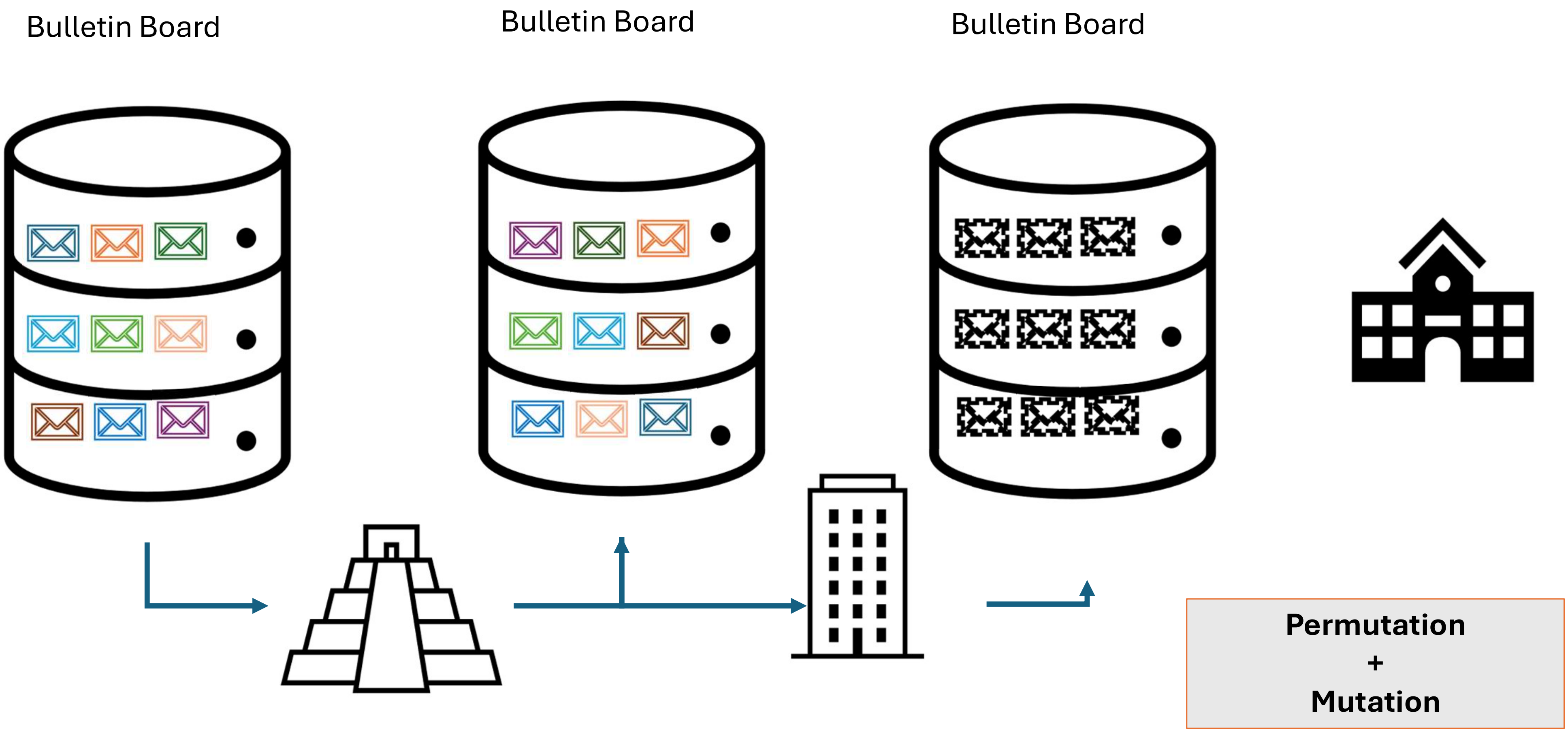
Mixnets


Bulletin Board
Bulletin Board
Bulletin Board
Permutation
+
Mutation

RSA Decryption Mixnets

- Each mixer i has a pair of RSA keys (sk_i, pk_i)
- The voter encrypts their choice using the public RSA keys of the mixers in reverse
  - $b_i = Enc_1(Enc_2(\ldots Enc_m(v_i) \ldots))$
  - $L_0 = (b_i)_{i=1}^{n}$
- Each mixer permutes the list of ballots using a random permutation π_i
- and decrypts using their private key (mutation)
- The first mixer will append to the BB:
  - $L_1 = \left(Dec_1(b_i)\right)_{\mathrm{i}=\pi_1^{-1}(1)}^{\pi_1^{-1}(n)}$

