

Advanced Voting

Panagiotis Grontas

16/05/2024

NTUA - Advanced topics in cryptography (2023-2024)

- Electronic voting with cryptography is quite old
 - First reference: Chaum (1981)
 - First E-voting PhD: Benaloh (1986)
 - Recall: DH Key Exchange (1976)
 - Recall: RSA (1978)
- Why can't we vote electronically (online) after 40 years?
 - Note: Efficiency reasons have been solved

Because of a set of conflicting properties

Verifiability The most important advantage over traditional elections

- Individual
- Universal
- Eligibility
- E2E Verifiability

Privacy

- Ballot secrecy
- Receipt freeness
- Coercion resistance
- Everlasting privacy
- Participation privacy

Other properties:

- Accountability
- Efficiency
- Fairness
- Robustness
- Usability

The Helios Voting System

- Electronic elections [A08] in the browser
 - E2E Verifiable: All can check that every vote is included in the tally unaltered
 - Open-Audit: Public and independent access to all election data
- Many elections: IACR, ACM, Universities etc.
 - 2.000.000 votes cast so far - heliosvoting.org
- Based on well known cryptographic protocols
 - Sako-Killian Mixnet (Eurocrypt '95) - Helios 1.0
 - CGS homomorphic tallying (Eurocrypt '97) - Helios 2.0
 - Added Cast-As-Intended Verifiability (Benaloh challenge)
- Many variations: Zeus, Helios-C (Belenios)

Characteristics

- Use of Non-Interactive Σ protocols for verifiability
- Force the EA and corrupted voters to follow the protocol
- Distributed Decryption
 - Votes are encrypted on the client
 - No decryption key leaves each trustee's computer
 - The Helios Server sees only the result
- **Trust no one for integrity, trust the trustee's for privacy**

Disadvantages

- Untrusted clients: A corrupted computer can ultimately display whatever it wants, despite auditing
- Few guarantees against coercion in the unsupervised setting (Countermeasure: Last vote counts)
- Assumption: The voter has access to a trusted computer at some point before the election ends

Participants

- Election administrator: Create the election, add the questions, combine partial tallies
- Bulletin' Board BB: Maintain votes (BTC) and audit data
- Voter V_{EL} : Eligible voters **optionally** identified by random alias or external authentication service (Google, Facebook, LDAP)
- Authenticated channel between voter and BB
- Trustees (Talliers) TA: Partially decrypt individual (in Helios 1.0) or aggregated (in Helios 2.0) ballots
- Registrars (Helios-C) RA: Generate cryptographic credentials for voters
- $EA = (RA, TA, BB)$

- Cast as intended: Benaloh challenge
 - After ballot creation (encryption) but *before* authentication, each voter can choose if they will audit or cast the ballot
 - On audit: Helios releases the encryption randomness and the voter can recreate the ballot using software of their choice
 - An audited ballot cannot be submitted
- Recorded as cast:
 - Each encrypted ballot and related data are hashed to a tracking number
 - Check if assigned number exists in the Ballot Tracking Center (BTC)

- Talled as recorded:
 - Retrieve ballots from (BTC)
 - Compare identities with eligible voters (if applicable)
 - Recompute tracking numbers and verify proofs
 - Aggregate the ballots and check equality with official encrypted tally before decryption
 - Verify decryption proofs

Individual verifiability: Verify cast as intended / recorded as cast

Universal verifiability: tallied as recorded

Eligibility verifiability

Model of Helios

$\mathcal{V}S^{\text{Helios}} =$

(Setup, SetupElection, Vote, Append, Valid, VerifyVote, Publish, Tally, Verify)

- **Setup**(1^λ) = $(\mathbb{G}, q, g, H : \{0, 1\}^* \rightarrow \mathbb{Z}_q, \text{BB} = \emptyset, (\text{DLProve}, \text{DLVerify})(\text{EqDLProve}, \text{EqDLVerify}), (\text{DisjProve}, \text{DisjVerify}))$
where:

- \mathbb{G} is a group where the DDH is hard (for ElGamal encryption)
- Computationally Sound and Honest Verifier Zero Knowledge (Non-Interactive using Fiat-Shamir Heuristic)
 - (DLProve, DLVerify) = $\text{NIZK}_H\{(g, \text{pk}), (\text{sk}) : \log_g \text{pk} = \text{sk}\}$
 - (EqDLProve, EqDLVerify) = $\text{NIZK}_H\{(g, \text{pk}, h, R), (\text{sk}) : \log_g \text{pk} = \log_h R\}$
 - (DisjProve, DisjVerify) = $\text{NIZK}_H\{(g, \text{pk}, R, S), (r) : (R, S) = \text{Enc}_{\text{pk}}(g^0) \text{ OR } (R, S) = \text{Enc}_{\text{pk}}(g^1)\}$

- **SetupElection** = $(sk \leftarrow \mathbb{Z}_q, pk = g^{sk}, V_{EL}, CS = \{0, 1\})$

$BB \leftarrow \{pk, V_{EL}, CS, H(pk||V_{EL}||CS)\}$

Distributed Key Generation:

- Each member of the TA: $sk_i \leftarrow \mathbb{Z}_q$
- Publish $pk_i := g^{sk_i}, \text{DLProve}(g, pk_i, sk_i)$
- $pk := \prod_{TA} pk_i$

Distributed Decryption of (R, S) :

- Each member of the TA computes:
 $D_i := R^{sk_i}, \text{EqDLProve}(g, pk_i, R, D_i, sk_i)$
- Plaintext $S / \prod_{TA} D_i$

Security analysis: TA modelled as a single entity

Formal Description iii

- **Vote**(i, v) :
 - $(R, S) = (g^r, g^v \text{pk}^r) = \text{Enc}_{\text{pk}}(g^v, r)$, $r \leftarrow \mathbb{Z}_q$
 - $\pi_{\text{Vote}} = \text{DisjProve}(g, \text{pk}, R, S, r)$
- **Valid**(b) $\in \{0, 1\}$:
Return 1 iff $i \in V_{\text{EL}}$ **AND** $\text{DisjVerify}(\pi_{\text{Vote}}) = 1$
well-formed ballots
- **Append**(b, BB)
If **Valid**(b) = 1 then the ballot is post on the BB
Some other checks might also be performed (i.e. check if there is an identical ballot)
well-formed BB contains only valid ballots
- **VerifyVote**(BB, b) $\in \{0, 1\}$ Check if $b \in \text{BB}$
- **Publish**(BB) = PBB where $\text{PBB} = \{(R, S), \pi_{\text{Vote}}\}$
only the last unique ballots appear for each voter without any id
typically occurs after all voters have voted

- **Tally**(PBB, sk)
 - Validates all ballots in BB
 - $(R_\Sigma, S_\Sigma) := \prod_{b \in \text{PBB}} (R_b, s_b)$
 - $g^t := \text{Dec}_{\text{sk}}(R_\Sigma, S_\Sigma)$
 - Compute small t
 - $\pi_{\text{Tally}} = \text{EqDLProve}(g, \text{pk}, R_\Sigma, S_\Sigma g^{-t}, \text{sk})$
- **Verify**(PBB, t , π_{Tally})
 - Check correct construction of PBB (last vote counts, no duplicate ciphertexts, $i \in L$, valid π_{Vote} for all ballots)
 - $(R_\Sigma, S_\Sigma) := \prod_{b \in \text{PBB}} (R_b, S_b)$
 - Check if (R_Σ, S_Σ) match values in π_{Tally}
 - Check if $\text{EqDLVerify}(\pi_{\text{Tally}}) = 1$

The Σ protocol (EqDLProve, EqDLVerify) (Schnorr)

$$\text{NIZK}_H\{(g, \text{pk}), (\text{sk}) : \log_g \text{pk} = \text{sk}\}$$

DLProve(g, pk, sk)

- $T := g^t, \quad t \leftarrow \mathbb{Z}_q$
- $c := H(g, \text{pk}, T)$
- $s := t - \text{sk} \cdot c$
- return (T, c, s) or (c, s)

EqDLVerify(T, c, s)

return if $T = g^s \text{pk}^c$ or alternatively: check if $c = H(g, \text{pk}, g^s \text{pk}^c)$

As a Σ -protocol it can be simulated by selecting the challenge before the commitment

Simulate(g, pk, c)

- $s \leftarrow \mathbb{Z}_q$
- $T := g^s \text{pk}^c$
- return (T, c, s)

The Σ protocol (EqDLProve, EqDLVerify) (Chaum Pedersen)

$$\text{NIZK}_H\{(g, \text{pk}, h, R), (\text{sk}) : \log_g \text{pk} = \log_h R = \text{sk}\}$$

