International eVoting PhD workshop

Practical considerations for the implementation of end-to-end verifiable eVoting

May 2011
Researcher
Sandra.Guasch@scytl.com

scytl
Secure Electronic Voting

Index

- Introduction
- Types of verifiability
- Some verification methods
- Conclusions

- Remote electronic voting is a challenging field of research when voting in cryptography. Some security requirements specific for electoral processes have to be accomplished.

Secrecy of vote

Remote voter authentication

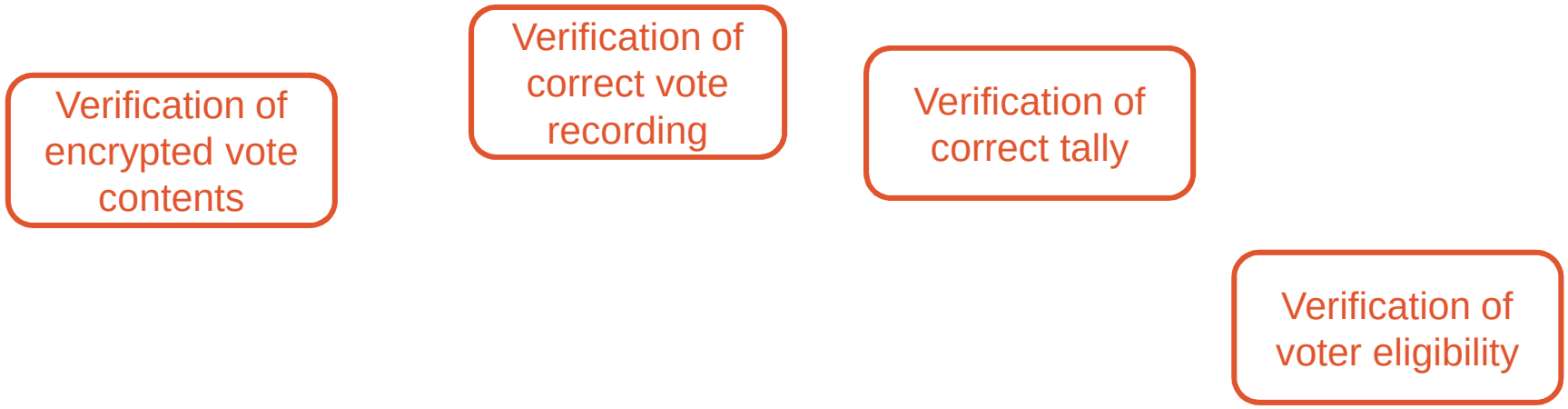
Integrity of election results

Receipt freeness

- Cryptographic tools are usually used in remote electronic voting protocols:
  - Encryption
  - Signatures
  - Zero Knowledge Proofs

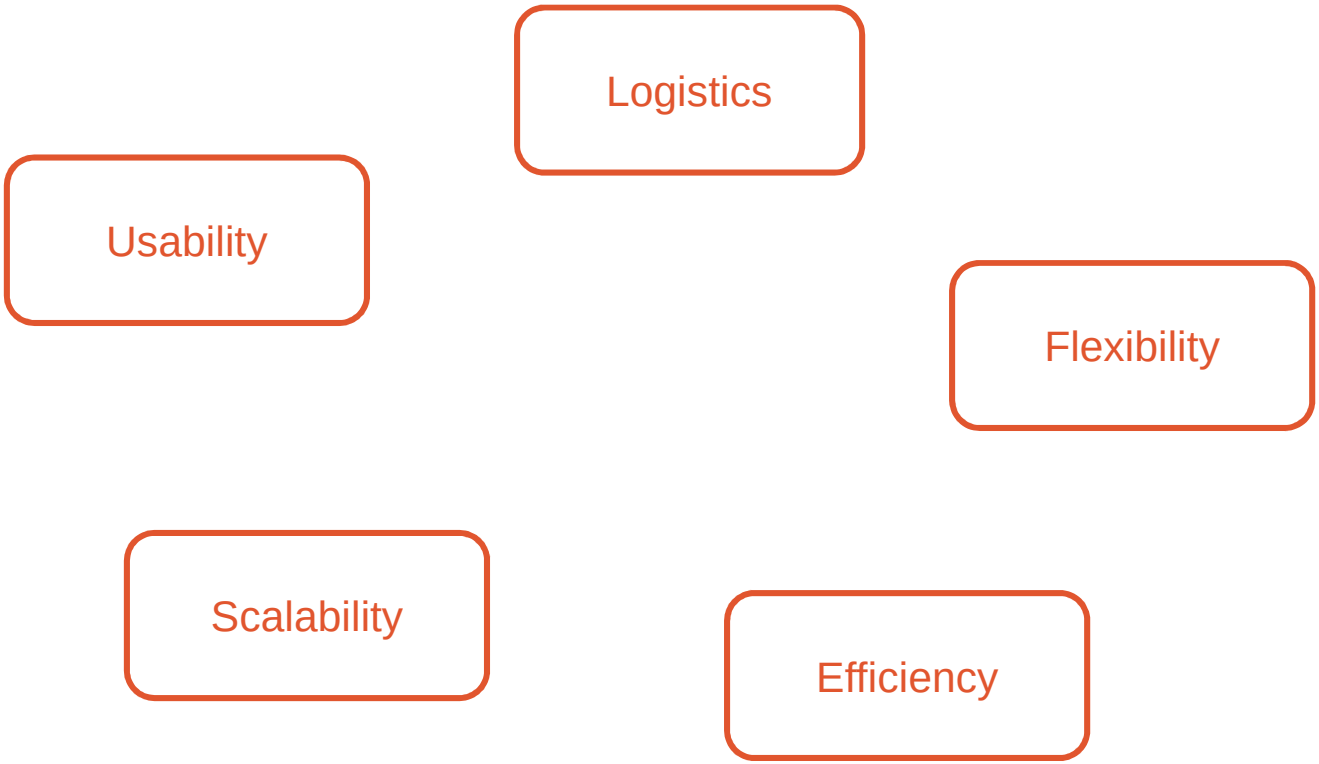
- Nice, but what about…
  - How much time does it cost to encrypt a vote? Is the voter going to wait that long?
  - If a voter is expected to digitally sign her vote before casting it, how does she get her signing keys?
  - How and where do we generate keys for the election?
  - How do we deliver the keys to their intended recipients?
  - How can we generate true random values for key generation and encryption? What about the voting client?