RSA Decryption Mixnets

- This process is repeated for every mixer
- In the end, the BB contains

$$L_m = (v_i)_{\mathrm{i}=\pi_m^{-1}\circ\cdots\circ\pi_1^{-1}(1)}^{\pi_m^{-1}\circ\cdots\circ\pi_1^{-1}(n)}$$

- Remarks:
  - The permutation could simply be to sort the encryptions as binary string
  - The last mixer knows the plaintext but not the voter identity
  - One honest mixer should be enough for security
    - Mixers should be entities with conflicting interests
  - Computationally expensive for the voter: $O(m)$ encryptions
  - Allows counting through the use of complex voting rules

ElGamal Decryption Mixnets

- Each mixer M_j has a key pair: $\left(sk_j, pk_j\right) = (x_j, g^{x_j})$
- The combined public key of the mixnet is $Y = \prod_j pk_j = g^{\sum x_j}$
- The voter encrypts their choice using Y
    - $b_{i0} = Enc_Y(v_i) = (g^{r_{i0}}, v_i Y^{r_{i0}})$
- Each M_j removes an encryption layer using their private key
    - $b_{ij} = Dec_{x_j}\left(b_{ij-1}\right)$
    - Applies new randomness r_{ij}
    - $L_j = \left\{b_{ij}\right\}_{i=1}^{n} = \left\{(g^{\sum_{k=0}^{j} r_{ik}}, v_i g^{\sum_{k=j+1}^{m} x_k \sum_{k=0}^{j} r_{ik}})\right\}_{i=1}^{n}$
    - Permutes using π_j

ElGamal Reencryption Mixnets

- Each mixer M_j reencrypts and permutes the ballot list using Y
- On input $L_{j-1} = \{Enc_Y(v_i, r_i)\}_{i=1}^n$
- Selects $\left\{r_{ij} \overset{\$}{\leftarrow} \mathbb{Z}_q\right\}_{i=1}^n$
- Computes
  - $L_j = \left\{Enc_Y\left(v_i, r_{ij}\right) \cdot Enc_Y\left(1, r_{ij}\right)\right\}_{i=1}^n = \left\{\left(g^{\sum_{k=0}^{j} r_{ik}}, v_i Y^{\sum_{k=0}^{j} r_{ik}}\right)\right\}_{i=1}^n$
- Permutes using π_j
- All mixers jointly decrypt after L_m has been posted

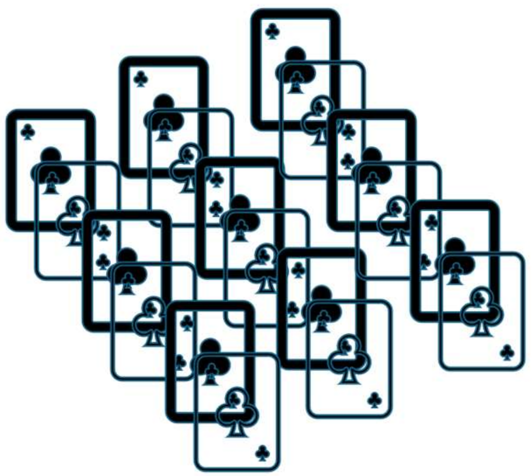
The tagging attack

- A generic attack applicable to all types of anonymous channels!
- Adversarial goal: reveal the input of V_i with the help of a corrupted user V_j willing to sacrifice their input
- The adversary
  - Retrieves the initial input of V_i: $c_{i0} = (g^r, v_i Y^r)$
  - Selects $\tau \xleftarrow{\$} \mathbb{Z}_q$ and computes $c_{i0}^{\tau} = (g^{r\tau}, v_i^{\tau} Y^{r\tau})$
  - Replaces V_j's input with c_{i0}^{τ}
- The output of the mixnet contains both v_i, v_i^{τ}
- The adversary computes for all outputs $x \rightarrow x^{\tau}$ and checks for duplicates

Pfitzmann, B.: Breaking an efficient anonymous channel. In: EUROCRYPT'94. pp. 332–340 (1995)