EqDLProve(g, pk, h, R)

- $T_1 := g^t, T_2 := h^t, t \leftarrow \mathbb{Z}_q$
- $c := H(g, \text{pk}, h, R, T_1, T_2)$
- $s := t - \text{sk} \cdot c$
- return (T_1, T_2, c, s)

EqDLVerify((g, pk, h, R), (T_1, T_2, c, s))

return

$$T_1 = g^s \text{pk}^c \text{ AND } T_2 = h^s R^c$$

As a Σ -protocol it can be simulated by selecting the challenge before the commitment

Simulate(g, pk, R, S, c)

- $s \leftarrow \mathbb{Z}_q$
- $T_1 := g^s \text{pk}^c, T_2 := h^s R^c$
- return (T_1, T_2, c, s)

The Σ protocol (DisjProve, DisjVerify) (Witness indistinguishable Chaum - Pedersen)

$$\text{NIZK}_H\{(g, \text{pk}, R, S), (r) : (R, S) = \text{Enc}_{\text{pk}}(g^0) \text{ OR } (R, S) = \text{Enc}_{\text{pk}}(g^1)\}$$

$$(R, S) = \text{Enc}_{\text{pk}}(g^0) \text{ OR } (R, S) = \text{Enc}_{\text{pk}}(g^1)$$

$$(R, S) = (g^r, g^0 \text{pk}^r) \text{ OR } (R, S) = (g^r, g^1 \text{pk}^r)$$

$$\log_g R = \log_{\text{pk}} S \text{ OR } \log_g R = \log_{\text{pk}}(S/g)$$

$$\text{EqDLProve}(g, \text{pk}, R, S, r) \text{ OR } \text{EqDLProve}(g, \text{pk}, R, S/g, r)$$

Assuming the voter has voted for 0:

$$\pi = \text{EqDLProve}(g, \text{pk}, R, S, r) \parallel \text{Simulate}(g, \text{pk}, R, S/g, c_S)$$

where: $c_r + c_S = c_H$

Pitfalls of the Fiat-Shamir Heuristic

Bernhard, Pereira, Warinschi (2012) How Not to Prove Yourself:
Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios.
ASIACRYPT 2012

- Weak FS: Input to hash function contains only commitment
 $c = H(T)$
- Strong FS: Input to hash function contains commitment,
statement to be proved and all public values generated so far.

If the prover is allowed to select their statement **adaptively** then the weak FS yields **unsound** proofs

Proofs created using the weak FS have implications to the privacy and verifiability of Helios and other similar voting systems.

The Strong Fiat - Shamir Transform

Pitfalls of the Fiat-Shamir Heuristic (cont'd)

$\text{DLProve}(g, \text{pk})$ proves knowledge of DLOG for a **particular** $\text{pk} \in \mathbb{G}$
given as input to the prover

If pk can be selected adaptively (after the proof):

- Select $T \leftarrow \mathbb{G}$
- Compute $c := H(T)$
- Select $s \leftarrow \mathbb{Z}_q$
- The tuple (T, c, s) is a proof of knowledge for $\text{pk} = (g^{-s}T)^{\frac{1}{c}}$ for which **sk** is not known!
- Indeed: $g^s \text{pk}^c = g^s (g^{-s}T)^{c \frac{1}{c}} = T$

Pitfalls of the Fiat-Shamir Heuristic (cont'd)

Assume that in $\text{EqDLProve}(g, \text{pk}, h, R, \text{sk})$ the prover can select the statement (g, pk, h, R) adaptively.

- Select $a, b, r, f \leftarrow \mathbb{Z}_q$
- Compute: $T_1 := g^a, T_2 := g^b, h := g^r, R := g^f$
- Compute: $c := H(T_1, T_2)$
- Compute $s := \frac{b-cf}{r}$
- Set $\text{sk} = \frac{a-s}{c}$

The proof verifies

$$g^s \text{pk}^c = g^s (g^{\frac{a-s}{c}})^c = g^a = T_1$$

$$h^s R^c = (g^r)^{\frac{b-cf}{r}} g^{fc} = g^b = T_2$$

but $\log_g \text{pk} \neq \log_h R$ (unsound!)

$$\log_g \text{pk} = \frac{a-s}{c} = \frac{a}{c} - \frac{b-cf}{rc} = \frac{a}{c} - \frac{b}{rc} + \frac{f}{r} = \frac{f}{r} + \frac{ra-b}{rc}$$

$$\text{and } \log_h R = \log_{g^r} g^f = \log_{g^r} g^f = \frac{f}{r}$$

Implications: Non-Malleable encryption

Malleability: Transform a ciphertext into another valid ciphertext

Enc + PoK: A common way to achieve non malleability

append a NIZK PoK of randomness to the ciphertext

For input $m \in \mathbb{G}$:

$\text{Enc}_{\text{pk}}(m) = (g^r, m \cdot \text{pk}^r, c, s)$ where: $r \leftarrow \mathbb{Z}_q, (c, s) = \text{DLProve}(g, g^r, r)$

If wFS is used then the scheme is malleable:

For $c_1 = (R, S, c, s)$ select $u \leftarrow \mathbb{Z}_q$ and create $c_2 = (R \cdot g^u, S \cdot \text{pk}^u, c, s - cu)$

The ciphertext was changed, but the proof $(c, s - cu)$ verifies.

$g^{s-cu} (Rg^u)^c = (g^s R^c) g^{-cu} g^{cu} = (g^s R^c) = T$ (valid from the original proof)

Theorem

Enc + PoK with sFS provides NM – CPA

Implications to Helios - Denial of Service attack

Each member TA_i computes: $D_i = R^{sk_i}$, $\text{EqDLProve}(g, pk_i, R, D_i, sk_i)$ for specific pk_i

A malicious TA_i can cheat by *first* creating a proof and *then* selecting D_i such that:

- Select $(a, b) \leftarrow \mathbb{Z}_q$
- Compute: $T_1 := g^a, T_2 := g^b$
- Compute: $c := H(T_1, T_2)$
- Compute $s := a - sk_i \cdot c$
- Compute $D_i := (R^{-s}T_2)^{\frac{1}{c}}$

The proof verifies: $g^s pk_i^c = g^a = T_1$ and $R^s D_i^c = R^s (R^{-s}T_2) = T_2$

However: $\log_g pk_i = sk_i$ but

$$\log_R D_i = \log_R R^{-\frac{s}{c}} + \log_R g^{\frac{b}{c}} = sk_i - \frac{a}{c} + \frac{rb}{c} = sk_i + \frac{rb-a}{c} \text{ where } g^r = R$$

This means that tally decryption yields a random group element \Rightarrow instead of g^t

Denial of service attack to compute DLOG

Application to Helios - Undetectably alter result

Goal:

Announce election result $t' \neq t$

Assumptions:

- The TA is corrupted and can eavesdrop on the randomness of all voters (realistic assumption since Helios generates it)
- Actively corrupt a single voter - casts a last vote

The TA creates a 'proof' (c, s) of correct tallying:

- Select $(a, b) \leftarrow \$_\mathbb{Z}_q$
- Compute: $T_1 := g^a, T_2 := g^b$
- Compute: $c := H(T_1, T_2)$
- Compute $s := a - sk \cdot c$

Application to Helios - Undetectably alter result (cont'd)

- All voters vote, except for the **corrupt voter**.
- The current result is t and encrypted as $(R, S) = (g^r, g^t \text{pk}^r)$
- The TA knows it and can compute t by decrypting
- From individual randomness they know $r = \sum_i r_i$
- The TA creates but does not release the proof (c, s)
- The TA selects $r' := \frac{b-c(t-t')}{s+c \cdot \text{sk}}$
- The **corrupt voter** casts $(g^{r'-r}, \text{pk}^{r'-r})$ which is a valid 0 vote.
- The complete product is: $(R', S') = (g^{r'}, g^t \text{pk}^{r'})$
- The encrypted tally does not change, but...

