Voting with Mixnets

Panagiotis Grontas

$\mu\Pi\lambda\forall$ - CoReLab Crypto Group

1/2/2013 - 15/2/2013

Outline

- Election System Abstraction And Properties
- Crypto Preliminaries
- Mixnets: Definition And Types
- Zero Knowledge Verifiable Mixnets
- Other Types of Verifiable Mixnets

High Level properties [?]

- **Integrity**
  - Votes are cast as intended
  - Votes are counted as cast
- **Ballot Secrecy** Nobody can figure out how you voted (privacy), even if you try to prove it (selling/coercion).
- Authentication and Authorisation
  - Only authorised voters can vote
  - a specified number of times as stated in election law
- Enfranchisement All voters must have the opportunity to vote
- Availability (DoS,Tallying)
- Efficiency (Cost,Time)

Remark: Authentication vs Secrecy vs Enfranchisement

Low Level Properties [?]

- **Individual Verifiability** Each voter can check that their ballot was included in the outcome
- **Universal Verifiability** All voters can check that a voter's ballot was included in the outcome
- **Receipt - Freeness** A voter cannot prove how she voted even if she wants to.
- **Robust** Nobody can disrupt an election
- **Fairness** No partial results are known.
- **Eliminate Vote Duplication** A voter can figure another vote by duplicating it and checking the bulletin board for a correlated vote.
- **Vote Cancelling** Vote in such way as to nullify a vote (without knowing it)

Remark: Individual Verifiability vs Receipt - Freeness

ElGamal Encryption

- Randomised Public Key Encryption From Diffie Hellman
- KeyGen From Cyclic Group $< g >$ of prime order q
  - $x \in_R \mathbb{Z}_q$
  - $y = g^x mod p$
  - return $(pk =<< p, q, g >, y >, sk = x)$
- Encrypt Message m with public key y
  - encode $m \in< g >$
  - $r \in_R \mathbb{Z}_q$
  - $G = g^r \, mod p$
  - $M = m \cdot y^r \, mod p$
  - return (G, M)
- Decrypt message (G, M) with secret key x
  - return $M / G^x \, mod p$

ElGamal Re Encryption - Re Randomisation [?]

$$\begin{aligned} ReEnc(c, r') &= c \cdot Enc(1, r') \\ &= (g^r, m \cdot (g^x)^r) \cdot (g^{r'}, (g^x)^{r'}) \\ &= (g^{r+r'}, m \cdot (g^x)^{r+r'}) \end{aligned}$$

- No knowledge of secret key is required.
- Re encryption does not affect decryption
- Without the secret key or the re randomisation factor it is infeasible to show that two messages are re reencryptions of each other (subject to the DDH)

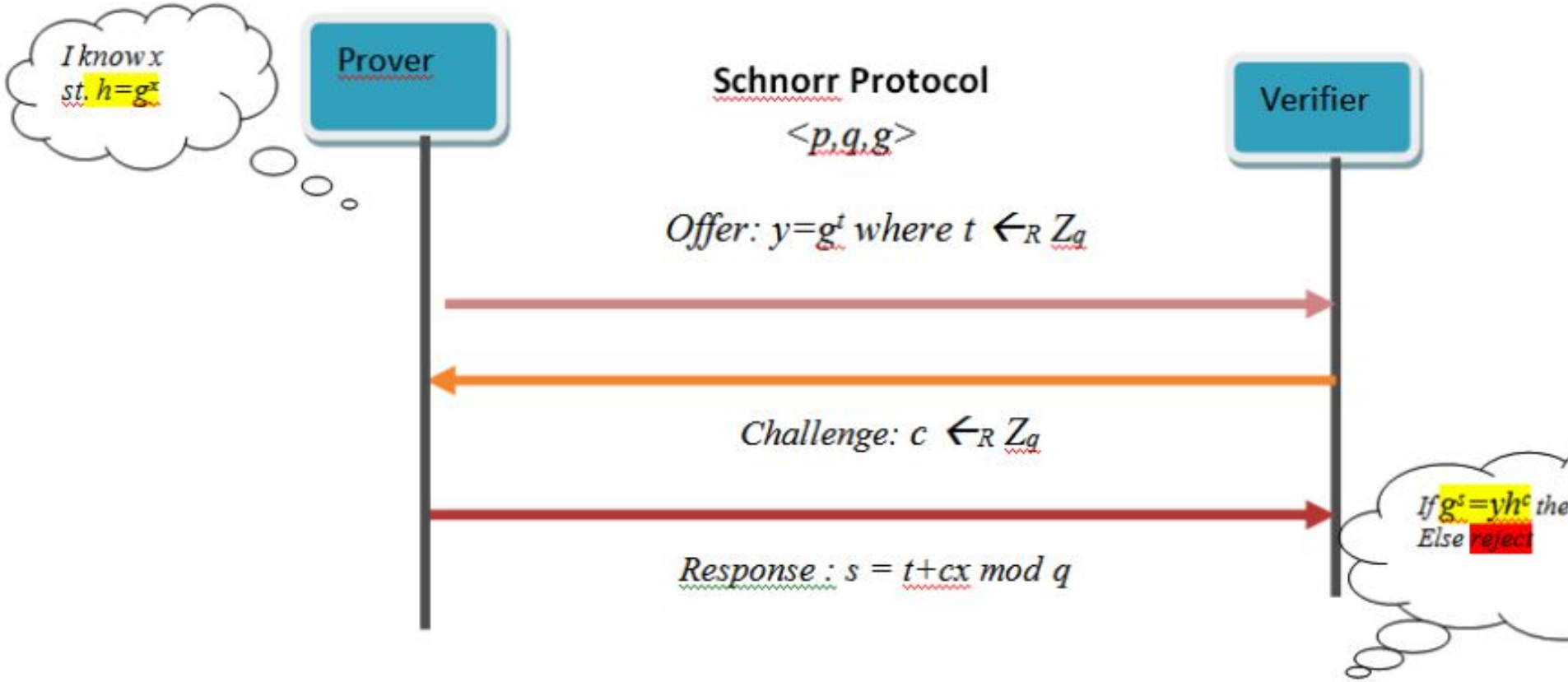
ElGamal Multiplicative Homomorphism

- Let m_1, m_2 plaintexts. Then: $Enc(m_1) \cdot Enc(m_2) = Enc(m_1 \cdot m_2)$
-

$$\begin{aligned} Enc(m_1) \cdot Enc(m_2) = \\ (g^{r_1}, m_1 \cdot y^{r_1}) \cdot (g^{r_2}, m_2 \cdot y^{r_2}) = \\ (g^{r_1+r_2}, m_1 \cdot m_2 y^{r_1+r_2}) = \\ Enc(m_1 \cdot m_2) \end{aligned}$$