Verifiable mixnets – Proofs of Shuffles

- Protect against corrupted mixers that aim to omit or alter inputs
- The mixer provides a proof of (correct) shuffle that:
  - No plaintexts were *modified*
  - No ciphertexts were *removed* or *inserted*
  - The output ciphertexts are only a reencryption and permutation of the input ciphertexts.
- Without revealing:
  - The permutation π
  - The reencryption factors r_i
- Many solutions in the literature



A simple 2 × 2 verifiable shuffle

- Input
  - $c_0 = Enc_Y(m_0, r_0), c_1 = Enc_Y(m_1, r_1)$
- Output
  - $c'_0 = ReEnc(c_b) = Enc_Y(m_b, r_b{}'), c'_1 = ReEnc(c_{1-b}) = Enc_Y(m_{1-b}, r_{1-b}{}')$
- Proof that $c'_i = ReEnc(c_i)$
  - Prove that they encrypt the same message
  - If $c_i = (G, mR)$ then $c_i{}' = (G', mR')$
  - This means that $DL_g(G \cdot G'^{-1}) = DL_Y(R \cdot R'^{-1})$
  - Use the Chaum – Pedersen Protocol
- Proof of correct shuffle
  - Prove that $\{c'_0, c'_1\}$ is a shuffle of $\{c_0, c_1\}$
  - Prove that $c'_i = ReEnc(c_i)\ AND\ c'_{1-i} = ReEnc(c_{1-i})\ OR\ c'_i = ReEnc(c_{1-i})\ AND\ c'_{i-1} = ReEnc(c_i)$
  - Composition of Chaum – Pedersen Protocols

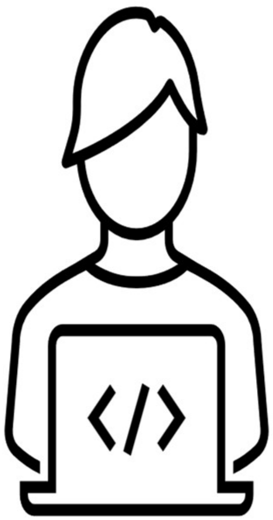
Bayer – Groth Proof of Shuffle

- Public Input
  - Two sets of ciphertexts $C_1, \ldots, C_n$ and $C_1', \ldots, C_n{}'$ in a group $\mathbb{G}$ of prime order q
  - Encrypted with pk
- Private input $\pi, \boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$ such that
  - $C_i' = C_{\pi(i)} \cdot Enc_{pk}(1, \rho_i)$
- Proof of Knowledge of Permutation
  - Product Argument: A set of committed values has a particular product
- Proof of Knowledge of Reencryption Factors
  - Mult exponentiation argument: The product of a set of ciphertexts raised to a set of committed exponents yields a particular ciphertext

Bayer, S., Groth, J. (2012). Efficient Zero-Knowledge Argument for Correctness of a Shuffle—EUROCRYPT 2012

Bayer – Groth Proof of Shuffle

9 round HVZK argument



$Commit(\pi(1), \dots, \pi(n))$

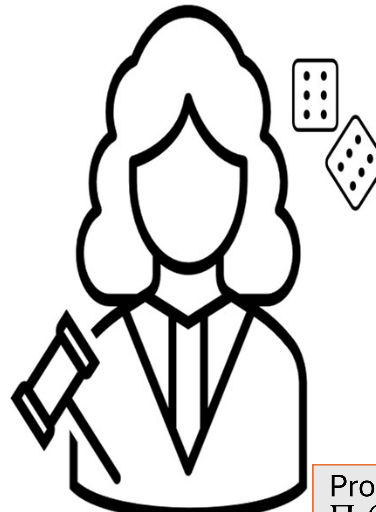
$x \xleftarrow{\$} \mathbb{Z}_q$

$Commit(x^{\pi(1)}, \dots, x^{\pi(n)})$

$c, z \xleftarrow{\$} \mathbb{Z}_q$

Prove that $\prod_i (d_i - z) = \prod_i (x^i + ic - z)$ using the product argument

Prove that $Enc_{pk}(1, r)\ \prod_i {C'}_i^{x^{\pi(i)}} = \prod_i C_i^{x^i}$ using the multiexponentiation argument



The prover must convince the verifier that the same permutation has been used for $1, \dots, n$ and $x^1, \dots, x^n$

