Proof of Vote® - Appendix

Contents

A Cryptographic & Computational Security

A.1 Security of Distributed Key Generation

In the Proof of Vote protocol, a distributed t-out-of-w threshold key sharing mechanism is employed. Each of w private key holders shares his private key among the others such that any t of them can reconstruct the private key. Every key holders sums up the key shares for him such that any t of them can reconstruct the sum of all the holders' private key, which is the vote decryption key and its corresponding public key is the vote encryption key.

In our protocol, a distributed ElGamal cipher protocol is used by trustees to generate a shared key. A distributed paillier cipher is explored for pedagogical reasons, but is rejected due to efficiency considerations.

The following security properties are desired in threshold key sharing.

Definition 6 *(threshold key sharing). In a t-out-of-w threshold key sharing protocol, there exists a secret but unique share generating polynomial such that any share holder P_i can verify that his share d_i is generated by the polynomial as the i^{th} share without knowing any other information about the polynomial than his share.*

Definition 7 *(Bindingness). There exists a secret but unique share generating polynomial in a binding threshold key sharing protocol such that any received share d_i passing its verification of the share validity is guaranteed to be generated by the polynomial as the i^{th} share with an overwhelmingly large probability.*

Definition 8 *(Hidingness). A threshold key sharing protocol is hiding if its verification mechanism does not reveal any information about the private key or any of its shares. More precisely, the two transcripts of proof of validity of the sharing of any two different private keys should have the same distribution and thus are indistinguishable.*

A.1.1 Distributed ElGamal Cipher

ElGamal encryption is the first choice in our e-voting system for ballot encryption. A merit of El Gamal is that a threshold encryption algorithm with distributed key generation is easily supported such that the private key to decrypt the votes can only be available with the cooperation of a threshold number of private key holders. Each of w private key holders shares his private key among the others such that any t of them can reconstruct the private key. Every key holders sums up the key shares for him such that any t of them can reconstruct the sum of all the holders' private key, which is the vote decryption key and its corresponding public key is the vote encryption key. t-out-of-w threshold ElGamal is generated and used in a distributed pattern as follows.

1. Parameter setting:
   G with order q is a cyclic subgroup of Z_p^* where q and $p = 2q + 1$ are large primes, g is a generator of G. Parameters p, q, g, y are published. There are w key sharers $P_1, P_2, \dots, P_w$, who cooperate to manage the threshold ElGamal cipher.
2. Key generation:
   Each P_i chooses his private key seed $x_i \in Z_q$ and publishes $y_i = g^{x_i} \bmod p$ for $i = 1,2,\dots,w$. Each P_i generates a polynomial $F_i(x) = \sum_{k=0}^{t-1} f_{i,k} x^k \ mod \ q$ where $f_{i,0} = x_i$ and $f_{i,1}$, $f_{i,2}$, , $f_{i,t-1}$ are random integers in Z_q. Each P_i distributes $d_{i,j} = F_i(j)$ to other key sharers P_j for $i = 1,2,\dots,w$ and $j = 1,2,\dots,i-1,i+1,i+2,\dots,w$. Each P_j calculates his private key share $d_j = \sum_{i=1}^{w} d_{i,j} \bmod q$.
3. Encryption:
   The public key of the ElGamal cipher is publicly available as $y = \prod_{i=1}^{w} y_i \bmod p$. A message $m \in Z_p^*$ is encrypted into a ciphertext pair (a, b) where $a = g^r \bmod p$, $b = my^r \bmod p$ and r is randomly chosen from Z_q.
4. Distributed decryption:
   For $j = 1,2,\dots,w$, every cooperating P_j provides his part of decryption

   $s_j = a^{d_j} \bmod p$.
5. Combination:
   If a set S of at least t participants provide correct shares,
   $s = \frac{b}{\prod_{P_i \in S} s_i^{u_i}} \ mod \ p$ can be recovered where $u_i = \prod_{P_j \in S, j \neq i} \frac{j}{j-i}$.

As the whole e-voting system must be end-to-end verifiable, so are the distributed key generation mechanism and the distributed decryption. Each share holder needs to verify that he can sum up the shares for him to a private key share to recover an unknown but unique private key, namely cooperation of any t share holders must result in the same private key. This is called verifiable key sharing, which is necessary as none of the private key sharers is trusted (otherwise we can just trust a private key holder and do not need threshold decryption at all) and must be prevented from cheating. Moreover, any one including the independent observers must be able to verify correctness of the distributed decryption, namely the cooperating key sharers use their private key shares to correctly decrypt the votes. The verification mechanism works as follows.

1. Public commitment:
   Each P_i publishes $E_{i,k} = g^{f_{i,k}} \bmod p$ for $k = 0,1,\dots,t-1$.

2. Verifiable key sharing:
   Each P_j verifies validity of $d_{i,j}$ sent to him by P_i against an equation

$$g^{d_{i,j}} = \prod_{k=0}^{t-1} E_{i,k}^{j^k} \bmod p \quad for \quad i = 1,2,\dots,j-1,j+1,j+2,\dots,w$$

If and only if the verification is satisfied, each P_j can be sure that d_j is the j^{th} share of $\sum_{i=1}^{n} \log_g E_{i,0} \bmod q$.

3. Verifiable decryption:
   Validity of the partial decryption result s_j can be publicly proved by P_j using a ZK proof of
   $$\log_g D_j = \log_a s_j \qquad (9)$$
   where
   $$D_j = \prod_{k=0}^{t-1} \prod_{i=1}^{w} E_{i,k}^{j^k} \bmod p$$
   If and only if the verification is satisfied, s_j is valid and t instances of it can reconstruct the message m.

A.1.2 Distributed Paillier Cipher

Paillier encryption is additive homomorphic and so is desired in some e-voting (especially homomorphic voting) scheme. However, it is hard to generate a Paillier key pair by multiple parties in a distributed way, although distributed decryption is still feasible with Paillier encryption. In the following, a centrally-generated Paillier private key is shared among w parties such that any t of them can reconstruct the private key or cooperate to implement a decryption.

1. **Key generation:**
   $N = pq$, $p = 2p' + 1$ and $q = 2q' + 1$ where p and q are primes and $gcd(N, \varphi(N)) = 1$. Integers a, b are randomly chosen from Z_N^* and $g = (1+N)^a + b^N \bmod N^2$. The private key is $\beta p' q'$ where β is randomly chosen from Z_N^*. The public key consists of N, g and $\theta = a\beta p' q'$. $P_1, P_2, \dots, P_w$ are the private key sharers. Let $F(x) = \sum_{k=0}^{t-1} f_k x^k$ where $f_0 = \beta p' q'$ and f_1, f_2, , f_{t-1} are random integers in $Z_{p\prime q\prime}$. The share $d_j = F(j) \bmod p' q' N$ is distributed to P_j for $j = 1,2,\dots,w$. G is the cyclic subgroup containing all the quadratic residues in $Z_{N^2}^*$ where an integer y in Z_{N^2} is an N^{th} residue if there exist an integer x such that $x^N = y \bmod N^2$.
   Random integer v is a generator of G and $v_j = v^{\Delta d_j} \bmod N^2$ for $j = 1,2,\dots,w$ where $\Delta = w!$. Integers v and v_j for $j = 1,2,\dots,w$ are published together with $E_k = v^{\Delta f_k} \bmod N^2$ for $k = 0,1,\dots,t-1$.

2. **Key share verification:**

Each P_j can verifies validity of d_j by

$$v^{d_j\Delta} = v_j \bmod N^2$$

$$v_j = \prod_{k=0}^{t-1} E_k^{j^k} \bmod N^2$$

If and only if the verification is satisfied, d_j is accepted.

3. **Encryption:**
   A message $m \in Z_N$ is encrypted into $c = g^m r^N \bmod N^2$ where r is randomly chosen from Z_N^*.

4. **Partial decryptions of ciphertext c:**
   For $j = 1,2,\ldots,w$, A_j provides his part of decryption $s_j = c^{2\Delta d_j} \bmod N^2$ and proves validity of his partial decryption by a ZK proof

   $$\log_{c^{4\Delta}} s_j^2 = \log_{v^\Delta} v_j \qquad (10)$$

   If and only if the verification is satisfied, s_j is accepted.

5. **Combination of partial decryptions:**
   The final decryption result can be recovered as $s = L(\prod_{j\in S} s_j^{2u_i}) \times \frac{1}{4\Delta^2\theta}$ where set S contains the indices of t correct partial decryptions and $u_j = \Delta \prod_{1\le j'\le M, j'\ne j} \frac{j'}{j'-j}$.

A.1.3 Conclusion

Both threshold ciphers achieves bindingness and hidingness and the security analysis is omitted due to space limitation. Distributed key generation is possible with Paillier cipher in theory although in practice it is too complicated and costly to be employed. We therefore use an ElGamal cipher for distributed key generation.

A.2 Security of Re-Encryption

Receiving a ciphertext as input, a re-encryption function outputs another ciphertext containing the same message. In our e-voting scheme, re-encryption mix network is employed to shuffle the encrypted ballots for anonymity of the voters. So, re-encryption function of the employed encryption algorithm is important and its security needs more study and exploration.

A.2.1 Re-encryption Functions

The first choice of encryption algorithm is ElGamal encryption, whose setting and re-encryption function are as follows.

1. **Parameter setting:**
   G with order q is a cyclic subgroup of Z_p^* where q and $p = 2q + 1$ are large primes, g is a generator of G.
2. **Key setting:**
   A random integer $x \in Z_q$ is the private key, while g and $y = g^x \bmod p$ is the public key. The keys may be generated in a distributed way by multiple key sharers such that decryption is feasible if and only if the number of cooperating key sharers is over a threshold as detailed in the CDP 74.
3. **Encryption:**
   A message $m \in Z_p^*$ is encrypted into a ciphertext pair (a, b) where $a = g^r \text{mod} p$, $b = my^r \text{mod} p$ and r is randomly chosen from Z_q.
4. **Re-encryption:**
   A ciphertext $c = (a, b)$ is re-encrypted into a new ciphertext pair (a', b') where $a' = ag^{r'} \bmod p$, $b' = by^{r'} \bmod p$ and r' is randomly chosen from Z_q.

If homomorphic tallying is adopted as a choice, usually an additively-homomorphic encryption algorithm is needed. Paillier encryption is a good choice and its setting and re-encryption function are as follows.

1. **Parameter setting:**
   Choose a large modulus $N = \alpha\beta$, where α and β are large prime integers. Choose an integer δ so that its order αN is a multiple of N modulo N^2.
2. **Key setting:**
   The public key is N and δ and the private key is $\lambda(N) = \lambda(N) = lcm(\alpha - 1, \beta - 1)$. The keys may be shared by multiple key sharers such that decryption is feasible if and only if the number of cooperating key sharers is over a threshold as detailed in CDP 74.
3. **Encryption:**
   To encrypt a message $m \in Z_N$, choose a random integer $r \in Z_N^*$ and compute the ciphertext $c = \delta^m r^N \bmod N^2$.
4. **Re-encryption:**
   A ciphertext $c \in Z_{N^2}$ is re-encrypted into $c = cr'^N \bmod N^2$ where r' is randomly chosen from Z_N^*.

