Helios Extensions For Privacy - Decentralized voting

Panagiotis Grontas

15/04/2021

NTUA -Advanced topics in cryptography (2020-2021)

Privacy

Modelling Secrecy i

- Secrecy in voting defers 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

Modelling Secrecy ii

3 approaches to defining ballot privacy

- Simulation-Based: Securely realize some ideal functionality that is secret
- Entropy-Based: Measure the amount of information the voting scheme leaks
- Game-Based: Model the privacy property as a game between the system and an attacker
  - Easier to work
  - Many attempts that differ on their relation to the result function and other properties of the voting scheme (e.g. verifiability)

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 the $\mathcal{A}$ tries to distinguish between different voting scenarios

Workflow:

- $\mathcal{C}$ setups the protocol and begins with 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 result functions

Impact of voting rules

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

The candidate with majority is declared the winner and a in case of a tie

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 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.

$$b_1 = (1, (R_1, S_1), \pi_{\mathsf{Vote},1})$$
$$b_2 = (2, (R_2, S_2), \pi_{\mathsf{Vote},2})$$

The voter replays an exact replica of b_1

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

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
- 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.

Overview of BPRIV

$\mathcal{A}$ 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 i

Algorithm 1: $\mathsf{BPRIV}^{b}_{\mathcal{A},\mathsf{VS}}$

Input : security parameter λ

Output: $\{0, 1\}$

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

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

$b' \leftarrow \mathcal{A}^{\mathbf{Vote},\mathbf{Cast},\mathbf{Tally},\mathbf{Board}}(\mathsf{pk}_{\mathsf{TA}})$

return $b = b'$

BPRIV

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

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

The BPRIV definition ii

Algorithm 2: 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 $\text{Valid}(b_0, \text{BB}_0)$ AND $\text{Valid}(b_1, \text{BB}_1)$ then
  - $\text{BB}_0 \Leftarrow b_0$
  - $\text{BB}_1 \Leftarrow b_1$
- else
  - return $\perp$

Oracle *Cast*(i, b, b)
- if $\text{Valid}(b, \text{BB}_{\text{b}})$ then
  - $\text{BB}_0 \Leftarrow b$
  - $\text{BB}_1 \Leftarrow b$
- else
  - return $\perp$

Oracle *Tally*(b)
- if $\text{b} = 0$ then
  - $(T, \pi_T) := \text{Tally}(\text{sk}_{\text{TA}}, \text{CS}, \text{BB}_0)$
  - return (T, π_T)
- else
  - $(T, \pi_T) := \text{Tally}(\text{sk}_{\text{TA}}, \text{CS}, \text{BB}_0)$
  - $\pi'_T := \mathcal{S}(\text{sk}_{\text{TA}}, \text{CS}, \text{BB}_0, \text{BB}_1, T)$
  - return (T, π'_T)

Oracle *Board*(b)
- return $\text{Publish}(\text{BB}_{\text{b}})$

Helios is BPRIV i

Observation
The visible BB consists of tuples $(i, b, \cdot)$ posted using **Cast**, **Vote**

If the tuple originates from **Vote**, the challenger can internally map it to a tuple $(i, b_0, vt_0, b_1, vt_1)$