- Let’s talk about verifiability:



- Usually, verifiability implies using more cryptographic tools, which increases the cost and usability of the eVoting platform.

- Some issues have to be considered when designing the eVoting cryptographic protocol



Index

- Introduction
- Types of verifiability
- Some verification methods
- Conclusions

Types of verifiability

Based on who can verify

- Individual verifiability
  - Focused on the voter: only the voter that casts the vote is able to implement the verification process.
  - Audit of the correct encoding of the voting options, correct vote reception, and presence of the vote on the final count.
  - Security concerns: preservation of voter privacy and prevention of vote selling/coercion practices.

- Universal verifiability
  - Focused on the public, not restricted to voters.
  - Audit of the correct vote counting.
  - Security concerns: preservation of voter privacy.

Types of verifiability

Based on what is verified

- Cast as intended (Karlof et al.)
  - Voters can verify that their cast votes really represent their voter intent.

- Counted as cast (Karlof et al.)
  - Any observer can verify that the final tally is an accurate count of the ballots cast.

End-to End verifiability (Benaloh‘06)= cast as intended + counted as cast

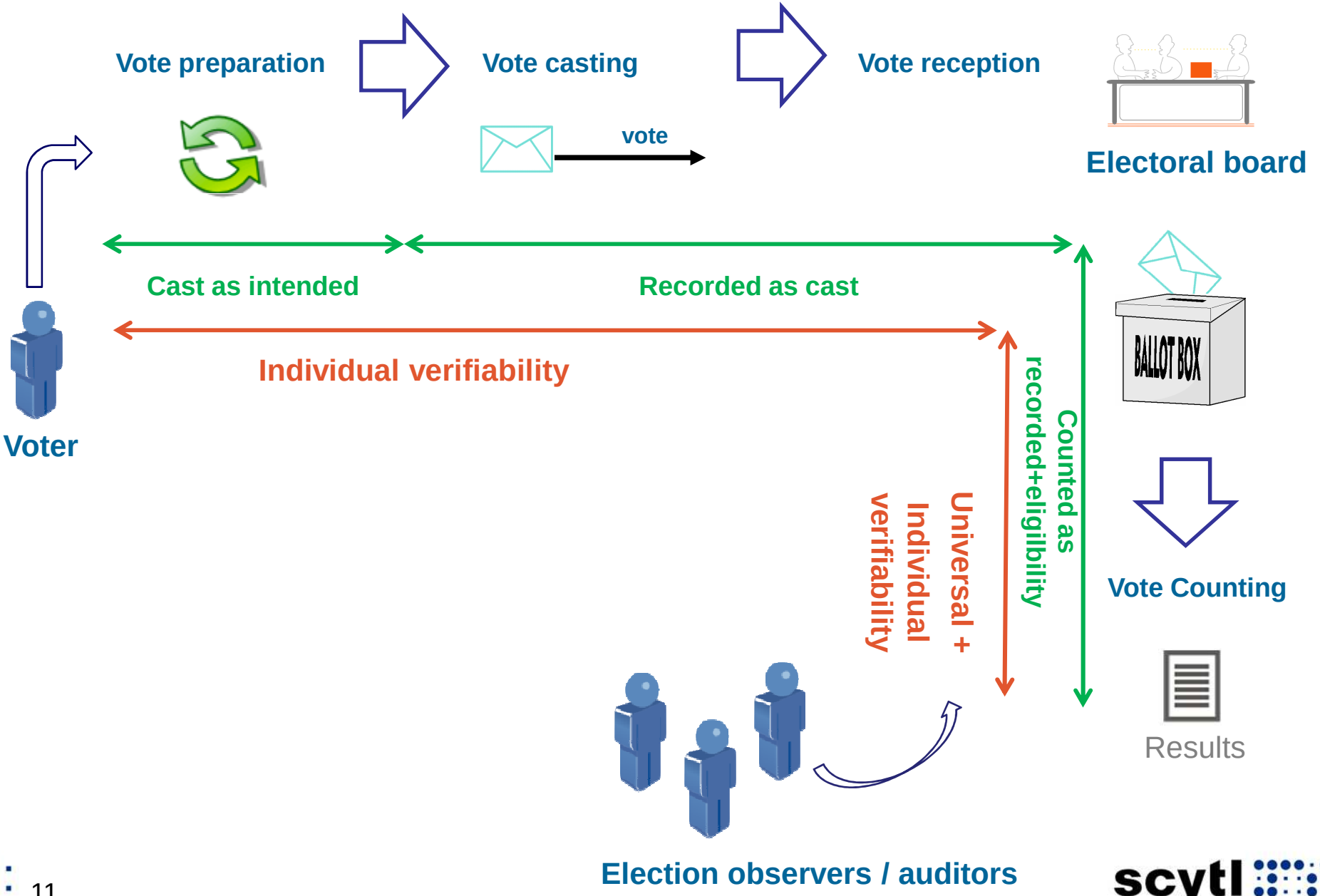
Types of verifiability

Based on what is verified

We can divide more:

- Cast as intended (Karlof et al.)
  - Voters can verify that their cast votes really represent their voter intent.
- Recorded as cast
  - Voters can verify that their cast votes have been properly stored (recorded) in the ballot box.
- Counted as recorded
  - Any observer can verify that only votes belonging to eligible voters are properly tallied.
  - A voter can verify that her vote has been included in the tally.

