Helios: Attacks and Formal Models for Verifiability

Panagiotis Grontas

30/03/2023

NTUA - Advanced topics in cryptography (2022-2023)

Introduction

- 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

History i

- 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)

History ii

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
- $\mathsf{EA} = (\mathsf{RA}, \mathsf{TA}, \mathsf{BB})$

Auditing Process - E2E Verifiability i

- 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)

Auditing Process - E2E Verifiability ii

- Tallied 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

$\mathsf{VS}^{\mathsf{Helios}} =$
$(\mathbf{Setup}, \mathbf{SetupElection}, \mathbf{Vote}, \mathbf{Append}, \mathbf{Valid}, \mathbf{VerifyVote}, \mathbf{Publish}, \mathbf{Tally}, \mathbf{Verify})$

- **Setup**$(1^\lambda) = (\mathbb{G}, q, g, \mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q, \mathsf{BB} = \emptyset, (\mathsf{DLProve}, \mathsf{DLVerify})(\mathsf{EqDLProve}, \mathsf{EqDLVerify}), (\mathsf{DisjProve}, \mathsf{DisjVerify}))$ 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)
    - $(\mathsf{DLProve}, \mathsf{DLVerify}) = \mathsf{NIZK_H}\{(g, \mathsf{pk}), (\mathsf{sk}) : log_g \mathsf{pk} = \mathsf{sk}\}$
    - $(\mathsf{EqDLProve}, \mathsf{EqDLVerify}) = \mathsf{NIZK_H}\{(g, \mathsf{pk}, h, R), (\mathsf{sk}) : log_g \mathsf{pk} = log_h R\}$
    - $(\mathsf{DisjProve}, \mathsf{DisjVerify}) = \mathsf{NIZK_H}\{(g, \mathsf{pk}, R, S), (r) : (R, S) = \mathsf{Enc_{pk}}(g^0) \text{ OR } (R, S) = \mathsf{Enc_{pk}}(g^1)\}$

Formal Description ii

- **SetupElection** $= (\mathsf{sk} \leftarrow\$\ \mathbb{Z}_q, \mathsf{pk} = g^{\mathsf{sk}}, \mathsf{V}_{\mathit{El}}, \mathsf{CS} = \{0, 1\})$
  $\mathsf{BB} \Leftarrow \{\mathsf{pk}, \mathsf{V}_{\mathit{El}}, \mathsf{CS}, \mathsf{H}(\mathsf{pk}||\mathsf{V}_{\mathit{El}}||\mathsf{CS})\}$
  Distributed Key Generation:
  - Each member of the TA: $\mathsf{sk}_i \leftarrow\$\ \mathbb{Z}_q$
  - Publish $\mathsf{pk}_i := g^{\mathsf{sk}_i}, \mathsf{DLProve}(g, \mathsf{pk}_i, \mathsf{sk}_i)$
  - $\mathsf{pk} := \prod_{\mathsf{TA}} \mathsf{pk}_i$

  Distributed Decryption of (R, S):
  - Each member of the TA computes:
    $D_i := R^{\mathsf{sk}_i}, \mathsf{EqDLProve}(g, \mathsf{pk}_i, R, D_i, \mathsf{sk}_i)$
  - Plaintext $S / \prod_{\mathsf{TA}} D_i$

  Security analysis: TA modelled as a single entity

- **Vote**(i, v) :
  - $(R, S) = (g^r, g^v \mathsf{pk}^r) = \mathsf{Enc}_{\mathsf{pk}}(g^v, r)$, $r \leftarrow\$ \mathbb{Z}_q$
  - $\pi_{\mathsf{Vote}} = \mathsf{DisjProve}(g, \mathsf{pk}, R, S, r)$
- **Valid**$(b) \in \{0, 1\}$:
  Return 1 iff $i \in \mathsf{V}_{El}$ AND DisjVerify$(\pi_{\mathsf{Vote}}) = 1$
  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**$(\mathsf{BB}, b) \in \{0, 1\}$ Check if $b \in \mathsf{BB}$
- **Publish**$(\mathsf{BB}) = \mathsf{PBB}$ where $\mathsf{PBB} = \{(R, S), \pi_{\mathsf{Vote}}\}$
  only the last unique ballots appear for each voter without any id
  typically occurs after all voters have voted