A.2.2 Re-encryption and Semantic Security

Re-encryption is only possible for public-key probabilistic encryption algorithms, where a probabilistic encryption of a message m by a key k involves a random integer r when generating the ciphertext $c = E_k(m, r)$. An encryption algorithm with encryption function $E()$ is semantically secure if no polynomial adversary can win the following game with a probability non-negligibly larger than 0.5.

1. The adversary chooses two different messages m_1 and m_2 in ny way he likes and sends them to an encryption oracle.

2. The encryption oracle randomly chooses i from $\{1,2\}$ and outputs $c = E(m_i)$.

3. The adversary wins the game if it receives c and finds out i.

As multiple ciphertexts are re-encrypted in a mix network, traditional semantic security is extended to a new property called semantic security with respect to multiple re-encryptions in security analysis of mix networks as follows.

Definition 9 *An encryption algorithm is semantically secure with respect to multiple re-encryptions if given three groups of ciphertexts* $c_{1,1}, c_{1,2}, \dots, c_{1,k}$, $c_{2,1}, c_{2,2}, \dots, c_{2,k}$ *and* $c_1, c_2, \dots, c_k$*, such that* $c_j = RE(c_{i,j})$ *for* $j = 1,2,\dots,k$ *and only one* i *in* $\{1,2\}$ *where* $RE()$ *denotes re-encryption, there is no polynomial algorithm to find out the* i *to satisfy* $c_j = RE(c_{i,j})$ *with a probability non-negligibly larger than 0.5.*

A purpose of probabilistic encryption is to achieve semantic security so that privacy of the encrypted message (e.g. encrypted ballots in e-voting systems) can be formal and strong, while in comparison deterministic encryption algorithms like RSA and AES cannot provide such strong privacy based on formal definitions. Both ElGamal encryption algorithm and Paillier encryption algorithm are semantically secure regarding encryption and multiple re-encryptions. The key reason for their semantic security is the involvement of the random integer in encryption. Moreover, re-encryption becomes possible after the random integer is added.

A.2.3 Re-encryption and Chosen Plaintext Attack

A concrete and practical functionality of the abstract concept of semantic security is invulnerability to cryptographic attacks. For example, semantically secure encryption algorithms are inherently immune to the famous chosen plaintext attack. In a chosen plaintext attack, an adversary has access to an encryption oracle, which receives a plaintext and returns its encryption. Chosen plaintext attack can be modified to chosen re-encryption attack, where an adversary accesses a re-encryption oracle and obtains the re-encryption result of a chosen ciphertext.

As encryption functions (and especially public keys of asymmetric ciphers) are usually widely provided to common users and access to an encryption client is quite easy, its threat is widespread when the employed encryption algorithm is not semantically secure. For example, if RSA or a symmetric cipher like AES is employed, an adversary can test whether a possible message (e.g. a certain selection in a ballot) is encrypted in a ciphertext by simply querying the encryption oracle and comparing the return of the oracle where the rate of success is 100 percent. Chosen re-encryption attack is a possible way to compromise anonymity in a mix network by treating a mixer as a re-encryption oracle.

More complex attacking strategy like adaptive chosen plaintext attack, which repeatedly querying the encryption oracle and choosing the next input plaintext according to the returns of the past queries, can even extract information about the private/secret key and completely compromise the encryption system. The same adaptive strategy can be applied to chosen re-encryption attack. However, this kind of attack is not our main concern in e-voting.

Obviously, any semantically secure encryption algorithm is resistant to chosen plaintext attack and chosen re-encryption attack against encrypted secret, otherwise a successful attack can be used to break semantic security. Although RSA and AES can be strengthened against chosen plaintext attack by adopting padding to the messages to be encrypted, informal mechanisms like padding cannot completely eliminate the threat of chosen plaintext attack either in theory or in practice.

A.2.4 Extended Exploration of Chosen Ciphertext Attack

Traditionally, a chosen ciphertext attack is an attack model for cryptanalysis where the cryptanalyst can gather information by obtaining the decryptions of chosen ciphertexts. From these pieces of information the adversary can attempt to recover the hidden secrets. Obviously, chosen ciphertext attack is not so widespread like chosen plaintext attack as access to a decryption oracle is much more difficult in practice. A possible decryption oracle may be a digital signature service, which may be easier to access than decryption services but its threat is still limited.

In an e-voting system a more practical and direct threat by chosen ciphertext attack is the so-called relation attack. In a relation attack, homomorphism of the employed encryption algorithm is exploited to re-encrypt an encrypted ballot into a related encrypted ballot such that a special relation exists between the contents of the two ballots. After the related ballot is decrypted into an invalid vote and recognised, the content of the original vote is discovered by reversing the relation between the two votes. A simple relation attack works as follows.

1. To trace a ciphertext c_i, an adversary computes one or more ciphertexts

encrypting messages related to the message encrypted in c_i, for example $c' = Re(c_i E(m))$ where m is chosen by the attacker. The ciphertexts are mixed with the normal and valid input ciphertexts and are shuffled together. They may appear as input ciphertexts to a mix network and then be shuffled by all the mixers or be inserted into the mix network by a dishonest mixer and replace some ciphertexts received by the mixer and then be shuffled by the following mixers.

2. The key idea of the attack is that there are some special relations between the message in c_i and the messages in the ciphertexts generated by the adversary. For example, $D(c') = D(c_i)m$ if ElGamal encryption is employed. The special relations are chosen by the adversary and can be recognized by him.

3. After the output messages of the mix network are published, the adversary recognises $D(c')$ and extracts the message in c_i as $D(c')/m$.

A.3 Security of Zero Knowledge Proofs

A.3.1 Introduction

In cryptographic applications like our e-voting system, it is often required to prove the knowledge of a secret satisfying a certain condition without revealing the secret (e.g. proof of knowledge or validity of vote, proof of correctness of decryption, proof of correctness of shuffling and designated digital signature). What is needed is the so-called ZK (zero knowledge) proof, which proves knowledge of secret integers in a certain language (or informally speaking, satisfying some certain properties) without revealing any information about the secret. We exploit homomorphism of the employed encryption algorithm and commitment algorithms to design our ZK proof primitive on the basis the commitment-challenge-response principle and the Fiat-Shamir heuristic, instead of using the related techniques like ZK-SNARK as explained in the following.

Succinct Non-Interactive Arguments of Knowledge (ZK-SNARK) gives cryptocurrency users the ability to hide all transaction data. ZK-SNARK requires the sender to produce a proof, in zero-knowledge, of the ability to spend an amount greater than or equal to the value of the transaction they are submitting. Zk-SNARK satisfy perfect completeness and computational soundness properties, in addition to the claimed property of succinctness, which simply means that the proof is polynomial in the security parameter.

The claimed succinctness actually refers to the verification side, while inefficiency on the proof side is an obvious drawback of ZK-SNARK. The proofs are currently computationally expensive to produce, taking several minutes to generate a key pair, and several more to generate a zero knowledge proof on a powerful desktop. The proofs are, however, very computationally efficient to verify, and when being applied to digital signature the signatures are very small in size --- 322 bytes for the signature, and “a few milliseconds” for the verifier’s computation.

The most famous application of ZK-SNARK is ZCash, which is a cryptocurrency attempting to address the lack of privacy in blockchain transaction through use of ZK-SNARK. ZCash uses ZK-SNARK over a Merkle tree that contains all previous transactions. Used naively, this could leak information about the latest transaction, as the transaction would be in the most recent set but not the previous one. To prevent this, the Merkle tree is of constant size, 264 --- for scale, if ZCash transactions occur at an average rate of 1000 transactions per second, it would take around 292 million years for the Merkle tree to be filled. ZCash is based on Zerocash which itself is an improvement of a protocol called Zerocoin, designed by a team of researchers at John Hopkins University in 2013. Zerocoin was designed to work atop the bitcoin blockchain system, with transaction origin hidden through a process involving minting and pouring of coins off the bitcoin blockchain and into zerocoins to obfuscate the otherwise transparent bitcoin transactions. Zerocoin was implemented as an extension to bitcoin, but was not commonly used, due to efficiency problems and imperfections of the scheme. Most importantly, using Zerocoin requires a 25kb proof to be produced for every coin spent, resulting in a total transaction size of 49kB. The transactions also only exist in single-denomination values, meaning many large proofs must be computed, and multiple transactions sent, in order to transfer a large number of zerocoins. In 2014, the scheme was extended and renamed Zerocash, a full cryptocurrency that used ZK-SNARK to protect the transaction value, sender and recipient address. ZCash has since been developed as an independent cryptocurrency with a further refined version of the Zerocash protocol.

We can see that users’ patterns and their required privacy in our e-voting system are quite different from those of ZCash users. What we need is the unlinkability between the voters’ identities and their votes while all the votes will be unsealed and tallied in the end. It is a one-time unlinkability and usually will not be reused. In comparison, ZCash requires a trusted set up stage, but after that the system is entirely anonymous. Due to the nature of the system and its use of zero-knowledge proofs, after the first transaction involving a coin, all coins are entirely anonymous and the blockchain is ‘opaque’ revealing nothing about senders, recipients, or transaction values.

Therefore, ZK-SNARK cannot be directly employed as the privacy guarantee in our e-voting scheme although it is the best way to make use of its advantages. An alternative way to employ ZK-SNARK is to replace our ZK proof protocols with it. Unfortunately, this idea is limited by low efficiency of ZK-SNARK. Even in ZCash, compared with other blockchain transactions, the proofs and resulting transactions of ZK-SNARK are very costly and there are stages which take a considerable amount of time. When being applied to the ZK proof in e-voting to implement the ZK proof operations in vote validity check and the employed mix network, the computational and communicational costs are even higher, especially in a large-scale election with a big number of voters.

In addition, there is another drawback: all of the schemes based on ZK-SNARK require a trusted setup stage, without which the secret information used in the construction of the system can be used dishonestly to fake transactions and create new, counterfeit transactions like coins. To avoid a trust on a single party, which is unacceptable in secure applications like e-voting, A multi-party setup is needed to share the trust among multiple parties and efficiency will be further compromised.

Our conclusion is that although blockchain is the best platform for our e-voting scheme, the security mechanisms of the blockchain-based cryptocurrencies cannot be always inherited in our application. Their privacy and anonymity requirements usually focus on untraceability of re-usable crypto coins and the related transactions, while we are concerned about linkability of one-time votes to their voters. So, the best tools to implement privacy in the two kinds of applications may be different. Moreover, efficiency issues affect them in different aspects and some necessary costs of one kind applications may be intolerable to the other.

A character of the operations and their proof of validity is that all of them are carried out in a batch. Each voter casts a vote in the same format, encrypted by the same encryption algorithm, processed by the same authorities in the same procedure and tallied together. All the operations in the same batch are proved and verified to be valid using the same ZK proof and verification protocols. So, efficiency of the e-voting schemes can be improved by batch proof and verification, which may be inappropriate for the proof and verification protocols to guarantee untraceability of crypto coins in the cryptocurrency applications of blockchains.

Suppose ElGamal encryption algorithm is employed, the following ZK proof primitives are needed in our e-voting system:

- ZK proof of knowledge of discrete logarithm
- ZK proof of equality of logarithms
- ZK proof of multiple pairs of equal logarithms in two products
- ZK proof of knowledge of 1-out-of-k logarithms
- ZK proof of 1-out-of-k equality of logarithms

If Paillier encryption is employed, the needed ZK proof primitive is ZK proof of knowledge of root. In all the needed ZK proof primitives, not only the existence of the secret in the certain language (to satisfy the required properties) needs to be proved but also the prover's knowledge of the secret must be guaranteed.