Verifiability and election processes



Index

- Introduction
- Types of verifiability
- Some verification methods
- Conclusions

Some verification methods in remote e-voting

- Individual verifiability:
  - Cast as intended verification.
  - Recorded as cast verification.
  - Counted as recorded verification.

- Universal verifiability:
  - Counted as recorded verification.
  - Eligibility verification.

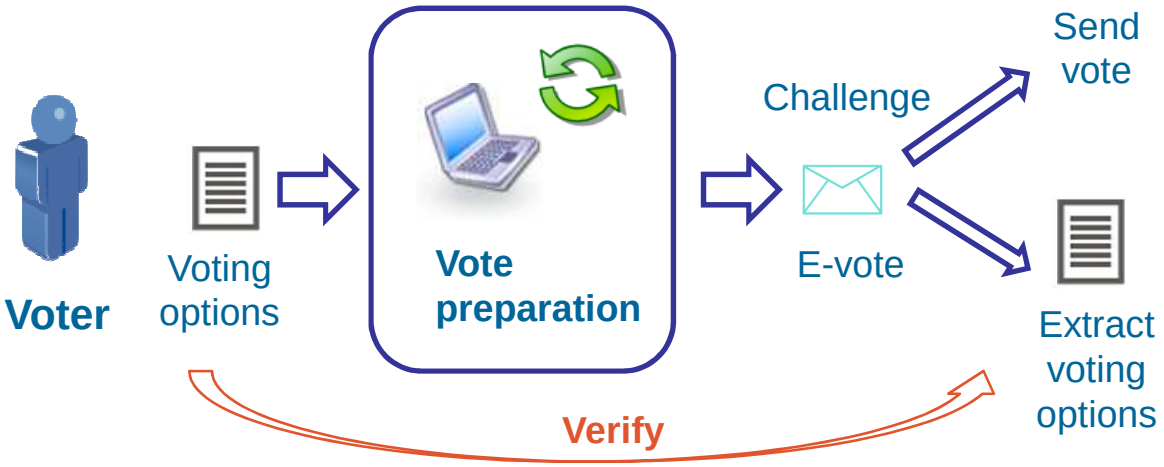
Cast as intended verification

➢ Vote encryption challenge (1/3)

- Proposed by Benaloh (Simple Verifiable Elections), used in Helios (Ben Adida).
- Separation of vote creation and casting functions for cast as intended verification.
- A malicious voting client that does not know when it is more likely to successfully cheat cannot do better than cheat at random.
- Random audit process.

➢ Vote encryption challenge (2/3)

- Vote encryption + encryption proof (e.g., hash of the encrypted vote).
- Voter can challenge the application to verify the proper encryption of the voting options:
  - Challenge: application shows the secret random parameters used to encrypt the vote.
  - Verification: voter uses the random parameters to verify if the encrypted vote contains her voter intent.
  - New encryption: the vote is encrypted again with new random parameters, and a new encryption proof is generated.



Vote encryption challenge (3/3)

- Practical issues:
  - Usability:
    - Independent voter verifiability is restricted: the voter cannot verify the correct encryption by herself.
    - An external tool has to be provided in order to make the calculations: numbers are very large, so that a simple calculator is not useful. The voter has to trust the entity providing the external tool.
    - The verification process may be complex for an average user.

➢ Pollsterless or code voting (1/3)

- Malkhi et al. Pollster: software and/or hardware artifact that participates in a voting protocol on behalf of the human voter (p.e., voting software performing the cryptographic operations).

- Pollsterless scenario:

  - Each voter has a Voting Card with a set of voter unique code pairs related to the voting options: candidate codes and return codes.

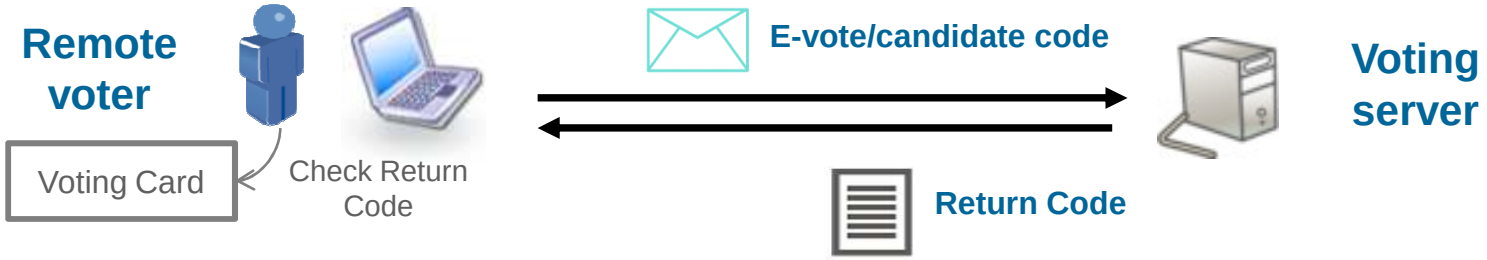
  - Voters use pre-encrypted candidate codes to vote for a particular candidate.