Formal Description iv

- **Tally**(PBB, sk)
  - Validates all ballots in BB
  - $(R_\Sigma, S_\Sigma) := \prod_{b \in \mathsf{PBB}} (R_b, s_b)$
  - $g^t := \mathsf{Dec}_{\mathsf{sk}}(R_\Sigma, S_\Sigma)$
  - Compute small t
  - $\pi_{\mathsf{Tally}} = \mathsf{EqDLProve}(g, \mathsf{pk}, R_\Sigma, S_\Sigma g^{-t}, \mathsf{sk})$
- **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)
  - $(R_\Sigma, S_\Sigma) := \prod_{b \in \mathsf{PBB}} (R_b, S_b)$
  - Check if (R_Σ, S_Σ) match values in π_{Tally}
  - Check if $\mathsf{EqDLVerify}(\pi_{\mathsf{Tally}}) = 1$

The Σ protocol (EqDLProve, EqDLVerify) (Schnorr)

$$\mathsf{NIZK}_{\mathsf{H}}\{(g, \mathsf{pk}), (\mathsf{sk}) : log_g \mathsf{pk} = \mathsf{sk}\}$$

DLProve$(g, \mathsf{pk}, \mathsf{sk})$

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

EqDLVerify(T, c, s)

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

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

Simulate(g, pk, c)

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

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

$$\mathsf{NIZK_H}\{(g, \mathsf{pk}, h, R), (\mathsf{sk}) : log_g \mathsf{pk} = log_h R = \mathsf{sk}\}$$

EqDLProve(g, pk, h, R)

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

EqDLVerify$((g, \mathsf{pk}, h, R), (T_1, T_2, c, s))$

return

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

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

Simulate$(g, \mathsf{pk}, R, S, c)$

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

The Σ protocol $(\mathsf{DisjProve}, \mathsf{DisjVerify})$ (Witness indistinguishable Chaum - Pedersen)

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

$$\begin{aligned}
(R, S) = \mathsf{Enc}_{\mathsf{pk}}(g^0) &\text{ OR } (R, S) = \mathsf{Enc}_{\mathsf{pk}}(g^1) \\
(R, S) = (g^r, g^0 \mathsf{pk}^r) &\text{ OR } (R, S) = (g^r, g^1 \mathsf{pk}^r) \\
log_g R = log_{\mathsf{pk}} S &\text{ OR } log_g R = log_{\mathsf{pk}}(S/g) \\
\mathsf{EqDLProve}(g, \mathsf{pk}, R, S, r) &\text{ OR } \mathsf{EqDLProve}(g, \mathsf{pk}, R, S/g, r)
\end{aligned}$$

Assuming the voter has voted for 0:

$\pi = \mathsf{EqDLProve}(g, \mathsf{pk}, R, S, r)||\mathsf{Simulate}(g, \mathsf{pk}, R, S/g, c_{\mathcal{S}})$

where: $c_r + c_{\mathcal{S}} = c_{\mathsf{H}}$

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 $c = \mathsf{H}(T)$
- 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)

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

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

Assume that in $\mathsf{EqDLProve}(g, \mathsf{pk}, h, R, \mathsf{sk})$ 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: $c := \mathsf{H}(T_1, T_2)$
- Compute $s := \frac{b-cf}{r}$
- Set $\mathsf{sk} = \frac{a-s}{c}$

The proof verifies

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

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

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

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

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

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 $m \in \mathbb{G}$:

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

If wFS is used then the scheme is malleable:

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

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: $D_i = R^{\mathsf{sk}_i}$, $\mathsf{EqDLProve}(g, \mathsf{pk}_i, R, D_i, \mathsf{sk}_i)$ 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: $c := \mathsf{H}(T_1, T_2)$
- Compute $s := a - \mathsf{sk}_i \cdot c$
- Compute $D_i := (R^{-s}T_2)^{\frac{1}{c}}$

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

However: $log_g \mathsf{pk}_i = \mathsf{sk}_i$ but
$log_R D_i = log_R R^{\frac{-s}{c}} + log_R g^{\frac{b}{c}} = \mathsf{sk}_i - \frac{a}{c} + \frac{rb}{c} = \mathsf{sk}_i + \frac{rb-a}{c}$ 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: $c := \mathsf{H}(T_1, T_2)$
- Compute $s := a - \mathsf{sk} \cdot c$

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

- All voters vote, except for the corrupt voter.
- The current result is t and encrypted as $(R, S) = (g^r, g^t \mathsf{pk}^r)$
- 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 $r' := \frac{b - c(t-t')}{s + c \cdot \mathsf{sk}}$
- The corrupt voter casts $(g^{r'-r}, \mathsf{pk}^{r'-r})$ which is a valid 0 vote.
- The complete product is: $(R', S') = (g^{r'}, g^t \mathsf{pk}^{r'})$
- The encrypted tally does not change, but...