Application to Helios - Undetectably alter result (cont'd)

- The proof (c, s) is also valid for the relation $\log_g \text{pk} = \log_{R'}(S' g^{-t'})$
- So the announced tally is verified as t'

Since s is valid for $\log_g \text{pk} = sk$: $T_1 = g^s \text{pk}^c$

$$\begin{aligned} R'^s (S' g^{-t'})^c &= (g^{r'})^s (g^t \text{pk}^{r'} g^{-t'})^c \\ &= g^{r'(a-c \cdot sk) + ct + c \cdot sk \cdot r' - t'c} \\ &= g^{ar' + c(t-t')} \\ &= g^{\frac{b-c(t-t')}{a} a + c(t-t')} \\ &= g^b \\ &= T_2 \end{aligned}$$

Application to Swiss Voting

Swiss e-voting trial offers \$150,000 in bug bounties to hackers

The white hat hacking begins February 24th



S. J. Lewis, O. Pereira, and V. Teague, "How not to prove your election outcome: The use of non-adaptive zero knowledge proofs in the Scytl-SwissPost Internet voting system, and its implications for decryption proof soundness"

R. Haenni, "Swiss post public intrusion test: Undetectable attack against vote integrity and secrecy" but in Australia:

NSW Electoral Commission iVote and Swiss Post e-voting

(borrowed from https://git.openprivacy.ca/sarah/presentations/raw/branch/master/20191017--sarah-jamie-lewis-on-e-voting-et-al_slides.pdf)

FROZEN HEART (FoRging Of ZERo kNowledge proofs)

- Girault's proof of knowledge protocol (Schnorr over a composite modulus)
- Bulletproofs
- PLONK

Takeaway

The Fiat-Shamir hash computation must include all public values from the zero-knowledge proof statement and all public values computed intermediately the proof (i.e., all random “commitment” values)

<https://blog.trailofbits.com/2022/04/13/part-1-coordinated-disclosure-of-vulnerabilities-affecting-girault-bulletproofs-and-plonk/>

Verifiability

Verifiability

The property that enables voters to regain the trust endangered by the volatile nature of computer systems that implement e-voting and the adversarial motives of voting authorities (systemic errors or malice)

Subnotions:

- Individual Verifiability (cast as intended / recorded as cast)
- Universal Verifiability (tallied as recorded)
- Eligibility Verifiability (avoid ballot stuffing)

Trust Assumptions: EA members are **totally corrupted** and cooperate to affect the election result to their advantage

- Universal Verifiability: TA is corrupted
 - Eligibility Verifiability: identify if a ballot was cast by a voter with a right to vote
- Corruption of BB: depends on the model

\mathcal{A} controls a subset of the voters

Verifiability does not mean verification

Do all the voters verify their ballots?

Individual Verifiability

Intuition

The voters verify that their ballots are included in the tally

A necessary condition

All ballots are unique

Clash attacks

Two or more voters are pointed to verify the same ballot

\mathcal{A} has at least one ballot to use to affect the result

Note

Paper-based voting systems do not possess individual verifiability

The use of aliases greatly affects the adversarial capability of mounting clash attacks

Helios without aliases

ElGamal probabilistic encryptions: If two voters find the same ciphertext then a clash attack has been mounted

A natural clash occurs with negligible probability

Helios with aliases

- Assumption: **Adversarial EA** that knows 2 or more voters might vote for the same candidate
- Attack:
 - Provides them with the same alias
 - Modifies user interface to **always** select the same random coins for these voters (regardless of the number of audits)
 - Note: audit does not require that successive ballots are different (i.e. the use of different random coins)
- All voters verify the same ballot → individual verifiability succeeds
- The **EA** then submits a ballot containing its preferred option in the free slot

Countermeasures

- The Bulletin' Board is published *after* each vote and not in the end
- Voters always observe the BB before vote casting
- Voters check audited ballots for exact duplicates
- Voters contribute to the encryption randomness (e.g. by typing a random phrase)
- Use unique real world identities (external authentication)
 - But: This might leak abstention or not
 - Illegal in some jurisdictions (e.g. France)
 - Might also mean repercussions for those who voted / did not vote
 - Relevant property: Participation privacy

Individual verifiability formal model

Algorithm 1: Individual verifiability $\text{IndVer}_{\text{VS}, \mathcal{A}}$

Input : security parameter λ

Output: $\{0, 1\}$

$(\text{CS}, \mathbf{vt}_0, \mathbf{vt}_1) \leftarrow \mathcal{A}(1^\lambda)$

$b_0 := \text{VS.Vote}(\mathcal{A}(), V_i(\mathbf{vt}_0), \text{pk}_{\text{EA}}, \text{CS}, \text{BB})$

$b_1 := \text{VS.Vote}(\mathcal{A}(), V_i(\mathbf{vt}_1), \text{pk}_{\text{EA}}, \text{CS}, \text{BB})$

if $b_0 = b_1$ **AND** $b_1 \neq \perp$ **then**

 | return 1

else

 | return 0

end

Individual verifiability definition

Definition

A voting scheme VS satisfies individual verifiability if

$$\forall \text{PPT } \mathcal{A} : \Pr[\text{IndVer}_{\text{VS}, \mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$$

Helios without aliases satisfies IndVer assuming honest generation of random coins

Since

$$\text{VS.Vote} \equiv \text{Enc} \Rightarrow \Pr[\text{IndVer}_{\text{VS}, \mathcal{A}}(1^\lambda) = 1] = \Pr[\mathbf{b}_0 = \mathbf{b}_1] = \text{negl}(\lambda).$$

Helios with aliases does not satisfy IndVer

Because of the clash attack

Note

This model deals only with the recorded as cast part of individual verifiability

Voter intent is not taken into account (cast as intended)

Even if it did, could there be a negligible probability of success?

Intuition

Everyone (voters, external auditors) can verify that the tally corresponds to the voter's selections

Adversarial Goal

Present a tally along with fabricated evidence that passes verification

A baseline is needed: A function result that correctly captures the tally:

Definition

$$\text{result}(\text{pk}_{\text{TA}}, \text{BB}, \text{CS})[v] = n_v \Leftrightarrow \exists^{n_v} \mathbf{b} \in \text{BB} : \mathbf{b} = \mathbf{Vote}(v)$$

Problem: How to calculate it in proofs - two approaches:

- Construction using an extractor that retrieves votes from ballots (does not apply if ballots are information-theoretically protected)
- Mere existence of corrupted votes + (honest votes are known to the challenger)

Universal Verifiability - A first definition

Algorithm 2: Universal Verifiability $\text{UniVer}_{\text{VS}, \mathcal{A}}$

Input : security parameter λ

Output: $\{0, 1\}$

$(\text{CS}, \text{BB}, T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}) \leftarrow \mathcal{A}(1^\lambda)$

$T \leftarrow \text{result}(\text{BB})$

if $T_{\mathcal{A}} \neq T$ **AND** $\text{VS.Verify}(T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}, \text{pk}_{T_{\mathcal{A}}}, \text{BB}, \text{CS}) = 1$ **then**

 | return 1

else

 | return 0

end

Definition

A voting scheme VS satisfies universal verifiability if

$$\forall \text{PPT } \mathcal{A} : \Pr[\text{UniVer}_{\text{VS}, \mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$$

Definition

A voting scheme VS (with external authentication) satisfies election verifiability if

$$\forall \text{PPT } \mathcal{A} : \Pr[\text{IndVer}_{\text{VS}, \mathcal{A}}(1^\lambda) = 1] + \Pr[\text{UniVer}_{\text{VS}, \mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$$

Universal Verifiability - Additional Considerations

- Are all the ballots in the BB valid? Is there revoting?
- Do *all* voters verify their ballots? If the verifiability definition demands it then it is too strong.
- Is a registration authority RA required? Is it corrupted? (External vs internal authentication)
- Is the BB passive - simply stores all the ballots? Is it corrupted (ballot stuffing)?