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

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

Helios is BPRIV ii

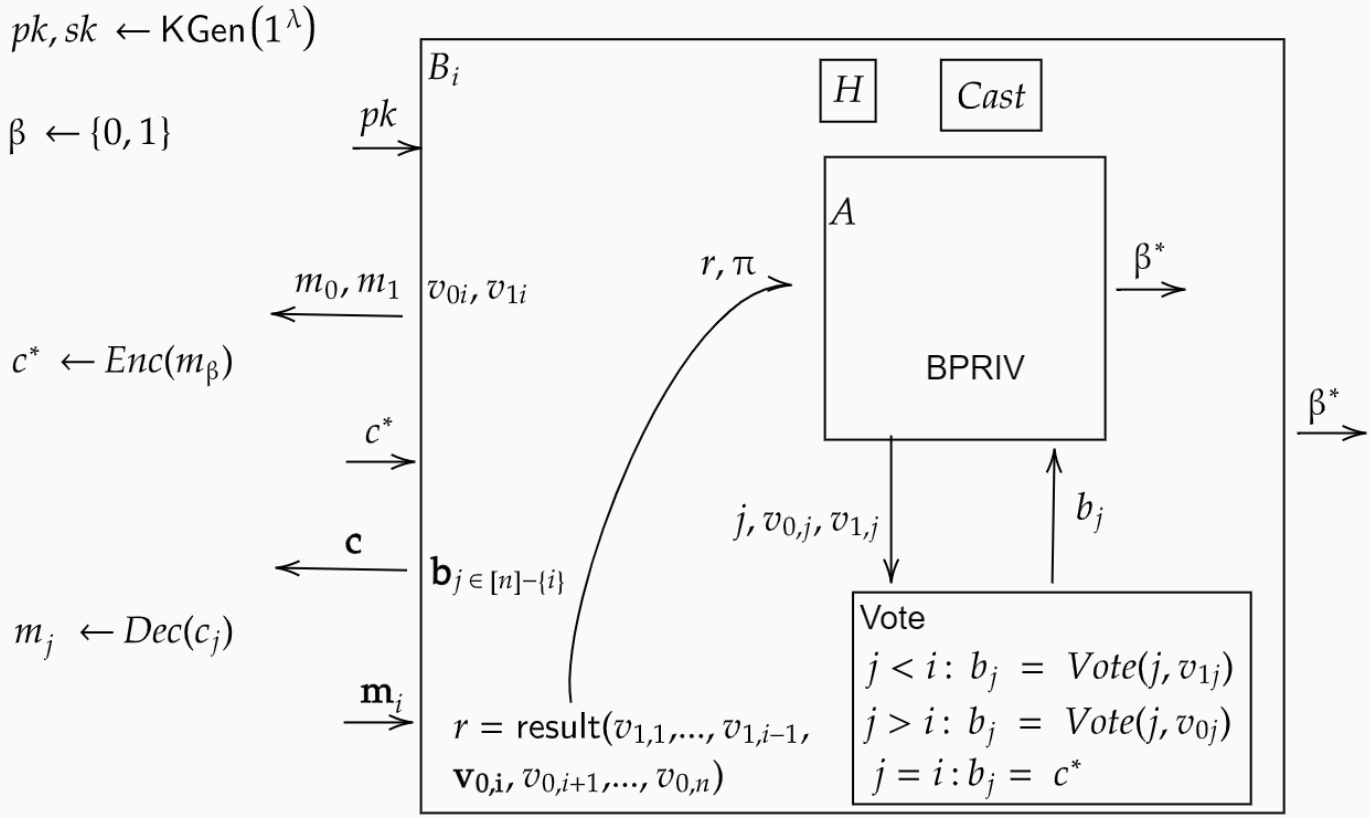
$Game_0$: BPRIV^0 - $\mathcal{A}$ sees BB_0 and the real result of the tally

$Game_1$: The tally proof in BPRIV^0 is simulated as
$\pi'_T = \text{Simulate}(g, \text{pk}, R, Sg^{-t}, c)$.
Based on the ZK property of π_T, the $\mathcal{A}$ has a negligible advantage in distinguishing between $Game_0$,$Game_1$

$\{Game_{2i}\}_{i=1}^{n_{Hon}}$
For all ballots cast using **Vote** (challenger entries $(i, b_0, vt_0, b_1, vt_1)$) replace b_{0i} in BB_0 with b_{1i} from BB_1.
$\mathcal{A}$ notices this with negligible probability due to the $NM - CPA$ property of Enc + PoK

$Game_{2n}$ is BPRIV^1

$NM - CPA \Rightarrow$ BPRIV



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

Approaches i

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

Approaches ii

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)

$(b_i, \mathrm{ok}_i) \leftarrow \mathrm{Commit}(\mathrm{ck}, vt_i)$

$[\mathrm{vt}_i] \leftarrow \mathrm{Enc}(\mathrm{pk}_{\mathrm{TA}}, vt_i)$

$[\mathrm{ok}_i] \leftarrow \mathrm{Enc}(\mathrm{pk}_{\mathrm{TA}}, \mathrm{ok}_i)$

$\mathrm{BB} \Leftarrow b_i$

Send $[vt_i], [\mathrm{ok}_i]$ **to** TA

Tally$(\mathrm{sk}_{\mathrm{TA}})$

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

$$vt = \mathrm{Dec}(\mathrm{sk}_{\mathrm{TA}}, [vt])$$

$$\mathrm{ok} = \mathrm{Enc}(\mathrm{pk}_{\mathrm{TA}}, \sum_{i=1}^{n} \mathrm{ok}_i) = \prod_{i=1}^{n} [\mathrm{ok}_i]$$

$$\mathrm{ok} = \mathrm{Dec}(\mathrm{sk}_{\mathrm{TA}}, [\mathrm{ok}])$$

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

if $\mathrm{CS.Open}(b, vt, \mathrm{ok}) = 1$ **then**

return vt

endif

endfor

Helios Extensions for receipt-freeness and participation privacy

Intuition

Privacy does not deal with malicious *voters*, that willingly interact with an attacker before/after voting to sell their votes

Assumption: $\mathcal{A}$ does not monitor voter during vote casting. Only at distinct moments before / after.

