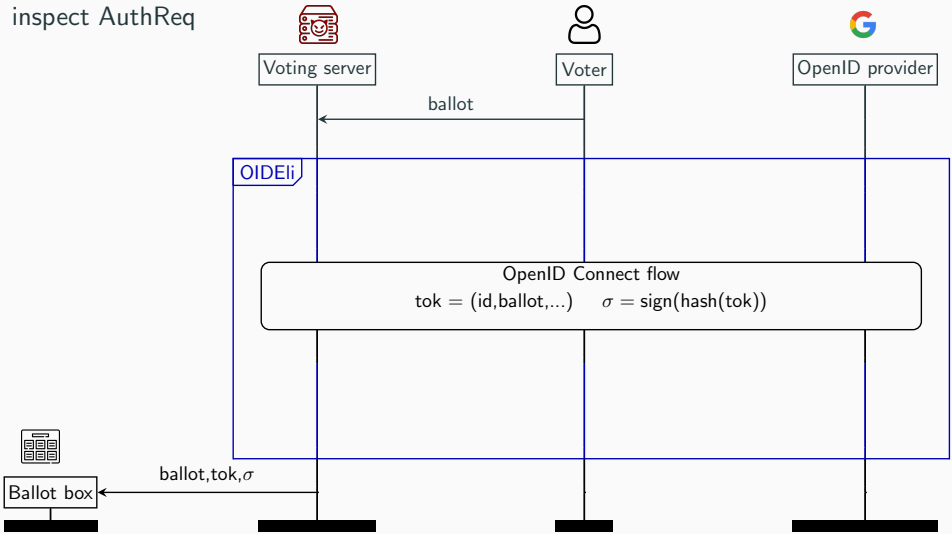


OIDEli protocol: OpenID for Eligibility



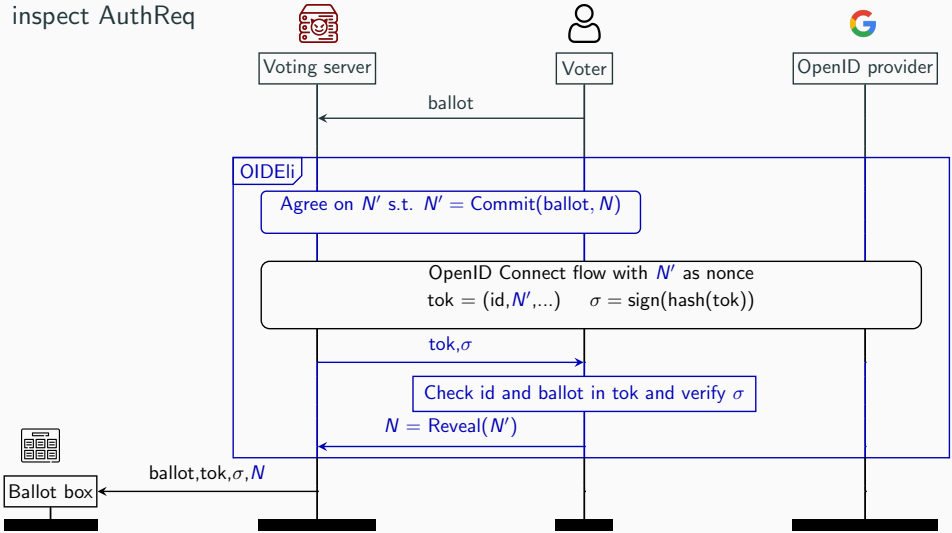
OIDEli protocol: OpenID for Eligibility

1. commit to ballot and reveal commitment after OpenID Connect flow
2. inspect AuthReq



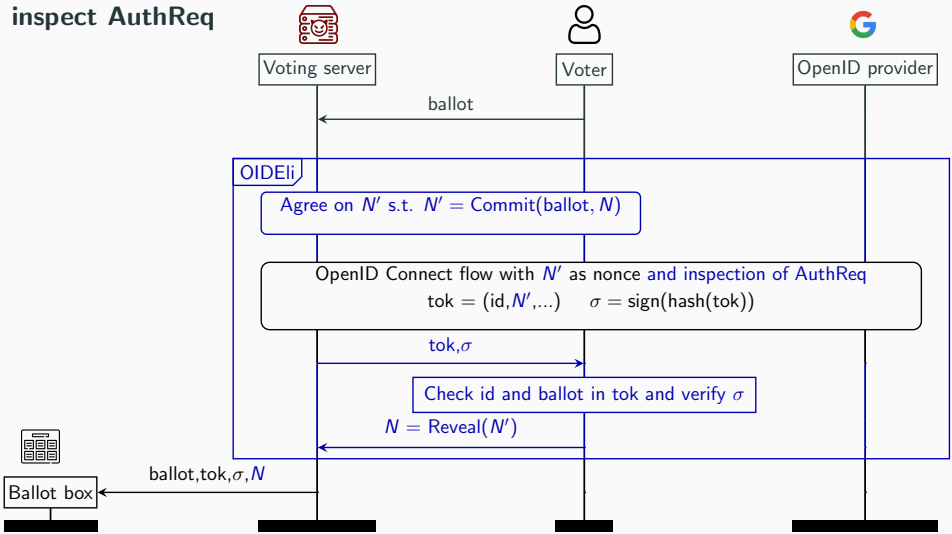
OIDEli protocol: OpenID for Eligibility

1. **commit to ballot and reveal commitment after OpenID Connect flow**
2. inspect AuthReq