Universal Verifiability - A Finer Grained Approach

Universal Verifiability with RA and BB

- RA provide cryptographic credentials to the voters
- **Vote** includes these credentials
- BB is not passive: can add or remove ballots
- *Weak* Universal Verifiability: Both the RA and the BB are honest.
- *Strong* Universal Verifiability: The RA and the BB are not corrupted at the same time.
 - Against corrupt RA
 - Against corrupt BB

The RA's objective is to counter BB corruption and vice versa

\mathcal{A} 's objective

Cause a tally to be accepted if either:

- **ballot stuffing occurs** - the number of corrupted votes *exceeds* the number of corrupted voters. **However, the choices should be admissible.**
- **verification was bypassed** - some of the votes of the honest voters that did verify **are not taken** into account
- some of the votes of honest voters that **did not check** are not taken into account - **all would be too strong**

Algorithm 3: Oracles for Universal Verifiability Definitions

Oracle *Register*(i)

| $(sk_i, pk_i) := VS.Register(RA(sk_{RA}), V_i())$
| $V_{EL} \leftarrow (i, pk_i)$

Oracle *Corrupt*(i)

| if $i \in V_{EL}$ then
| | $V_{Corr} \leftarrow (i, pk_i, sk_i)$
| end

Oracle *Vote*(i, vt_i)

| if $i \in V_{EL}$ AND $i \notin V_{Corr}$ then
| | if $\exists (i, \cdot, \cdot) \in V_{Hon}$ then
| | | $V_{Hon} := V_{Hon} \setminus \{(i, \cdot, \cdot)\}$
| | end
| | $b := VS.Vote(i, vt_i, sk_i)$
| | $V_{Hon} \leftarrow (i, vt_i, b)$
| end

Oracle *Cast*(i, b)

| $BB \leftarrow (i, b)$

Weak universal verifiability

Algorithm 4: Weak universal verifiability game W-UniVer

Input : security parameter λ

Output: $\{0, 1\}$

$(\text{prms}, \text{pk}_{\mathcal{T}_A}, \text{sk}_{\mathcal{T}_A}) \leftarrow \text{VS.Setup}(\lambda)$

$(T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}) \leftarrow \mathcal{A}^{\text{Register, Corrupt, Vote, Cast}}()$

if $\text{VS.Verify}(T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}, \cdot) = 0$ **OR** $T_{\mathcal{A}} = \perp$ **then**

 | return 0

end

if $\exists n_{V_{\text{Corr}}} : 0 \leq n_{V_{\text{Corr}}} \leq |V_{\text{Corr}}|$ **AND** $\exists \{\text{vt}_i^{V_{\text{Corr}}}\}_{i=1}^{n_{V_{\text{Corr}}}} \in \text{CS} :$

$T_{\mathcal{A}} = \text{result}(\text{vt}_i^{V_{\text{Corr}}}) \oplus \text{result}(\text{vt}_i^{V_{\text{Hon}}})$ **then**

 | return 0 // \mathcal{A} fails if all honest and some corrupted votes
 | are included in the final valid tally

else

 | return 1

end

Note:

\mathcal{A} controls only EA and V_{Corr} - cannot add-delete ballots but may try to input invalid options or alter tally.

Algorithm 5: Strong universal verifiability game (with malicious BB) S-UniVer-BB

Input : security parameter λ

Output: $\{0, 1\}$

$(\text{prms}, \text{pk}_{\mathcal{T}_A}, \text{sk}_{\mathcal{T}_A}) \leftarrow \text{VS.Setup}(1^\lambda)$

$(\text{BB}, \mathcal{T}_A, \pi_{\mathcal{T}_A}) \leftarrow \mathcal{A}^{\text{Register, Corrupt, Vote}}()$

if $\text{VS.Verify}(\text{BB}, \mathcal{T}_A, \pi_{\mathcal{T}_A}, \cdot) = 0$ OR $\mathcal{T}_A = \perp$ then

 return 0

end

$V_{\text{Chck}} = \{(\text{ID}_i^{\text{Chck}}, \text{vt}_i^{\text{Chck}}, \text{b}_i^{\text{Chck}})\}_{i=1}^{n_{\text{Chck}}} // \text{ Voters who verified}$

if $\exists n_{V_{\text{Corr}}} : 0 \leq n_{V_{\text{Corr}}} \leq |V_{\text{Corr}}|$ AND $\exists \{\text{vt}_i^{V_{\text{Corr}}}\}_{i=1}^{n_{V_{\text{Corr}}}} \in \text{CS}$ AND

$\exists n' : 0 \leq n' \leq |V_{\text{Hon}}| - |V_{\text{Chck}}|$ AND $\exists \{\text{vt}'_i\}_{i=1}^{n'}$ // *Voters that did not check*

such that: $\mathcal{T}_A = \text{result}(\text{vt}_i^{V_{\text{Corr}}}) \oplus \text{result}(\text{vt}_i^{V_{\text{Chck}}}) \oplus \text{result}(\text{vt}'_i)$ then

 return 0 // \mathcal{A} fails if the final valid tally corresponds to valid
 votes of all who checked, some that did not and ballots were
 not stuffed/deleted

else

 return 1

end

Note: The corrupted BB might add, replace or delete ballots

Strong Universal Verifiability

Algorithm 6: Strong universal verifiability game (with malicious RA)

S-UniVer-RA

Input : security parameter λ

Output: $\{0, 1\}$

$(\text{prms}, \text{pk}_{\mathcal{T}_A}, \text{sk}_{\mathcal{T}_A}) \leftarrow \text{VS.Setup}(1^\lambda)$

$(\mathcal{T}_A, \pi_{\mathcal{T}_A}) \leftarrow \mathcal{A}^{\text{Cast, Corrupt, Vote}}()$

if $\text{VS.Verify}(\text{BB}, \mathcal{T}_A, \pi_{\mathcal{T}_A}, \cdot) = 0$ OR $\mathcal{T}_A = \perp$ then

 return 0

end

$V_{\text{Chck}} = \{(\text{ID}_i^{\text{Chck}}, \text{vt}_i^{\text{Chck}}, b_i^{\text{Chck}})\}_{i=1}^{n_{\text{Chck}}} // \text{ Voters who verified}$

if $\exists n_{V_{\text{Corr}}} : 0 \leq n_{V_{\text{Corr}}} \leq |V_{\text{Corr}}|$ AND $\exists \{\text{vt}_i^{V_{\text{Corr}}}\}_{i=1}^{n_{V_{\text{Corr}}}} \in \text{CS}$ AND

$\exists n' : 0 \leq n' \leq |V_{\text{Hon}}| - |V_{\text{Chck}}|$ AND $\exists \{\text{vt}'_i\}_{i=1}^{n'}$ such that:

$\mathcal{T}_A = \text{result}(\text{vt}_i^{V_{\text{Corr}}}) \oplus \text{result}(\text{vt}_i^{V_{\text{Chck}}}) \oplus \text{result}(\text{vt}'_i)$ then

 return 0 // \mathcal{A} fails if the final valid tally corresponds to the
 votes of all who checked, some that did not and ballots were
 not cancelled

else

 return 1

end

Note: Ballot stuffing/erasing does not occur through the BB but through the RA (via invalid credentials)

Correctness

Honest executions yield the expected result:

honest ballots are accepted

tally is verified

the output of tally corresponds to the output of result

$$\Pr \left[\begin{array}{l} (T, \pi_T) = \text{Tally}(\{b_1, \dots, b_n\}) \text{ where} \\ \{b_i = \text{Vote}(i, v_i, sk_i), v_i \in \text{CS}\}_{i=1}^n; \\ \text{Valid}(b_i) = 1 \text{ AND} \\ \text{Verify}(\{b_1, \dots, b_n\}, T, \pi_T) = 1 \text{ AND} \\ T = \text{result}(v_1, \dots, v_n) \end{array} \right] = 1$$

Tally uniqueness

A correct tally of an election is unique

$$\Pr \left[\begin{array}{l} (BB, T_1, \pi_{T_1}, T_2, \pi_{T_2}) \leftarrow \mathcal{A}(1^\lambda); \\ T_1 \neq T_2; \\ \text{Verify}(BB, T_1, \pi_{T_1}) = 1 \text{ AND} \\ \text{Verify}(BB, T_2, \pi_{T_2}) = 1 \end{array} \right] = \text{negl}(\lambda)$$

You cannot get two results from the same ballots and verify the result.

Accuracy

VS has accuracy if $\forall b$ (even adversarial):

- **Valid**(b) = 1 AND **Verify**($\{b\}, T_b, \pi_{T_b}$) = 1 $\Rightarrow v_b \in \text{CS}$ AND $T_b = \text{result}(v_b)$
 - Any ballot that passes the validity test is a valid vote
 - Even if it is generated by the adversary
- **Verify**(BB, **Tally**(BB, sk)) = 1, $\forall \text{BB}$
 - Any honestly generated tally and proof passes verification
 - usually holds by design

Partial counting

$\text{result}(S_1 \cup S_2) = \text{result}(S_1) \oplus \text{result}(S_2)$ where S_1, S_2 are sequences of votes

Partial tallying

If $(T_1, \cdot) = \mathbf{Tally}(BB_1, sk)$,

$(T_2, \cdot) = \mathbf{Tally}(BB_2, sk)$,

$(T, \cdot) = \mathbf{Tally}(BB_1 \cup BB_2, sk)$ and

$BB_1 \cap BB_2 = \emptyset$ then:

$$T = T_1 \oplus T_2$$

Sufficient conditions for weak universal verifiability i

Theorem

If VS satisfies correctness, tally uniqueness, partial tallying, and accuracy then it provides weak universal verifiability

Let (BB, T, π_T) the output of VS such that $\mathbf{Verify}(BB, T, \pi_T) = 1$ and $T \neq \perp$