A.3.2 Three-Move Proof

The needed ZK proof protocols in our e-voting system are built on some basic ZK proof primitives in the form of three-move proof. A three-move proof of relation $R:\{0,1\}^* \rightarrow \{0,1\}^*$ is as follows. Two values x and w are in relation $R(x,w)$ where finding w

from x is hard. A prover holds w as a secret while x is public. The prover wants to prove the relation $R(x,w)$ to a verifier without revealing x. In the first move, the prover chooses a secret random integer, commits to it and sends the commitment to the verifier. In the second move the verifier sends a t-bit challenge to the prover. In the third move, the prover sends a response (generated from w and the secret random value in Move One) to the verifier. The verifier can use the commitment, the challenge and the response to verify the validity of the proof.

A three-move protocol for relation $R(x,w)$ is secure if the following three conditions are satisfied:

1. **Comleteness**: If the prover and verifier follow the protocol, the verifier always accepts. In this thesis, this property is called correctness.
2. **Special soundness**: If two conversations with a unique commitment and two different challenges can pass the verification, w can be calculated efficiently from the two conversations.
3. **Honest-verifier zero-knowledge**: There exists a polynomial-time extractor to generate a simulated valid conversation with a same distribution as a conversation between an honest prover and an honest verifier.

A stronger and stricter kind of soundness can be defined using the following definition, although in practice special soundness is easier to understand and prove. It is enough to guarantee that a prover without knowledge of w can pass the verification with a t-bit challenge with a probability no more than 2^{-t}.

Definition 10 *Let* $\kappa()$ *be a function from bit strings to the interval* $[0,1]$. *A protocol is said to be a proof of knowledge for the relation* R *with knowledge erroe* κ*, if the following are satisfied:*

- **Completeness** On common input x, if the honest prover gets as private input w such that $(x,w) \in R$, then the verifier always accepts.
- **Knowledge soundness** There exists a probabilistic algorithm M called the knowledge extractor. This M gets input x and rewindable black-box access to the prover and attempts to compute w such that $(x,w) \in R$. We require that the following holds: For any prover, let $\varepsilon(x)$ be the probability that the verifier accepts on input x. There exists a constant c such that whenever $\varepsilon(x) > \kappa(x)$, M will output a correct w in expected time at most $\langle x \rangle^c / (\varepsilon(x) - \kappa(x))$ where access to the prover counts as one step only.

Several common ZK proof primitives frequently employed in secure applications like e-voting are detailed in Appendix 13

A.3.3 ZK Proof in E-voting

The employed proof primitives are correct (complete) and specially sound and verifier honest. So, the probability that a prover without knowledge of a correct secret can pass the verification in the proof protocols is negligible (1 out of the numbers of all possible challenges) and the prover's secret knowledge is not revealed. Therefore, they can be employed in our e-voting system to achieve our desired security properties.

In e-voting schemes, it is impossible for every different verifier to interactively communicate with the voters and verify their real-time ZK proofs. Instead, non-interactive transferable proof and verification protocols are usually needed. Using the well known Fiat-Shamir heuristic, a hash function can be employed to generate the challenge, so that the protocol becomes non-interactive. In the rest of our work, unless otherwise specified the proofs are non-interactive when applied. If the hash function can be seen as a random oracle, security is not compromised in the non-interactive proof. When ZK proof protocols must be non-interactive but not transferable, ZK proof of knowledge of 1-out-of-2 secrets (e.g. ZK proof of knowledge of 1-out-of-2 logarithms) can be employed to build designated proof like in the designated digital signature schemes.

In our e-voting application, the non-inerative versions of the ZK proof primitives are employed in proofs of validity of various operations. When ElGamal encryption is used, the first two primitives are necessary. For example, proof of knowledge of his selection in an encrypted ballot by a voter is a ZK proof of equality of logarithms, while proof of valid shuffling by a mixer is a multiple-round[1] combination of ZK proof of multiple pairs of equal logarithms in Two Products and other proof primitives. In homomorphic e-voting, ZK Proof of knowledge of 1-out-of-k logarithms, ZK proof of 1-out-of-k equality of logarithms or their optimisation is needed to prove validity of ballots. Designated digital signature needed in e-voting can be implemented through ZK proof of knowledge of 1-out-of-2 secrets. When RSA or Paillier encryption is employed, ZK proof of knowledge of root is the basic modular to build the proof protocols. Proof of validity of decryption in distributed threshold version of any of the three encryption algorithms depends on ZK proof of equality of logarithms.

A.4 Interactive Device Challenge Verification

A.4.1 Background

The original idea of Benaloh verification comes from a paper by Benaloh in 2006 [5], which is a proposal of mix network. In a mix network, a few mixers each carries out a shuffling to re-encrypt and permute some ciphertexts. More precisely, in a shuffling

[1] sometimes larger than 3 rounds is needed.

input ciphertexts $c_1, c_2, \dots, c_n$ are shuffled to $c'_i = ReEn(c_{\pi(i)}, r_i)$ for $i = 1,2,\dots,n$ where $ReEn()$ stands for re-encryption and $\pi()$ is a random permutation of $\{1,2,\dots,n\}$. To prove validity of shuffling, Benaloh employs a special cut-and-choose challenge. The mixer carries out k instances of additional shuffling and generates k groups of intermediary re-encrypted and permuted ciphertexts. Then the set $S = \{1,2,\dots,n\}$ is randomly divided into two subsets S_1 and S_2. The mixer has to show that each group of ciphertexts in S_1 is an re-encryption and permutation of $c_1, c_2, \dots, c_n$ and each group of ciphertexts in S_2 can be re-encrypted and permuted to $c'_1, c'_2, \dots, c'_n$. Each shuffling in Benaloh's mix network is as follows.

1. The mixer carries out k instances of additional shuffling such that

   $$c'_{j,i} = Re(c_{\pi_j(i)}, r_{j,i}) \quad for \quad j = 1,2,\dots,k \quad and \quad i = 1,2,\dots,n \tag{11}$$

   where $\pi_j()$ is a random permutation of $\{1,2,\dots,n\}$ for $j = 1,2,\dots,k$.

2. A random subset of $S = \{1,2,\dots,n\}$ is chosen and denoted as S_1 such that the mixer publishes $\pi_j()$ and $r_{j,i}$ for $j \in S_1$ and $j = 1,2,\dots,k$. Anyone can publicly verify that

   $$c'_{j,i} = Re(c_{\pi_j(i)}, r_{j,i}) \quad for \quad j \in S_1 \quad and \quad i = 1,2,\dots,n \tag{12}$$

3. For $S_2 = S - S_1$, the mixer publishes permutation $\pi'_j() = \pi(\pi_j^{-1}())$ for $j \in S_2$ and $r'_{j,i} = r_i ! r_{j,i}$ for $j \in S_2$ and $j = 1,2,\dots,k$ where $!$ stands for subtraction in the case of ElGamal encryption or division in the case of Paillier encryption. Anyone can publicly verify that

   $$c'_i = Re(c'_{j,\pi'_j(i)}, r'_{j,i}) \quad for \quad j \in S_2 \quad and \quad i = 1,2,\dots,n \tag{13}$$

Soundness of the shuffling depends on the difficulty for a cheating mixer to guess in advance what S_1 and S_2 will be such that he can re-encrypt and permute $c_1, c_2, \dots, c_n$ into $c_{j,1}, c_{j,2}, \dots, c_{j,n}$ for $j \in S_1$ and $c'_1, c'_2, \dots, c'_n$ into $c_{j,1}, c_{j,2}, \dots, c_{j,n}$ for $j \in S_2$ but $c'_1, c'_2, \dots, c'_n$ is not a shuffling of $c_1, c_2, \dots, c_n$.

A.4.2 The General Principle and Our Application

How difficult is it for a cheating mixer to respond correctly to Benaloh's challenge (e.g. in the Benaloh mix network)? The probability of success is one out of the number of possible divisions of S into S_1 and S_2, so it is 2^{-k}. When $k = 20$, the probability is smaller than 1 out of one million; when $k = 30$, the probability is smaller than 1 out of one billion.

With a k not too large, the probability of successful cheating can be small enough for practical applications. So, Benaloh's special cut-and-choose idea may be adopted in many other applications than mix network to detect and prevent cheating. However, there is a condition for the idea to work: there must be multiple, say k, verifiable intermediary statuses between the initial input and the desired output so that both the operation from the initial input to any intermediary status and the operation from any intermediary status to the final output can be verified. Moreover, verification of the operation from the initial input to any intermediary status or the operation from any intermediary status to the final output will not reveal the opertion from the initial input to the final output.

We need to be aware that this condition is not always satisfied in any circumstance. For example, it is satisfied and so Benaloh verification can work in re-encryption mix network as re-encryption is malleable and can produce unlimited instances of intermediary re-encryption. In comparison, in a decryption mix network based on a symmetric-cipher encryption chain, it is hard to adopt Benaloh verification as it is difficult to find any verifiable intermediary status for a block-cipher (e.g AES) decryption operation.

In our e-voting system, what we need to verify is the client agent whose source and behaviour must be checked and accepted by the users without any computational capability by hand. If Benaloh verification is employed, we need to consider what the intermediary statuses are. Another application similar to ours is how to verify the digital code in a smart contract in a block chain system so that the public used to physical contract can accept it. More precisely, The current physical world has to adapt to the contract code amicably with human consensus. Currently, there are three solutions to this challenge — multi-signature transactions, prediction markets and oracles, each of which has its own drawbacks. Benaloh verification has advantages over those methods when verifying an operation/behaviour.

A.4.3 Application to Range Proof

In the Section of range proof, to prove that an integer m is in a range $\{a, a+1, \dots, b\}$, it is committed to in $c = g^m h^r \mathrm{mod} N$ where N is a large composite. However, as ElGamal encryption is employed, m i actually encrypted into $e = (a, b) = (f^t \mathrm{mod} p, m y^t \mathrm{mod} p)$ where p is a large prime. So, we need to prove that the same m is committed to in c and encrypted in e. The proof can be implemented using the following Benaloh verification.

1. The prover publishes $c_i = g^{m_i} h^{r_i} \mathrm{mod} N$ and $e_i = (a_i, b_i) = (f^{t_i} \mathrm{mod} p, m_i y^{t_i} \mathrm{mod} p)$ for $i = 1,2, \dots, k$ where m_i, r_i, t_i are raondom integers.

2. A random subset of S is chosen as S_1 where $S = \{1,2, \dots, k\}$. The prover publishes m_i, r_i, t_i for $i \in S_1$ and any one can verify $c_i = g^{m_i} h^{r_i} \mathrm{mod} N$

and $e_i = (a_i, b_i) = (f^{t_i} \mathrm{mod} p, m_i y^{t_i} \mathrm{mod} p)$ for $i \in S_1$.

3. For $S_2 = S - S_1$, the prover publishes $C_i = c^{m_i} \mathrm{mod} N$ and use ZK proof of knowledge of discrete logarithm and ZK proof of quality of discrete logarithms to prove his knowledge of m_i, r_i to satisfy $C_i = c^{m_i} \mathrm{mod} N$ and $c_i = g^{m_i} h^{r_i} \mathrm{mod} N$ for $i \in S_2$.