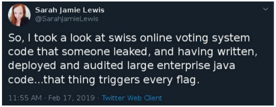
Application to Helios - Undetectably alter result (cont'd)

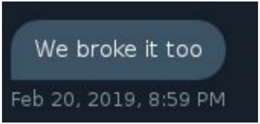
- The proof (c, s) is also valid for the relation $log_g \mathsf{pk} = log_{R'}(S' g^{-t'})$
- So the announced tally is verified as t'

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

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

Application to Swiss Voting





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:

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

Application to modern constructions

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

Introduction i

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)

Introduction ii

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

Clash attacks on Helios

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

Clash attacks on Helios

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 $\rightarrow$ individual verifiability succeeds
- The EA then submits a ballot containing its preferred option in the free slot

Clash attacks on Helios

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 $\mathsf{IndVer}_{\mathsf{VS},\mathcal{A}}$

Input : security parameter λ

Output: $\{0,1\}$

$(\mathsf{CS}, vt_0, vt_1) \leftarrow \mathcal{A}(1^\lambda)$
$b_0 := \mathsf{VS}.\mathbf{Vote}\langle \mathcal{A}(), \mathsf{V}_i(vt_0), \mathsf{pk}_{\mathsf{EA}}, \mathsf{CS}, \mathsf{BB}\rangle$
$b_1 := \mathsf{VS}.\mathbf{Vote}\langle \mathcal{A}(), \mathsf{V}_i(vt_1), \mathsf{pk}_{\mathsf{EA}}, \mathsf{CS}, \mathsf{BB}\rangle$
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

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

Helios without aliases satisfies IndVer assuming honest generation of random coins

Since

$\text{VS}.\mathbf{Vote} \equiv \text{Enc} \Rightarrow \mathbf{Pr}\left[\text{IndVer}_{\text{VS},\mathcal{A}}(1^\lambda) = 1\right] = \mathbf{Pr}[b_0 = b_1] = \text{negl}(\lambda).$

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?

Universal Verifiability i

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:

Universal Verifiability ii

Definition

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

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)

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

Input : security parameter λ

Output: $\{0,1\}$

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

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

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

  return 1

else

  return 0

end

Definition

A voting scheme VS satisfies universal verifiability if

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

Definition

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

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

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

Universal Verifiability - Additional Considerations

$\mathcal{A}$'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

Helper Oracles

Algorithm 3: Oracles for Universal Verifiability Definitions

Oracle *Register*(i)
- $(\mathsf{sk}_i, \mathsf{pk}_i) := \mathsf{VS.Register}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i()\rangle$
- $\mathsf{V}_{El} \Leftarrow (i, \mathsf{pk}_i)$

Oracle *Corrupt*(i)
- **if** $i \in \mathsf{V}_{El}$ **then**
  - $\mathsf{V}_{Corr} \Leftarrow (i, \mathsf{pk}_i, \mathsf{sk}_i)$
- **end**

Oracle *Vote*(i, vt_i)
- **if** $i \in \mathsf{V}_{El}$ **AND** $i \notin \mathsf{V}_{Corr}$ **then**
  - **if** $\exists(i, \cdot, \cdot) \in \mathsf{V}_{Hon}$ **then**
    - $\mathsf{V}_{Hon} := \mathsf{V}_{Hon} \setminus \{(i, \cdot, \cdot)\}$
  - **end**
  - $b := \mathsf{VS}.\mathbf{Vote}(i, vt_i, \mathsf{sk}_i)$
  - $\mathsf{V}_{Hon} \Leftarrow (i, vt_i, b)$
- **end**