Receipt-Freeness
A (malicious) voter cannot prove how she voted, even if she wants to

Helios is not receipt-free
The malicious voter can reveal the randomness used in ballot creation (encryption) $\mathcal{A}$ can recompute the encryption and check the BB for the ballot

Revoting does not help. $\mathcal{A}$ can check which ballot finally appears in the published BB.

Solutions

BeleniosRF

- Rerandomization Server
- Changes the randomization contained in the ballot
- The voter is not alone in contributing randomness
- So the voter has no use in revealing it

Helios-KTV

- Deniable vote updating
- The voter casts b_1 (the selled vote vt_1)
- Later updates the ballot to b'_1 (a new vote vt'_1)
- Anonymization layer required
- Revoting is not allowed in many jurisdictions

Both define receipt-freeness as an extension of BPRIV

BPRIV for receipt-freeness i

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

- **Register**(i): Generate credentials for the voter
- **Corrupt**(i): Corrupt a voter for receipt-freeness (get the voter's key)
- **Receipt**(b_0, b_1): The corrupted voter casts two ballots to the respective BB. They are assumed to encode information that will allow vote selling (e.g. set last bit to one)

$\mathcal{A}$ can call **Tally** only - once the rest at will

$\mathcal{A}$ must guess which BB is tallied

A ballot b_b submitted through **Receipt**(b_0, b_1) can be swapped with $b_{1-\mathrm{b}}$ without $\mathcal{A}$ noticing

BPRIV for receipt-freeness ii

Algorithm 3: BPRIV extensions for BeleniosRF

Oracle *Register*(i)
- $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Register}(i)$
- $\mathsf{V}_{El} \Leftarrow (i, \mathsf{pk}_i, \mathsf{sk}_i)$
- return pk_i

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

Oracle *Receipt*(i, b_0, b_1)
- **if** $i \in \mathsf{V}_{Corr}$ **then**
  - **if** $\mathsf{Valid}(b_0, \mathsf{BB}_0)$ **AND** $\mathsf{Valid}(b_1, \mathsf{BB}_1)$ **then**
    - $\mathsf{BB}_0 \Leftarrow b_0$
    - $\mathsf{BB}_1 \Leftarrow b_1$

- The ballot b_0 can be replaced with b_1 without $\mathcal{A}$ noticing
- No strategy on behalf of the voter needs to be applied
  - More practical schemes (single pass)
  - Revoting is not always legal
- Helios is not receipt-free: $\mathcal{A}$ can encode information in $\mathsf{BB}_0, \mathsf{BB}_1$ (randomness) to distinguish them

Signatures on Randomizable Ciphertexts i

- $(\mathsf{dk}, \mathsf{pk}, \mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m, \mathsf{pk})$
- $\{0,1\} \leftarrow \textbf{Verify}(\mathsf{vk}, \sigma, m)$
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m, \mathsf{vk})$
- $m \leftarrow \mathsf{Dec}(\mathsf{dk}, c)$

$$(c', \sigma') \leftarrow \mathsf{Random}(\mathsf{pk}, \mathsf{vk}, c, \sigma) \approx (\mathsf{Enc}(\mathsf{pk}, m, \mathsf{vk}), \mathsf{Sign}(\mathsf{sk}, c', \mathsf{pk}))) \text{ where}$$

$$c = \mathsf{Enc}(\mathsf{pk}, m, \mathsf{vk}) \sigma = \mathsf{Sign}(\mathsf{sk}, c, \mathsf{pk})$$

$$\textbf{Vote}(id, \mathsf{pk}, \mathsf{vk}, vt) = (c' = \mathsf{Enc}(\mathsf{pk}, m, \mathsf{vk}), \mathsf{Sign}(\mathsf{sk}, c', \mathsf{pk}))$$

$$\mathsf{Append}(b) = \mathsf{Random}(\mathsf{pk}, \mathsf{vk}, b)$$

Verifiability: Validate the signatures except the proofs

Helios-KTV i

Participation privacy: The voter cannot prove *if she voted or abstained*.