4. The prover publishes $M_i = mm_i$, $R_i = rm_i$ and $T_i = tt_i$ for $i \in S_2$. Any one can verify that $C_i = g^{M_i} h^{R_i}$ and $(aa_i, bb_i) = (f^{T_i} \mathrm{mod} p, M_i y^{T_i} \mathrm{mod} p)$ for $i \in S_2$.

The probability for unmatched c and e to pass the proof and verification is 2^{-k} while the computational cost is linear in k. As a range proof of the message in c has a constant cost independent of the range size, with the help of Belanoh verification, vote validity proof in our e-voting scheme has a cost independent of the range size. When k is 40, the probability of error is less than one out of one trillion. Therefore, efficiency of vote validity test is not compromised, especially when the range is large.

A.4.4 Open Question

Besides other verification applications of Benaloh challenge, mix network is a focus of our research too. We are exploring is the possibility of a more advanced re-encryption mix network scheme using Benaloh verification. Of course we will not use the original mix network by Benaloh due to its following drawbacks in efficiency:

- k additional re-encryption operations cost $O(k)$ exponentiations. Moreover, all the exponentiations are separate with full-length exponents at least hundreds of bits long for the sake of encryption privacy so that efficiency improvement is not easy.
- The additional re-encryption operations cannot be carried out in advance to save the real-time computation.
- Batch verification cannot be effectively applied to the verification of re-encryption for the purpose of efficiency optimisation.

Our target is to use the idea of Benaloh verification more flexibly such that the efficiency improving tools including batch verification, pre-computation and exponentiation optimisation can be effectively applied to greatly improved the performance of mix network. Especially, we notice that exponents scores of bits long are strong enough for practical soundness and we will try to use much shorter exponents than those used by Benaloh without compromising security in the real world. We will make most of the costly computations pre-computable to reduce the real-time overhead. We may even extend batch verification to batch proof and verification. With all the new techniques employed, a Benaloh-verified mix network much more efficiency can be designed with practically strong security.

A.5 Security of Public Key Infrastructure

Whenever encryption is needed, there exists the question of key distribution. To distribute the keys of symmetric ciphers, asymmetric (public key) encryption is employed, which uses a pair of keys, a public key and a private key. To be workable, a public key encryption algorithm needs to distribute its own public key as well. Another important cryptographic technique needing public key distribution is digital signature, where a signer has to distribute its public key so that the verifiers can use the public key to verify validity of his digital signatures.

Public key distribution is implemented through a key certificate by an authority. To authenticate the certificate, the authority needs to sign the certificate it issues digitally using its own private key. For others to verify the signature, the authority's public key needs to be distributed too. So, the authority needs a higher-level authority to certify its key in the same way. Therefore, a certification tree containing all the authority is built up, such that every authority depends on its father node for certification and the root node is the highest authority independent of the certification system. This is the background and motivation of PKI (public key infrastructure).

In our e-voting system, both encryption and digital signature are needed and thus PKI is necessary. Although identity-based key generation mechanism free of public key distribution has been proposed, it is a special-purposed technique liable to many limitations and not suitable for our e-voting application. Moreover, vote privacy and voter anonymity required in the e-voting system (through mix network and pseudonym), we have special privacy requirement on PKI. We call it privacy preserved PKI.

Privacy infringement problems are widespread in today's computing environments and grow as the number of networking devices increases over the Internet. There is a strong demand for privacy-preserving policies and mechanisms, while abuse based on anonymity and user-controlled pseudonyms is another serious problem. To cope with these conflicting challenges, a number of methods have been developed. In e-voting, privacy-preserving techniques that provide unlinkability of pseudonyms is a great concerns for privacy preservation. For prompt deployment in legacy systems, it would be desirable to develop privacy-preservation methods in the standard paradigm.

During the past decade, Public Key Infrastructures (PKIs) has been standardized and deployed world-wide to support various kinds of communication sessions and electronic transactions over the Internet. A PKI plays an important role in asserting the ownership of keys for users and SPs on the Internet, but not for their privacy. A public key and corresponding data should be signed by an authorized entity according to the current standard, X.509, so that the signed one can be accepted as an identity certificate.

Authorization attributes are signed to yield an attribute certificate that will be used for authorization.

These standard paradigms are far from preserving privacy. The signed certificate should be publicized by the authority, for example, in a directory system, by disclosing the user information in an "authentic" manner. An anonymized certificate, saying that the true identity is not entered in a subject field, could enhance privacy to some extent. However, an authorized issuer may become, from the view of privacy concerns, a big brother even unintentionally since he can always trace the registered user according to the certificate. What we mean by "trace" is that all pseudonymous or anonymous transactions are bound explicitly with corresponding principals' true identities, and vice versa. An anonymized certificate should be made traceable and revocable "conditionally" by distributed authorities, not by a big brother. In the belief that PKIs will continue to play its significant roles, there have been practical approaches for enhancing privacy in those paradigms, and also several approaches for borrowing them in even more complex unlinkable schemes.

We have suggested to adopt the X.509 standard in our e-voting system. However, we still need to take into account an important fact: our network environment is a decentralized blockchain network. We notice that some decentralized PKI has been implemented in blockchains like Blockstack[1] based on Bitcoin blockchain and Certcoin [12] based on Namecoin [19] blockchain. They in general employ the idea in [14] to use either Namecoin or Bitcoin to issue identity credentials in settings such as anonymous peer-to-peer networks, where one does not have trusted credential issuers. Their ideal environment setting is an e-currency or e-payment system, where there is no trusted central authority and privacy and anonymity of the users is absolute and self-managed. Slightly different proposal like [13] distributes the process of certificate issuance to not only provide transparency into the process but also prevent misbehavior in the first place. In this sense, they resemble the so-called web of trust technique, where anybody in the web can certify other party's identity and public key.

Robustness of decentralized PKL was discussed in[4], but its comprehensive security and vulnerability has not been systematically analysed. Apart from the potential security concerns, we are not convinced of applicability of decentralized PKI to our e-voting scheme. Although those decentralized PKI protocols seem more inherently consistent with blockchains, they are not suitable for e-voting applications. In an election, every vote must be authenticated, verifiable and accounted for and the self-managed-by-participant pattern is not appropriate. Moreover, unlike the decentralized peer-to-peer payment applications, there are central authorities in elections (at least in political voting applications) to manage the election and authenticate the voters. The authentication by other peers in the web-of-trust model is not official and trustworthy enough, not to mention the multiple-authenticators-per-user mechanism complicated the trust structure and identity management.

So, central PKI managed by some election authorities and public key certificates signed by them seem to be a better solution, although they do not inherently exist in a blockchain system. As explained before, we do not embed our e-voting scheme into a blockchain network but run it on the top of a blockchain platform. Of course, since blockchain is our platform to run the election, we will not ignore its influence. For example, the choice of blockchain is an interesting question. While using an application designed blockchain can provide greater flexibility, they have less mining power than Bitcoin, which can have security implications for schemes built on top of them, see the security analysis of the Namecoin blockchain in [2].

A.5.1 X.509 standard

A Public Key Infrastructure (PKI) is considered one of the most important techniques used to propagate trust in authentication over the Internet. This technology is based on a trust model defined by the original X.509 (1988) standard and is composed of three entities: the certification authority (CA), the certificate holder (or subject), and the Relying Party (RP). The CA plays the role of a trusted third party between the certificate holder and the RP. In many use cases, this trust model has worked successfully.

X.509 is an ITU-T standard for PKI/PMI, which specifies standard formats for public key certificates, certificate revocation lists, attribute certificates, and certification path validation algorithms. Currently, X.509 is rather implying the standard documented by the PKIX (PKI based on X.509) working group of the IETF and the PKCS (Public Key Cryptography Standards) developed by RSA Security, Inc. Within the framework of the X.509 standard, a Certificate Authority (CA) is an entity that issues digital certificates, while a Registration Authority (RA) is manipulates users' requests and their accounts. There are many commercial CAs that charge for their services over the world, and also many free CAs.

Version One of X.509 was established by the ITU in November of 1988. At this time the focus of the standard was directory services authentication and did not include any mechanisms to obtain the public key or revocation information. Version Two was proposed Nov 1993 but short lived. The key change from X.509v1 to X.509v2 was the improved flexibility in identifying the issuer and the subject.? Version 3 of the X.509 certificate is the most common format found today and can satisfy our PKI application. In this version of the certificate a single field was added that allows the extensibility required to support multiple applications and is likely the reason that this version is still in use today.

There are two kinds of useful digital certificates, such as an identity (or public key) certificate and an attribute (or authorization) certificate, both of which are standardized by X.509. The former binds a public key to a unique identity of its owner in PKIs, while the latter binds multiple attributes of permissions to its holder for authorization purposes and is not our focus. An X.509 v3 (version 3) certificate contains its version, serial number, certificate signature algorithm, issuer's name, validity period, subject's name, subject's public key information, options including extensions, and certificate signature itself. An attribute certificate replaces subject and subject's public key information fields with its holder information and attributes of permissions. In today's environments, there are already a number of applications and protocols that support the X.509 standard certificates, such as SSL/TLS, S/MIME, IPsec, SSH, EAP, LDAP, and even P2P.

An common example is SSL/TLS x.509 Certificate. An SSL Certificate is a digital computer file that has two specific functions. The first is authentication and verification: the SSL Certificate has information about the authenticity of details around the identity of a host or site. When you click on the padlock displayed or check the trust mark the certificate chain details prove where the certificate is generated from. The second is data encryption: The SSL Certificate enables encryption, which means that the sensitive information exchanged via the web site cannot be intercepted and read by anyone other than the intended recipient. An SSL Certificate is most reliable when issued by a trusted Certificate Authority (CA). The CA has to follow very strict rules and policies about who may or may not receive an SSL Certificate. So, when you have a valid SSL Certificate from a trusted CA, there is a higher degree of trust.

X.509 certificate files are written using Abstract Syntax Notation (ASN.1) and require a converter to become human-readable.?The most common encoding of individual certificates are DER encoded binary x.509 or Base-64 encoded X.509, or Public Key Cryptography Standard ($PKCS$) #7 formats (see RFC 2315 http://tools.ietf.org/html/rfc2315 and RFC 5652 http://tools.ietf.org/html/rfc5652).? $PKCS$ #7 also enables the storage of multiple certificates in a single file and can be useful to establish trusts with non-public certification authorities where the AIA path may not be accessible.?

In our e-voting application, X.509 is our implementation of PKI to support key distribution for encryption and digital signature. An election authority will act as a cross-signing CA to issue public key certificates and certify node permissions.

B Mix Networks

B.1 Details of the Implementation of the Mix Network

The details are as follows.

1. $S \longrightarrow V$: $\alpha_i = a'^{v_i}_i g^{x_i} \bmod p,$ $\beta_i = b'^{v_i}_i y^{x_i} \bmod p$ for $i = 1, 2, \dots, n$

$$a = g^{w_1} \bmod p, \qquad b = y^{w_1} \bmod p$$
$$\alpha'' = (\textstyle\prod_{i=1}^{n}(a''^{v_i}_i)/g^{w_2} \bmod p, \qquad \beta'' = (\textstyle\prod_{i=1}^{n}(b''^{v_i}_i)/y^{w_2} \bmod p$$

where $w_1, w_2 \in_R Z_q$ and $v_i, x_i \in_R Z_q$ for $i = 1, 2, \dots, n$.

