Bern University
of Applied Sciences

# Verifiable Internet Elections with Ever-lasting Privacy and Minimal Trust

*Rolf Haenni (co-work with Philipp Locher)*

Scytl, Barcelona, April 23, 2015

# Outline

# Vote Privacy Assumptions

"Any adversary is polynomial-time bounded."

"A threshold number of authorities is trustworthy."

# Protocol Overview

- Goal: Make vote privacy independent of
  - computational intractability assumptions
  - trusted authorities
- Involved parties
  - election administration
  - voters
  - public bulletin board
  - verifiers (the public)
- Cryptographic ingredients: perfectly hiding commitments, non-interactive zero-knowledge proofs (NIZKP)

# Step 1: Registration

The voter ...

- creates a pair of private and public credentials
- sends the public credential to the election administration (over an authentic channel)

# Step 2: Election Preparation

The election administration . . .

- publishes the list of public voter credentials on bulletin board

# Step 3: Vote Casting

The voter ...

- creates ballot consisting of
  - commitment to public credential
  - NIZKP that the commitment contains a valid public credential
  - NIZKP of knowing the corresponding private credential
  - vote
- sends ballot to bulletin board (over an anonymous channel)

# Step 4: Public Tallying

The verifier . . .

- retrieves the election data from bulletin board
- checks proofs contained in each ballot
- computes the election result

# Outline

- Introduction and Protocol Overview
- Cryptographic Preliminaries
    - Set Membership Proof
    - Representation Proof
- Detailed Protocol Description
- Properties and Extensions
- Performance and Implementation
- Conclusion

# Cryptographic Setup

- Let $\mathcal{G}_p$ be a cyclic group of prime order $p$ with independent generators $g_0, g_1$
- Let $\mathbb{G}_q \subset \mathbb{Z}_p^*$ be a sub-group of prime order $q \mid (p-1)$ with independent generators $h_0, h_1, \ldots, h_N$
- Assume that DL has no efficient solution in $\mathcal{G}_p$ and $\mathbb{G}_q$

# Pedersen Commitments

- Pedersen commitment over $\mathcal{G}_p$, for $u, r \in \mathbb{Z}_p$

$$\mathrm{com}_p(u, r) = g_0^r g_1^u$$

- Pedersen commitment over $\mathbb{G}_q$, for $v, s \in \mathbb{Z}_q$

$$\mathrm{com}_q(v, s) = h_0^s h_1^v$$
$$\mathrm{com}_q(v_1, \ldots, v_N, s) = h_0^s h_1^{v_1} \cdots h_N^{v_N}$$

- Perfectly hiding, computationally binding, homomorphic

# Non-Interactive Preimage Proofs

- Goal: prove knowledge of preimage of a given value

$$NIZKP[(a) : b = \phi(a)]$$

- Secret input
  - $a \in X$
- Public inputs
  - Homomorphic one-way function $\phi : X \to Y$
  - $b = \phi(a) \in Y$

- Standard construction
  - $\Sigma$-protocol
  - Fiat-Shamir heuristic using hash function
  - Proof transcript: $\pi = (t, s) \in Y \times X$

# Examples of Preimage Proofs

- Knowledge of discrete logarithm (Schnorr)

$$NIZKP[(a) : b = g^a]$$

- Equality of discrete logarithms (Chaum-Pedersen)

$$NIZKP[(a) : b_1 = g_1^a \wedge b_2 = g_2^a]$$

- Ability of opening a Pedersen commitment

$$NIZKP[(u, r) : c = \mathrm{com}_p(u, r)]$$

- Knowledge of ElGamal plaintext

$$NIZKP[(m, r) : e = \mathrm{ElGamal}_{pk}(m, r)]$$

# Set Membership Proof

# Set Membership Proof

- Goal: prove that a committed value belongs to a given set

$$NIZKP[(u, r) : c = \mathsf{com}_p(u, r) \wedge u \in U]$$

- Secret inputs
  - $u, r \in \mathbb{Z}_p$
- Public inputs
  - Commitment $c = \mathsf{com}_p(u, r) \in \mathcal{G}_p$
  - Set $U = \{u_1, \ldots, u_M\}$ of values $u_i \in \mathbb{Z}_p$