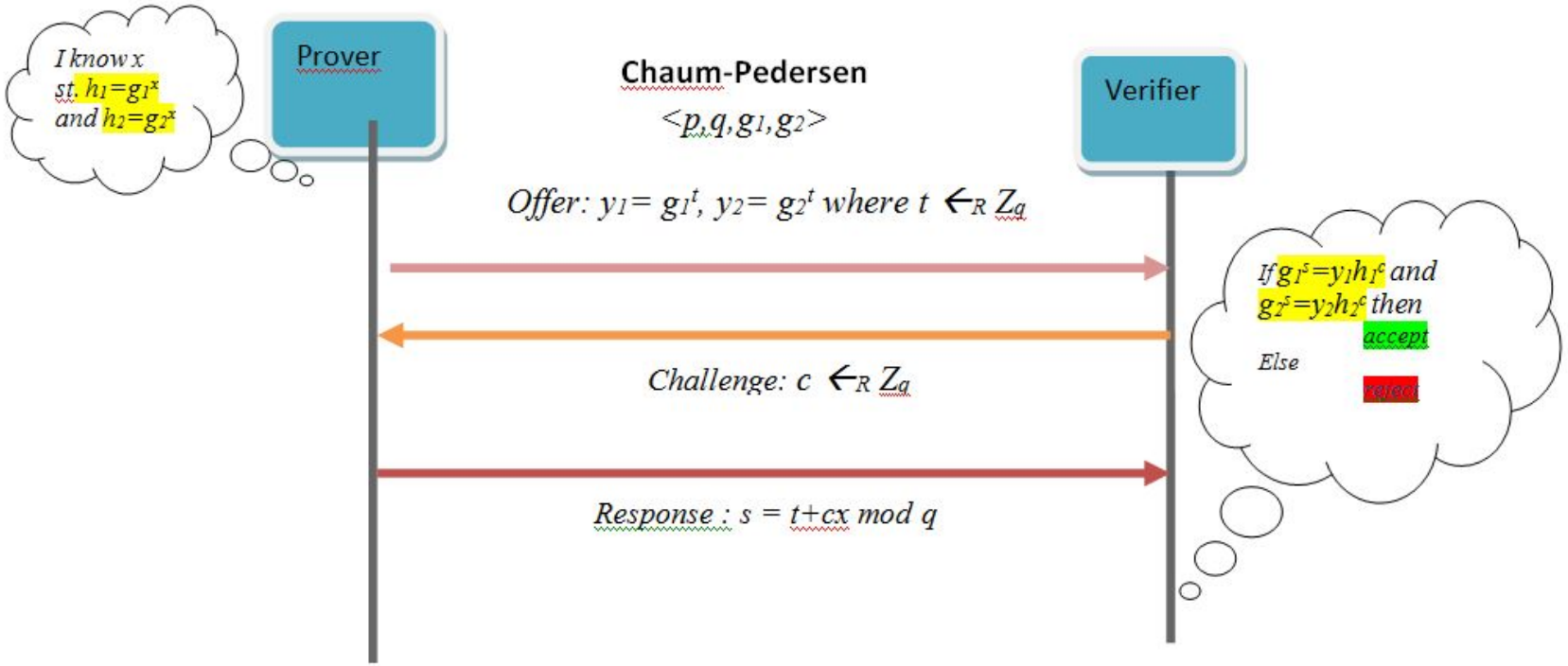
- In elections we would desire additive homomorphism.

Prove that you know DLOG (Schnorr)



- Non interactive version (Fiat-Shamir)
- Replace challenge with hash $H(g^t)$

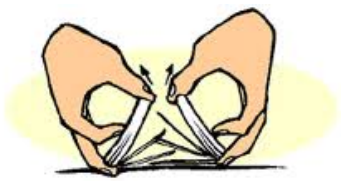
Prove DLOG equality (Chaum-Pedersen)