2. $V \longrightarrow S$: $c \in_R Z_{2^L}$

3. $S \longrightarrow V$: $\theta_i = x_i + r'_i c \bmod q$ for $i = 1, 2, \dots, n$

$$z_1 = w_1 + R_1 c \bmod q, \qquad z_2 = w_2 + R_2 c \bmod q$$
$$\gamma_i = v_i + t_i c \bmod q \qquad \text{for } i = 1, 2, \dots, n,$$

Anyone can verify that

$$\alpha_i a''^{c}_i = a'^{\gamma_i}_i g^{\theta_i} \bmod p, \qquad \beta_i b''^{c}_i = b'^{\gamma_i}_i y^{\theta_i} \bmod p \qquad \text{for } i = 1, 2, \dots, n \quad (14)$$
$$aA_1^c = g^{z_1} \bmod p, \qquad bB_1^c = y^{z_1} \bmod p \quad (15)$$
$$\alpha'' A_2^c g^{z_2} = \textstyle\prod_{i=1}^{n} a''^{\gamma_i}_i \bmod p, \qquad \beta'' B_2^c y^{z_2} = \textstyle\prod_{i=1}^{n} b''^{\gamma_i}_i \bmod p \quad (16)$$

where $A_1 = \prod_{i=1}^{n} a''_i / \prod_{i=1}^{n} a_i^{s_i} \bmod p$, $B_1 = \prod_{i=1}^{n} b''_i / \prod_{i=1}^{n} b_i^{s_i} \bmod p$, $A_2 = \prod_{i=1}^{n} a_i^{s_i^2} \bmod p$ and $B_2 = \prod_{i=1}^{n} b_i^{s_i^2} \bmod p$.

Fig. 16. Public proof and verification

B.2 General or Separate Verification

According to how correctness is verified, mix networks can be classified into GMN (general verification) and SMN (separate verification).

GMN does not provide a verification of correct shuffling by each server separately. Instead, correctness of the shuffling by the whole mix network is verified after the outputs are produced in plaintext. GMN is illustrated in Figure 12.

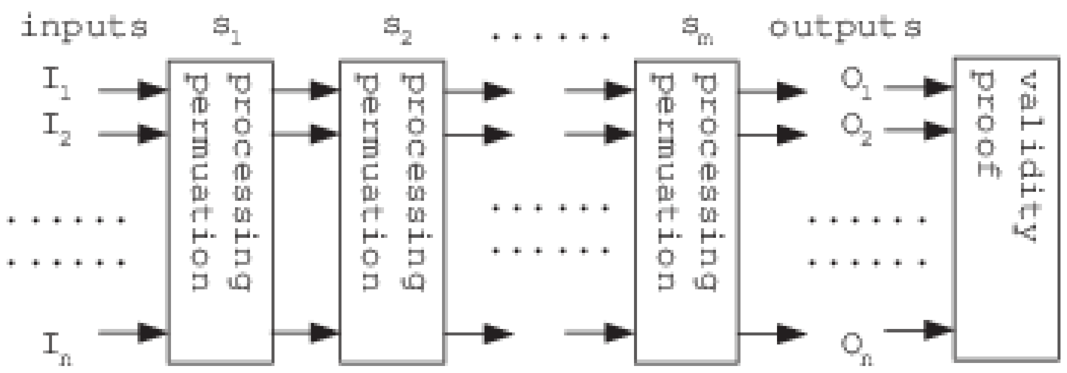


Figure 12: General Verification Mix Network

An advantage of GMN is that schemes in this category are usually very efficient as only one final verification for shuffling validity is needed. A key drawback of this category is that a malicious server and its invalid shuffling cannot be found instantly.

In SMN, each server proves that his shuffling is correct. So whenever incorrect shuffling is performed, it is detected and the mixing stops instantly. Therefore no plaintext is obtained if the mixing is not correct. SMN is illustrated in Figure 13

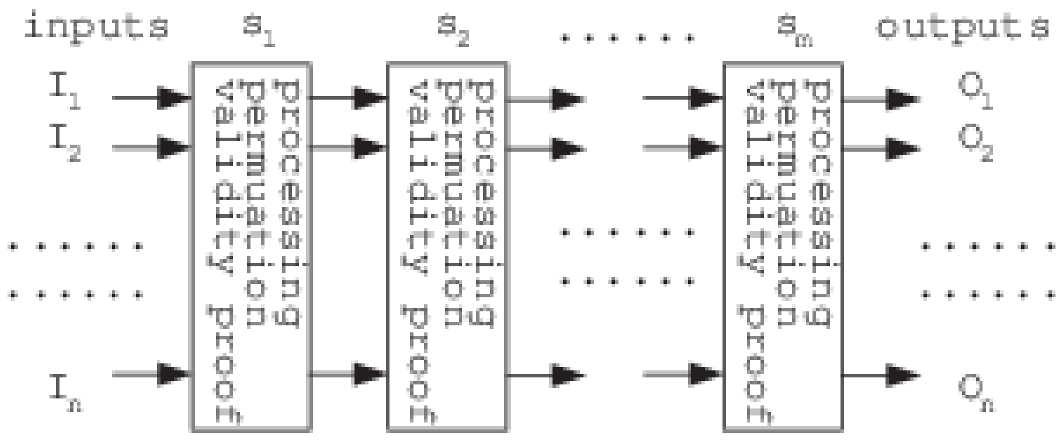


Figure 13: Separate Verification Mix Network

The most serious problem in this approach is inefficiency. Our first choice is re-encryption mix network with separate verification. However, decryption mix network with general verification is also studies in case it is needed for the sake of efficiency or resistence against quantum attacks. A design is proposed in Appendix 10 as an alternative.

B.3 Decryption Mix Network with General Verification

General verification is specially suitable for decryption mix network. In this section a new mix network is designed to implement efficient vote shuffling, which is composed of two rounds. Each voter commits his inputs first and the commitments are mixed in the first round. Any voter can easily verify that his commitment is shuffled correctly and protest if the verification fails. Any protest can be publicly verified and the malicious mixer can be identified and removed. The inputs are mixed in the second round. After the outputs are produced, they can be verified publicly against the commitments published in the first round. Both decryption chain and re-encryption can be applied to implement the shuffling. In the following, the protocol is described in detail in the case of decryption mix network. The employed encryption algorithm can be any semantically secure encryption algorithm like ElGamal or Paillier.

1. Encryption key(s) are generated. If public key encryption is employed, the same public key is used to encrypt all the votes and the corresponding private key are shared by the mixers. If secret key encryption is employed, a different key is shared between every voter and every mixer and each vote is encrypted by all the keys shared between the corresponding voter and all the mixers.
2. Each voter P_i chooses an input b_i and commits to it as $c_i = H(b_i, v_i)$ where $H()$ is a one-way and collision-resistant hash function and v_i is a random number.
3. Commitments c_i for $i = 1,2, \ldots, n$ are encrypted to e_i for $i = 1,2, \ldots, n$, which are signed by the corresponding voters and submitted to the mix network.
4. Each mixer in the mix network partially decrypts and permutes e_i for $i = 1,2, \ldots, n$ in sequence. The mixers do not need to prove correctness of their shuffling.
5. Let c'_i for $i = 1,2, \ldots, n$ denote the outputs of the mix network.
6. Each voter P_i verifies that his commitment c_i is among the published shuffled commitments. Any voter can complain if he cannot find his commitment in the published outputs. Any dispute is solved as follows.
   (a) The protesting voter P_i publishes e_i and his signature on e_i.
   (b) Each mixer (from the first one to the last one) has to prove his shuffling of e_i is correct by publishing his output for e_i and his decryption operations of e_i.
   (c) If any mixer fails to prove his innocence, he is removed (or replaced if necessary) and the mixing is performed again. If all the mixers prove their innocence, the protesting voter is identified as a cheater and removed.

7. The pair of input (b_i, v_i) is encrypted to the ciphertext pair $(e1_i, e2_i)$ and signed by P_i for $i = 1,2, \ldots, n$.
8. The encrypted inputs $(e1_i, e2_i)$ for $i = 1,2, \ldots, n$ are submitted to the mixers, who perform the second round of mixing to shuffle them. As in the first round, shuffling in the second round is also composed of partial decryption and permutation and not proved by the mixers to be valid. Ciphertext pair $(e1_i, e2_i)$ for $i = 1,2, \ldots, n$ are shuffled to (b'_i, v'_i) for $i = 1,2, \ldots n$.
9. Anyone can publicly verify that every published output is correctly committed to some commitment. If an output (b'_i, v'_i) of the second round cannot be matched to any published commitments c'_j for $j = 1,2, \ldots, n$, the problem is solved as follows.
   (a) Each mixer (from the last one to the first one) has to prove his shuffling leading to $(e1'_j, e2'_j)$ is correct by publishing his input for $(e1'_j, e2'_j)$ and his operations on it. If any mixer has performed an incorrect shuffling, he will be discovered.
   (b) If any mixer fails to prove his innocence, he is removed (or replaced if

necessary) and the mixing is performed again. If all the mixers prove their innocence, the output in dispute has been traced back to an input of the mix network. The voter submitting the input in dispute is identified as a cheater and removed.

B.4 Preliminaries

ElGamal encryption as follows and the symbols and glossaries in Table 1 are used in this paper.

L	a security parameter, such that $2^L < q$ where q is defined in the ElGamal setting
M'	the transpose matrix of a matrix M
Permutation matrix	a matrix with exactly one 1 in every row and exactly one 1 in every column and zeros in other positions
$x \in_R S$	integer x is randomly chosen from a set S.
$P[e]$	the probability of event e
$P[e_1 \| e_2]$	the probability of event e_1 when event e_2 has happened
$ST\ (\ x_1, x_2, \dots, x_k \mid f_1, f_2, \dots, f_l\ )$	a statement of knowledge of $x_1, x_2, \dots, x_k$ such that $f_i(x_1, x_2, \dots, x_k) = 0$ for $i = 1, 2, \dots, l$.

Table 1 - Symbols and glossaries

- G is a cyclic subgroup of Z_p^* with prime order q and g is a generator of G. q should be large enough in the current security standard (e.g. 1024 bits long).

- Private key is an integer x, which is generated by multiple distributed DAs (decryption authorities) as described in our design of distributed key generation protocol, such that no single party knows the private key and private key recovery and decryption are only possible when the number of cooperating decryption authorities is over a sharing threshold. Public key is y, which is equal to $g^x \bmod p$.

- Encryption: a message m in G is encrypted into $(g^r \bmod p, my^r \bmod p)$ where $r \in_R Z_q$.

- Re-encryption: a ciphertext (a, b) is re-encrypted into $(g^r a \bmod p, y^r b \bmod p)$ where $r \in_R Z_q$.

- Decryption: a ciphertext $c = (a, b)$ is decrypted into $D(c) = \frac{b}{a^x} \bmod p$ by the DAs.

- Multiplication: $c_1 c_2 \bmod p = (a_1 a_2 \bmod p, b_1 b_2 \bmod p)$ where $c_1 = (a_1, b_1)$

and $c_2 = (a_2, b_2)$.