More serious attack for Internet voting due to larger scale

Basic idea: *dummy* ballots

- cast by a TTP - the posting trustee
- at random voting intervals during the voting phase
- contain a null vote
- many dummy ballots per voter - id $\{\textbf{VoteDummy}(i) = \textbf{Vote}(i, \mathrm{pk}, \cdot, 0)\}_{i=1}^{m}\ m \leftarrow\!\!\$\ [n]$
- before tallying: homomorphically aggregate all ballots per voter id
- the end result contains the real vote of voter id

Helios-KTV ii

For Receipt-Freeness: Deniable vote updating

- the $\mathcal{A}$ wants to cast a ballot $b_{\mathcal{A}} = \textbf{Vote}(i, \text{pk}, \text{sk}_i, vt_{\mathcal{A}})$
- the voter obeys
- at a different time (earlier or later) the voter casts a different ballot b_v that updates to correspond to vote vt_v
- $\textbf{DeniablyUpdate}(i, vt_v, vt_{\mathcal{A}}) = \textbf{Vote}(i, \text{pk}, \text{sk}_i, vt_v - vt_{\mathcal{A}})$
- need to know previous vote

Assumptions

- available public key infrastructure
- anonymous channel
- $\mathcal{A}$ does not watch voter during the complete election
- trusted posting trustee, BB, TA

Extend BPRIV using two oracles

Helios-KTV iii

Main idea: Vote swapping with abstention $\mathcal{A}$ cannot distinguish if V_{i_0} abstained and V_{i_1} voted at most k times or vice versa

Algorithm 4: BPRIV extension for KTV-Helios

Oracle $\mathit{VoteAbstain}_{\mathbb{P}}(i_0, i_1, vt_1, \cdots, vt_k)$

$m_b, m_{1-b} \leftarrow\$ \mathbb{P}$

$\{b_{i_b,i}\}_{i=1}^{m_b} \leftarrow \mathsf{VoteDummy}(i_b)$

$\{b_{i_{1-b},i}\}_{i=1}^{m_{1-b}} \leftarrow \mathsf{VoteDummy}(i_{1-b})$

$\{b_{i_b,i} \leftarrow \mathsf{Vote}(i_b, \mathsf{pk}, \mathsf{sk}_{i_b}, vt_i)\}_{i=1}^{k}$

$\{\mathsf{BB}_b \Leftarrow b_{i_b,i}\}_{i=1}^{k+m_b}$

$\{\mathsf{BB}_{1-b} \Leftarrow b_{i_{1-b},i}\}_{i=1}^{m_{1-b}}$

The adversarial advantage depends on distinguishing $k + m_b$ from m_{1-b} where $m_b, m_{1-b} \leftarrow\$ \mathbb{P}$

Helios-KTV iv

For b $= 0$ the voter sells her vote. For b $= 1$ the voter tries to fool the coercer and executes **DeniablyUpdate**

distinguishing factor - number of ballots in BB_0, BB_1 - obfuscation using **VoteDummy**

Algorithm 5: BPRIV extensions for KTV-Helios

Oracle *Receipt*$(i, vt_{\mathcal{A}}, vt_v)$

- $b_{\mathcal{A}} =$ **Vote**$(i, \mathsf{pk}, \mathsf{sk}_i, vt_{\mathcal{A}})$
- $BB_0 \Leftarrow b_{\mathcal{A}}$
- $BB_1 \Leftarrow b_{\mathcal{A}}$
- $b_v =$ **DeniablyUpdate**$(i, vt_v, vt_{\mathcal{A}})$
- $BB_1 \Leftarrow b_v$
- $BB_0 \Leftarrow$ **VoteDummy**(i)
- $BB_1 \Leftarrow$ **VoteDummy**(i)