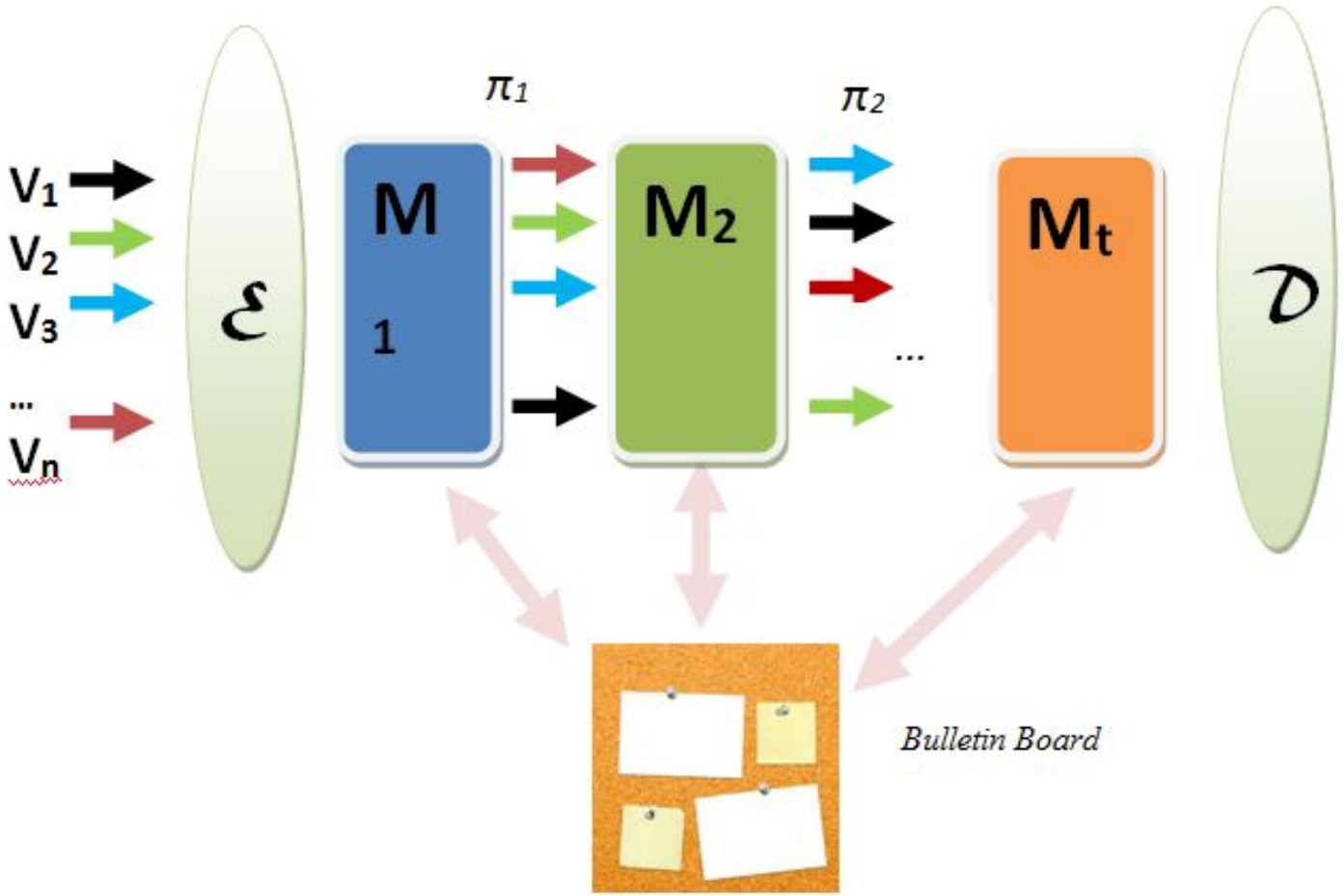
- Honest-Verifier Zero Knowledge

Mixnet Overview

- Introduced in [**?**]
- Primitive for anonymous channel
- Many uses: elections (of course), anonymous browsing etc.
- Operation: Different mistrusting parties
- Anonymity based on encryption+shuffling
- Output a random permutation of the input
- No single output item, must match an input item
- Enter encryption



Mixnets



Bulletin Board

- Essential Component of All Election Systems
- Integrity Preserving Tallying: Post all votes, Public tallying
- Authenticated
- Accessible to the public
- Tamper Proof
- Resistant to DoS attacks
- Each mix server reads input and appends output
- Implemented via Byzantine agreement algorithms

Voting With Mixnets:Main Idea

- Create Ballots
- Initial Encryption $C_{10}, C_{20}, ..., C_{n0} = Enc(B_1, ..., B_n)$
- Each mix server processes and permutes the encrypted items
- After all mixing has occured (optionally) decrypt items
- Post the **unencrypted ballots** to the bulletin board
- Perform the tally

Decryption Mixnets

- The original [**?**] idea
- Encrypt the ballot with the public key of the mix servers in reverse order
- $C_{i0} = E_{pk1} \circ E_{pk2} \circ ... \circ E_{pkt}(B_i)$
- Each mix server decrypts with its secret key (akin to onion peeling) and performs the shuffle
- After the final stage, all ballots are decrypted
- The final decryption key might shared.
- **Remarks:**
  - A mix server can block the mixing process
  - The ciphertext size is proportional to the number of mix servers

ReEncryption Mixnets [?]

- Two variations both of them proposed in [?]
- Version 1
  - Each mix server re randomises the ballots by reencryption
  - A final decryption stage is needed
  - The decryption key must be shared to various parties
  - The decryption is jointly done by all mix servers.
- Version 2
  - Each mix server partially decrypts the ballots by applying its secret key.
  - Then re randomises by reencryption
  - The last mix server decrypts
- Elgamal, Pallier Cryptosystems

Is everything OK?

Yes provided that:

- We trust the original encryption
- We trust at least one mix server
- We trust the decryption (Honest Majority Needed)
- We only want individual verifiability, but not universal.

Universal Verifiability - Verification

- for each decryption/reencryption
- for the correct permutation

without compromising anonymity

What can go wrong

- At the initial encryption stage, a different vote might be encrypted (vote changing, vote copying, vote cancelling)
- **Solution:Zero - knowledge proof of the contents of the vote**
- A mix server can change some of the input votes, by replacing them in the output.
- **Solution:Zero - knowledge proof of correct shuffling for the mix-servers**
- A subset of the mix servers might cooperate to break anonymity
- **Solution:Privacy is guaranteed if at least one of the mix servers is honest**

General Idea

- Correctness of initial encryption $\Leftrightarrow$ Prove that you know DLOG
- Correctness of Shuffling[**?**] $\Leftrightarrow$
  Prove that each encrypted vote in the input, appears in the output $\Leftrightarrow$
  Prove that each encrypted vote in the input, has a reencryption in the output $\Leftrightarrow$
  Prove the following NP statement

$$\exists \text{ permutation } \pi \text{ on } \{1, \cdots, n\}, \exists s_1, ..., s_n,$$
$$\forall \text{ vote } i \in \{1, \cdots, n\},$$
$$\forall \text{ mix server } j \in \{1, \cdots, k\} :$$
$$\text{If } C_{i,j} = (a_{i,j}, b_{i,j}) \text{ and } C_{\pi(i),j+1} = (a_{\pi(i),j}, b_{\pi(i),j}) \text{ then}$$
$$a_{\pi(i),j+1} = a_{i,j} \cdot g^{s_i} \text{ and } b_{\pi(i),j+1} = b_{i,j} \cdot y^{s_i}$$

- NP-statement $\Rightarrow$ Zero-Knowledge Proof
- How fast?

An example:Millimix [?]

- A mixnet with the following properties:
  - Privacy: Correlation of input/output no better than a random guess
  - Robustness: The output of each mich server is a correct reencryption and permutation
  - Public Verifiability: Anybody (participant or not) can verify the correct operation
  - Computational Efficiency on small batches

Millimix Architecture I

Sorting Networks

- A collection of wires and comparators.
- A 2×2 comparator is a function f such that:

$$f(x, y) = \begin{cases} (x, y), & \text{if } x < y \\ (y, x), & \text{if } x \geq y \end{cases}$$

- n input wires, n output wires that are connected by comparators
- Result: Inputs are outputted in sorted order.
- **Remark:** Can implement all $n!$ permutations.
- At least $\Omega(nlogn)$ comparators.