Prove that the $P(k) = \prod_i (d_i - k) - \prod_i (x^i + ic - k)$ is the zero polynomial

Schwartz-Zippel lemma: This can be cheated with negligible probability if the permutation is not known

Bayer, S., Groth, J. (2012). Efficient Zero-Knowledge Argument for Correctness of a Shuffle—EUROCRYPT 2012

Bayer – Groth Proof of Shuffle

- State of the art in proof size $O(\sqrt{n})$
- Verification time $O(n)$
- Prover time $O(log(\sqrt{n})n)$
- Main trick for efficient communication complexity:
  - Arrange the input ciphertexts into a $k \cdot l$ matrix where $k = O(\sqrt{n})$
  - Use Generalised Pedersen Commitment to commit to columns
- First prover message
  - Send $\mathbf{cm}_{\Pi} = GPC(\boldsymbol{\pi}_{\mathbf{k}}, \mathbf{r})$ were $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^k$ and $\bigcup_k \boldsymbol{\pi}_{\mathbf{k}} = \boldsymbol{\pi}$
- Second prover message
  - Send $\mathbf{cm}_{\mathbf{X}} = GPC(\mathbf{x}^{\boldsymbol{\pi}_{\mathbf{k}}}, \mathbf{s})$ were $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$ and $\bigcup_k \boldsymbol{\pi}_{\mathbf{k}} = \boldsymbol{\pi}$
  - The permutation was fixed before the prover saw x

- **Generalized Pedersen Commitment**
- Commitment to a vector $\mathbf{m} = (m_1, \dots, m_n)$
- $\mathbb{G}$ is a cyclic group of prime order q generated by $g_1, \dots, g_n, h$
  - $GPC(\mathbf{m}, r) = h^r \prod_i g_i^{m_i}$

Bayer, S., Groth, J. (2012). Efficient Zero-Knowledge Argument for Correctness of a Shuffle—EUROCRYPT 2012

Bayer – Groth Proof of Shuffle

- Third message: Both prover and verifier compute
  - $\mathbf{cm}_{-z} = \mathrm{GPC}(-\mathbf{z}, \mathbf{0})$
  - $\mathbf{cm}_{\mathbf{D}} = \mathbf{cm}_{\boldsymbol{\Pi}}^{c} \otimes \mathbf{cm}_{\mathbf{X}} = \mathrm{GPC}(c \cdot \boldsymbol{\pi}(\mathbf{i}) + \mathbf{x}^{\boldsymbol{\pi}(\mathbf{i})})$ which is a commitment to $c\pi(i) + x^{\pi(i)}$ with randomness $cr_i + s_{\mathrm{i}}$
  - The verifier does not know $\pi(i), r_i, s_{\mathrm{i}}$ but can compute the values homomorphically
  - $\mathbf{cm}_{D} \otimes \mathbf{cm}_{-z} = \mathrm{GPC}(\mathbf{d} - \mathbf{z})$ where $d_i = c\,\pi(i) + x^{\pi(i)}$
  - Use the product argument to show knowledge of d_i, r_i, s_i such that:
    - $\prod_i (d_i - z) = \prod_i (x^i + ic - z)$ a polynomial and its permutation in z - identical roots
    - The value $\prod_i (x^i + ic - z)$ can be computed by the verifier

Bayer, S., Groth, J. (2012). Efficient Zero-Knowledge Argument for Correctness of a Shuffle–EUROCRYPT 2012

Bayer – Groth Proof of Shuffle

- Third message: The prover computes
  - $\rho \leftarrow \boldsymbol{\rho} \odot \mathbf{s}$
  - $\mathbf{C}^{\mathbf{x}} = Enc(1, \rho) \cdot \mathbf{C}'^{\mathbf{x}_\pi}$ where $\mathbf{x} = (x^1, \ldots, x^n)$
  - The verifier can compute $\mathbf{C}^{\mathbf{x}}$
  - Using the multi exponentiation argument it convinces the verifier that $Enc(1, \rho) \cdot \mathbf{C}'^{s}$ was computed correctly
- Note that because of the homomorphic properties
  - $\prod_i m_i^{x^i} = \prod_i m_i'^{x^i} \Rightarrow \log \sum (m_i)\, x^i = \log \sum \left(m_{\pi^{-1}(i)}{}'\right) x^i$
  - This means that **wvhp** $m_{\pi(i)} = m_i{}'$
- The shuffle was performed correctly

Bayer, S., Groth, J. (2012). Efficient Zero-Knowledge Argument for Correctness of a Shuffle–EUROCRYPT 2012

Voting Paradigms

Helios and extensions
JCJ – Coercion Resistance
Voting with blind/ring signatures
OpenVote

Helios

Helios' Facts

Trust the vote.