BB is honest $\Rightarrow \forall b \in BB : \mathbf{Valid}(b) = 1$

Split BB into disjoint honest and corrupt parts $BB = BB_{Hon} \cup BB_{Corr}$

BB_{Hon}

BB is honest \Rightarrow no honest ballot has been deleted

From correctness and partial tallying:

$(T_{Hon}, \pi_1) = \mathbf{Tally}(BB_{Hon}, sk)$ with $T_{Hon} = \mathbf{result}(\{v_i\}_{i=1}^{n_{Hon}})$ where
 $\{b_i = \mathbf{Vote}(i, v_i)\}_{i=1}^{n_{Hon}}$

Sufficient conditions for weak universal verifiability ii

BB_{Corr}

Since BB is honest means at most one ballot per voter:

$$|BB_{Corr}| \leq |V_{Corr}|$$

Compute

$$(T_{Corr}, \pi_2) = \mathbf{Tally}(BB_{Corr}, sk)$$

From accuracy (2) and the honest BB condition:

$$\mathbf{Verify}(BB_{Corr}, T_{Corr}, \pi_2) = 1$$

From tally uniqueness this tally is unique

From accuracy (1): $T_{Corr} = \mathbf{result}(\{v_i\}_{i=1}^{n_{Corr}})$

From partial tallying: $T = \mathbf{Tally}(BB_{Corr} \cup BB_{Hon}, sk)$

Helios is weakly verifiable under the DLOG assumption in the random oracle model

Correctness: Follows from the correctness of El-Gamal and completeness of Schnorr, Chaum - Pedersen and NIZK.

Tally Uniqueness: A verified tally passes proofs generated by DisjProve, EqDLProve. Uniqueness follows from the special soundness of the Σ protocols (DLOG)

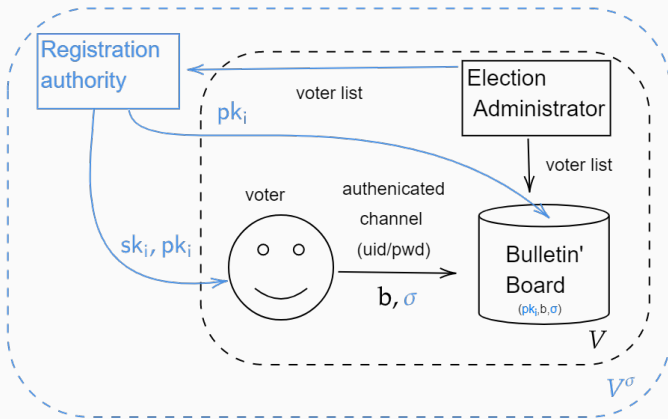
Accuracy: If $\text{Valid}(\cdot) = 1$ and $\text{Verify}(\cdot) = 1$ then $v \in \text{CS}$ with negligible soundness error $\frac{1}{q}$ because of Σ protocol (DisjProve, DisjVerify).

Helios and strong universal verifiability i

A generic construction from VS to VS^σ with strong universal verifiability:

- **EUFCMA**-secure signature scheme
- Registration authority that hands credentials (pk_u, sk_u) to the voters
- Registration authority publishes public credential list
- Voters use the credentials for ballot signing (with last vote counts update)
- BB uses an identification scheme to allow the voters to cast a ballot (**This means that each voter has two credentials**)
- Ballot validation (by the BB) includes signature verification and every public credential is unique and registered
- BB maintains correspondence between (id, pk_i) to avoid multiple impersonation attacks

Helios and strong universal verifiability ii



Lemma

If VS has weak verifiability, tally uniqueness and σ is EUF-CMA then VS^σ provides verifiability against a corrupted BB

Every adversary \mathcal{A}^σ against VS^σ is as powerful as an adversary \mathcal{A} against VS, unless he can break **EUF-CMA**.

Facts (from strong verifiability definition):

- $T \neq \perp$
- BB^σ is well-formed, since it passes **Verify** $^\sigma$. All ballots are valid.

As a result: $\forall (T, \pi) : \mathbf{Verify}(BB^\sigma, T, \pi) = \mathbf{Verify}(BB, T, \pi)$

- Every vote $\mathbf{vt} \in V_{Chck}$ that has a corresponding ballot $b = (pk_u, a, \sigma)$ in BB^σ has also a ballot a in BB . a is valid from weak verifiability.

Helios and strong universal verifiability iv

- Every vote $\mathbf{vt} \in V_{Hon} \setminus V_{Chck}$ that has a corresponding ballot in BB^σ corresponds to an honest vote (output of **Vote**)
If not: since it is placed in BB^σ it must have a valid signature. Since σ does not come from **Vote** then it must have been forged (**contradiction**).
Conclusion: Every $\mathbf{vt} \in V_{Hon} \setminus V_{Chck}$ comes from **Vote**.
- $n_{Corr} \leq |V_{Corr}|$
If not:
There are two (at least 2) ballots in BB^σ with the same credential. But BB^σ is well-formed.
Or: \mathcal{A}^σ added a valid ballot without calling **Corrupt** (without knowing sk_i). This contradicts unforgeability again.

Privacy

Intuition

Nobody can learn the vote cast by a particular voter.

- Secrecy in voting differs from secrecy in messaging
- Ballot privacy is not absolute
- The result leaks information
 - In a unanimous vote, everyone knows how everyone voted
 - In an all-but-one vote, the one that differs knows how everyone else voted
 - The result also yields a probability of a particular vote
 - Important for small voting populations

Game-based definitions for ballot secrecy

Threat model: \mathcal{A} corrupts voters. TA is honest!

Intuition: Indistinguishability games for cryptographic secrecy

Instead of distinguishing between message encryptions, \mathcal{A} tries to distinguish between different voting scenarios.

Workflow:

- \mathcal{C} setups the protocol and begins with an empty BB
- \mathcal{A} corrupts voters and casts ballots on their behalf
- For an honest voter, \mathcal{A} selects 2 choices and hands them to \mathcal{C}
- \mathcal{C} flips a coin and selects one of the choices
- \mathcal{C} creates and casts the respective ballot
- \mathcal{A} casts more corrupted ballots
- The \mathcal{A} must guess the coin

This must be combined with verifiability and apply to all possible voting rules (i.e. result functions) - voting rules.

Impact of voting rules - A toy example [?]

Assume the majority result function and candidates $\{a, b\}$

The candidate with majority is declared the winner

In case of a tie: a is the winner.

There are two honest voters and one corrupt voter

In scenario 1: $\{(i_0, a)(i_1, a)\}$ winner: a

In scenario 2: $\{(i_0, a)(i_1, b)\}$ winner: a

\mathcal{A} can distinguish the scenario by casting a ballot for b

Lack of ballot independence can break ballot privacy [?] i

Ballot independence

A voter **cannot repost** an exact replica of another voter's ballot in the BB

Assume an election with 3 voters, where V_1, V_2 are honest and V_3 is corrupt.

$$b_1 = (1, (R_1, S_1), \pi_{\text{Vote},1})$$

$$b_2 = (2, (R_2, S_2), \pi_{\text{Vote},2})$$

The voter replays an exact replica of b_1

$$b_3 = (3, (R_1, S_1), \pi_{\text{Vote},1})$$

The candidate that receives more than 2 votes is the one preferred by V_1

Lack of ballot independence can break ballot privacy [?] ii

Alternative implementation: Use malleability and construct a different ballot for the same vote

Countermeasures:

- Ballot weeding
- Strong Fiat-Shamir Heuristic to counter encryption malleability
Recall: Enc + PoK provides non malleability
- Add voter id to the hash function

Despite being naive such attacks provide a sanity check for voting systems

The BPRIV framework [?]

Goal

A private protocol does not leak any more information than what is leaked from the tally.

Main idea

\mathcal{A} : Tries to distinguish between two worlds by having access to:

- The 'real' BB that contains honest and adversarial ballots. The adversary sees the real result.
- A 'fake' BB. The adversary sees the real results. Proofs of correctness are simulated.