Millimix Architecture II

- Each mix server S_i simulates a sorting network.
- Generates a random permutation π_i
- Output E_u, E_v based on the comparison $\pi_1(u), \pi_1(v)$
- The mix network as a whole simulates the permutation: $\pi = \pi_n \circ \cdots \circ \pi_1$

Millimix:Proof Of Reencryption

Statement

The ciphertext $c_2 = (a_2, b_2) = (g_u, m_2 \cdot y_u)$ is a reencryption of the ciphertext $c_1 = (a_1, b_1) = (g_t, m_1 \cdot y_t)$

Proof

c_2 is a reencryption of c_1 iff they both encrypt the same message, meaning that $m_1 = m_2$.

$$m_1 = m_2 \Leftrightarrow \frac{m_1}{m_2} = 1 \Leftrightarrow \frac{m_1 \cdot y^t}{m_2 \cdot y^u} = \frac{y^t}{y^u} \Leftrightarrow$$

$$\frac{b_1}{b_2} = \frac{a_1^x}{a_2^x} \Leftrightarrow log_{a_1} b_1 = x \text{ and } log_{a_2} b_2 = x \Leftrightarrow$$

$$log_{a_1} b_1 = log_{a_2} b_2 \Leftrightarrow \text{ Chaum - Pedersen Protocol}$$

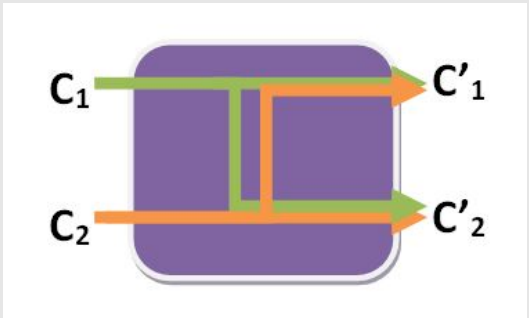
Millimix:Proof Of Permutation I

Statement

Prove that $\{c'_1, \cdots, c'_n\}$ is a reencryption of $\{c_1, \cdots, c_n\}$ without revealing the correspondence.

Proof

Denote $c_1 \approx c_2$ iff c_1 is a reencryption of c_2. We shall first prove for a 2×2 mix server that $(c'_1 \approx c_1 \bigwedge c'_2 \approx c_2) \bigvee (c'_1 \approx c_2 \bigwedge c'_2 \approx c_1)$



$\Leftrightarrow (c'_1 \approx c_1 \vee c'_1 \approx c_2) \bigwedge (c'_1 \approx c_1 \vee c'_2 \approx c_1) \bigwedge (c'_2 \approx c_2 \vee c'_1 \approx c_2) \bigwedge (c'_2 \approx c_2 \vee c'_2 \approx c_1)$

Millimix:Proof Of Permutation II

Prove $(c'_1 \approx c_1 \vee c'_1 \approx c_2)$

Use the Chaum-Pedersen Protocol and accept if $c'_1 \approx c_1$ or $c'_2 \approx c_2$.

- Generalize for $n \times n$ mixnets using sorting networks.
- Instead of comparators use 2×2 mixnets.

Killian-Sako Verifiable Mixnet [?] I

- The first universally verifiable mixnet
- Elgamal Reencryption based Mixnet [**?**]
- Message m (represents the voter's ballot)
- Mix Server i:
  - Secret Key: x_i
  - Public Key: $y_i = g^{x_i}$
- Voter Encryption: $(G_1, M_1) = (g^{r_0}\ mod p, (y_1 ... y_n)^{r_0} \cdot m\, mod p)$
- Each Mix Server
  - $G_{i+1} = G_i \cdot g^{r_i} = g^{r_0 + r_1 + ... + r_i}$
  - $M_{i+1} = M_i \cdot (y_{i+1} ... y_n)^r_i / G_i^{x_i}$ ie. adds one random exponent and drops one private key
- Decryption(Final) Mix Server: $M_n / G_n^{x_n} = m$
- Permute and announce

Killian-Sako Verifiable Mixnet [?] II

- Verification
  - Prove correct partial decryption
  - Prove correct re encryption and shuffling
- Solution: Cut and Choose Protocol
- 50% soundness. A mix server might cheat and getaway with reencryption and shuffling.
- Alternative: Use the Chaum-Pedersen Protocol for 100% soundness.

Killian-Sako Verifiable Mixnet [?] III

Cut - And - Choose Proof Of Correct Partial Decryption

Prove knowledge of secret key and decryption.

1. **Input**: $(G, g, y = g^x mod p, H)$
2. **Output**: Prove that $H = G^x mod p$
3. Prover: Send $(y', G') = (g^r mod p, G^r mod p)$ where $r \in_r \mathbb{Z}_{p-1}$
4. Verifier: With probability $\frac{1}{2}$ request r or $r' = r - x$
5. Prover: Reveal r or r'
6. Verifier:
   - If r is revealed then check that $y' = g^r mod p$ and $G' = G^r mod p$
   - If r' is revealed then check that $y' = g^{r'} y mod p$ and $G' = H \cdot G^{r'} mod p$

Killian-Sako Verifiable Mixnet [?] IV

Batch Verification

- We proved that for each message $H = G^x mod p$
- Instead of proving correct encryption for each vote, a mix server proves correct encryption of all the votes.
- The verifier chooses random exponents c_i (one for each vote).
- Calculate $\prod_i H^{c_i}$ and $\prod_i (G^x)^{c_i}$
- Prove that $\prod_i H^{c_i} = \prod_i (G^x)^{c_i}$
- The proof implies that all encryptions are done correctly

Killian-Sako Verifiable Mixnet [?] V

Cut - And - Choose Proof Of Correct Shuffle - Main idea [?]

- Generate another permutation and randomisation values
- Perform reencryption and shuffling according to them (secondary shuffle)
- Reveal secondary shuffle or the difference between primary and secondary shuffle
- Validate.

Killian-Sako Verifiable Mixnet [?] VI

Cut - And - Choose Proof Of Correct Shuffle - Specifics

- **Input**: (g, w, A, B) where
  - g, w are constants
  - $A = (a_i^{(1)}, a_i^{(2)})$
  - $B = (a_{\pi(i)}^{(1)} \cdot g^{r'_{\pi(i)}}, a_{\pi(i)}^{(2)} \cdot w^{r'_{\pi(i)}})$
  - $a_i^{(1)}, a_i^{(2)}$ refer to G and M/H
- **Output**: Prove that B can be generated from A

Killian-Sako Verifiable Mixnet [?] VII

Cut - And - Choose Proof Of Correct Shuffle

Generate a permutation λ and randomization values t_i.