Oracle *Cast*(i, b)
- $\mathsf{BB} \Leftarrow (i, b)$

Algorithm 4: Weak universal verifiability game W-UniVer

Input : security parameter λ
Output: $\{0, 1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathsf{VS}.\mathbf{Setup}(\lambda)$
$(T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}) \leftarrow \mathcal{A}^{\mathsf{Register},\mathsf{Corrupt},\mathsf{Vote},\mathsf{Cast}}()$
if $\mathsf{VS}.\mathbf{Verify}(T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}, \cdot) = 0$ OR $T_{\mathcal{A}} = \perp$ **then**
  return 0
end
if $\exists n_{\mathsf{V}_{Corr}} : 0 \leq n_{\mathsf{V}_{Corr}} \leq |V_{Corr}|$ AND $\exists \{vt_i^{\mathsf{V}_{Corr}}\}_{i=1}^{n_{\mathsf{V}_{Corr}}} \in \mathsf{CS}$:
$T_{\mathcal{A}} = \mathsf{result}(vt_i^{\mathsf{V}_{Corr}}) \oplus \mathsf{result}(vt_i^{\mathsf{V}_{Hon}})$ **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.

Strong Universal Verifiability

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

Input : security parameter λ
Output: $\{0, 1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathsf{VS}.\mathbf{Setup}(1^\lambda)$

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

if $\mathsf{VS}.\mathbf{Verify}(\mathsf{BB}, T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}, \cdot) = 0$ OR $T_{\mathcal{A}} = \perp$ **then**
    return 0
end

$\mathsf{V}_{Chck} = \{(\mathsf{ID}_i^{Chck}, vt_i^{Chck}, b_i^{Chck})\}_{i=1}^{n_{Chck}}$ *// Voters who verified*

if $\exists n_{\mathsf{V}_{Corr}} : 0 \leq n_{\mathsf{V}_{Corr}} \leq |\mathsf{V}_{Corr}|$ AND $\exists \{vt_i^{\mathsf{V}_{Corr}}\}_{i=1}^{n_{\mathsf{V}_{Corr}}} \in \mathsf{CS}$ AND
$\exists n' : 0 \leq n' \leq |\mathsf{V}_{Hon}| - |\mathsf{V}_{Chck}|$ AND $\exists \{vt_i'\}_{i=1}^{n'}$ *// Voters that did not check*
such that: $T_{\mathcal{A}} = \mathsf{result}(vt_i^{\mathsf{V}_{Corr}}) \oplus \mathsf{result}(vt_i^{\mathsf{V}_{Chck}}) \oplus \mathsf{result}(vt_i')$ **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: $\{0, 1\}$
$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathsf{VS}.\mathbf{Setup}(1^\lambda)$
$(T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}) \leftarrow \mathcal{A}^{\mathsf{Cast},\mathsf{Corrupt},\mathsf{Vote}}()$
if $\mathsf{VS}.\mathbf{Verify}(\mathsf{BB}, T_{\mathcal{A}}, \pi_{T_{\mathcal{A}}}, \cdot) = 0$ **OR** $T_{\mathcal{A}} = \perp$ **then**
    return 0
end
$\mathsf{V}_{Chck} = \{(\mathsf{ID}_i^{Chck}, vt_i^{Chck}, b_i^{Chck})\}_{i=1}^{n_{Chck}}$ `// Voters who verified`
if $\exists n_{\mathsf{V}_{Corr}} : 0 \leq n_{\mathsf{V}_{Corr}} \leq |\mathsf{V}_{Corr}|$ **AND** $\exists \{vt_i^{\mathsf{V}_{Corr}}\}_{i=1}^{n_{\mathsf{V}_{Corr}}} \in \mathsf{CS}$ **AND** $\exists n' : 0 \leq n' \leq |\mathsf{V}_{Hon}| - |\mathsf{V}_{Chck}|$ **AND** $\exists \{vt'_i\}_{i=1}^{n'}$ *such that:*
  $T_{\mathcal{A}} = \mathsf{result}(vt_i^{\mathsf{V}_{Corr}}) \oplus \mathsf{result}(vt_i^{\mathsf{V}_{Chck}}) \oplus \mathsf{result}(vt'_i)$ **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

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

Tally uniqueness

A correct tally of an election is unique

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

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

Proving weak universal verifiability - Helper notions iii

Accuracy

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

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

Partial counting

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

Proving weak universal verifiability - Helper notions iv

Partial tallying

If $(T_1, \cdot) = \textsf{Tally}(\mathsf{BB}_1, \mathsf{sk})$,
$(T_2, \cdot) = \textsf{Tally}(\mathsf{BB}_2, \mathsf{sk})$,
$(T, \cdot) = \textsf{Tally}(\mathsf{BB}_1 \cup \mathsf{BB}_2, \mathsf{sk})$ and
$\mathsf{BB}_1 \cap \mathsf{BB}_2 = \emptyset$ then:
$T = T_1 \oplus T_2$