\mathcal{A} has access to the following oracles:

- **Board**(b): Retrieve the contents of the BB_b (**Publish**)
- **Vote**(v_0, v_1): Select two votes and post the corresponding ballots to BB_0, BB_1 respectively - ballots are honestly created
- **Cast**(b): Cast a ballot to both BB - ballot is adversarially created
- **Tally**(b): Obtain the result of BB_0 . Yield real or simulated proofs

\mathcal{A} can call the oracles **Board**, **Vote**, **Cast** at will

\mathcal{A} can call **Tally** only once

\mathcal{A} must guess which BB is presented

The BPRIV definition

Algorithm 7: $\text{BPRIV}_{\mathcal{A}, \text{VS}}^b$

Input : security parameter λ

Output: $\{0, 1\}$

$(pk_{\text{TA}}, sk_{\text{TA}}) \leftarrow \text{VS.Setup}(1^\lambda)$

$\text{CS} \leftarrow \mathcal{A}()$

$b' \leftarrow \mathcal{A}^{\text{Vote, Cast, Tally, Board}}(pk_{\text{TA}})$

return $b = b'$

BPRIV

A voting system VS supports ballot privacy if there exists a simulator for \mathcal{S} such that $\forall \text{PPT } \mathcal{A}$:

$$|\Pr[\text{BPRIV}_{\mathcal{A}, \text{VS}}^0(\lambda) = 1] - \Pr[\text{BPRIV}_{\mathcal{A}, \text{VS}}^1(\lambda) = 1]| = \text{negl}(\lambda)$$

Algorithm 8: Oracles for BPRIV definition

Oracle *Vote*(i, vt_0, vt_1)

```
   $b_0 := \text{Vote}(i, vt_0)$ 
   $b_1 := \text{Vote}(i, vt_1)$ 
  if Valid( $b_0, BB_0$ ) AND Valid( $b_1, BB_1$ ) then
    |  $BB_0 \leftarrow b_0$ 
    |  $BB_1 \leftarrow b_1$ 
  else
    | return  $\perp$ 
```

Oracle *Cast*(i, b, b)

```
  if Valid( $b, BB_b$ ) then
    |  $BB_0 \leftarrow b$ 
    |  $BB_1 \leftarrow b$ 
  else
    | return  $\perp$ 
```

Oracle *Tally*(b)

```
  if  $b = 0$  then
    |  $(T, \pi_T) := \text{Tally}(sk_{TA}, CS, BB_0)$ 
    | return  $(T, \pi_T)$ 
  else
    |  $(T, \pi_T) := \text{Tally}(sk_{TA}, CS, BB_0)$ 
    |  $\pi'_T := \mathcal{S}(sk_{TA}, CS, BB_0, BB_1, T)$ 
    | return  $(T, \pi'_T)$ 
```

Oracle *Board*(b)

```
  return Publish( $BB_b$ )
```

Observation

The visible BB consists of tuples (i, \mathbf{b}, \cdot) posted using **Cast**, **Vote**

If the tuple originates from **Vote**, the challenger can internally map it to a tuple $(i, \mathbf{b}_0, \mathbf{vt}_0, \mathbf{b}_1, \mathbf{vt}_1)$

If the tuple originates from **Cast**, then it can be mapped to $(i, \mathbf{b}, \perp, \mathbf{b}, \perp)$

Proof overview

A sequence of games beginning from BPRIV^0 and ending to BPRIV^1 with indistinguishable differences for \mathcal{A}

Game₀: BPRIV⁰ - \mathcal{A} sees BB_0 and the real result of the tally

Game₁: The tally proof in BPRIV⁰ is simulated as

$\pi'_T = \text{Simulate}(g, \text{pk}, R, Sg^{-t}, c)$ where $c \leftarrow \mathbb{Z}_q$.

Based on the ZK property of π_T , \mathcal{A} has a negligible advantage in distinguishing between **Game₀**, **Game₁**.

$\{\text{Game}_{2,i}\}_{i=1}^{n_{Hon}}$

For all ballots cast using **Vote** (challenger entries $(i, \mathbf{b}_0, \mathbf{vt}_0, \mathbf{b}_1, \mathbf{vt}_1)$) replace \mathbf{b}_{0i} in BB_0 with \mathbf{b}_{1i} from BB_1 .

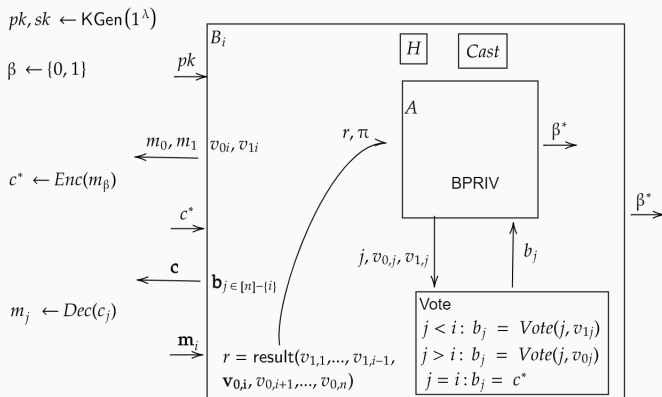
\mathcal{A} notices this with negligible probability due to the property of Enc + PoK.

Game_{2, n_{Hon}} is BPRIV¹

Helios is BPRIV iii

$\Rightarrow \text{Game}_{2,i} \equiv \text{Game}_{2,i-1}$

Definition of from [?]



Everlasting Privacy

Motivation

- Ballot secrecy is provided through encryption schemes
- Protection relies on computational hardness assumptions
- What if these assumptions are broken?

- Vote contents might be useful to a future oppressive government
- The need for verifiability makes election data publicly available
- But such a regime might also use insider information
- This threat might constitute an indirect coercion attempt

Everlasting Privacy [?]

Perfectly hiding commitment schemes

- Ballot: Perfectly hiding commitment of vote
- For counting: openings are required
- How do voters post the openings?
 - Encrypt the openings
 - Secret sharing with the tallying authorities

Practical Everlasting Privacy: Everlasting privacy towards the public
[?]

External adversary has the same view with the voters

Excludes insiders

Anonymous casting (+ blind signatures) [?]

- Disassociate identity with ballot
- Provide a blind signature to identified ballot to signal eligibility
- Identity is no longer required
- Blinded ballots cast anonymously

A further advantage: TA is not required to be trusted for privacy

Applying Everlasting Privacy to Helios [?] i

- Pedersen commitment $\text{Commit}(m, r) = g^m h^r$ $g, h \leftarrow_{\$} \mathbb{G}$
- Homomorphic:
 $\text{Commit}(m_1, r_1) \cdot \text{Commit}(m_2, r_2) = \text{Commit}(m_1 + m_2, r_1 + r_2)$
- Perfectly hiding - computationally binding
- Problem: ElGamal cannot be used for encrypting m, r
- Use a compatible cryptosystem (many approaches)

Applying Everlasting Privacy to Helios [?] ii

Vote(i, vt_i)

$(b_i, ok_i) \leftarrow \text{Commit}(ck, vt_i)$

$[vt_i] \leftarrow \text{Enc}(pk_{TA}, vt_i)$

$[ok_i] \leftarrow \text{Enc}(pk_{TA}, ok_i)$

$BB \leftarrow b_i$

Send $[vt_i], [ok_i]$ to TA

Tally(sk_{TA})

$b = \prod_{i=1}^n b_i$

$[vt] = \text{Enc}(pk_{TA}, \sum_{i=1}^n vt_i) = \prod_{i=1}^n [vt_i]$

$vt = \text{Dec}(sk_{TA}, [vt])$

$[ok] = \text{Enc}(pk_{TA}, \sum_{i=1}^n ok_i) = \prod_{i=1}^n [ok_i]$

$ok = \text{Dec}(sk_{TA}, [ok])$

if CS.Open(b, vt, ok) = 1 **then**

return vt

endif

Coercion Resistance

A stronger adversary

Active methods for attack:

- Vote for a specific candidate / randomly
- Totally abstain from voting
- Yield private keys - allow to be simulated

Passive methods for attack:

- Monitor voting system throughout election
- Same for voter except for a moment of privacy

Goal: Internet voting

Note: Coercion Resistance \Rightarrow Receipt-Freeness

The JCJ coercion resistance framework [?]