Note that with the ElGamal parameter setting in this paper (where q is large) it is easy to map messages in Z_q into G in an invertible manner. For example, when $p - 1 = 2q$, a message m in Z_q is mapped to $-m \bmod p$ before being encrypted if it is not a quadratic residue and unchanged otherwise. As it is impossible for m and $-m \mathrm{mod} p$ to be in Z_q simultaneously, there is no collision in this mapping. After a ciphertext is decrypted into m, m is mapped to $-m \bmod p$ if it is not in Z_q and unchanged otherwise. Soundness of the new shuffling scheme in the new setting is based on an assumption called multiple discrete logarithm assumption.

Definition 1 *A logarithm relation of $m_1, m_2, \ldots, m_n$, all in G, is found if non-negative integers $l_1, l_2, \ldots, l_n$, not all zero (modulo q if q is known), are found such that $\prod_{i=1}^{n} m_i^{l_i} = 1 \; mod \; p$. Multiple discrete logarithm assumption on a polynomial mixer with respect to $m_1, m_2, \ldots, m_n$ states that the mixcer cannot find a logarithm relation of $m_1, m_2, \ldots, m_n$ with a non-negligible probability.*

When $m_1, m_2, \ldots, m_n$ are randomly chosen from G, multiple discrete logarithm assumption is satisfied as DL problem is hard. In this case, even if an adversary knows $m_1, m_2, \ldots, m_n$, it is still difficult for him to break multiple discrete logarithm assumption, not to mention without the private key it is hard for any mixer to know any information about the encrypted messages. As illustrated later in Section 4.9.6, even if $m_1, m_2, \ldots, m_n$ are not randomly chosen multiple discrete logarithm assumption can still be satisfied in a special setting based on two widely accepted hard problems, factorization problem and DDH problem.

B.5 Efficiently Handling Security Assumption and Further Optimisation

As stated before, when the input messages are randomly chosen multiple-discrete-logarithm assumption with respect to the shuffled messages is satisfied if the discrete logarithm problem is hard. However, sometimes the input messages are not randomly chosen. In some circumstances a voter may even collude with the shuffling node, submitting carefully chosen input messages and revealing them to the shuffling node. In this case, multiple-discrete-logarithm assumption with regard to the shuffled messages cannot be satisfied in the proposed shuffling protocol in general. There are a few solutions to this problem. A simple measure is to randomly divide the input ciphertexts $c_1, c_2, \ldots, c_n$ into $c'_1, c'_2, \ldots, c'_n$ and $c''_1, c''_2, \ldots, c''_n$ such that $c_i = c'_i c''_i$ and shuffle the two sets of ciphertexts using the same permutation. On one hand, shuffling of two groups of ciphertexts using the same permutation can be easily implemented as Theorem can be extended to guarantee the same permutation. On the other hand, the two groups of output ciphertexts can be combined to recover the shuffled ciphertexts.

C Details of the ZK Protocols

The following seven proof protocols are complete (a prover with knowledge of the logarithm can always pass the verification) and sound (the probability to pass the verification is negligible if the prover does not know the logarithm).

C.1 Proof of Knowledge of Logarithm

Suppose G is a multiplicative cyclic groups of order q where the modulus in multiplication is p. Let g be a generator of G and y a member in G. Below is a protocol to prove knowledge of $\log_g y$ without revealing x.

1. The prover chooses randomly $r \in Z_q$ and calculates $a = g^r \bmod p$ and sends it to the verifier.
2. Then the verifier chooses a challenge c and sends it to the prover.
3. The prover calculates $s = r - cx \bmod q$ and sends it to the verifier.
4. The verifier checks $a = g^s y^c \bmod p$. The proof is accepted only when this verification is passed.

C.2 ZK Proof of Equality of Logarithms

Suppose G is a multiplicative cyclic groups of order q where the modulus in multiplication is p. Let g_1, g_2 be two generators of G. $Y_1 = g_1^x$ and $Y_2 = g_2^x$ while $x \in Z_q$. Below is a protocol to prove $\log_{g_1} Y_1 = \log_{g_2} Y_2$ without revealing x.

1. The prover chooses randomly $r \in Z_q$ and calculates $a = g_1^r \bmod p$, $b = g_2^r \bmod p$ and sends them to the verifier.
2. Then the verifier chooses a challenge c and sends it to the prover.
3. The prover calculates $s = r - cx \bmod q$ and sends it to the verifier.
4. The verifier checks $a = g_1^s Y_1^c \bmod p$ and $b = g_2^s Y_2^c \bmod p$. The proof is accepted only when this verification is passed.

C.3 ZK Proof of Multiple Pairs of Equal Logarithms in Two Products

Suppose G is a multiplicative cyclic groups of order q where the modulus in multiplication is p. Let $g_1,\ g_2, \dots, g_n$ and $h_1,\ h_2, \dots, h_n$ be generators of G. $Y_1 =$

$\prod_{i=1}^{n} g_i^{x_i}$ and $Y_2 = \prod_{i=1}^{n} gh_i^{x_i}$ while $x_i \in Z_q$. Below is a protocol to prove knowledge of $x_1,\ x_2, \dots, x_n$ to satisfy $Y_1 = \prod_{i=1}^{n} g_i^{x_i}$ and $Y_2 = \prod_{i=1}^{n} gh_i^{x_i}$ without revealing x.

1. The prover chooses randomly $r \in Z_q$ and calculates $a = \prod_{i=1}^{n} g_i^{r_i} \bmod p$, $b = \prod_{i=1}^{n} h_i^{r_i} \bmod p$ for $i = 1,2,\dots,n$ and sends them to the verifier.
2. Then the verifier chooses a random challenge c and sends it to the prover.
3. The prover calculates $s_i = r_i - cx_i \bmod q$ for $i = 1,2,\dots,n$ and sends them to the verifier.
4. The verifier checks $a = \prod_{i=1}^{n} g_i^{s_i} Y_1^c \bmod p$ and $b = \prod_{i=1}^{n} h_i^{s_i} Y_2^c \bmod p$. The proof is accepted only when this verification is passed.

C.4 ZK Proof of knowledge of 1-out-of-k Logarithm

Sometimes we need a proof of knowledge of 1-out-of-k Logarithm, namely knowledge of $\log_g y_1 \ \vee \ \log_g y_1 \ \vee \ \log_g y_k$ where y_i for $i = 1,2,\dots,k$ are within a cyclic group G, which has a generator g and integer x is less than the order of G. The proof protocol is as follows. For simplicity and without losing generality, suppose the prover knows $x_1 = \log_g y_1$.

1. The prover chooses r_1 from $Z_{[G]}$ randomly and calculates $a_1 = g^{r_1}$. The prover chooses w_i and c_i for $i = 2,3,\dots,k$ from $Z_{[G]}$ randomly and calculates $a_i = g^{w_i} z_i^{c_i}$ for $i = 2,3,\dots,k$. He sends a_i and b_i for $i = 1,2,\dots,k$ to the verifier.
2. The verifier chooses a random challenge c from $Z_{[G]}$ and sends it to the prover.
3. The prover calculates $c_1 = c - \sum_{i=2}^{k} c_i \bmod [G]$ and $w_1 = r_1 - c_1 x_1 \bmod [G]$. Then he sends c_i and w_i for $i = 1,2,\dots,k$ to the verifier.
4. The verifier checks $c = \sum_{i=1}^{k} c_i$ and $a_i = g^{w_i} z_i^{c_i}$ for $i = 1,2,\dots,k$.

C.5 ZK Proof of 1-out-of-k Equality of Logarithms

Adapting a proof of knowledge of logarithm to a proof of equality of logarithms is straight-forward, so it is not difficult to combine the last two proof techniques to prove $\log_h z_1 = \log_g y_1 \quad \vee \quad \log_h z_2 = \log_g y_1 \quad \vee \quad \log_h z_k = \log_g y_k$ where $z_i \in G$ for $i = 1,2,\dots,k$ and h is another generator of G. The proof protocol is as follows. For simplicity and without losing generality, suppose $\log_h z_1 = \log_g y_1$ and $x_1 = \log_h z_1$ is known to the prover.

1. The prover chooses r_1 from $Z_{[G]}$ randomly and calculates $a_1 = h^{r_1}$, $b_1 = g^{r_1}$. The prover chooses w_i and c_i for $i = 2,3,\dots,k$ from $Z_{[G]}$ randomly and calculates $a_i = h^{w_i} z_i^{c_i}$, $b_i = g^{w_i} y_i^{c_i}$ for $i = 2,3,\dots,k$. He sends a_i and b_i for $i = 1,2,\dots,k$ to the verifier.
2. The verifier chooses a random challenge c from $Z_{[G]}$ and sends it to the prover.
3. The prover calculates $c_1 = c - \sum_{i=2}^{k} c_i \bmod [G]$ and $w_1 = r_1 - c_1 x_1 \bmod [G]$. Then he sends c_i and w_i for $i = 1,2,\dots,k$ to the verifier.
4. The verifier checks $c = \sum_{i=1}^{k} c_i$ and $a_i = h^{w_i} z_i^{c_i}$, $b_i = g^{w_i} y_i^{c_i}$ for $i = 1,2,\dots,k$.

C.6 ZK Proof of Knowledge of Root

To prove knowledge of x, a m^{th} root of y with composite modulus n with unknown factorization, the following protocol is used.

1. The prover randomly chooses $r \in Z_n$ and publishes $b = r^m \bmod n$.
2. The verifier randomly chooses e with a sufficiently large length and sends it to the prover.
3. The prover calculates $w = rx^e \bmod n$ and sends it to the verifier.
4. The verifier checks $w^m = by^e \bmod n$. He only accepts the proof when this equation holds.

C.7 Range Proof

In cryptographic applications, it is often necessary to test that a secret message is in a certain interval range. When the secret message is known by a party, the party is often required to prove that he knows a secret integer in the interval range. The party chooses an integer from an interval range R, encrypts it or commits to it and publishes the ciphertext or commitment. Then he has to prove that the integer encrypted in the ciphertext or committed in the commitment is in R. The proof cannot reveal any information about the integer except that it is in the range. This proof operation is called range proof.

The most straightforward range proof technique is ZK proof of partial knowledge (e.g. ZK Proof of knowledge of 1-out-of-k Logarithm), which proves that the secret integer may be each integer in the range one by one and then link the multiple proofs with OR logic. This method has a drawback: the number of computations it needs is linear in the size of the range, which leads to very low efficiency. It is well known that the straightforward method can be optimised in efficiency by sealing the secret integer bit by bit and proving that each commitment contains a bit. We can also employs the u-base coding mechanism and proves that each u-base digit of the secret integer is in $\{0,1,\dots,u-1\}$. However, the optimised solution is still not efficient enough especially when the range size is large.

More efficient range proof techniques can be employed in our e-voting system. For example, it can work as follows to prove that an integer m is in a range $\{a,a+1,\dots,b\}$.

-- An integer m is in a range $\{a,a+1,\dots,b\}$ if and only if $(m-a+1)(b-m+1)$ is positive.

-- If $w^2(m-a+1)(b-m+1)$ is positive, $(m-a+1)(b-m+1)$ is positive where w is a integer.

-- To prove that $w^2(m-a+1)(b-m+1)$ is positive, it is divided into three shares m_1,m_2,m_3, whose sum is $w^2(m-a+1)(b-m+1)$. Moreover, m_3 is a square and thus non-negative. If both $sm_1+m_2+m_3$ and $m_1+tm_2+m_3$ are positive where s and t are random positive integers, then $w^2(m-a+1)(b-m+1)$ is positive with an overwhelmingly large probability.