1. Prover: Send $C = (a^1_{\lambda(i)} \cdot g^{t_{\lambda(i)}} mod p, a^2_{\lambda(i)} \cdot w^{t_{\lambda(i)}} mod p)$
2. Verifier: With probability $\frac{1}{2}$ request (λ, t_i) or $(\lambda' = \lambda \circ \pi^{-1}, t'_i = t_i - r'_i)$
3. Prover: Reveal requested items
4. Verifier:
   - if λ, t_i is requested then check C by definition
   - if λ', t'_i is requested then for $B = (b^{(1)}_i, b^{(2)}_i)$ check if $C = (b^{(1)}_{\lambda'(i)} \cdot g^{t'_{\lambda'(i)}}, b^{(2)}_{\lambda'(i)} \cdot w^{t'_{\lambda'(i)}} mod p)$ ie. $\lambda(i) = \lambda' \circ \pi(i)$ or $\lambda(i) = \lambda \circ \pi^{-1} \circ \pi(i)$

Mix with verification work independent of number of servers [?],[?]

- An extension of the cut and choose protocol by [**?**]
- The mix-servers present a chained proof which is checked by the verifier.
- Each mix server cuts by performing a secondary mix **based on the secondary mix of the previous server** and commits to it.
- Verifier chooses:
  - The secondary mix
  - The difference between the primary and secondary mix
- Simultaneous revelation
- The verifier checks a single shuffle

Furukawa and Sako Mixing [?] I

- Very fast - $18n$ exponentiations.
- Mixing is represented as matrix multiplication.
- Permutation matrix

$$A_{ij} = \begin{cases} 1, \pi(i) = j \\ 0, \text{otherwise} \end{cases}$$

- For example the permutation $\pi(1, 2, 3) = (2, 3, 1)$ can be represented using the matrix:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Furukawa and Sako Mixing [?] II

- Shuffling and re encryption can be represented using a permutation matrix.

$$E_i = (g_i, m_i) \rightarrow_{mixing} E'_i = (g'_i, m'_i) \text{ where}$$

$$g'_i = g^{r_i} \cdot g_{\pi^{-1}(i)} = g^{r_i} \cdot \prod_{j=1}^{n} g_j^{A_{ji}}$$

$$m'_i = y^{r_i} \cdot m_{\pi^{-1}(i)} = y^{r_i} \cdot \prod_{j=1}^{n} m_j^{A_{ji}}$$

Furukawa and Sako Mixing [?] III

- Prove that r_i and $[A_{ij}]$ exists.

Key observation

A matrix $[A_{ij}]$ is a permutation matrix iff the sum of dot-products of all different columns is zero and 1 for the same columns, ie $\forall i, j, k$

$$P_{ij} = \sum_{h=1}^{n} A_{hi} \cdot A_{hj} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

$$P_{ijk} = \sum_{h=1}^{n} A_{hi} \cdot A_{hj} \cdot A_{hjk} = \begin{cases} 1 & i = j = k \\ 0 & \text{otherwise} \end{cases}$$

Furukawa and Sako Mixing [?] IV

Zero - Knowledge Proof

1. Prove that there exist $r_i, [A_{ij}]$ st: $g'_i = g^{r_i} \cdot g_{\pi^{-1}(i)} = g^{r_i} \cdot \prod_{j=1}^{n} g_j^{A_{ji}}$ where $[A_{ij}]$ is a matrix that satisfies the two conditions of the key observation.
2. Prove that the $r_i, [A_{ij}]$ used in the two conditions are identical
3. Prove that for each pair (g'_i, m'_i) the same $r_i, [A_{ij}]$ have been used.

Furukawa and Sako Mixing [?] V

Prove that:

$$g'_i = g^{r_i} \cdot g_{\pi^{-1}(i)} = g^{r_i} \cdot \prod_{j=1}^{n} g_j^{A_{ji}} \text{ where}$$

$$P_{ij} = \sum_{h=1}^{n} A_{hi} \cdot A_{hj} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

Furukawa and Sako Mixing [?] VI

Proof.

- Verifier: Issue challenge c_j
- Prover: Respond with $s = \sum_{j=1}^{n} r_j c_j$ and $s_i = s = \sum_{j=1}^{n} A_{ij} c_j$
- Verifier: Check that:

$$\sum_{i=1}^{n} s_i^2 = \sum_{j=1}^{n} c_j^2$$

$$g^s \prod_{i=1}^{n} {g_i}^{si} = \prod_{j=1}^{n} {g'_j}^{cj}$$

- In order not to leak information add randomisation and modify accordingly.

□

Neff Verifiable Mixnet I

- Introduced in [**?**] and improved in [**?**]
- The fastest verifiable mixnet so far, with $8n$ exponentiations.
- Unconditionally Sound
- Computational Zero Knowledge
- Proof protocol for proving mixing without knowledge of exponents from (prover - shuffler).
- Main Idea:
  - Key Property: A polynomial is unaffected by permutation of the roots. $\prod_{i=1}^{n}(m_i - x) = \prod_{i=1}^{n}(m_{\pi(i)} - x)$
  - Mix inputs and outputs are represented as polynomials P_{in}, P_{out} with coefficients $\in Z_q$
  - Verifier chooses a random $t \in Z_q$
  - Evaluates both input and output polynomials
  - The results match with very high probability

Neff Verifiable Mixnet II

- Comprised of three subprotocols:
  - Iterated Logarithmic Multiplication Proof Protocol (ILMPP)
    - Chaum-Pedersen Generalisation
  - Simple n-shuffle Proof Protocol
    - Evaluation of a pair of polynomials at a common random point
  - General n-shuffle Proof Protocol
    - Apply when not all exponents are known

ILMPP

- Public n element sequences $\{X_i\}, \{Y_i\}$ are known
- The prover knows $\{x_i = log_g X_i\}, \{y_i = log_g Y_i\}$
- Convince the verifier that $\prod_{i=1}^{n} x_i = \prod_{i=1}^{n} y_i$
- For $n = 2$ it is the Chaum Pedersen Protocol.