Intuition

\mathcal{A} will not be motivated to attack, if he cannot check if the attack succeeds

Techniques

- Multiple votes per voter
- Authentication using anonymous credentials

Registration phase: voter registers a real credential

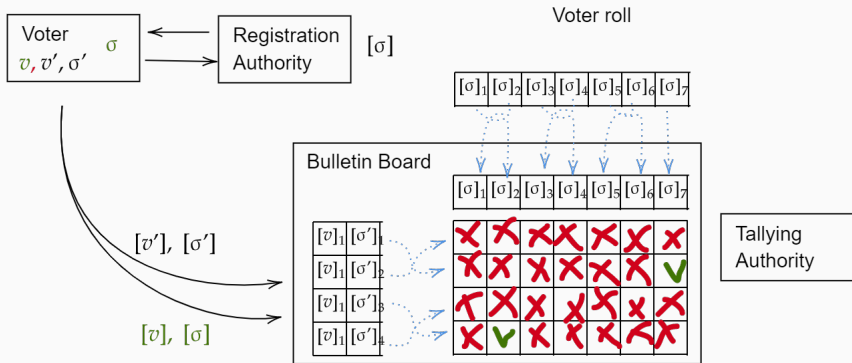
During coercion attack: voter supplies a fake credential (indistinguishable)

During moment of privacy: voter casts the real vote

During tallying: the TA must filter out **fake and duplicate votes** in a *verifiable manner* without disclosing which votes are not counted

How: Blind comparisons in the backend against a voter roll

The scheme



A key component: $\text{PET}(\text{Enc}_{\text{pk}}(m_1), \text{Enc}_{\text{pk}}(m_2)) = 1 \Leftrightarrow m_1 = m_2$

Plaintext Equivalence Test

Algorithm 9: PET for ElGamal ciphertexts

Input : $\mathbb{G}, g, q, \text{pk}_i: \prod_i^t \text{pk}_i = \text{pk}, c = (c_1, c_2), c' = (c'_1, c'_2)$

Private Input: $\text{sk}_i \in \mathbb{Z}_q: \sum_i^t \text{sk}_i = \text{sk}$

Output : $\{0, 1\}$

$c_{\text{PET}} := \frac{c}{c'} = \left(\frac{c_1}{c'_1}, \frac{c_2}{c'_2}\right)$

$z_i \leftarrow \mathbb{Z}_q$

$c_{i,\text{PET}} := c_{\text{PET}}^{z_i} = (c_{i1}, c_{i2}) = \left(\left(\frac{c_1}{c'_1}\right)^{z_i}, \left(\frac{c_2}{c'_2}\right)^{z_i}\right)$

$\pi_{i1} \leftarrow \text{NIZK}\left\{(\mathbb{G}, g, q, \text{pk}, c_{\text{PET}}, c_{i,\text{PET}}), (z_i) : c_{i,\text{PET}} = c_{\text{PET}}^{z_i}\right\}$

Publish $(c_{i,\text{PET}}, \pi_{i1})$ and wait until all players have posted. Verify the proofs π_{i1} posted from other players

$\phi := \prod_i^t c_{i,\text{PET}} = \left(\prod_i c_{i1}, \prod_i c_{i2}\right) = \left(c_{1i}^{\sum_i z_i}, c_{2i}^{\sum_i z_i}\right) = (x, y)$

$\psi_i := x^{\text{sk}_i}$

$\pi_{i2} \leftarrow \text{NIZK}\left\{(\mathbb{G}, g, q, \text{pk}, \psi_i), (\text{sk}_i) : \psi_i = x^{\text{sk}_i}\right\}$

Publish (ψ_i, π_{i2}) wait until all players have posted. Verify the proofs π_{i2} posted from other players

$\rho := y / \prod_i^t \psi_i$

return $\rho = 1$

Note: The strong Fiat-Shamir heuristic must be used If not verifiability can be broken

Assumptions

- Moment Of Privacy
- Untappable Registration (occurs one / used in multiple elections)
 - Secure transcript erasure
 - Simulation of transcript
- Anonymous casting
 - For forced abstention attack
- Coercer uncertainty about voter behavior
 - If all vote, then the abstention attack will always succeed
 - The voting authorities may inject chaffe votes on purpose

Algorithm 10: Real Coercion resistance game

```
(prms,  $V_{EL}, CS$ )  $\leftarrow$  Setup( $1^\lambda$ )
 $\{(sk_i, pk_i) \leftarrow \text{Register}(sk_{RA}, i)\}_{i=1}^n$ 
 $V_{Corr} \leftarrow \mathcal{A}(\text{corrupt})$  // Adversary corrupts voters
 $(j, vt_j) \leftarrow \mathcal{A}(V_{Hon}, \text{coerce})$  // Adversary coerces voter  $j$ 
 $b \leftarrow \{0, 1\}$ 
if  $b = 0$  then
    |  $sk_j^* \leftarrow \text{fakekey}(j)$  // generate fake credential
    |  $b_j \leftarrow \text{Vote}(j, vt_j, sk_j)$  // Moment of privacy
else
    |  $sk_j^* \leftarrow sk_j$  // yield credential
end
 $\{b_i \leftarrow \text{Vote}(i, vt_i, sk_i), \}_{i=1}^{|V_{Hon}|, \mathcal{D}}$ 
 $BB \leftarrow \text{Vote}(j, vt_j, sk_j^*)$   $\{BB \leftarrow \mathcal{A}(sk_i, vt_i, \text{cast})\}_{i=1}^{|V_{Corr}|}$ 
 $(T, \pi_T, \Gamma) := \text{Tally}(BB, sk_{TA})$ 
 $b' \leftarrow \mathcal{A}(T, \pi_T, \Gamma, BB, \text{guess})$ 
return  $b = b'$ 
```

Algorithm 11: Ideal Coercion resistance game

```
(prms,  $V_{EL}, CS$ )  $\leftarrow$  Setup( $1^\lambda$ )
 $\{(sk_i, pk_i) \leftarrow \text{Register}(sk_{RA}, i)\}_{i=1}^n$ 
 $V_{Corr} \leftarrow \mathcal{A}(\text{corrupt})$  // Adversary corrupts voters
 $(j, vt_j) \leftarrow \mathcal{A}(V_{Hon}, \text{coerce})$  // Adversary coerces voter  $j$ 
 $b \leftarrow \{0, 1\}$ 
if  $b = 0$  then
  |  $b_j \leftarrow \text{Vote}(j, vt_j, sk_j)$ 
end
// Moment of privacy
 $sk_j^* \leftarrow sk_j$  // always yield credential
 $\{b_i \leftarrow \text{Vote}(i, vt_i, sk_i), \}_{i=1}^{|V_{Hon}|, \mathcal{D}}$ 
 $BB \leftarrow \text{Vote}(j, vt_j, sk_j^*)$ 
 $\{BB \leftarrow \mathcal{A}(sk_i, vt_i, \text{cast})\}_{i=1}^{|V_{Corr}|}$ 
 $T := \text{ideal\_tally}(BB, sk_{TA})$ 
 $b' \leftarrow \mathcal{A}(T, \text{guess})$ 
return  $b = b'$ 
```

Why the ideal experiment?

- An alternative: Distinguish between $b = 0$ and $b = 1$?
- The tally might help the coercer distinguish if the coercion attempt succeeded
- For instance: The voter instruct a vote for 'Alice' but no 'Alice' votes are found (regardless of the cryptographic primitives used)
- We need to measure the effect of the cryptographic primitives

Ideal tallying functionality

- Ballots cast by V_{Hon} are treated normally
- Ballots cast by \mathcal{A} are added to the result
- Also performs weeding of double votes based on the extracted credential
- If $b = 0$ votes with sk_j^* are *not* counted
- If $b = 1$ votes with sk_j^* are counted

Main drawback Quadratic tallying complexity: $\mathcal{O}(v^2) + \mathcal{O}(nv)$ for duplicate detection and tallying

Goal $\mathcal{O}(n + v)$

3 approaches for better efficiency:

Anonymity sets

The ballot contains

- The current credential
- The real credential from the voter roll (rerandomized)
- Some $\beta - 1$ other random credentials from the BB
- The PET takes place only among the credentials of the ballot

Decentralized voting

2 Voting paradigms

- Large scale elections
 - Involvement of authorities (mixing / tallying)
 - Trust required for some properties
 - Each voter is only interested to cast their ballot (vote & go)
 - Existence of BB: contains all voting public data (broadcast channel with memory)
- Small(er) scale elections (boardroom)
 - Conducted by the voters themselves
 - No entity plays a special part
 - Robustness is more important: A voter cannot disrupt the election
 - Private channels lead to disputes

Can these two paradigms be combined?

Desirable properties for decentralized elections - [?]

- Self-Tallying (open tallying)
Any (external) entity should be able to count the ballots
(Implies verifiability)
- Perfect Ballot Secrecy
The partial election result for a particular subset of voters can be obtained only by a coalition that contains the rest of the voters
- Dispute-Freeness Embedded mechanisms avert disputes and make the participants follow the protocol (accountability)

Can these properties be achieved while minimizing communication complexity and voter-to-voter interaction?