# General Construction

- Proposed by Brands et al. (2007)
- Let $P(X) = \prod_{i=1}^{M}(X - u_i)$ satisfying $P(u_i) = 0$ for all $u_i \in U$
- Set membership proof

$$NIZKP[(u, r) : c = \mathrm{com}_p(u, r) \wedge u \in U]$$
$$\iff$$
$$NIZKP[(u, r) : c = \mathrm{com}_p(u, r) \wedge P(u) = 0]$$

# Polynomial Evaluation Proof

- Polynomial evaluation proof by Bayer and Groth (2013)

  $$NIZKP[(u, r, v, s) : c = \mathrm{com}_p(u, r) \wedge d = \mathrm{com}_p(v, s) \wedge P(u) = v]$$

- Performance (for $v = s = 0$)
  - Transcript: $4 \log M$ elements of $\mathcal{G}_p$, $3 \log M$ elements of $\mathbb{Z}_p$
  - Generation: $O(M \log M)$
    $8 \log M$ exponentiations in $\mathcal{G}_p$, $2M \log M$ multiplications in $\mathbb{Z}_p$
  - Verification: $O(M)$
    $6 \log M$ exponentiations in $\mathcal{G}_p$, $3M$ multiplications in $\mathbb{Z}_p$

**Public Input:** $c = \mathrm{com}_p(u, r) \in \mathcal{G}_p$, $P(X) = \sum_{i=0}^{M} a_i X^i \in \mathbb{Z}_p[X]$
**Secret Input:** $u, r \in \mathbb{Z}_p$
**Generation:**

1. For $j = 1, \dots, m$, pick $r_j \in_R \mathbb{Z}_p$ and compute $c_j = \mathrm{com}_p(u^{2^j}, r_j)$.
2. For $j = 0, \dots, m$, pick $\bar{a}_j, \bar{r}_j \in_R \mathbb{Z}_p$ and compute $\bar{c}_j = \mathrm{com}_p(\bar{a}_j, \bar{r}_j)$.
3. Compute new polynomial
$$\tilde{P}(X) = \sum_{j=0}^{m} \tilde{a}_j X^j = \sum_{i=0}^{M} a_i \prod_{j=0}^{m} (u^{2^j} X + \bar{a}_j)^{i[j]} X^{1-i[j]} \in \mathbb{Z}_p[X]$$
of degree $m$. For $j = 0, \dots, m$, pick $\tilde{r}_j \in_R \mathbb{Z}_p$ and compute $\tilde{c}_j = \mathrm{com}_p(\tilde{a}_j, \tilde{r}_j)$.
4. For $j = 0, \dots, m-1$, compute $\hat{a}_j = u^{2^j} \bar{a}_j$, pick $\hat{r}_j \in_R \mathbb{Z}_p$, and compute $\hat{c}_j = \mathrm{com}_p(\hat{a}_j, \hat{r}_j)$.
5. Compute $x = h(c, a_0, \dots, a_M, c_1, \dots, c_m, \bar{c}_0, \dots, \bar{c}_m, \tilde{c}_0, \dots, \tilde{c}_m, \hat{c}_0, \dots, \hat{c}_{m-1})$.
6. For $j = 0, \dots, m$, compute $\bar{a}'_j = \bar{a}_j + x u^{2^j}$.
7. For $j = 0, \dots, m$, compute $\bar{r}'_j = \bar{r}_j + x r_j$.
8. For $j = 0, \dots, m-1$, compute $\hat{r}'_j = \hat{r}_j + x r_{j+1} - b_j r_j$.
9. Compute $\tilde{r}' = \sum_{j=0}^{m} \tilde{r}_j x^j$.