-- As the the information revealed from $sm_1+m_2+m_3$ and $m_1+tm_2+m_3$ about m is statistically negligible except showing that m is in the range $\{a,a+1,\dots,b\}$, they can be published without compromising statistical privacy.

N is a large composite with unknown factorization and G is a cyclic subgroup of Z_N^* with a large order. We employ two large security parameters k_1 and k_2. k_1 is much smaller than k_2. However, k_1 is large enough such that $1/(k_1-1)$ is a negligible probability. A recommendation for their values is that k_1 is large but much smaller than the order of G (e.g. 160 bits long) and k_2 is larger than the order of G (e.g. longer than 1024 bits). A secret integer m is committed to in $c=g^m h^r \bmod N$ where r is a random integer in Z_{k_2}. A party with knowledge of m and r has to prove that the message committed in c is in $\{a,a+1,\dots,b\}$. The proof protocol and the corresponding verification are as follows:

1. The prover calculates and publishes $c_1=c/g^{a-1}\text{mod}N$ and $c_2=g^{b+1}/c\text{mod}N$.
2. He calculates and publishes $c'=c_1^{b-m+1}h^{r'}\text{mod}N$ and publicly gives a proof
$$EL(b-m+1,-r,r'|g,h,c_1,h|c_2,c'). \qquad (17)$$
where r' is a random integer in Z_{k_2} and $EL(\chi,r_1,r_2|g_1,h_1,g_2,h_2|y_1,y_2)$ proves knowledge of secret integers χ,r_1 and r_2 such that $g_1^{\chi}h_1^{r_1}=y_1\text{mod}N$

and $g_2^{\chi} h_2^{r_2} = y_2 \bmod N$ and is a straightforward combination of ZK proof of knowledge of discrete logarithm and ZK proof of equality of discrete logarithms.

3. He randomly chooses integers w and r'' respectively from $Z_{k_2} - \{0\}$ and Z_{k_2} and publishes

$$c'' = c'^{w^2} h^{r''} \text{mod} N.$$

He publicly gives a proof

$$SQR(w, r'' | c', h | c''). \tag{18}$$

where $SQR(\chi, r | g, h | y)$ proves knowledge of integers χ and r such that $y = g^{\chi^2} h^r \text{mod} N$ and is a straightforward extention of ZK proof of knowledge of discrete logorithm.

4. He randomly chooses three non-negative integers m_1, m_2 and m_4 smaller than $w^2(m - a + 1)(b - m + 1)$ such that

$$m_1 + m_2 + m_3 = w^2(m - a + 1)(b - m + 1)$$
$$m_3 = m_4^2.$$

He randomly chooses r_1, r_2, r_3 to satisfy $r_1 + r_2 + r_3 = w^2((b - m + 1)r + r') + r''$ and publishes

$$c'_1 = g^{m_1} h^{r_1} \text{mod} N$$
$$c'_2 = g^{m_2} h^{r_2} \text{mod} N$$
$$c'_3 = c'' / c'_1 c'_2 \text{mod} N.$$

He publicly gives a proof

$$SQR(m_4, r_3 | g, h | c'_3). \tag{19}$$

5. A verifier randomly chooses and publishes integers s and t in $Z_{k_1} - \{0\}$.

6. The prover publishes

$$x = sm_1 + m_2 + m_3$$
$$y = m_1 + tm_2 + m_3$$
$$u = sr_1 + r_2 + r_3$$
$$v = r_1 + tr_2 + r_3$$

7. Besides verification of (17), (18) and (19) the following equations have to be publicly verified

$$c_1 = c / g^{a-1} \text{mod} N \tag{20}$$
$$c_2 = g^{b+1} / c \text{mod} N \tag{21}$$
$$c'' = c'_1 c'_2 c'_3 \text{mod} N \tag{22}$$
$$c'^{s}_1 c'_2 c'_3 = g^x h^u \text{mod} N \tag{23}$$
$$c'_1 c'^{t}_2 c'_3 = g^y h^v \text{mod} N \tag{24}$$
$$x > 0 \tag{25}$$
$$y > 0 \tag{26}$$

Anyone can publicly perform the verification. Once all the integers involved in a verification equation are available, the equation is verified. If a verification fails, the proof protocol fails and stops. The proof protocol succeeds if and only if all the verification conditions are satisfied.

Note that range proof is applied to encrypted votes in our e-voting system, and so we need to prove that the same message m is encrypted by a vote-sealing encryption algorithm. If the employed encryption algorithm is Paillier, the additional proof is straightforward, a combination of ZK proof of knowledge of discrete logarithm, ZK proof of equality of discrete logarithms and ZK proof of knowledge of root. If ElGamal encryption is employed, Benaloh verification has to be employed to implement the additional proof as detailed in the section of Benaloh verification.

C.8 Membership Proof

C.8.1 Protocol 1: Efficient Membership Proof to Specify Vote Validity Check

Fisrtly, we design a membership proof protocol to prove that a secret integer is in a set of w integers at the cost of $O(\sqrt{w})$ exponentiations. It employs divide-and-conquer strategy to use ZK proof of partial knowledge in a more efficient way as follows.

1. For simplicity of description, suppose S can be divided into κ subsets $S_1, S_2, \dots, S_\kappa$ and each S_l contains k integers denoted as $\tau_{l,1}, \tau_{l,2}, \dots, \tau_{l,k}$.
2. V_i randomly chooses an integer s in Z_N and calculates for each S_l integers $b_{l,j}$ for $j = 1,2,\dots,k$ in Z_N to satisfy

$$\sum_{j=1}^{k} b_{l,j}\tau_{l,\rho}^{j} = s \bmod N \quad for \quad \rho = 1,2,\dots,k. \tag{27}$$

More precisely, integers $b_{l,j}$ for $l = 1,2,\dots,\kappa$ and $j = 1,2,\dots,k$ must satisfy

$$\begin{pmatrix} \tau_{l,1} & \tau_{l,1}^2 & \cdots & \tau_{l,1}^k \\ \tau_{l,2} & \tau_{l,2}^2 & \cdots & \tau_{l,2}^k \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \tau_{l,k} & \tau_{l,k}^2 & \cdots & \tau_{l,k}^k \end{pmatrix} \begin{pmatrix} b_{l,1} \\ b_{l,2} \\ \cdots \\ \cdots \\ b_{l,k} \end{pmatrix} = \begin{pmatrix} s \\ s \\ \cdots \\ \cdots \\ s \end{pmatrix}$$

for $l = 1,2,\dots,\kappa$ where in this paper computations in any matrix is carried out modulo N. As $\tau_{l,j} < N$ for $l = 1,2,\dots,\kappa$ and $j = 1,2,\dots,k$ and they are different integers,

$$M_l = \begin{pmatrix} \tau_{l,1} & \tau_{l,1}^2 & \cdots & \tau_{l,1}^k \\ \tau_{l,2} & \tau_{l,2}^2 & \cdots & \tau_{l,2}^k \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \tau_{l,k} & \tau_{l,k}^2 & \cdots & \tau_{l,k}^k \end{pmatrix}$$

is a non-singular matrix for $l = 1,2,\ldots,\kappa$ and there is a unique solution for $b_{l,1}, b_{l,2}, \ldots, b_{l,k}$:

$$\begin{pmatrix} b_{l,1} \\ b_{l,2} \\ \cdots \\ \cdots \\ b_{l,k} \end{pmatrix} = M_l^{-1} \begin{pmatrix} s \\ s \\ \cdots \\ \cdots \\ s \end{pmatrix}$$

for $l = 1,2,\ldots,\kappa$. Therefore, functions $F_l(x) = \sum_{j=1}^{k} b_{l,i} x^j \mathrm{mod} N$ for

$l = 1,2,\ldots,\kappa$ are obtained, each to satisfy

$$F_l(\tau_{l,i}) = s \quad for \quad j = 1,2,\ldots,k. \tag{28}$$

V_i publishes s. Note that $F_l()$ is actually the unique polynomial with degree at most k to satisfy (28) and $F_l(0) = 0$. Readers with basic knowledge in linear algebra should know a few efficient methods, which do not cost any exponentiation, to calculate $F_l()$ from $\tau_{l,j}$ for $j = 1,2,\ldots,k$. Our presentation of $F_l()$ through matrix calculations is only one of them, which seems formal and straightforward. Also note that if necessary calculation of $F_l()$ can be performed beforehand once S is published such that it is already available when the membership proof or exclusion proof starts.

3. V_i calculates $e_{i,j} = e_{i,j-1}^{m_i} R_i^{\gamma_{i,j} N} \mathrm{mod} N^2$ for $j = 1,2,\ldots,k-1$ where

   $m_i = \sum_{l=1}^{w} s_{i,l} s_l$, $e_{i,0} = C_i$ and $\gamma_{i,j}$ is randomly chosen from Z_N.
4. V_i proves validity of $e_{i,1}, e_{i,2}, \ldots, e_{i,k-1}$ using a zero knowledge proof that he

   knows m_i, $r_i = R_i^{\sum_{l=1}^{w} r_{i,l} s_l} \bmod N^2$ and $\gamma_{i,j}$ for $j = 1,2,\ldots,k-1$ such that

   $C_i = g^{m_i} r_i^N \bmod N^2$ and $e_{i,j} = e_{i,j-1}^{m_i} R_i^{\gamma_{i,j} N} \bmod N^2$ for $j = 1,2,\ldots,k-1$,

   which can be implemented through a simple combination of ZK proof of knowledge of discrete logarithm and ZK proof of equality of discrete logarithms.
5. V_i publishes $\omega_{i,l} = g^{s'_i} R'^N_i \mathrm{mod} N^2$ for $l = 1,2,\ldots,\kappa$ where

   $s'_i = \sum_{j=0}^{k-1} b_{l,j+1} m_i^{j+1} \bmod N$ and $R'_i = R_i^{\sum_{j=0}^{k-1} \Gamma_{i,j+1} b_{l,j+1}} \bmod N$ and
   $\Gamma_{i,j} = \Gamma_{i,j-1} m_i + \gamma_{i,j-1} \bmod N \quad for \quad j = 2,3,\ldots,k$ and
   $\Gamma_{i,1} = \sum_{l=1}^{w} r_{i,l} s_l \bmod N$.

6. Any verifier can check $\omega_{i,l} = \prod_{j=0}^{k-1} e_{i,j}^{b_{l,j+1}} \bmod N^2$ for $l = 1,2,\ldots,\kappa$ as

   follows.
   (a) He randomly chooses integers $\theta_1, \theta_2, \ldots, \theta_\kappa$ from Z_τ where τ is a large integer smaller than p and q.
   (b) For $i = 1,2,\ldots,n$ he checks

$$\prod_{l=1}^{\kappa} \omega_{i,l}^{\theta_l} = \prod_{j=0}^{k-1} e_{i,j}^{\sum_{l=1}^{\kappa} \theta_l b_{l,j+1}} \bmod N^2. \tag{29}$$

(c) He accepts validity of $\omega_{i,1}, \omega_{i,2}, \dots, \omega_{i,\kappa}$ iff (29) holds.