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

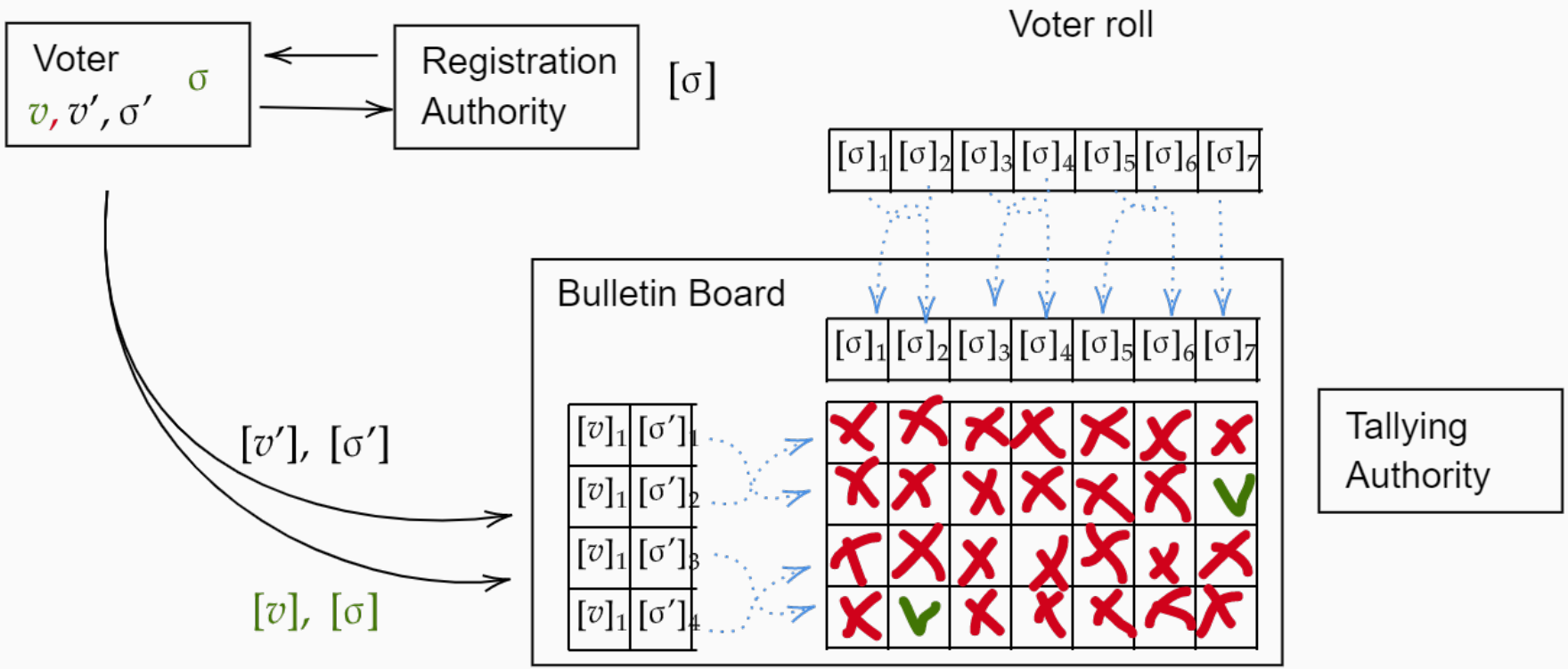
During coercion attack: voter supplies a fake credential (indistinguishable)

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: PET $(\mathsf{Enc}_{\mathsf{pk}}(m_1), \mathsf{Enc}_{\mathsf{pk}}(m_2)) = 1 \Leftrightarrow m_1 = m_2$

Plaintext Equivalence Test

Algorithm 6: PET for ElGamal ciphertexts

Input : $\mathbb{G}, g, q$, pk_i: $\prod_i^t \mathsf{pk}_i = \mathsf{pk}$, $c = (c_1, c_2)$, $c' = (c'_1, c'_2)$
Private Input: $\mathsf{sk}_i \in \mathbb{Z}_q$: $\sum_i^t \mathsf{sk}_i = \mathsf{sk}$
Output : $\{0, 1\}$

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

$z_i \leftarrow\$ \mathbb{Z}_q$

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

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

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

$\phi := \prod_i^t c_{i,\mathsf{PET}} = (\prod_i c_{1i}, \prod_i c_{2i}) = (c_{1i}^{\sum_i z_i}, c_{2i}^{\sum_i z_i}) = (x, y)$

$\psi_i := x^{sk_i}$

$\pi_{i2} \leftarrow \mathsf{NIZK}\{(\mathbb{G}, g, q, \mathsf{pk}, \psi_i), (sk_i) : \psi_i = x^{sk_i}\}$

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 7: Real Coercion resistance game