Theorem

Self-tallying is incompatible with robustness and privacy (at the same time)

Assume: Self-tallying and robust

Assume: $n - 1$ out of n voters show up

By self tallying: Anyone can compute the result

But: The same computation can take place even if all voters show up

This reveals the preference of the last (any) voter

Theorem

Robustness through threshold secret sharing is incompatible with perfect ballot secrecy

For perfect ballot secrecy the threshold must be set to n

But this is not robust

A solution: Decentralized voting with a BB (can be replaced with a blockchain)

Anonymous Voting by 2-Round Public Discussion - [?]

Preparation

Select a group $\mathbb{G} = \langle g \rangle$

Each voter has an identity V_i

Selects $a_i \leftarrow \mathbb{Z}_q$

Round 1 - Commitment

Each V_i posts g^{a_i} , $\text{NIZK}\{(a_i), g^{a_i}\}$

When this phase ends compute:

$$\prod_{j=1}^{i-1} g^{a_j} / \prod_{j=i+1}^n g^{a_j} = g^{y_i}$$

for some *unknown* $y_i \in \mathbb{Z}_q$

Round 2 - Voting

Each V_i selects $v_i \in \{0, 1\}$ and posts $(g^{y_i})^{a_i} g^{v_i}$

Round 3 - Self-tallying

Everyone computes

$$\prod_{i=1}^n (g^{y_i})^{a_i} g^{v_i} = \prod_{i=1}^n g^{v_i} = g^{\sum_{i=1}^n v_i}$$

Solve a simple DLP

Correctness

$$\sum_{i=1}^n a_i y_i = \sum_{i=1}^n \sum_{j<i} a_i a_j - \sum_{i=1}^n \sum_{j>i} a_i a_j = 0 \text{ since: } y_i = \sum_{j<i} a_j - \sum_{j>i} a_j$$

	x_1	x_2	x_3	x_4	x_5
x_1		-	-	-	-
x_2	+		-	-	-
x_3	+	+		-	-
x_4	+	+	+		-
x_5	+	+	+	+	

Problems

Robustness: If someone refuses to vote then the result cannot be computed

Fairness: The last voter can learn the result

Improvements for robustness and fairness [KSRH12]

Robustness

Assume that only voters in \mathcal{L} have voted.

Recovery Round

Each $V_i \in \mathcal{L}$ compute:

$$\hat{h}_i = \frac{\prod_{j \in \{i+1, n\} \setminus \mathcal{L}} g^{a_j}}{\prod_{j \in \{1, i-1\} \setminus \mathcal{L}} g^{a_j}}$$

and posts cancellation tokens $\hat{h}_i^{a_i}$, $\text{NIZK}\{(a_i) : \log_g g^{a_i} = \log_{\hat{h}} \hat{h}^{a_i}\}$

Tallying becomes: $V = \prod_{i=1}^n b_i = \prod_{i \in \mathcal{L}} \hat{h}_i^{a_i} (g^{y_i})^{a_i} g^{v_i} = g^{\sum_{i=1} v_i}$

No	First round	Second round	Third round	Recovery
1	g^{x_1}	commitment	$g^{x_1 y_1} = g^{x_1(-x_2-x_3-x_4-x_5)}$	$\hat{h}_1^{x_1} = g^{x_1(x_2+x_4)}$
2	g^{x_2}	commitment	Abort	-
3	g^{x_3}	commitment	$g^{x_3 y_3} = g^{x_3(x_1+x_2-x_4-x_5)}$	$\hat{h}_3^{x_3} = g^{x_3(x_4-x_2)}$
4	g^{x_4}	commitment	Abort	-
5	g^{x_5}	commitment	$g^{x_5 y_5} = g^{x_5(x_1+x_2+x_3+x_4)}$	$\hat{h}_5^{x_5} = g^{x_5(-x_2-x_4)}$

Open Vote Network [?]

- Implementation of [?] using Ethereum
- Smart contracts (voting, registration, tallying)
- Voters are authenticated with their Ethereum user-controlled accounts
- Ethereum restrictions:
 - integers of 256 bits
 - expensive cryptographic computations
 - one vote or six registrations per block
 - small number of allowed local variables
 - order of transactions in a block and timers
- Maximum number of voters: 50 (due to gas limit)
 - Linear number of operations for Tally and Voter List
- Cost per voter: 0.73\$

Scalable Open Vote Network [?]

Improvements

- Organize voters in Merkle Tree only the root is stored (256 bits)
- Instead of voter list a voter provides a proof of membership
- Tally off-chain by an **untrusted** tallier
 - Publish computation trace in Merkle Tree $((i, t_i))$
 - Subject to verification

```
def TallyVotes(c: array []):  
  t = 1  
  for i=1 to n:  
    t = Mul(c[i], t)  
  return t
```

Step i	c_i	t_{i-1}	t_i
1	c_1	$t_0 = 1$	$t_1 = c_1$
2	c_2	t_1	$t_2 = c_2 \cdot t_1$
.
n	c_n	t_{n-1}	$t_n = c_n \cdot t_{n-1}$

Conceptual similarity between blockchain and the BB

- Append-only
- Broadcast channel
- No central authority - anyone can be a miner (given enough computing power)
- Pseudonymity

Good for universal/individual verifiability (recorded as cast)

But...

- Registration/authentication/eligibility verifiability are inherently centralized
- Does not help with verifying voter intent
- Does not help with coercion-resistance / receipt-freeness





- Intensifies threats associated with everlasting privacy
- Is it actually decentralized? (concentration of mining power)

To sum up... 'using Blockchain for voting solves a *small* part of the problem with an unnecessarily big hammer' (Ben Adida, 2017)

However... it might be useful for different types of elections - new election paradigms on a smaller scale with many different permission-types of blockchains

References

1. Ben Adida. “Helios: web-based open-audit voting”. In: Proceedings of the 17th conference on Security symposium. USENIX Association, 2008, pages 335– 348.
2. Bernhard D., Pereira O., Warinschi B. (2012) How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. ASIACRYPT 2012.
3. Hao, F., Ryan, P.Y.A. (Eds.). (2016). Real-World Electronic Voting: Design, Analysis and Deployment (1st ed.). Auerbach Publications (Chapter 11: Voting with Helios Olivier Pereira)
4. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. “Election Verifiability for Helios under Weaker Trust Assumptions”. In: ESORICS (2). Volume 8713. Lecture Notes in Computer Science. Springer, 2014, pages 327–344.
5. Ben Smyth, Steven Frink, and Michael R. Clarkson. “Computational Election Verifiability: Definitions and an Analysis of Helios and JCJ”. In: IACR Cryptol. ePrint Arch. 2015 233.
6. V Cortier, D Galindo, R Küsters, J Müller, and T Truderung. “SoK: Verifiability Notions for E-Voting Protocols”. In: IEEE Security and Privacy Symposium. 2016, pages 779–798

-  Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan, *Practical everlasting privacy*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7796 LNCS, 2013, pp. 21–40.
-  David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi, *Sok: A comprehensive analysis of game-based ballot privacy definitions*, 2015 IEEE Symposium on Security and Privacy, 2015, pp. 499–516.
-  Mihir Bellare and Amit Sahai, *Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization*, Advances in Cryptology — CRYPTO' 99 (Michael Wiener, ed.).
-  Véronique Cortier and Ben Smyth, *Attacking and fixing helios: An analysis of ballot secrecy*, J. Comput. Secur. (2013), 89–148.

-  Denise Demirel, J Van De Graaf, and R Araújo, *Improving Helios with Everlasting Privacy Towards the Public*, EVT/WOTE'12 Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections (2012).
-  Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta, *A practical secret voting scheme for large scale elections*, AUSCRYPT (Jennifer Seberry and Yuliang Zheng, eds.), LNCS, Springer, 1992, pp. 244–251.
-  Feng Hao, Peter Ryan, and Piotr Zielinski, *Anonymous voting by two-round public discussion*, Information Security, IET **4** (2010), 62 – 67.
-  Ari Juels, Dario Catalano, and Markus Jakobsson, *Coercion-resistant electronic elections*, Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005, ACM, 2005, pp. 61–70.

-  Aggelos Kiayias and Moti Yung, *Self-tallying elections and perfect ballot secrecy*, Public Key Cryptography, 2002, pp. 141–158.
-  Tal Moran and Moni Naor, *Receipt-Free Universally-Verifiable Voting with Everlasting Privacy*, 2006, pp. 373–392.
-  Patrick McCorry, Siamak Shahandashti, and Feng Hao, *A smart contract for boardroom voting with maximum voter privacy*, pp. 357–375, 01 2017.
-  Mohamed Seifelnasr and Hisham Galal, *Scalable open-vote network on ethereum*, pp. 436–450, 08 2020.