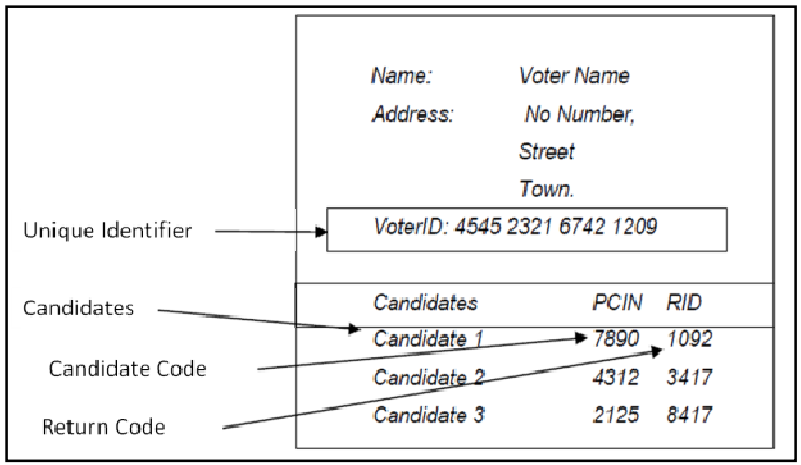
  - When casting a vote, the voting platform calculates return codes from the received encrypted vote and sends them to the voter.

  - The voter uses the Voting Card to verify that the received Return Codes match her selected candidates.

➢ Pollsterless or code voting (2/3)

- Usually two approaches:
  - Pre-encrypted ballots: Voting Card contains vote candidate codes for casting the vote.
  - Voter encrypted ballots: the vote is encrypted in the voting terminal (does not use pre-encrypted codes per candidate).





scytl Secure Electronic Voting

➢ Pollsterless or code voting (3/3)

- Practical issues:
  - Usability: voters are expected to enter complex codes and compare them for voting and verification.
    - This can be mitigated by using the second approach: click & select scheme with return codes for validation.
  - Logistics: voting cards have to be delivered to voters before or during the voting phase.
  - Scalability: elections with a high volume of parties and candidates can lead to poor usable voting cards.
  - Security: interception of Return Codes when validating can lead to a man-in-the-middle attack.

Cast as intended verification

➢ Return Codes for encrypted votes
(alternative code voting approach for the eValg-2011 project)

- Proposal:
  - Voter encrypted ballots: the vote is encrypted in the voting terminal using a voting client -> reduces usability issues.
  - Return codes are generated in the voting server and returned to the voter, who verifies them using the voting card. An alternative channel is used for sending the return codes (e.g., SMS) -> prevents man-in-the-middle attacks.
  - Voting cards are not needed to vote, but to verify the vote casting. Voting cards can be delivered to voters during the voting phase -> easier for logistics.
  - Return codes can represent positions in a party list instead of candidates -> more scalable. Voters have to make another check to verify the original order of party lists.

➢ Homomorphic tally (1/8)

- Single votes are never decrypted, but their aggregation.
- Homomorphic encryption:

$$E(m_1) \oplus E(m_2) = E(m_1 \otimes m_2)$$

- ElGamal encryption scheme:
  - Multiplicative homomorphism:

$$E(m_1) = (m_1 \cdot h^{r_1}, g^{r_1})$$

$$E(m_2) = (m_2 \cdot h^{r_2}, g^{r_2})$$

$$(m_1 \cdot h^{r_1}, g^{r_1}) \times (m_2 \cdot h^{r_2}, g^{r_2}) = (m_1 \cdot m_2 \cdot h^{r_1+r_2}, g^{r_1+r_2})$$

$$D(E(m_1) \times E(m_2) = m_1 \cdot m_2$$

➢ Homomorphic tally (2/8)

- ElGamal encryption scheme:
  - Additive homomorphism:

$$E(m_1) = (\lambda^{m_1} \cdot h^{r_1}, g^{r_1})$$

$$E(m_2) = (\lambda^{m_2} \cdot h^{r_2}, g^{r_2})$$

$$(\lambda^{m_1} \cdot h^{r_1}, g^{r_1}) \times (\lambda^{m_2} \cdot h^{r_2}, g^{r_2}) = (\lambda^{m_1} \cdot \lambda^{m_2} \cdot h^{r_1+r_2}, g^{r_1+r_2})$$

$$D(E(m_1) \times E(m_2)) = \lambda^{m_1+m_2}$$

  - Candidates are usually represented as individual encryptions where m1 and m2 denote selection/no selection.

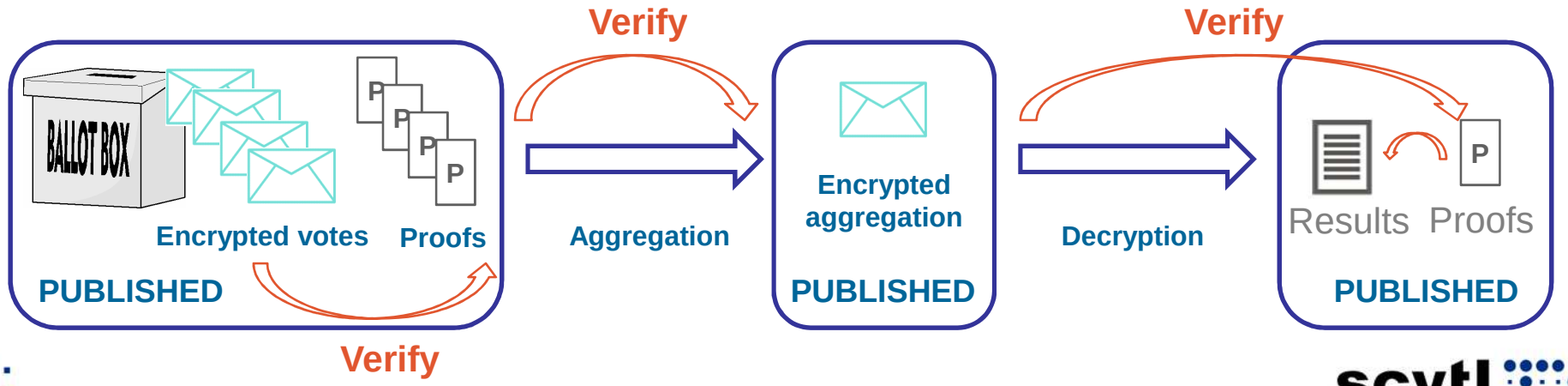
$$\begin{array}{ll} & \lambda^{m_1}, \lambda^{m_2}, \lambda^{m_2} \\ & \lambda^{m_1}, \lambda^{m_2}, \lambda^{m_2} \\ \text{X} & \lambda^{m_2}, \lambda^{m_1}, \lambda^{m_2} \end{array} \longrightarrow \text{Candidate 1 wins!}$$