**Transcript:**
$(c_1, \dots, c_m, \bar{c}_0, \dots, \bar{c}_m, \tilde{c}_0, \dots, \tilde{c}_m, \hat{c}_0, \dots, \hat{c}_{m-1}, \bar{a}'_0, \dots, \bar{a}'_m, \bar{r}'_0, \dots, \bar{r}'_m, \hat{r}'_0, \dots, \hat{r}'_{m-1}, \tilde{r}')$

**Verification:**

1. Compute $x = h(c, a_0, \dots, a_M, c_1, \dots, c_m, \bar{c}_0, \dots, \bar{c}_m, \tilde{c}_0, \dots, \tilde{c}_m, \hat{c}_0, \dots, \hat{c}_{m-1})$.
2. For $j = 0, \dots, m$, check $c_j^x \bar{c}_j = \mathrm{com}_p(\bar{a}'_j, \bar{r}'_j)$.
3. For $j = 0, \dots, m-1$, check $c_{j+1}^x \hat{c}_j = c_j^{\bar{a}'_j} \cdot \mathrm{com}_p(0, \hat{r}'_j)$.
4. Check
$$\prod_{j=0}^{m} \tilde{c}_j^{x^j} = \mathrm{com}_p\left( \sum_{i=0}^{M} a_i \prod_{j=0}^{m} \bar{a}'^{\,i[j]}_j x^{1-i[j]}, \tilde{r}' \right).$$

# Representation Proof

# DL-Representation

- Let $\mathbb{G}_q \subset \mathbb{Z}_p^*$ be a cyclic group of order $q$ and $h_1, \ldots, h_N \in \mathbb{G}_q$
- A tuple $(v_1, \ldots, v_N) \in \mathbb{Z}_q^N$ is a *DL-representation* of $u \in \mathbb{G}_q$ relative to $h_1, \ldots, h_N$, if

$$u = h_1^{v_1} \cdots h_N^{v_N}$$

- Note that $\mathbb{G}_q \subset \mathbb{Z}_p^* \subset \mathbb{Z}_p$ implies $u \in \mathbb{Z}_p$

# Representation Proof

- Goal: prove that a commitment contains a DL-representation of another committed value

$$NIZKP[(u, r, v_1, \ldots, v_N, s) : c = \mathrm{com}_p(u, r) \wedge d = \mathrm{com}_q(v_1, \ldots, v_N, s) \wedge u = h_1^{v_1} \cdots h_N^{v_N}]$$

- Secret inputs
  - $u, r \in \mathbb{Z}_p$
  - $v_1, \ldots, v_N, s \in \mathbb{Z}_q$
- Public inputs
  - Commitment $c = \mathrm{com}_p(u, r) \in \mathcal{G}_p$
  - Commitment $d = \mathrm{com}_q(v_1, \ldots, v_N, s) \in \mathbb{G}_q$

# Representation Proof

- Au, Susilo, Mu (2010) proposed an extension of the double discrete logarithm proof by Camenisch and Stadler (1997)
- Let $K$ be a security parameter (e.g. $K = 80$)
- Performance
  - Transcript: $K$ elements of $\mathcal{G}_p$, $\mathbb{G}_q$, $\mathbb{Z}_p$, $KN$ elements of $\mathbb{Z}_q$
  - Generation and verification: $O(KN)$
    $2K$ exponentiations in $\mathcal{G}_p$, $KN$ exponentiations in $\mathbb{G}_q$

**Public Input:** $c = \text{com}_p(u, r) \in \mathcal{G}_p$, $d = \text{com}_q(v_1, \ldots, v_N, s) \in \mathbb{G}_q$
**Secret Input:** $u, r \in \mathbb{Z}_p$, $v_1, \ldots, v_N, s \in \mathbb{Z}_q$
**Generation:**