Neff Verifiable Mixnet III

1. **Prover:** Pick $n-1$ random values θ_i
2. **Prover:** Calculate and send n values A_i as

$$A_1 = Y_1^{\theta_1} \cdots$$
$$A_i = X_i^{\theta_{i-1}} \cdot Y_i^{\theta_i} \cdots$$
$$A_n = X_n^{\theta_{n-1}}$$

3. **Verifier:** Generate random challenge γ
4. **Prover:** Calculate $n-1$ random values r_i st:

$$Y_1^{r_1} = A_1 \cdot X_1^{-\gamma} \cdots$$
$$X_i^{r_{i-1}} \cdot Y_i^{i} = A_i \cdots$$
$$X_n^{r_{n-1}} = A_n \cdot Y_n^{(-1)^{n-1} \cdot r_{n-1}}$$

5. **Verifier:** Accept if all above relations hold.
6. r_i are calculated by solving a system of $n \times n-1$ equations
7. Solution exists iff $\prod_{i=1}^{n} x_i = \prod_{i=1}^{n} y_i$

Neff Verifiable Mixnet IV

Simple n-shuffle Proof Protocol

- Public n element sequences $\{X_i\}, \{Y_i\}$ and commitments $\Gamma = g^{\gamma}$ are known
- Prover knows $\{x_i = log_g X_i\}, \{y_i = log_g Y_i\}$ and constant γ
- Convince the verifier that $\exists \pi$ permutation such that $Y_i = X_{\pi(i)}^{\gamma} \; \forall i \in \{1, ..., n\}$ which means that $y_i = \gamma \cdot x_{\pi(i)}$.

Neff Verifiable Mixnet V

Proof.

- Verifier challenges with random $w \in \mathbb{Z}_q$
- Both prover and verifier run the ILMPP with $2n$ sequences: $(X_1 \cdot g^w, \cdots, X_n \cdot g^w, \Gamma, \cdots, \Gamma)$ and $(Y_1 \cdot \Gamma^w, \cdots, Y_n \cdot \Gamma^w, g, \cdots, g)$
- Prover runs the ILMPP with with $2n$ sequences: $(x_1 + w, \cdots, x_n + w, \gamma, \cdots \gamma)$ and $(y_1 + w\gamma, \cdots, y_n + w\gamma, 1, \cdots 1)$
- As a result: $\gamma^k \prod_{i=0}^{n-1}(x_i + w) = \prod_{i=0}^{n-1}(y_i + w\gamma)$ which means: $\prod_{i=0}^{n-1}(x_i\gamma + w\gamma) = \prod_{i=0}^{n-1}(y_i + w\gamma)$
- Two polynomials are equal at a random point $\Rightarrow$ Equal with high probability
- As a result: $y_i = \gamma \cdot x_{\pi(i)}$

□

Neff Verifiable Mixnet VI

The shuffler can prove the shuffle even if no exponents are known.

General n-shuffle Proof Protocol - El Gamal Shuffle

- Public n element sequences $\{(X_i, Y_i)\}$ (input) and $\{(X'_i, Y'_i)\}$ (output) and commitments $g, h = g^t$ are known
- Prover knows $\beta_1, \cdots, \beta_n$ and π st:
- $(X'_i, Y'_i) = (g^{\beta_{\pi(i)}} X_{\pi(i)}, h^{\beta_{\pi(i)}} Y_{\pi(i)})$

Neff Verifiable Mixnet VII

Proof.

- Fix permutation π and β_i, ξ_i. Define: $(X'_i, Y'_i) = (g^{\beta_{\pi(i)}} X_{\pi(i)}, h^{\xi_{\pi(i)}} Y_{\pi(i)})$
- $\exists \pi$ st: $\beta_i = \xi_i \forall i$
- Show that for a random $\bar{s}$ vector: $\bar{s}\bar{\beta} = \bar{s}\bar{\xi}$
- For random $\bar{s} = \bar{r_\pi}$ show $\bar{s}\bar{x} - \bar{r}\bar{x} = \bar{s}\bar{y} - \bar{r}\bar{y}$
- Use simple n-shuffle Proof Protocol

□

Verifiable Mixnets: Performance

Cost = Total Number of Exponentiations For

- Reencryption
- Proof
- Decryption

Performance:

- [?] $2n + 7nlogn(2k - 1) + (2 + 4k)n$
- [?] $2n + 642nk + (2 + 4k)n$
- [?] $2n + 18n(2k - 1) + (2 + 4k)n$
- [?] $2n + 8n(2k - 1) + (2 + 4k)n$

In practice: Neff Mixnet: 10^6 votes -> 20 hours to mix and verify.
Lesson:Zero Knowledge Proofs Are Computationally Expensive.

Randomised Partial Checking [?] I

- **Idea**: Give up the expensive notion of **proof**
- Provide **strong evidence** that the mix server has operated correctly (ie. the output is a permutation of the input)
- Strong Evidence = Probabilistic Verification
- Partial Revelation of the input/output correspondence.
- For n items, choose randomly $\frac{n}{2}$ and reveal the input/output relation.
- The mix server has no control of which items are revealed.
- A cheater cannot get away with altering too many votes
- **Tradeoff:** Privacy

Randomised Partial Checking [?] II

Method Overview

- System Setup
- Ballot B_i preparation and encryption $C_{i,0} = E_{PK}(B_i)$.
- Remark:Both reencryption and decryption mixnets are supported.
- Ballot Validity Checking.
- Each mix server commits to a permutation.
- Mix net processing. $C_{ij} = X_j(C_{ij-1})$
- Correctness By Partial Checking.

Randomised Partial Checking [?] III

Permutation Commitment

- $\zeta_w[i]$ commitment to integer i using witness w
- How can server S_j commit to a private permutation
  - Commit to the mappings of input elements to output elements $\Gamma^{IN} = \{\zeta_{w_{ij}}[\pi_j(i)]\}_{i=1}^{n}$
  - Commit to the mappings of output elements to input elements $\Gamma^{OUT} = \{\zeta_{w_{ij}}[\pi_j^{-1}(i)]\}_{i=1}^{n}$
- γ_{ij}: The $i-th$ commitment of S_j
- For speed: $\zeta_w[i] = h(w||i)$ for hash function h.

Randomised Partial Checking [?] IV

Mapping Revelation

Reveal a fraction $p > 0$ of the input - output correspondence.