➢ Homomorphic tally (3/8)

- "Multi-Authority Election Scheme" (*Cramer, Gennaro, Schoenmakers*):
  - Threshold scheme for public key encryption:
    - Each authority has a share of the private key (Shamir threshold secret sharing scheme).
    - Public commitment in order to ensure that the shares are correct.
  - ElGamal cryptosystem for encrypting the votes:
    - Additive homomorphic encryption.
  - Voting client also generates Zero Knowledge Proofs of validity: to ensure that the vote is correct.
  - Votes encrypted and digitally signed are published on a bulletin board.

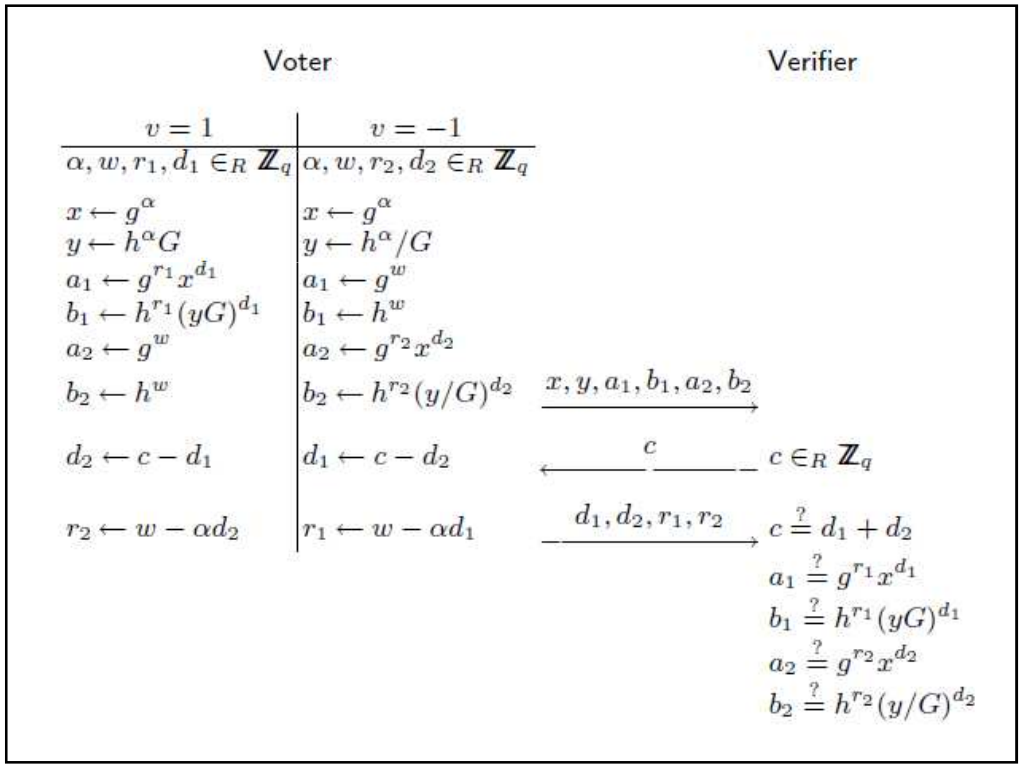
➢ Homomorphic tally (4/8)

- "Multi-Authority Election Scheme" (*Cramer, Gennaro, Schoenmakers*):
  - At the end of the voting phase, votes published on the bulletin board are multiplied.
  - Each authority makes a partial decryption and generates a ZKP of correct decryption.
  - Final decryption values are calculated, and values for each candidate are obtained.



➢ Homomorphic tally (5/8)

- "Multi-Authority Election Scheme" (Cramer, Gennaro, Schoenmakers):
  - Practical issues (I):
    - Scalability: as many encryptions in the voter side as voting choices in the election. One ZKP of validity per voting option (encryption).



8 modular exponentiations per voting option in the voter side!

scytl Secure Electronic Voting

➢ Homomorphic tally (6/8)

- What about using multiplicative homomorphism to enhance scalability?
  - Only selected voting options need to be encrypted.
  - ZKP of validity increases its cost: proof of encrypting m1 or m2 translates into proof of encrypting a value between as many options as choices in the election.