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

Algorithm 8: Ideal Coercion resistance game

$(\text{prms}, \mathsf{V}_{El}, \text{CS}) \leftarrow \textbf{Setup}(1^{\lambda})$
$\{(\text{sk}_i, \text{pk}_i) \leftarrow \text{Register}(\text{sk}_{\text{RA}}, i)\}_{i=1}^{n}$
$V_{Corr}, \leftarrow \mathcal{A}(\textbf{corrupt})$ *// Adversary corrupts voters*
$(j, vt_j) \leftarrow \mathcal{A}(V_{Hon}, \textbf{coerce})$ *// Adversary coerces voter j*
$\text{b} \leftarrow\$ \{0, 1\}$
if $\text{b} = 0$ **then**
  $b_j \leftarrow \textbf{Vote}(j, vt_j, \text{sk}_j)$
end
// Moment of privacy
$\text{sk}_j^* \leftarrow \text{sk}_j$ *// always yield credential*
$\{b_i \leftarrow \textbf{Vote}(i, vt_i, \text{sk}_i),\}_{i=1}^{|V_{Hon}|, \mathcal{D}}$
$\text{BB} \Leftarrow \textbf{Vote}(j, vt_j, \text{sk}_j^*)$
$\{\text{BB} \Leftarrow \mathcal{A}(\text{sk}_i, vt_i, \textbf{cast})\}_{i=1}^{|V_{Corr}|}$
$T := \text{ideal_tally}(\text{BB}, \text{sk}_{TA})$
$\text{b}' \leftarrow \mathcal{A}(T, \textbf{guess})$
return $\text{b} = \text{b}'$

The model iii

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 (invariant of the protocol)
- Ideal tallying functionality ideal_tally:
  - ballots cast by V_{Hon} are treated specially
  - For ballots cast by $\mathcal{A}$:
  - 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

JCJ evolution

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 real credential

- 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

JCJ evolution (cont'd)

Blinded hashing

- The credential $[\sigma]$ is blinded to obtain $[\sigma]^z$
- The value z is jointly computed by the RA members
- Instead of PET $[\sigma]^z$ is decrypted to obtain σ^z (credential fingerprint)
- This fingerprint is sent through a hashtable $\{(\sigma^z, b)\}$
- For verifiability: σ^z is made public so that everybody can check the hashtable

Vulnerable to the tagging attack (Pfitzmann)

- $\mathcal{A}$ forces a voter to reveal the credential
- Then $\mathcal{A}$ posts two ballot one with $[\sigma]$ and one with $[\sigma^2]$
- Two fingerprints will be produced σ^z and one with σ^{2z}
- $\mathcal{A}$ squares all elements in the BB.
- If a square matches another element and both have been discarded then $\mathcal{A}$ learns that σ was fake

Note: This attack applies only on fake credentials, not on duplicates
So $\mathcal{O}(v^2)$ becomes $\mathcal{O}(v)$

JCJ evolution (cont'd)

Structured credentials

Check of validity is self contained in the credential

- A credential is a tuple $(r \leftarrow_{\$} \mathbb{Z}_q, a \leftarrow_{\$} \mathbb{G}, b = a^y, c = a^{x+rxy})$ where $x, y \leftarrow_{\$} \mathbb{Z}_q$ are secret keys of the RA
- r should be kept secret by the voter
- A fake credential can be made by selecting a new r
- If (r, a, b, c) is valid then (r, a^l, b^l, c^l) is valid
- $b = ([\boldsymbol{vt}], [a], [a^r], [b^r], [c], o^r) = (\cdot, u, v, w, z, \cdot)$
- Duplicates are identified by o^r
- Ballot validity if $z \cdot u^{-x} \cdot w^{-x}$ decrypts to 1

Decentralized voting

Introduction

2 Voting paradigms

- Large scale elections
  - Involvement of authorities (mixing / tallying)
  - Trust required for some properties
  - Each voters plays a small part (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 - [KY02]

- Self-Tallying (open tallying)
  Any (external) entity should be able to count the ballots
  (Implies verifiability) it does not depend on secret keys
- 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

Relationships

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

Proof.
Assume: Self-tallying and robust

Assume: $n - 1$ from 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 But this reveals the preference of the last (any) voter □

Theorem
Robustness through threshold secret sharing is incompatible with perfect ballot secrecy

Proof.
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 - [HRZ10]

Preparation
Select a group $\mathbb{G} = g$

Each voter has an identity V_i

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

Round 1 - Commitment
Each V_i posts g^{a_i}, NIZK$\{(a_i), g^{a_i}\}$

When this phase ends:
$g^{y_i} = \prod_{j=1}^{i-1} g^{a_i} / \prod_{j=i+1}^{n} g^{a_i}$