1. Pick $\bar{u}, \bar{r} \in_R \mathbb{Z}_p$ and compute $\bar{c} = \text{com}_p(\bar{u}, \bar{r})$.
2. For $j = 1, \ldots, K$,
   (a) pick $\bar{v}_{1,j}, \ldots, \bar{v}_{N,j} \in_R \mathbb{Z}_q$ and compute $\bar{u}_j = h_1^{\bar{v}_{1,j}} \cdots h_N^{\bar{v}_{N,j}}$,
   (b) pick $\bar{r}_j \in_R \mathbb{Z}_p$ and compute $\bar{c}_j = \text{com}_p(\bar{u}_j, \bar{r}_j)$,
   (c) pick $\bar{s}_j \in_R \mathbb{Z}_q$ and compute $\bar{d}_j = \text{com}_q(\bar{v}_{1,j}, \ldots, \bar{v}_{N,j}, \bar{s}_j)$.
3. Compute $x = h(c, d, \bar{c}, \bar{c}_1, \ldots, \bar{c}_k, \bar{d}_1, \ldots, \bar{d}_k)$.
4. Compute $\bar{u}' = \bar{u} - xu$ and $\bar{r}' = \bar{r} - xr$.
5. For $j = 1, \ldots, K$,
   (a) for $i = 1, \ldots, N$, compute $\bar{v}'_{i,j} = \bar{v}_{i,j} - x[j]v_i$,
   (b) compute $\bar{r}'_j = \bar{r}_j - x[j] \cdot \text{com}_q(\bar{v}'_{1,j}, \ldots, \bar{v}'_{N,j}, r)$,
   (c) compute $\bar{s}'_j = \bar{s}_j - x[j]s$.

**Transcript:**

$$(\bar{c}, \bar{c}_1, \ldots, \bar{c}_k, \bar{d}_1, \ldots, \bar{d}_k, \bar{u}', \bar{r}', \bar{v}'_{1,1}, \ldots, \bar{v}'_{N,K}, \bar{r}'_1, \ldots, \bar{r}'_k, \bar{s}'_1, \ldots, \bar{s}'_k)$$

**Verification:**

1. Compute $x = h(c, d, \bar{c}, \bar{c}_1, \ldots, \bar{c}_k, \bar{d}_1, \ldots, \bar{d}_k)$.
2. Check $\bar{c} = c^x \cdot \text{com}_p(\bar{u}', \bar{r}')$.
3. For $j = 1, \ldots, K$,
   (a) check $\bar{d}_j = d^{x[j]} \cdot \text{com}_q(\bar{v}'_{1,j}, \ldots, \bar{v}'_{N,j}, \bar{s}'_j)$,
   (b) compute $\bar{u}'_j = h_1^{\bar{v}'_{1,j}} \cdots h_N^{\bar{v}'_{N,j}}$, and check

$$\bar{c}_j = \begin{cases} \text{com}_p(\bar{u}'_j, \bar{r}'_j), & \text{if } x[j] = 0, \\ c^{\bar{u}'_j} \cdot \text{com}_p(0, \bar{r}'_j), & \text{if } x[j] = 1. \end{cases}$$

# Outline

# Step 1: Registration

The voter . . .

- creates a pair of private and public credentials
- sends the public credential to the election administration (over an authentic channel)

# Step 1: Registration

The voter ...

- creates a pair of private and public credentials

$$\alpha, \beta \in_R \mathbb{Z}_q$$
$$u = h_1^{\alpha} h_2^{\beta} \in \mathbb{G}_q$$

- sends the public credential $u$ to the election administration (over an authentic channel)

# Step 2: Election Preparation

The election administration . . .

- publishes the list of public voter credentials on bulletin board

# Step 2: Election Preparation

The election administration ...

- defines the list of public voter credentials $U = \{u_1, \ldots, u_M\}$
- computes coefficients $a_0, \ldots, a_M$ of polynomial

$$P(X) = \prod_{i=1}^{M}(X - u_i) = \sum_{i=0}^{M} a_i X^i$$

- selects independent election generator $\hat{h} \in \mathbb{G}_q$
- publishes $(U, a_0, \ldots, a_M, \hat{h})$ on bulletin board

# Step 3: Vote Casting

The voter ...