- Elections in the browser
  - Open-Audit: Everyone has access to all election data for verifiability
  - **Trust no one for integrity – trust the server for privacy**
  - Low coercion environments
- 2.000.000 votes cast so far
  - ACM, IACR and university elections
  - Can be used online https://vote.heliosvoting.org/ or deployed locally
- Based on:
  - Verifiable mixnets – Helios 1.0 (Sako-Killian, Eurocrypt 95)
  - **Homomorphic tallying – Helios 2.0 (Cramer-Genaro-Shoenmakers, Eurocrypt 97)**
  - **Benaloh Challenge**
- Many variations
  - Belenios (Helios-C)
  - Zeus

Ben Adida. 2008. Helios: web-based open-audit voting. In Proceedings of the 17th conference on Security symposium (SS'08). USENIX Association, USA, 335–348.

Participants

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

Auditing Process

- Individual Verifiability
  - Cast as intended
    - 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.
    - Every voter can check if the assigned number exists in the Ballot Tracking Center (BTC).

Auditing Process

- Universal Verifiability
  - Tallied as recorded - Every interested party may
    - Retrieve ballots from BTC
    - Compare identities with eligible voters (if applicable)
    - Recompute tracking numbers
    - Aggregate the ballots and check equality with official encrypted tally before decryption
  - Verify decryption proofs

Formal Description: Setup

- Executed by the Election Administrator
- Creates cryptographic groups, defines message space etc.
- Reusable for many elections

$$Setup(1^\lambda) = \begin{cases} \mathbb{G}, q, g \\ H_q: \{0,1\} \rightarrow \mathbb{Z}_q \\ (\boldsymbol{DLPRV}(x, g, Y)\,, \boldsymbol{DLVF}(g, Y, \pi)\,) \\ (\boldsymbol{EQPRV}(x, g_1, Y_1, g_2, Y_2), \boldsymbol{EQVF}(g_1, Y_1, g_2, Y_2, \pi)) \\ (\boldsymbol{DJPRV}(x_1, x_2, g, Y_1, Y_2), \boldsymbol{DJVF}(g, Y_1, Y_2, \pi)) \\ BB \leftarrow \emptyset \end{cases}$$

Formal Description: SetupElection

- The members of the TA cooperate to create their **joint** public key
  - Compute member key pair: $sk_i \xleftarrow{\$} \mathbb{Z}_q, pk_i \leftarrow g^{\mathrm{sk}_i}$
  - Publish $pk_i, DLPRV(sk_i, g, pk_i)$
  - Compute election public key: $pk \leftarrow \prod_i pk_i$
- Create list of eligible voters V_l
- Create list of candidates $CS = \{0,1\}$ (for simplicity)
- Publish everything into BB
  - $BB \Leftarrow \{pk_i, pk, V_l, CS\}$

Formal Description: Voting

Vote(i,v):

$v \in \{g^0, g^1\}$

$Enc_{\mathrm{pk}}(g^v) \rightarrow (g^r, g^v \cdot \mathrm{pk}^r) = (R, S)$

$EQPRV(r, g, R, pk, S)\ OR\ EQPRV(r, g, R, pk, Sg^{-1}) \rightarrow \pi_V$

$\mathrm{b} = (R, S, \pi_V)$

Valid(i,b):

Return 1 if $i \in V_l$ and $EQVF(\pi_V) = 1$

Append(I,b):

$BB \leftarrow (i, b)$ if $Valid(b) = 1$

VerifyVote(i,b,BB):

Return 1 if $b \in BB$ **and** $Valid(i, b) = 1$

Publish(BB):

Return $\mathrm{PBB} = \{b\}$ i.e. remove id's from ballots and keep one ballot per voter id

Occurs after all voters have voted

Formal Description: Tally

Tally(PBB, sk_i):