- Let $\pi_j(k) = i$ and $C_{ij} = X_j(C_{kj-1})$
- Reveal (k, i, R_{ijk}) where R_{ijk} is the necessary information to reconstruct the processing (padding,randomness etc.)
- Reveal the in-commitment γ_k or the out-commitment γ_i.

Randomised Partial Checking [?] V

What to reveal

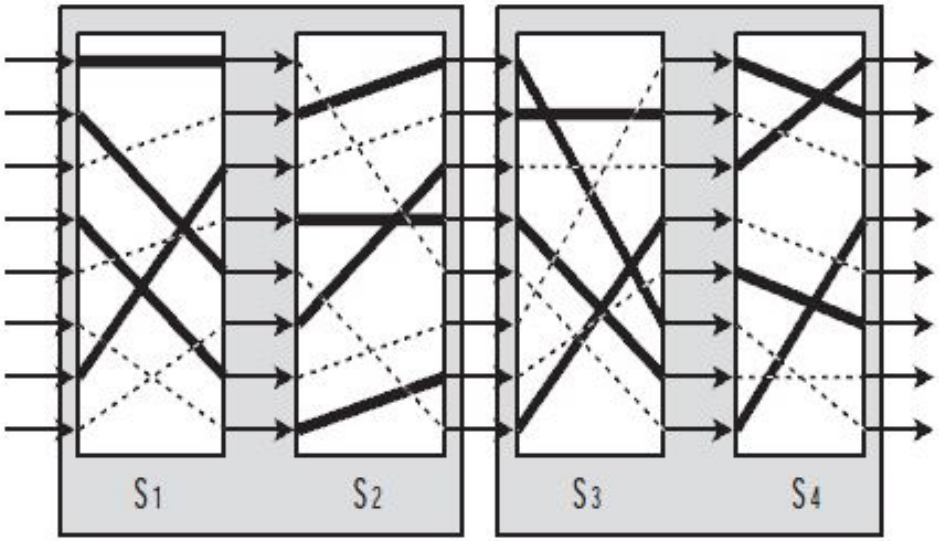
- Each S_j commits to random value r_j
- Let $R = \oplus_j r_j$
- Let $Q = h(R, BB)$ where BB are the contents of the bulletin board
- $Q_j = h(Q, j)$ determines which values must be revealed.
- $P_{IN}(Q_j, k) = true \Rightarrow$ reveal the pair with k as input
- $P_{OUT}(Q_j, i) = true \Rightarrow$ reveal the pair with i as output

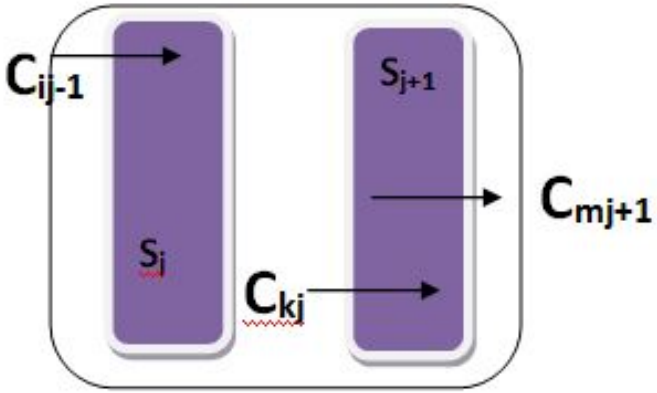
Randomised Partial Checking [?] VI

What about privacy

- Main idea:Pair the servers so that we never reveal the same pair twice.
- At least one honest pair is needed for privacy
- Let j odd and (S_j, S_{j+1}) a server pair . Then
  - $P_{IN}(Q_j, k) = false$
  - $P_{OUT}(Q_j, i) = true$ with probability $\frac{1}{2}$
  - $P_{IN}(Q_{j+1}, i) = \bar{P_{OUT}}(Q_j, i)$
  - $P_{IN}(Q_{j+1}, m) = false$

Randomised Partial Checking [?] VII





Randomised Partial Checking [?] VIII

Boundary Check

- Let k minimum number of votes to alter election result.
- What is the probability Pr_{und} that someone switches k votes without detection; If $p = \frac{1}{2}n$ permutations are revealed then $Pr_{und} = \frac{1}{2^k}$
- Proof idea:
  - Since mix servers are paired, we check only predecessors or successors of intermediate items.
  - If a mix server is not honest, we will find it out by checking predessor or successor.
  - Testing is independent.

Almost Entirely Correct Mixing [?] I

- **Idea**: We might not need a 100% correct proof of mixing, especially if the margin of victory is wide.
- An almost correct proof of mixing might do.
- 10^6 votes -> instant proof and 99% correct proof of mixing.

Method

- Select a random subset S of mix server inputs
- Calculate the product π_s
- Reveal S to the mix server.
- Ask to product a set of outputs S' st. $\pi_s = \pi_{s'}$
- Honest mix server: Simply apply the permutation
- Cheating mix server: Might be impossible to find such S'.

Almost Entirely Correct Mixing [?] II

Operation

- Input Ciphertexts $C_i = (g^{r_i}, m_i \cdot y^{r_i})$
- Output Ciphertexts $C_{\pi(i)} = (g^{r'_i}, m'_i \cdot y^{r'_i})$
- Each server M_j generates and commits to random r_j. $r = \oplus_j r_j$
- Agree on a security parameter α.
- Verify correct format of output with a single exponentiation.
- Each Mix-Server M_j proves that $\prod_{i=1}^{n} m_i = \prod_{i=1}^{n} m'_i$ using El Gamal homomorphism and Chaum-Pedersen.
- Generate α sets $S_1, \cdots, S_\alpha, S_i \subset S$ by including each input item with probability $\frac{1}{2}$ where randomness stems from r.

Almost Entirely Correct Mixing [?] III

Operation

- The sets $S_1, \cdots, S_\alpha$ are given to M_j.
- M_j must produce $S'_1, \cdots, S'_\alpha$ st. $\forall i \in \{1, \cdots \alpha\} \|S_i\| = \|S'_i\|$ and $\prod_{k \in S_i} m_i = \prod_{k \in S'_i} m'_i$. Proof is given using El Gamal homomorphism and Chaum-Pedersen.
- If a server fails in the previous test it is considered dishonest and is excluded and the protocol is restarted.
- If the protocol succeeds then the ballots are decrypted.