- creates ballot consisting of
    - commitment the public credential
    - NIZKP that the commitment contains a valid public credential
    - NIZKP of knowing the corresponding private credential
    - vote
- sends ballot to bulletin board (over an anonymous channel)

# Step 3: Vote Casting

The voter ...

- creates ballot $B = (c, d, e, \hat{u}, \pi_1, \pi_2, \pi_3)$ consisting of
  - commitment to public credential $c = \mathrm{com}_p(u, r)$

$$\pi_1 = NIZKP[(u, r) : c = \mathrm{com}_p(u, r) \wedge P(u) = 0]$$

  - commitment to private credential $d = \mathrm{com}_q(\alpha, \beta, s)$

$$\pi_2 = NIZKP[(u, r, \alpha, \beta, s) : c = \mathrm{com}_p(u, r) \wedge d = \mathrm{com}_q(\alpha, \beta, s) \wedge u = h_1^{\alpha} h_2^{\beta}]$$

  - vote $e$
  - election credential $\hat{u} = \hat{h}^{\beta}$

$$\pi_3 = NIZKP[(\alpha, \beta, s) : d = \mathrm{com}_q(\alpha, \beta, s) \wedge \hat{u} = \hat{h}^{\beta}]$$

- sends ballot $B$ to bulletin board (over an anonymous channel)

# Step 4: Public Tallying

The verifier ...

- retrieves the election data from bulletin board
- checks proofs contained in each ballot
- computes the election result

# Step 4: Public Tallying

The verifier . . .

- retrieves the election data from bulletin board

$$U, a_0, \ldots, a_M, \hat{h}, \mathcal{B}$$

- checks proofs $\pi_1, \pi_2, \pi_3$ contained in each ballot $B \in \mathcal{B}$
- detects ballots with identical values $\hat{u}$ and resolve conflicts
- computes the election result from votes $v$ contained in $\mathcal{B}' \subseteq \mathcal{B}$

# Outline

# Adversary Model

- Present adversaries are polynomial-time bounded and thus ...
  - unable to solve DL efficiently in $\mathcal{G}_p$ and $\mathbb{G}_q$
  - unable to compute $\text{hash}^{-1}(h)$
- Future adversaries will have unrestricted computational resources and are therefore
  - able to solve DL efficiently in $\mathcal{G}_p$ and $\mathbb{G}_q$
  - able to compute $\text{hash}^{-1}(h)$

# Correctness

Attack by present adversary (during or shortly after election)

- Case 1: Present adversary $\neq$ voter
  - Find representation $(\alpha', \beta')$ for some $u \in U$
    $\rightarrow$ equivalent to solving DL
  - Simulate $\pi_1, \pi_2, \pi_3$ without valid secret inputs $(\alpha', \beta')$
    $\rightarrow$ equivalent to solving DL or inverting hash function
- Case 2: Present adversary $=$ voter
  - Use different $\beta' \neq \beta$ in a second ballot and simulate $\pi_3$
    $\rightarrow$ equivalent to solving DL or inverting hash function

# Privacy

Attack by future adversary (possibly in the far future)

- For every $B = (c, d, e, \hat{u}, \pi_1, \pi_2, \pi_3) \in \mathcal{B}$
  - compute $\beta$ satisfying $\hat{u} = \hat{h}^{\beta}$
  - compute $(\alpha', \beta)$ satisfying $u' = h_1^{\alpha'} h_2^{\beta}$ for every $u' \in U$
- Therefore, uncovering $\beta$ from every ballot does not reveal anything about the links between $\mathcal{B}$ and $U$
- Note that $c, d$ are perfectly hiding and $\pi_1, \pi_2, \pi_3$ are perfect zero-knowledge

# Extensions

- To achieve fairness, the vote must be encrypted
  - Generate encryption key pair $(sk, pk)$ during election preparation
  - Encrypt vote using $pk$ during vote casting
  - Publish $sk$ to initiate public tallying
- Extended credentials are required to vote multiple times
  - Private credentials $(\alpha, \beta_1, \dots, \beta_L)$
  - Public credentials $u = h_1^{\alpha} h_2^{\beta_1} \cdots h_{L+1}^{\beta_L}$
  - Use different $\beta_i$ for each election
- To allow vote updating, some other minor adjustments are necessary