OIDEli protocol: OpenID for Eligibility

1. commit to ballot and reveal commitment after OpenID Connect flow
2. **inspect AuthReq**

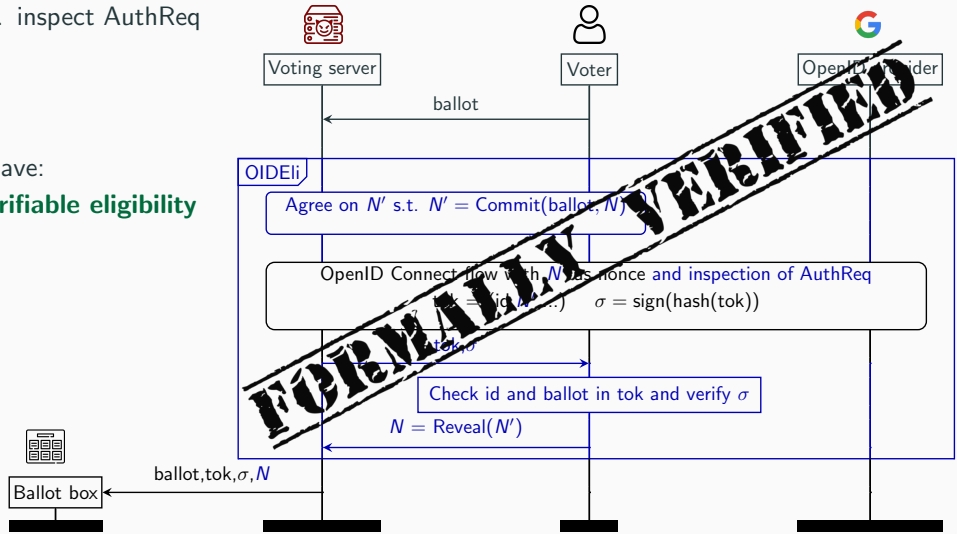


OIDEli protocol: OpenID for Eligibility

1. commit to ballot and reveal commitment after OpenID Connect flow
2. inspect AuthReq

We have:

✓ **verifiable eligibility**



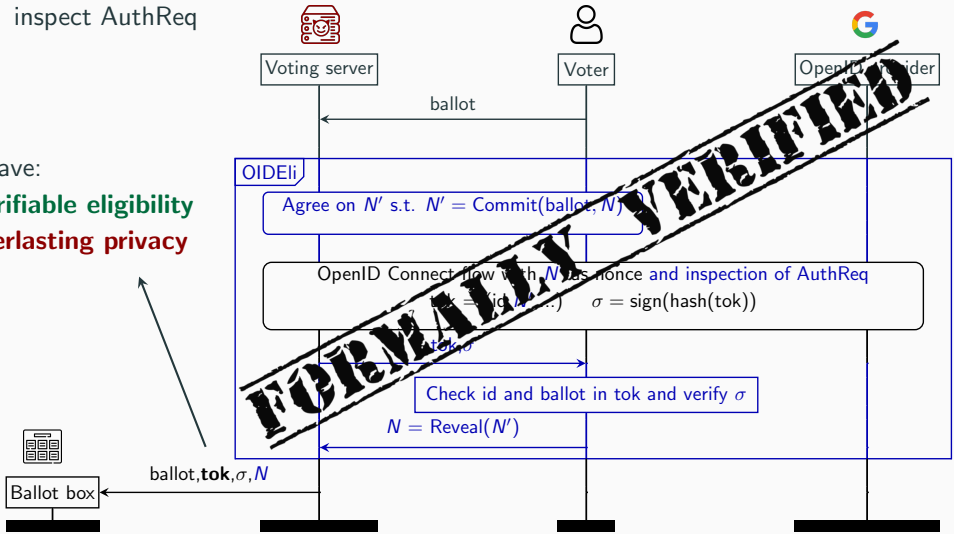
OIDEli protocol: OpenID for Eligibility

1. commit to ballot and reveal commitment after OpenID Connect flow
2. inspect AuthReq

We have:

✓ **verifiable eligibility**

✗ **everlasting privacy**



OIDEli with privacy

Idea: replace token containing voter identity with **proof of signature by eligible voter**

OIDEli with privacy

Idea: replace token containing voter identity with **proof of signature by eligible voter**