Almost Entirely Correct Mixing [?] IV

Properties for k mix servers and n inputs

- Proof Cost $= 2\alpha(2k-1)$ exponentiations
- Decryption Cost $= (2+4k)n$
- Privacy Every input is hidden among $n/2^{\alpha}$.
- Cheating will be detected with probability $1-\frac{5}{8}^{\alpha}$ or the DLOG problem can be solved in polynomial time.

Example

- Election with 160K voters
- $\alpha = 6$ subsets to check per server
- Every vote is hidden in 2500 votes
- Cheating will be detected with probability 94%

Optimistic Mixing [?] I

- Exit - Poll Mixing
- El Gamal Re encryption Mixnet
- Fast proof when all mix servers are honest
- If a cheating mix server is found then:
  - No output is produced
  - A correct proof is executed ([?],[?])
  - Privacy is not compromised
- Design to execute several mix sessions with the same set of keys
- Proofs of knowledge are bound to a session id

Optimistic Mixing [?] II

Proof of Product with checksum

- Input: Encryption of Plaintexts with cryptographic checksum
- Proof of Correct Operation: $\prod_{input} plaintexts = \prod_{output} plaintexts$ without knowledge of the plaintexts (for privacy reasons)
- Product Preservation does not imply absence of cheating
- If cheating occurs some plaintexts will have been replaced so that the product is not altered
- The checksums will still be invalid
- **Cause:** Cheating Mix Server Or Invalid checksum in the first place?

Optimistic Mixing [?] III

Solution

- Restrict the input plaintexts m_i (*and* the output plaintexts m'_i) to a particular format so that it is infeasible $\{m_i\} \neq \{m'_i\}$ and $\prod m_i = \prod m'_i$
- **How:** Apply a hash function to the input.
- $(E(m, r), E(h(m), r'))$
- **Theorem (Wagner):** It is impossible to find two different lists of ciphertexts of equal length $\{(u_i, v_i)\}_{i=1}^N \neq \{(u'_i, v'_i)\}_{i=1}^N$ such that $\prod_{i=1}^N h(u_i, v_i) = \prod_{i=1}^N h(u'_i, v'_i)$ in a group where the discrete logarithm is hard

Optimistic Mixing [?] IV

Privacy compromise

- **Remark:**Cheating will be detected **after the fact**. Not much can be done if privacy has been compromised (there is no use in mixing again:
- For example:
  - User i submits $t_i = (\ E(m_i, r_i), E(h(m_i), r'_i)\ )$
  - Cheating mix server replaces u_1 with $t_1 \cdot t_2$ and t_2 with $(1, 1, 1, 1)$.
  - Cheating will be discovered after the decryption
  - The cheating server will be disqualified but
  - If mixing is restarted the cheating server will be able to distinguish the output of the first two users from the outputs of the rest by comparing the output of the second run with the output of the first run by checking for the product of the two plaintexts

Optimistic Mixing [?] V

Solution:Double Enveloping

- User input is encrypted twice
- Each user submits the triple:
  $t_i = (\, E(G_i, r_i), E(M_i, r'_i), E(h(M, G), r''_i)\,)$ where
  $(G_i, M_i) = (g^{r_i}, m \cdot y^{r_i}) = E(m_i, r_i)$
- Verification Step: Decrypt Once
- If verification succeeds (both product and checksum) then decrypt twice and tally
- If cheating is discovered then encrypted inputs the cause of cheating is sought:
  - Cheating senders: Solution: Ignore them, and proceed with the rest
  - Cheating mix servers: The messages will be forwarded to the slower mixnet with the mix server replaced
- Cheating will not expose privacy, since the cheating server will cheat on (single) ciphertexts

Optimistic Mixing [?] VI

Investigation of Invalid Triples

- Invalid triples $w_i \neq h(u_i, v_i)$
- Backtrace their path through the mixnet
- Last server reveals: (input, randomisation)
- Validate that the triple can be obtained from the input
- Repeat until the first server
- If the first server is reached then classify as user cheating
- If a server fails validation the classify as server cheating

Further Work:Review

- Attacks
- Voting based on Homomorphic Encryption
- Voting based on Blind Signatures
- Voter Verification Methods (visual cryptography)
- Implementations (Helios, Scratch & Vote, PretaVoter)

References I

Masayuki Abe.

Universally verifiable mix-net with verification work independent of the number of mix-servers.

In Kaisa Nyberg, editor, *Advances in Cryptology EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447. Springer Berlin Heidelberg, 1998.

B. Adida.

Advances in cryptographic voting systems.

PhD thesis, Massachusetts Institute of Technology, 2006.

Ben Adida.

Verifying elections with cryptography, 2007.

References II

Dan Boneh and Philippe Golle.

Almost entirely correct mixing with applications to voting.

In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 68–77, New York, NY, USA, 2002. ACM.

David Chaum.

Untraceable electronic mail, return addresses, and digital pseudonyms.

Commun. ACM, 24(2):84–88, 1981.

Jun Furukawa and Kazue Sako.

An efficient scheme for proving a shuffle.

In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 368–387, London, UK, UK, 2001. Springer-Verlag.

References III

Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels.

Optimistic mixing for exit-polls.

In *Asiacrypt 2002, LNCS 2501*, pages 451–465. Springer-Verlag, 2002.

J. Alex Halderman.

Securing digital democracy.

Coursera Online Course, September 2012.

Markus Jakobsson, Ari Juels, and Ronald L. Rivest.

Making mix nets robust for electronic voting by randomized partial checking.

In *In USENIX Security Symposium*, pages 339–353, 2002.

References IV

Joe Kilian and Kazue Sako.

Receipt-free MIX-type voting scheme - a practical solution to the implementation of a voting booth.

In *Proceedings of EUROCRYPT 1995*. Springer-Verlag, 1995.

Jakobsson Markus and Juels Ari.

Millimix: Mixing in small batches.

Technical report, 1999.

C. Andrew Neff.

A verifiable secret shuffle and its application to e-voting.

In *Proceedings of the 8th ACM conference on Computer and Communications Security*, CCS '01, pages 116–125, New York, NY, USA, 2001. ACM.

References V

C. Andrew Neff.

Verifiable mixing (shuffling) of elgamal pairs.

Technical report, In proceedings of PET '03, LNCS series, 2004.

Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa.

Efficient anonymous channel and all/nothing election scheme.

In *EUROCRYPT*, pages 248–259, 1993.

Ronald Rivest.

6.897 - selected topics in cryptography, 2004.