# Outline

- Introduction and Protocol Overview
- Cryptographic Preliminaries
  - Set Membership Proof
  - Representation Proof
- Detailed Protocol Description
- Properties and Extensions
- Performance and Implementation
- Conclusion

# Ballot Size

| Ballot Component | Elements of $\mathcal{G}_p$ | Elements of $\mathbb{Z}_p, \mathbb{G}_q$ | Elements of $\mathbb{Z}_q$ |
|---|---|---|---|
| $c, d, \hat{u}$ | 1 | 2 | – |
| $\pi_1$ | $4\lfloor \log M \rfloor + 2$ | $3\lfloor \log M \rfloor + 3$ | – |
| $\pi_2$ | $K + 1$ | $2K + 2$ | $K(L + 2)$ |
| $\pi_3$ | – | 2 | 4 |
| Entire Ballot | $4\lfloor \log M \rfloor + K + 4$ | $3\lfloor \log M \rfloor + 2K + 9$ | $KL + 2K + 4$ |

Table 1: Ballot size as a function of $M$, $K$, and $L$ (without encrypted vote and proof of known plaintext of the encrypted vote). Elements of $\mathbb{Z}_p$ and $\mathbb{G}_q$ are counted together.

# Ballot Size

| $M = \|U\|$ | Elements of $\mathcal{G}_p$ | Elements of $\mathbb{Z}_p, \mathbb{G}_q$ | Elements of $\mathbb{Z}_q$ | Single Ballot | $M$ Ballots |
|---:|:---:|:---:|:---:|:---:|---:|
| 10 | 96 | 178 | 244 | 39.0 KB | 0.4 MB |
| 100 | 108 | 187 | 244 | 41.6 KB | 4.1 MB |
| 1'000 | 120 | 196 | 244 | 44.3 KB | 43.2 MB |
| 10'000 | 136 | 208 | 244 | 47.8 KB | 466.5 MB |
| 100'000 | 148 | 217 | 244 | 50.4 KB | 4.8 GB |
| 1'000'000 | 164 | 229 | 244 | 53.9 KB | 51.4 GB |

Table 2: Ballot size for different numbers of voters and parameters $K = 80$, $L = 1$, $|p| = 1024$, and $|q| = 160$.

# Cost of Ballot Generation

| Ballot Component | Exponentiations in $\mathcal{G}_p$ | Exponentiations in $\mathbb{G}_q$ | Multiplications in $\mathbb{Z}_p$ |
|---|---|---|---|
| $c, d, \hat{u}$ | 2 | 4 | – |
| $\pi_1$ | $8\lfloor \log M \rfloor + 4$ | – | $2M\lfloor \log M \rfloor$ |
| $\pi_2$ | $2K + 2$ | $K(L+2)$ | – |
| $\pi_3$ | – | 4 | – |
| Entire Ballot | $8\lfloor \log M \rfloor + 2K + 8$ | $KL + 2K + 8$ | $2M\lfloor \log M \rfloor$ |

Table 3: Number of exponentiations and multiplications required to generate a single ballot (without encrypted vote and proof of known plaintext of the encrypted vote).

# Cost of Ballot Generation

| $M = \|U\|$ | Exponentiations in $\mathcal{G}_p$ | Exponentiations in $\mathbb{G}_q$ | Multiplications in $\mathbb{Z}_p$ | Estimated Time (Single Ballot) |
|---:|:---:|:---:|---:|:---:|
| 10 | 192 | 248 | 60 | 0.7 sec. |
| 100 | 216 | 248 | 1'200 | 0.7 sec. |
| 1'000 | 240 | 248 | 18'000 | 0.9 sec. |
| 10'000 | 272 | 248 | 260'000 | 2.2 sec. |
| 100'000 | 296 | 248 | 3'200'000 | 17.0 sec. |
| 1'000'000 | 328 | 248 | 40'000'000 | 3.4 min. |

Table 4: Cost of ballot generation for different numbers of voters and parameters $K = 80$, $L = 1$, $|p| = 1024$, and $|q| = 160$. The time estimates are based on 350 exponentiations per second in $\mathcal{G}_p$, 2'000 exponentiations per second in $\mathbb{G}_q$, and 200'000 multiplications per second in $\mathbb{Z}_p$.