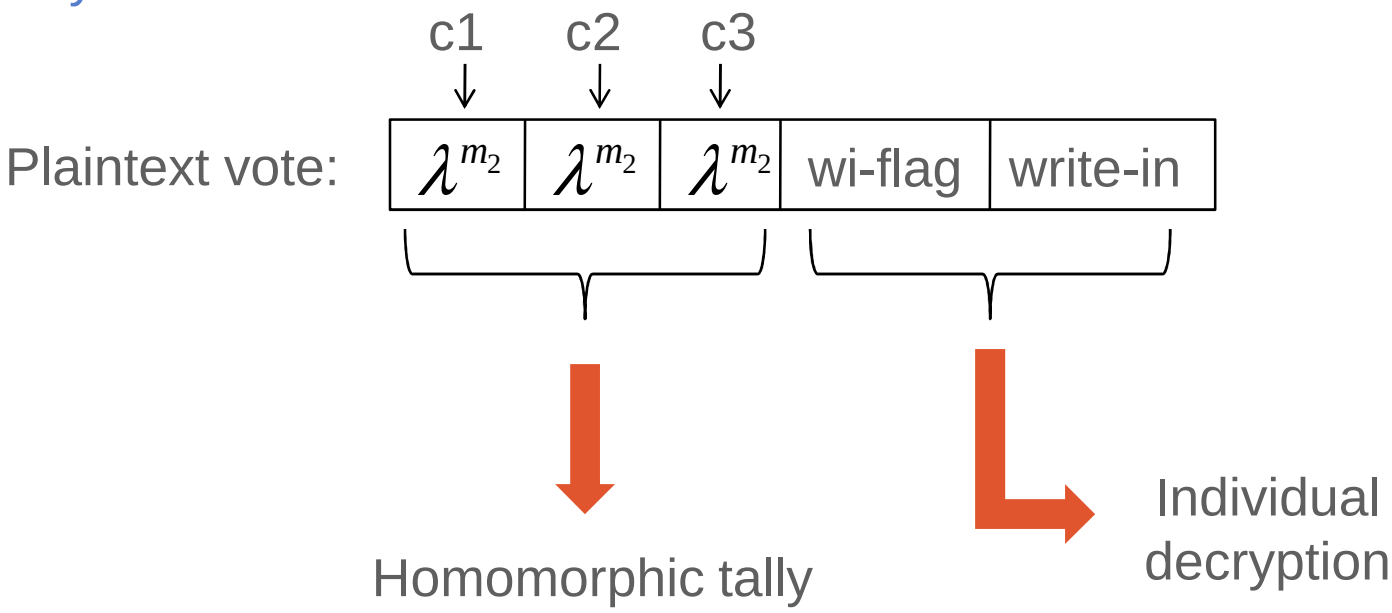
$$\log_g x = \log_h(y/m_0) \quad \vee \quad \log_g x = \log_h(y/m_1)$$

$$\log_g x = \log_h(y/G_1) \quad \vee \quad \cdots \quad \vee \quad \log_g x = \log_h(y/G_K)$$

  - Possible overflow when operating the votes.

➢ Homomorphic tally (7/8)

- Practical issues (II):
  - Flexibility: write-ins are not allowed in this scheme.
- "The Vector-Ballot E-Voting Approach" (*Kiayias, Yung*):
- Better approach for allowing write-ins in homomorphic tally schemes
 -> flexibility



➢ Homomorphic tally (8/8)

- "Batch ZK Proof and Verification of OR Logic" (*Kun Peng, Feng Bao*):

- More efficient proofs of validity -> scalability

- Additive homomorphism:
  - Multiple encryptions: one for each vote option.
  - PoV: each encryption contains one value from a group of two (selection/no selection):

$$\log_g x = \log_h(y/m_0) \quad \vee \quad \log_g x = \log_h(y/m_1)$$

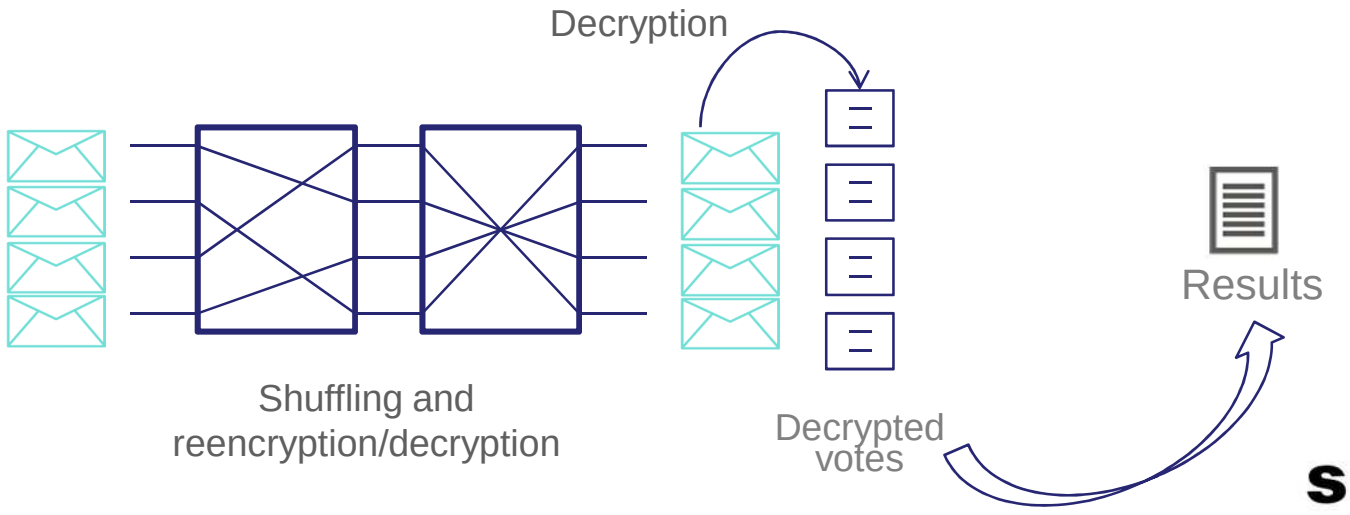
- Multiplicative homomorphism:
  - Less encryptions: one for each selected voting option.
  - PoV demonstrates that each encryption contains one value from the group of all the voting options.

$$\log_g x = \log_h(y/G_1) \quad \vee \quad \cdots \quad \vee \quad \log_g x = \log_h(y/G_K)$$

Batch ZK Proofs group multiple statements linked with OR.