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 $\mathbf{Verify}(\mathsf{BB}, T, \pi_T) = 1$ and $T \neq \perp$

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

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

BB_{Hon}

BB is honest $\Rightarrow$ no honest ballot has been deleted
From correctness and partial tallying:
$(T_{Hon}, \pi_1) = \mathbf{Tally}(\mathsf{BB}_{Hon}, \mathsf{sk})$ with $T_{Hon} = \mathsf{result}(\{v_i\}_{i=1}^{n_{Hon}})$ 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:

$$|\mathsf{BB}_{Corr}| \leq |V_{Corr}|$$

Compute

$$(T_{Corr}, \pi_2) = \textbf{Tally}(\mathsf{BB}_{Corr}, \mathsf{sk})$$

From accuracy (2) and the honest BB condition:

$$\textbf{Verify}(\mathsf{BB}_{Corr}, T_{Corr}, \pi_2) = 1$$

From tally uniqueness this tally is unique

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

From partial tallying: $T = \textbf{Tally}(\mathsf{BB}_{Corr} \cup \mathsf{BB}_{Hon}, \mathsf{sk})$

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.

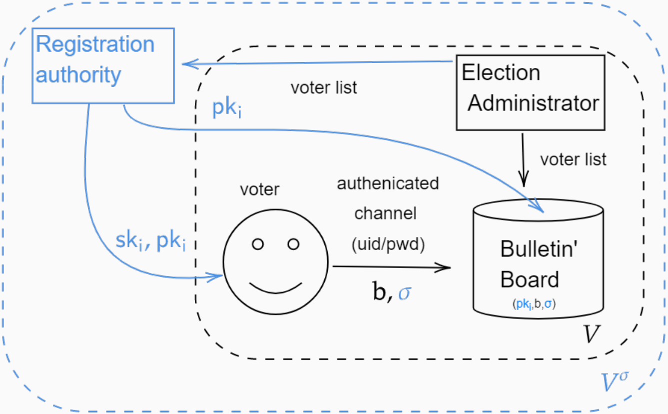
Tally Uniqueness: A verified tally passes proofs generated by DisjProve, EqDLProve. Uniqueness follows from the special soundness of the Σ protocols (DLOG)

Accuracy: If **Valid**$(\cdot) = 1$ and **Verify**$(\cdot) = 1$ then $v \in$ 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$^\sigma$ with strong universal verifiability:

- EUF-CMA-secure signature scheme
- Registration authority that hands credentials $(\mathsf{pk}_u, \mathsf{sk}_u)$ 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


Registration authority
Election Administrator
voter list
pk_i
voter list
voter
authenicated
channel
(uid/pwd)
sk_i, pk_i
b, σ
Bulletin' Board
(pk_i,b,σ)
V
V^σ

Helios and strong universal verifiability iii

Lemma

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

Every adversary $\mathcal{A}^\sigma$ 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 $\textbf{Verify}^\sigma$. All ballots are valid.

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

- Every vote $\boldsymbol{vt} \in \text{V}_{Chck}$ that has a corresponding ballot $b = (\text{pk}_u, a, \sigma)$ in BB^σ has also a ballot a in BB. a is valid from weak verifiability.

- Every vote $\mathit{vt} \in \mathsf{V}_{\mathit{Hon}} \setminus \mathsf{V}_{\mathit{Chck}}$ 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 $\mathit{vt} \in \mathsf{V}_{\mathit{Hon}} \setminus \mathsf{V}_{\mathit{Chck}}$ comes from **Vote**.
- $n_{\mathit{Corr}} \leq |V_{\mathit{Corr}}|$
  If not:
  There are two (at least 2) ballots in BB^{σ} with the same credential. But BB^{σ} is well-formed.
  Or: $\mathcal{A}^{\sigma}$ added a valid ballot without calling **Corrupt** (without knowing sk_i). This contradicts unforgeability again.

Helios and strong universal verifiability v

Lemma

If VS has weak verifiability, tally uniqueness then VS^{σ} *provides verifiability against a corrupted* RA

Since the RA is corrupted $\mathcal{A}^{\sigma}$ has all the credentials.

However the authenticated channel between $\mathcal{A}$ and the honest BB forbids him from ballot stuffing

Result:

Theorem

A voting system with weak verifiability combined with an existentially unforgeable signature scheme provides strong universal verifiability.

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