# Cost of Ballot Verification

| Ballot Component | Exponentiations in $\mathcal{G}_p$ | Exponentiations in $\mathbb{G}_q$ | Multiplications in $\mathbb{Z}_p$ |
|---|---|---|---|
| $\pi_1$ | $6\lfloor \log M \rfloor + 6$ | – | $2M$ |
| $\pi_2$ | $2K + 1$ | $K(L+2)$ | – |
| $\pi_3$ | – | $6$ | – |
| Total | $6\lfloor \log M \rfloor + 2K + 7$ | $KL + k + 6$ | $2M$ |

Table 5: Number of exponentiations and multiplications required to verify a single ballot (without proof of known plaintext of the encrypted vote).

# Cost of Ballot Verification

| $M = \lvert U \rvert$ | Exponentiations in $\mathcal{G}_p$ | Exponentiations in $\mathbb{G}_q$ | Multiplications in $\mathbb{Z}_p$ | Estimated Time (Single Ballot) | Estimated Time ($M$ Ballots) |
|---:|:---:|:---:|---:|:---:|---:|
| 10 | 185 | 166 | 30 | 0.6 sec. | 6.1 sec. |
| 100 | 203 | 166 | 300 | 0.7 sec. | 1.1 min. |
| 1'000 | 221 | 166 | 3'000 | 0.7 sec. | 12.2 min. |
| 10'000 | 245 | 166 | 30'000 | 0.9 sec. | 2.6 hours |
| 100'000 | 263 | 166 | 300'000 | 2.3 sec. | 64.8 hours |
| 1'000'000 | 287 | 166 | 3'000'000 | 15.9 sec. | 4417.5 hours |

Table 6: Cost of ballot verification for different numbers of voters and parameters $K = 80$, $L = 1$, $\lvert p \rvert = 1024$, and $\lvert q \rvert = 160$. The time estimates are based on 350 exponentiations per second in $\mathcal{G}_p$, 2'000 exponentiations per second in $\mathbb{G}_q$, and 200'000 multiplications per second in $\mathbb{Z}_p$.

# Time Measurements with UniCrypt

| $M = \|U\|$ | Ballot Generation | Ballot Verification |
|---:|:---:|:---:|
| 10 | 1.3 sec. | 0.9 sec. |
| 100 | 1.4 sec. | 1.0 sec. |
| 1'000 | 1.6 sec. | 1.1 sec. |
| 10'000 | 3.0 sec. | 1.3 sec. |
| 100'000 | 18.2 sec. | 2.9 sec. |
| 1'000'000 | 3.3 min. | 18.8 sec. |

Table 7: Actual running times for generating and verifying a single ballot using the UniCrypt library.

# Outline

- Introduction and Protocol Overview
- Cryptographic Preliminaries
  - Set Membership Proof
  - Representation Proof
- Detailed Protocol Description
- Properties and Extensions
- Performance and Implementation
- Conclusion

# Summary

- New approach based on different cryptographic primitives
- Pros
  - Everlasting privacy
  - No trusted authorities (except for fairness)
  - Simplicity of voting process
  - Implementation available in UniCrypt
- Cons
  - Anonymous channel required for vote casting
  - Relatively expensive ballot generation/verification
  - Restricted scalability

# Outlook

- Optimize the implementation
  - multi-exponentiation
  - fix-base exponentiation
  - parallel execution on multiple cores
  - use polynomial evaluation proof by Brands et al. (2007) when $M$ gets very large
- Add receipt-freeness (we have a solution!) or coercion-resistance
- Generate return codes?