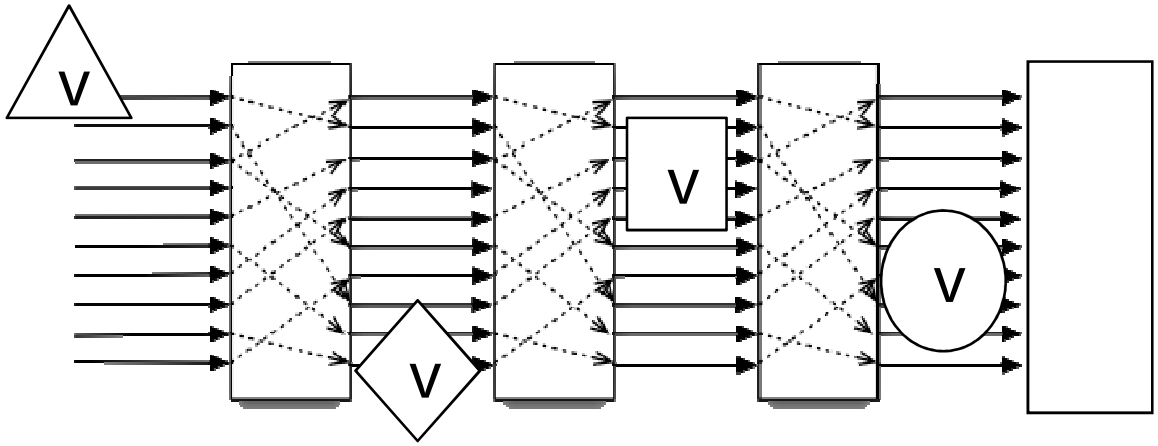
➢ Mix-Nets

- Several nodes shuffle and re-encrypt/decrypt the votes for breaking the correlation between the original input order and the output one.
  - Initially, encrypted votes are separated from their digital signatures.
  - The shuffled and re-encrypted/decrypted vote output from one node is used as the input of another one.
  - The vote contents are obtained (decrypted) at the last node.



➢ Universally verifiable Mix-Nets (1/4)

- Each Mix-Node “changes” the value of the encryption (re-encryption or decryption).



How do we know that a Mix-Node has not changed the vote?

- Verification:
  - Each mix-node calculates proofs of correct shuffling and correct re-encryption/decryption.
  - All the proofs are verifiable by anyone to detect that the input and output votes are based on the same original votes (i.e., have not been changed).

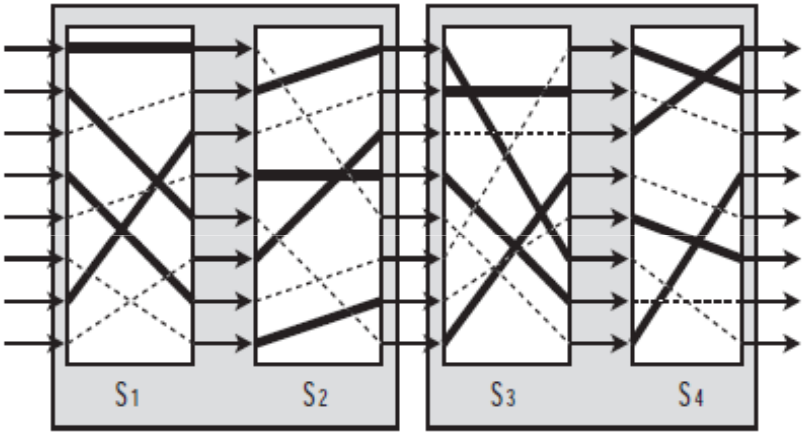
➢ Universally verifiable Mix-Nets (2/4)

- Practical issues:
  - Scalability:
    - Each vote is individually re-encrypted/decrypted at each node.
    - Generation and verification of ZK proofs of re-encryption/decryption is very costly.
    - Best robust proof is from Neff: 19n+7 modular exponentiations at each node.

➢ Universally verifiable Mix-Nets (3/4)

- Other proposals verify groups of votes or part of the votes at each node in order to increase efficiency:

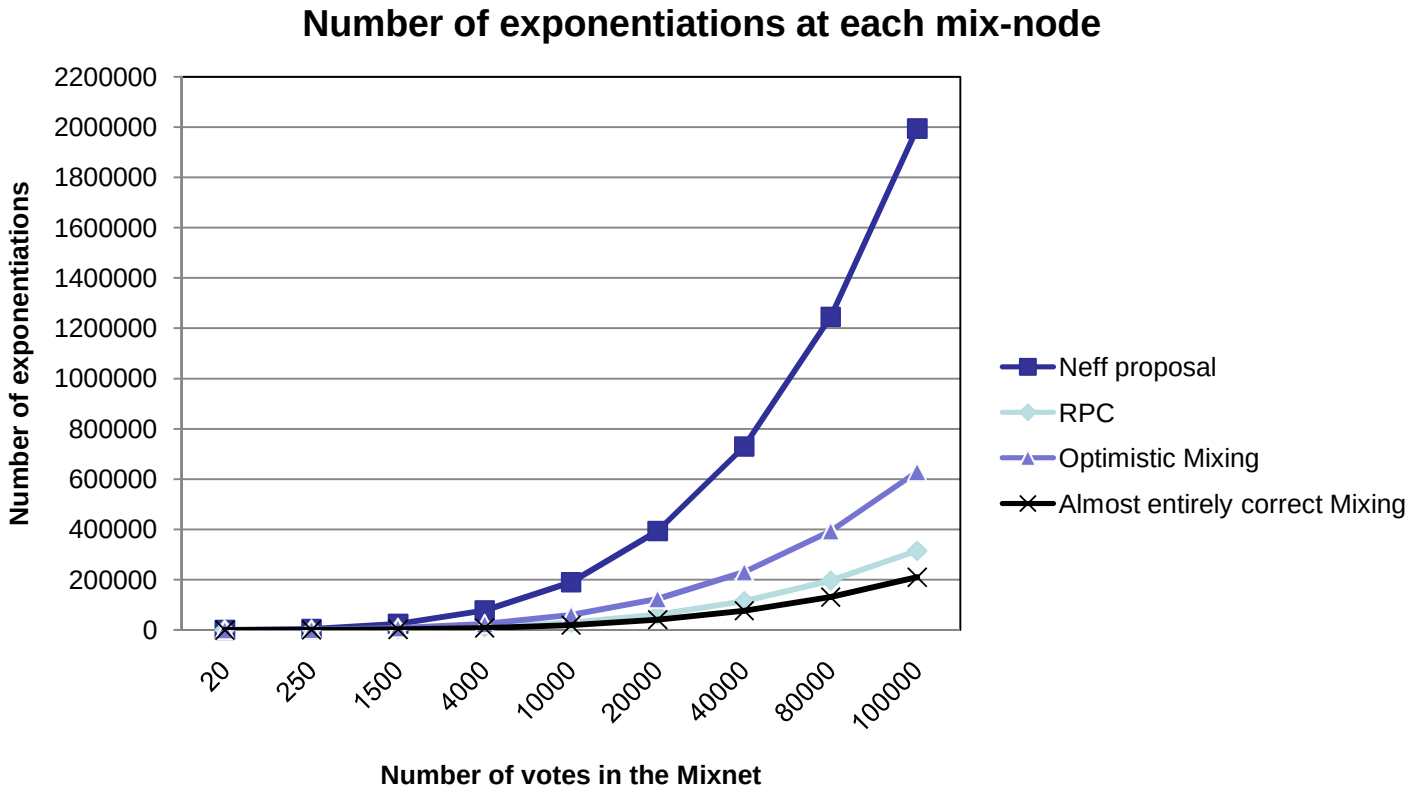
  - Random Partial Checking (*Jakobsson, Juels, Rivest*): only half of the votes are checked at each node (probabilistic verification). 3n modular exponentiations per mix-node.