Content of ballot box:

ballot
N
$\text{tok} = (\text{id}, \text{Commit}(\text{ballot}, N), ...)$
$\sigma = \text{sign}(H),\ H = \text{hash}(\text{tok})$

OIDEli with privacy

Idea: replace token containing voter identity with **proof of signature by eligible voter**

Content of ballot box:

$$
\begin{array}{l}
\quad \text{ballot} \\
\quad N \\
- \ \text{tok} = (\text{id}, \text{Commit}(\text{ballot}, N), \ldots) \\
\quad \sigma = \text{sign}(H),\ H = \text{hash}(\text{tok}) \\
+ \ \pi = \text{ZKP}(\exists\ \text{id s.t. id} \in \text{Eligible} \wedge H = \text{hash}((\text{id}, \text{Commit}(\text{ballot}, N), \_))
\end{array}
$$

OIDEli ZKP

$$\mathrm{ZKP}(\exists \text{ id s.t. id} \in \text{Eligible} \wedge H = \textbf{SHA256}((\text{id}, \text{Commit}(\text{ballot}, N), \_))$$

Challenge: OpenID Connect relies on SHA256 $\Rightarrow$ **high proving time**
multiple instances of SHA256 $\Rightarrow$ **super-high proving time!**

OIDEli ZKP

$$\mathrm{ZKP}(\exists \text{ id s.t. id} \in \text{Eligible} \wedge H = \textbf{SHA256}((\text{id}, \text{Commit}(\text{ballot}, N), \_))$$

Challenge: OpenID Connect relies on SHA256 $\Rightarrow$ **high proving time**
multiple instances of SHA256 $\Rightarrow$ **super-high proving time!**

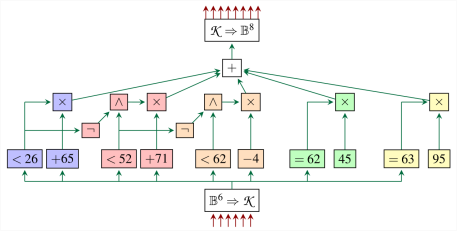
Solution: use **zk-SNARK** framework
optimize circuit (one SHA256 preimage)

$$\mathrm{ZKP}(\exists\ \mathrm{id}\ \text{s.t.}\ \mathrm{id} \in \mathrm{Eligible} \wedge H = \mathbf{SHA256}((\mathrm{id}, \mathrm{Commit}(\mathrm{ballot}, N), \_))$$

Challenge: OpenID Connect relies on SHA256 $\Rightarrow$ **high proving time**
multiple instances of SHA256 $\Rightarrow$ **super-high proving time!**

Solution: use **zk-SNARK** framework
optimize circuit (one SHA256 preimage)

In practice: implementation using Plonky2
✓ design remains framework agnostic



$$\mathrm{ZKP}(\exists\ \mathrm{id}\ \text{s.t.}\ \mathrm{id} \in \mathrm{Eligible} \wedge H = \textbf{SHA256}((\mathrm{id}, \mathrm{Commit}(\mathrm{ballot}, N), \_))$$

Challenge: OpenID Connect relies on SHA256 $\Rightarrow$ **high proving time**
multiple instances of SHA256 $\Rightarrow$ **super-high proving time!**

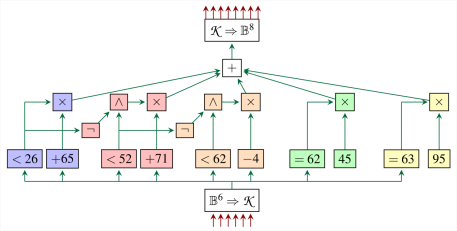
Solution: use **zk-SNARK** framework
optimize circuit (one SHA256 preimage)

In practice: implementation using Plonky2

- ✓ design remains framework agnostic

Performance:

- ✗ naive circuit: < 0.75 proof per hour
- ✓ our circuit: $\sim$1.3K proofs per hour
- ✓ highly parallelizable
- ✓ computation on server



Conclusion

✓ OIDEli protocol using OpenID to extend e-voting systems with verifiable eligibility

Conclusion

- ✓ OIDEli protocol using OpenID to extend e-voting systems with verifiable eligibility
- ✓ Design and optimization of zk-SNARK circuit needed for OIDEli

Conclusion

- ✓ OIDEli protocol using OpenID to extend e-voting systems with verifiable eligibility
- ✓ Design and optimization of zk-SNARK circuit needed for OIDEli
- ✓ Implementation of a proof of concept integrated in Belenios
- ✓ Formal security proofs

  Machine-checkable symbolic proofs using ProVerif

Conclusion

- ✓ OIDEli protocol using OpenID to extend e-voting systems with verifiable eligibility
- ✓ Design and optimization of zk-SNARK circuit needed for OIDEli
- ✓ Implementation of a proof of concept integrated in Belenios
- ✓ Formal security proofs

  Machine-checkable symbolic proofs using ProVerif

Thanks for your attention!