Validate all proofs in PBB

Compute $(R_\Sigma, S_\Sigma) \leftarrow \prod b$ for all $b \in PBB$

Distributed Decryption of $(R_\Sigma, S_\Sigma) \rightarrow g^t$

Each TA_i

posts $\left(D_i = R_\Sigma^{sk_i}, EQPRV(sk_i, g, pk_i, R_\Sigma, D_i)\right)$

computes $\frac{S_\Sigma}{\prod_i D_i} \rightarrow g^t$

solves small DLOG to get t

posts $\pi_T = EQPRV(sk_i, g, pk_i, R_\Sigma, S_\Sigma \cdot g^{-t})$

Formal Description: Verify

Verify(BB,PBB, t, π_T):

Check correct construction of PBB

- Only last ballot kept
- All kept ballots belong to eligible voters
- All kept ballots had valid proofs

Recompute $(R_\Sigma, S_\Sigma) \leftarrow \prod b$ for all $b \in PBB$

Verify π_T

Attacks by using wFS: Denial of Service

- In the proof $EQPRV(sk_i, g, pk_i, R_\Sigma, D_i)$ a malicious TA_i can cheat by **first creating the proof** and **then adaptively selecting D_i**
  - Compute $T_1 \leftarrow g^a, T_2 \leftarrow g^b$ where $a, b \xleftarrow{\$} \mathbb{Z}_q$
  - wFS: $c \leftarrow H(T_1, T_2)$
  - Compute $s \leftarrow a + c \cdot sk_i$
  - Select $D_i \leftarrow (R_\Sigma^{-s} T_2)^{-c^{-1}}$
- The proof (c, s) verifies
  - $g^s pk_i^{-c} = T_1$ and $R_\Sigma^s D_i^{-c} = R_\Sigma^s R^{-s} T_2 = T_2$ but $\log_{R_\Sigma} D_i = -s - c^{-1} \log_{R_\Sigma} T_2 \neq sk_i$
- What does this mean?
  - Tally decryption will yield a random group element instead of g^t
  - Efficient computation of t (assumed to be small DLOG) will not be feasible!

Attacks by using wFS: Undetectably alter result

- Goal: Announce election result $t \neq t'$
- Assumptions
  1. All TA_i's are corrupted – corrupted TA
  2. The TA can eavesdrop on the voter-selected encryption randomness
     - Realistic assumption if the voting device is corrupt
  3. Corrupt a single voter to cast the last vote
- The TA creates a 'proof' of correct 'tallying' before tallying
  1. Compute $T_1 \leftarrow g^a, T_2 \leftarrow g^b$ where $a, b \overset{\$}{\leftarrow} \mathbb{Z}_q$
  2. wFS: $c \leftarrow H(T_1, T_2)$
  3. Compute $s \leftarrow a + c \cdot sk$
- All voters vote except for the corrupt voter
  1. The current result is t and encrypted as $(R, S) = (g^{\sum r}, g^t pk^{\sum r})$
  2. By assumption 2: $\sum r$ is know to the TA
  3. The TA can compute t before the corrupt voter

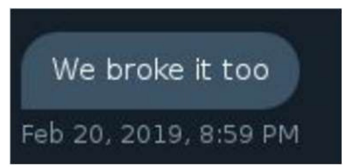
Attacks by using wFS: Undetectably alter result

- The TA selects $r' \leftarrow \frac{b+c(t-t')}{s-c \cdot sk}$
- Using the corrupt voter the TA casts the ballot $(g^{r'-\Sigma r}, g^0 pk^{r'-\Sigma r})$ which is a valid ballot
- The current encrypted tally is $(R', S') = (g^{r'}, g^t \cdot pk^{r'})$
- The encrypted tally does not change **but the proof $(\boldsymbol{c}, \boldsymbol{s})$ also verifies for $\boldsymbol{t'}$**
- $g^s pk^{-c} = T_1$ **(nothing has changed here)**
- $R'^s (S' \cdot g^{-t'})^{-c} = g^{sr' - ct - c \cdot sk \cdot r' + ct'} = g^{r'(s - c \cdot sk) - c(t - t')} = g^b = T_2$
- As a result, the corrupt TA can announce t' for the election result and everyone will be convinced by the proof.

NSW Electoral Commission iVote and Swiss Post e-voting

Similar attacks to other voting schemes

- 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”



Helios Extensions

Everlasting Privacy

Receipt Freeness

Eligibility Verifiability

Everlasting privacy

- 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
- But such a regime might also use insider information
- This threat might constitute an indirect coercion attempt
- The need for verifiability makes election data publicly available

- Helios does not have everlasting privacy!
- The functionality **Publish(BB)** releases the encrypted ballots
- An unbounded adversary can decrypt them!