  - Optimistic Mixing *(Golle et al.)*: generates one re-encryption proof for all the votes. 6n+6 modular exponentiations per mix-node.

➢ Universally verifiable Mix-Nets (4/4)

- Almost entirely correct Mixing *(Boneh, Golle)*: votes are grouped. The Mix-Nodes generate one re-encryption proof for each group of votes. 2n+30 modular exponentiations per mix-node.

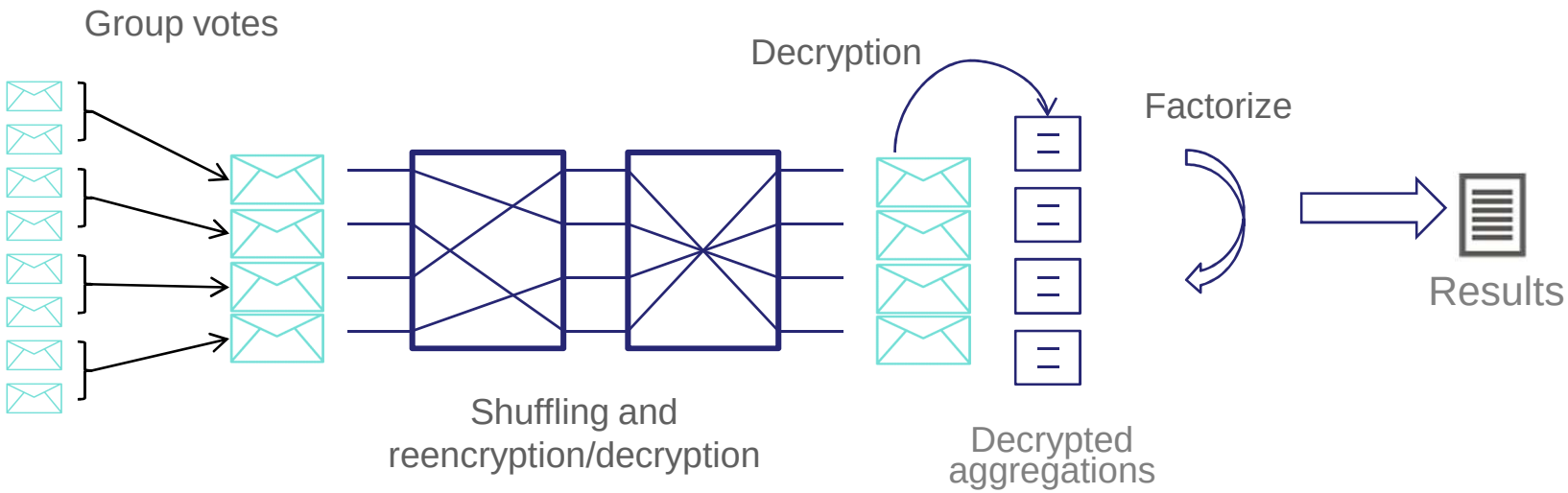


➢ Universally verifiable Mix-Nets + homomorphic encryption (1/2)

- "Multiplicative Homomorphic E-Voting" (*Peng, Aditya, Boyd, Dawson, Lee)*:
- Homomorphic tally is combined with a mixing scheme:
  - Votes are encrypted using a multiplicative homomorphic encryption algorithm (ElGamal).
  - In traditional homomorphic tally, votes are multiplied and result is decrypted.
    - Values encrypted may overflow the ElGamal module, so that decryption is not possible. Solution: divide votes to be operated in small groups.
    - Privacy problem: anonymity is only achieved within a subset of votes. Solution: mix the groups of votes before decrypting.
- The number of votes to be processed by the Mix-Net is reduced.

➢ Universally verifiable Mix-Nets + homomorphic encryption (2/2)

- "Multiplicative Homomorphic E-Voting" (*Peng, Aditya, Boyd, Dawson, Lee)*:



- Combining both methods their drawbacks are reduced.

Conclusions

- Cast as intended verification can pose some usability, scalability and logistic issues. It has to be designed carefully so that the most amount of voters are able to validate their votes. There is not a perfect system, but conventional proposals can still be enhanced.

- For counted as recorded verification, the verification system must be chosen depending on the election characteristics (homomorphic tally or Mixing). Some nice work is being done by Peng et al. to improve the scalability of these systems.

www.scytl.com