Round 2 - Voting
Each V_i selects $vt_i \in \{0, 1\}$ and posts $(g^{y_i})^{a_i} g^{vt_i}$ for some unknown $y_i \in \mathbb{Z}_q$

Round 3 - Self-tallying
Everyone computes
$\prod_{i=1}^{n} (g^{y_i})^{a_i} g^{vt_i} = \prod_{i=1}^{n} g^{vt_i} = g^{\sum_{i=1}^{n} vt_i}$

Solve a simple DLP

Analysis [HRZ10]

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

For multiple candidates

- Parallel voting (reject/accept) $0, 1$ for k candidates
- Baudron [BFPPS01] counters:
  - Select m st: $2^m > n$
  - To vote for candidate i cast: $\boldsymbol{vt}_i = 2^{(i-1)m}$
  - During tallying: $\sum_i \boldsymbol{vt}_i = 2^0 c_1 + 2^m c_2 + \cdots + 2^{(k-1)m} c_k$
  - Retrieve c_i

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:

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

and posts cancellation tokes $\hat{h}_i^{a_i}$, 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 [MSH17]

- Implementation of [HRZ10] 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 [SGY20]

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}	$t_n = c_n \cdot t_{n-1}$

Blockchain and voting i

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

Blockchain and voting ii

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

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

Relations between properties

Verifiability i

Orthogonal properties

IV without UV

- Construct a scheme with $(vt_i, r_i) \leftarrow \textbf{Vote}(i, vt_i)$
- Individual verifiability because of r_i
- Allows ballots stuffing
- BeleniosRF: RA and rerandomization server collusion can change contents of vote

UV without IV if everyone votes

- Construct a scheme with $vt_i \leftarrow \textbf{Vote}(i, vt_i)$
- Clash for same preferences

Verifiability ii

EV implies IV

- Nobody can construct a ballot unless they know sk_i (private voter credential)
- sk_i allows individual verifiability

Verifiability and Privacy i

Universal verifibiality is uncompatible with unconditional privacy

- Unconstrained $\mathcal{A}$
- Compute election tally of a sublist of $n - 1$ voters
- Decryption and comparison with the tally of n voters
- Reveal the option of the remaining voter

Verifiability and Privacy ii

Privacy implies individual verifiability

- Assume no individual verifiability
- A corrupt EA can replace all ballots except one
- Learn the particular preference

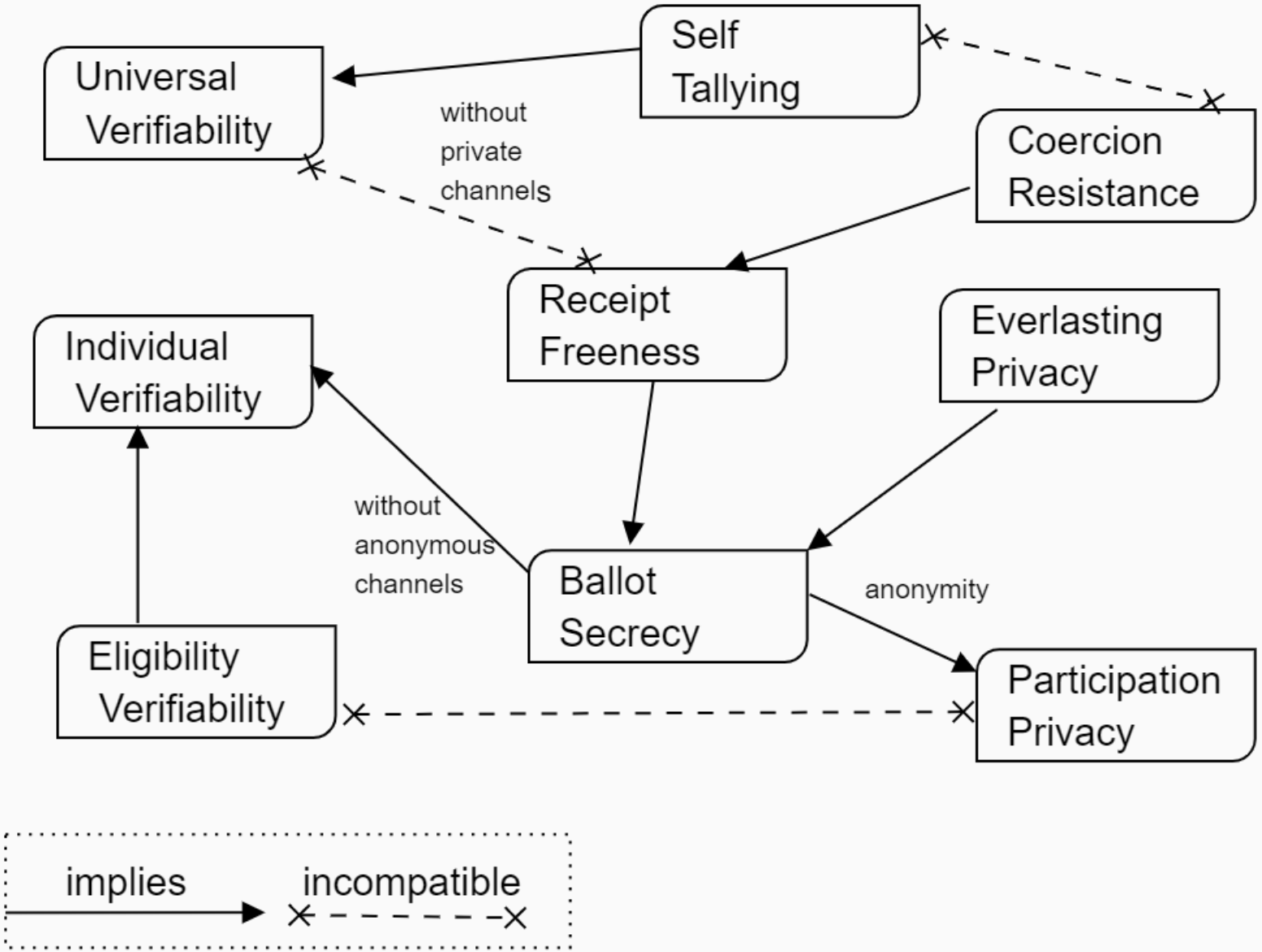
Receipt-freeness is incompatible with universal/individual verifiability