Approaches to everlasting privacy

- **Perfectly Hiding Commitments**
  - Instead of encryption
  - But: Counting requires the openings.
  - How do voters send them?
    - Through Private Channels
      - Encrypted
      - Directly sent to the authorities
    - Not available to a future attacker
      - Unless they control part of the authorities
    - **Practical Everlasting Privacy**

- **Anonymous casting**
  - Disassociate identity from ballot
  - Use anonymous credentials to signal ballot eligibility or validity
    - Blind signatures
    - Ring signatures
  - An important advantage:
    - No trust required for privacy!

- Haines, T., Mueller, J., Mosaheb, R., & Pryvalov, I. (2023). SoK: Secure E-Voting with Everlasting Privacy. In Proceedings on Privacy Enhancing Technologies (PoPETs).
- Grontas, P., Pagourtzis, A. Anonymity and everlasting privacy in electronic voting. Int. J. Inf. Secur. 22, 819–832 (2023). https://doi.org/10.1007/s10207-023-00666-2

Adding everlasting privacy to Helios

Voters:

- Instead of encryption, use committments
  - $v \in \{0,1\}$,
  - $c = Commit(v, s) \rightarrow (g^{v} \cdot h^{s})$
  - $c_1 = Enc_{pk}(v) \rightarrow (g^{r_1}, g^{v} pk^{r_1})$
  - $c_2 = Enc_{pk}(s) \rightarrow (g^{r_2}, g^{s} pk^{r_2})$
- Proof of validity of v
- Proof that v, s are the same in c, c_1, c_2
- Post c in BB
- Send c_1, c_2 to TA through **private channels**

Talliers:

- Compute
  - $\prod_{v \in V} c$. Yields $\mathbf{c} = Commit(\sum v, \sum s)$
  - $\prod_{v \in V} c_1$. Yields $\mathbf{c_1} = Enc_{pk}(\sum v)$
  - $\prod_{v \in V} c_2$. Yields $\mathbf{c_2} = Enc_{pk}(\sum s)$
- Posts decryptions of $\boldsymbol{c_1}, \boldsymbol{c_2}$
- Everyone can validate the commitment $\mathbf{c}$

Do you see a problem?

Denise Demirel, J Van De Graaf, and R Araújo. "Improving Helios with Everlasting Privacy Towards the Public". In: *EVT/WOTE'12 Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections* (2012).

Adding everlasting privacy to Helios

- $Enc_{pk}(\sum s) = (g^{\sum r_2}, g^{\sum s} pk^{\sum r_2})$
- Need to solve DLP to get $\sum s$.
- This is not feasible!
  - Randomness is not in the same range as the result
- **Solution:**
  - **Use Paillier cryptosystem**
  - Encryption in the exponent
  - DLP for free!

Denise Demirel, J Van De Graaf, and R Araújo. “Improving Helios with Everlasting Privacy Towards the Public”. In: *EVT/WOTE’12 Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections* (2012).

Receipt-Freeness

Josh Benaloh and Dwight Tuinstra. "Receipt-free secret-ballot elections (extended abstract)". In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing - STOC '94*. ACM Press, 1994, pages 544– 553.

- Extensions for privacy against malicious voters
  - Voters that wish to sell their vote
- The attack scenario:
  - A voter agrees to sell their vote before the election
  - Proceeds to vote on their own
  - **The buyer does not monitor the voter when casting the ballot**
  - The voter presents evidence *after* voting to receive payment

A voting system is receipt free if a malicious voter cannot prove how they voted even if the want to

Helios is not receipt-free

- The malicious voter will offer as evidence:
  - the encryption randomness r
  - the position of the claimed ballot b in the BB
- The buyer will:
  - Encrypt the claimed choice with r
  - Compare with b
- Revoting does not help against coercion resistance
  - The published BB contains the final version of b

Adding receipt - freeness

- **Main idea:** The voter is not the sole contributor of encryption randomness for the ballot
  - They do not know the final randomness used - the voter – generated randomness as receipt is spoiled!
- A rerandomization authority reencrypts the ballot
  - Trusted for receipt-freeness
  - Not trusted for integrity/verifiability and privacy
- Sends a proof of correct reencryption to the voter
  - Use of designated verifier proofs
  - The voter (DV) cannot use it to convince the voter buyer

Martin Hirt and Kazue Sako. "Efficient receipt-free voting based on homomorphic encryption". In. EUROCRYPT'00

Adding receipt – freeness

- Each voter has a private-public key pair (sk_V, pk_V).
- They encrypt their ballot deterministically (i.e. $r = 0$) and send it to the EA
- The EA is split into $EA_1, \cdots, EA_n$ which operate a verifiable mixnet
  - Each vote is shuffled and reencrypted
  - Public proof of correct shuffling
- Each authority privately proves to each voter how the list was shuffled and reencrypted
  - The proof uses pk_V so it is designated-verifier
  - The voter can pinpoint their ballot in their final list to verify it, but they cannot prove to a vote seller its position
    - Non-transferability

Martin Hirt and Kazue Sako. "Efficient receipt-free voting based on homomorphic encryption". In. EUROCRYPT'00

Eligibility verifiability

- Anyone can verify that:
  - Every ballot was cast by a voter with the right to vote
  - No voter cast more than two counted ballots
  - Prevent **ballot stuffing**
- A simple solution:
  - Equip voters with credentials (PKI)
  - Sign encrypted ballots
  - Keep only one ballot / public key
  - Verify against eligible voter list

Belenios: Helios with credentials

- Extension to provide eligibility verifiability
- Adds a registration (credential) authority
- The BB generates login information for the voters (username, password)
- The voters receive both credentials $\langle (pk_i, sk_i), (uid, pwd) \rangle$using a private channel
- The voter logins to the BB using (uid, pwd)
- The ballot consists of
  - Vote encryption c
  - NIZK proof π of vote validity
  - A signature on c

Belenios: A Simple Private and Verifiable Electronic Voting System. Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. In Foundations of Security, Protocols, and Equational Reasoning, pp. 214-238, 2019.

Belenios: Helios with credentials

- The BB keeps one ballot per (id, pk)
  - Last one if multiple exist
- The BB checks signatures and proofs
- The voters check that their ballots appear on the BB (individual verifiability)
- Ballot stuffing can occur only if both the BB and the RA are corrupt
  - Stuffed ballots need to have both a vk and an id
- Eligibility verifiability:
  - Everyone can check that a ballot comes from a valid voter
  - But: This reveals who abstained - illegal in some countries

Belenios: A Simple Private and Verifiable Electronic Voting System. Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. In Foundations of Security, Protocols, and Equational Reasoning, pp. 214-238, 2019.

Private eligibility verifiability (KTV-Helios)

- Participation privacy + Universal verifiability
- **Main idea: Add null votes + vote update capabilities**
- Voting proxies:
  - Entities that add null votes for a voter
- Properties of null votes:
  - They do not add to the result
  - They are indistinguishable from regular votes
  - Proofs that each vote is either a null vote or a normal vote
  - Anonymous casting
- Also provide (some degree) of receipt freeness
  - The voter may prove that he cast c, but..
  - If there exists another ballot c' cast for them, they cannot prove that
    - $c' \neq c'' \cdot c^{-1}$ (which updates their true ballot to c'')

Kulyk, O., Teague, V., Volkamer, M. Extending Helios Towards Private Eligibility Verifiability. Vote-ID 2015.

BeleniosRF: Belenios with receipt-freeness

- Use a rerandomizing server
  - Rerandomizes all the ballots before publishing them to the BB
  - This breaks the validity of signatures!
- Solution: Signatures on Randomizable Ciphertexts
  - Given a ciphertext, signature pair (c, σ)
    - Rerandomize the ciphertext to c'
      - Without the decryption key
    - Adapt the signature so that it publicly verifies for c'
      - Without the signing key

Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In *23rd ACM Conference on Computer and Communications Security (CCS'16)*, pages 1614–1625, Vienna, Austria, 2016.

BeleniosRF: Belenios with receipt-freeness

- No need for proofs of correct rerandomization for RF
  - The EA rerandomizes the ciphertexts and adapts the signatures of validity
- Security:
  - Rerandomization appears as fresh encryption
  - One-more unforgeability: The signer can create signatures on *messages* they have never seen
- Is it enough?
  - The voter might sell their keys and passwords!

Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In *Public Key Cryptography - PKC 2011*