- Encryption randomness can be used as a receipt
- Without randomness: receipt-freeness
- With randomness: verifiability
- No private or anonymous channels are used

Privacy

- Receipt freeness implies privacy
- Everlasting privacy implies privacy
- The reverse does not hold
- Everlasting privacy is related to coercion resistance
- Coercion resistance and receipt-freeness: Depends on the model

Overview of relationships



Open questions

- Security Analysis of Zeus
  - Check if the pitfalls of the Fiat - Shamir Heuristic apply
  - Formal proof of verifiability and privacy
  - Security issues
- Decentralized voting with coercion resistance
  - Ring signatures
  - Self-tallying incompatibility
- Relations between properties
  - Formal proofs

References

References

1. Non-malleable Encryption: Equivalence between Two Notions, and an Indistinguishability-Based Characterization, Mihir Bellare and Amit Sahai,Crypto'99, 519–536.
2. Cortier, Véronique and Smyth, Ben. 'Attacking and Fixing Helios: An Analysis of Ballot Secrecy'. Journal of Computer Security, 1 Jan. 2013 : 89 – 148.
3. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions". In: 2015 IEEE Symposium on Security and Privacy, SP 2015, pages 499–516.
4. A. Kiayias and M. Yung (2002). Self-tallying elections and perfect ballot secrecy. In International Workshop on Public Key Cryptography, pages 41-58. Springer,2002.
5. F. Hao, P. Y. Ryan, and P. Zielinski (2010). Anonymous voting by two-round public discussion. IET Information Security, 4(2):62-67
6. D. Khader, B. Smyth, P. Y. Ryan, and F. Hao (2012). A fair and robust voting system by broadcast. In 5th International Conference on Electronic Voting, volume 205, pages- 285 299.
7. Baudron O., Fouque P., Pointcheval D., Poupard G., Stern J.: 'Practical multi-candidate election system'. Proc. 20th ACM Symp. on Principles of Distributed Computing, 2001, pp. 274–283.
8. Nasser, Y., Okoye, C., Clark, J., Ryan, P. Y. A. (2016). Blockchains and voting: Somewhere between hype and a panacea.
9. Patrick McCorry, Siamak Shahandashti and Feng Hao (2017), A Smart Contract for Boardroom Voting with Maximum Voter Privacy the 21st International Conference on Financial Cryptography and Data Security (FC'17).
10. Mohamed Seifelnasr and Hisham S. Galal and Amr M. Youssef, Scalable Open-Vote Network on Ethereum, (FC'20), eprint 2020/0033.