7. The verifier (e.g. the talliers)
   (a) calculates $b_{l,j}$ for $l = 1,2,\dots,\kappa$ and $j = 1,2,\dots,k$ to satisfy (27) like the voter does where s is provided by the voter;
   (b) verifies V_i's proof of validity of $e_{i,1}, e_{i,2}, \dots, e_{i,k-1}$.

8. Vote validity proof is reduced to proof of encryption in one of multiple ciphertexts: s is encrypted in one of the κ ciphertexts $\omega_{i,1}, \omega_{i,2}, \dots, \omega_{i,\kappa}$ where

$$\omega_{i,l} = \prod_{j=0}^{k-1} e_{i,j}^{b_{l,j+1}} \bmod N^2.$$

9. V_i runs ZK proof of partial knowledge and ZK proof of knowledge of N^{th} root to implement the proof by showing that he knows one of κ instances of N^{th} root: $(\omega_{i,1}/g^s)^{1/N}$, $(\omega_{i,2}/g^s)^{1/N}$, ..., $(\omega_{i,\kappa}/g^s)^{1/N}$.

The new vote validity proof protocol above employs zero knowledge proof primitives and so achieves honest verifier zero knowledge. It guarantees validity of the votes as illustrated in Theorem 7.

Theorem 7. The new vote validity proof protocol is sound. More precisely, if a polynomial voter can extract an opening (m_i, r_i) to C_i such that $m_i \neq s_l \bmod q$ for $l = 1,2,\dots,w$, then the probability that the voter can pass the verification for validity of his vote is negligible.

Proof: If the voter extracts m_i, r_i and passes the verification in the vote validity proof with a non-negligible probability while $C_i = g^{m_i} r_i^N \bmod N^2$ and $m_i \neq s_l \bmod N$ for $l = 1,2,\dots,w$, a contradiction can be found as follows. As he passes the verification in the vote validity proof with a non-negligible probability, he must have successfully proved validity of $e_{i,1}, e_{i,2}, \dots, e_{i,k-1}$ with a non-negligible probability. As proof of validity of $e_{i,1}, e_{i,2}, \dots, e_{i,k-1}$ is based on ZK proof of knowledge of N^{th} root and ZK proof of equality of discrete logarithms, whose soundness is formally proved when they are proposed, it is guaranteed with a non-negligible probability that the voter can calculate integers m_i, r_i and $\gamma_{i,j}$ for $j = 1,2,\dots,k-1$ in polynomial time such that

$$C_i = g^{m_i} r_i^N \bmod N^2 \tag{30}$$

$$e_{i,j} = e_{i,j-1}^{m_i} \gamma_{i,j}^N \bmod N^2 \quad for \quad j = 1,2,\dots,k-1 \tag{31}$$

where $e_{i,0} = C_i$.

As he passes the verification in the vote validity proof with a non-negligible probability, the voter must have successfully passed the zero knowledge proof of knowledge of one out of κ N^{th} roots with a non-negligible probability. As soundness of zero knowledge proof of partial knowledge is formally proved when it is proposed, it is guaranteed that for some l in $\{1,2,\dots,\kappa\}$ the voter can calculate integers s and R in polynomial time such that

$$g^s R^N = \omega_{i,l} \bmod N^2.$$

Namely,

$$g^s R^N = \prod_{j=0}^{k-1} e_{i,j}^{b_{l,j+1}} \bmod N^2 \tag{32}$$

with a non-negligible probability where $e_{i,0} = C_i$.

(30), (31) and (32) imply that with a non-negligible probability

$$g^s R^N = \prod_{j=0}^{k-1} g^{b_{l,j+1} m_i^{j+1}} \Gamma_{i,j+1}^{N b_{l,j+1}}$$

$$= g^{\sum_{j=0}^{k-1} b_{l,j+1} m_i^{j+1}} (\prod_{j=0}^{k-1} \Gamma_{i,j+1}^{b_{l,j+1}})^N \bmod N^2$$

where

$$\Gamma_{i,j} = \Gamma_{i,j-1}^{m_i} \gamma_{i,j-1} \bmod N^2 \quad for \quad j = 2,3,\dots,k$$

and $\Gamma_{i,1} = r_i$. So

$$s = D(g^s R^N) = D(g^{\sum_{j=0}^{k-1} b_{l,j+1} m_i^{j+1}} (\prod_{j=0}^{k-1} \Gamma_{i,j+1}^{b_{l,j+1}})^N)$$

$$= \sum_{j=0}^{k-1} b_{l,j+1} m_i^{j+1} = \sum_{j=1}^{k} b_{l,j} m_i^j \bmod N$$

with a non-negligible probability.

Note that $b_{l,1}, b_{l,2}, \dots, b_{l,k}$ are generated through

$$\sum_{j=1}^{k} b_{l,j} \tau_{l,\rho}^j = s \bmod N \quad for \quad \rho = 1,2,\dots,k.$$

So with a non-negligible probability

$$\begin{pmatrix} \tau_{l,1} & \tau_{l,1}^2 & \dots & \tau_{l,1}^k \\ \tau_{l,2} & \tau_{l,2}^2 & \dots & \tau_{l,2}^k \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \tau_{l,k} & \tau_{l,k}^2 & \dots & \tau_{l,k}^k \\ m_i & m_i^2 & \dots, & m_i^k \end{pmatrix} \begin{pmatrix} b_{l,1} \\ b_{l,2} \\ \dots \\ \dots \\ b_{l,k} \end{pmatrix} = \begin{pmatrix} s \\ s \\ \dots \\ \dots \\ s \\ s \end{pmatrix} \tag{33}$$

However, as $m_i \neq s_l \bmod N$ for $l = 1,2,\dots,w$ and all the calculations in the matrix is performed modulo N, $\begin{pmatrix} \tau_{l,1} & \tau_{l,1}^2 & \dots & \tau_{l,1}^k & s \\ \tau_{l,2} & \tau_{l,2}^2 & \dots & \tau_{l,2}^k & s \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \tau_{l,k} & \tau_{l,k}^2 & \dots & \tau_{l,k}^k & s \\ m_i & m_i^2 & \dots, & m_i^k & s \end{pmatrix}$ is a non-singular matrix and thus (33) absolutely and always fails. Therefore, a contradiction is found and the probability that a voter can pass the vote validity proof is negligible if the integer he encrypts in C_i is not in S.

C.8.2 Protocol 2: Efficient Vote Validity Proof Without Any Trust Assumption on the Verifier

The main idea of the second new vote validity proof technique is based on the Diffie-Hellman handshake. The talliers set up w different Diffie-Hellman keys, each of whose discrete logarithm is the product of two key roots. The two key roots for the j^{th} Diffie-Hellman key include a constant key root chosen by the talliers and $\log_{R_i^N} y_j$ where $y_j = C_i/g^{s_j} \bmod N^2$. The talliers then seal a random secret message m into w commitments, using a hash function $H()$ and the w Diffie-Hellman keys. The talliers publish the w commitments and his Diffie-Hellman commitment to his key root and then ask the voter V_i to extract m. Obviously, V_i can extract m iff he knows one of the n Diffie-Hellman keys, while Diffie-Hellman assumption (to be detailed later) implies that he knows the j^{th} Diffie-Hellman key iff he knows $\log_{R_i^N} y_j$. The proof protocol is described as follows to guarantee that the voter knows one of $\log_{R_i^N} y_1, \log_{R_i^N} y_2, \dots, \log_{R_i^N} y_w$ under Diffie-Hellman assumption.

1. For simplicity and without losing generality, suppose V_i knows $x_t = \log_{R_i^N} y_t$ where $1 \leq t \leq w$. A one-way and collision-resistent hash function $H()$ from Z_{N^2} to Z_L is employed where $L << N$.
2. The talliers randomly choose an integers r from Z_N and a random message m from Z_L. They calculate and conceal

$$z_j = y_j^r \bmod N^2 \quad for \quad j = 1,2,\dots,w.$$

They calculate and publish

$$c_j = H(z_j) \oplus m \quad for \quad j = 1,2,\dots,w$$
$$z = (R_i^N)^r \bmod N^2.$$

3. V_i publishes

$$u = H(z_t) \oplus c_t$$

where $z_t = z^{x_t} \text{mod} N^2$.

4. The talliers verify

$$u = m.$$

They accept the proof iff this equation holds.

This new vote validity proof protocol is quite efficient. Especially the prover only needs a low constant computational cost independent of w. So it is suitable for low-capability voters using mobile devices. Moreover, as $L << N$, it greatly improves communicational efficiency as well. Theorem 8 shows that it guarantees validity of the votes under Diffie-Hellman assumption as defined in Definition 11, while it is proved to be honest-verifier zero knowledge in Theorem 9.

Definition 11 *(Diffie-Hellman assumption) Given $h^a modN^2$ and $h^b modN^2$, it is hard to calculate $h^{ab} modN^2$ unless either a or b is known.*

Theorem 8. Passing the new vote validity proof protocol guarantees the voter's knowledge of one of $\log_{R_i^N} y_1, \log_{R_i^N} y_2, \dots, \log_{R_i^N} y_w$ under Diffie-Hellman assumption.

Proof: Passing the new vote validity proof protocol implies $u = m$ and thus the user knows m. As the only published information about m is

$$c_j = H(z_j) \oplus m \quad for \quad j = 1,2, \dots, w,$$

the user must know a z_J where $1 \leq J \leq w$. So according to Diffie-Hellman assumption, the user must know $x_J = \log_{R_i^N} y_J$ as

$$z_J = (R_i^N)^{x_J r} \text{ mod } N^2$$

and r is kept secret from the prover. □

Theorem 9. The new vote validity proof protocol achieves honest-verifier zero knowledge.

Proof: The proof transcript of the new vote validity proof protocol contains $c_1, c_2, \dots, c_w, z, u$. A party without any knowledge of any secret can simulate the proof transcript as follows.

1. He randomly chooses an integers r from Z_N and a random message m from Z_L.
2. He calculates $z = (R_i^N)^r \text{mod} N^2$ and $z_j = y_j^r \text{ mod } N^2 \quad for \quad j = 1,2, \dots, w$.
3. He calculates $c_j = H(z_j) \oplus m \quad for \quad j = 1,2, \dots, w$
4. He sets $u = m$.

In both the real transcript with an honest verifier and the simulating transcript,

-- u is uniformly distributed in Z_L.

-- z is uniformly distributed in the cyclic group generated by the generator R_i^N.

-- $c_j = H(z_j) \oplus u$ for $j = 1,2,\dots,w$ where $z_j = y_j^{\log_{R_i^N} z} \bmod N^2$.

As the two transcripts have just the same distribution, the proof transcript can be simulated without any difference by a party without any knowledge of any secret if the verifier is honest and does not deviate from the proof protocol. So the new vote validity proof protocol achieves honest-verifier zero knowledge. □

Note that proof of zero knowledge in Theorem 9 assumes that the verifier is honest and does not deviate from the proof protocol. If the verifier is dishonest and encrypts different messages in $c_1, c_2, \dots, c_w$, he can break privacy of the proof. When the verifier cannot be trusted, the vote validity proof protocol can be further optimised as follows such that before returning m the voter can verify that the verifier is honest and all the $c_1, c_2, \dots, c_w$ are encryptions of the same message.

1. Suppose V_i knows $x_t = \log_{R_i^N} y_t$ where $1 \le t \le w$. Besides one-way and

collision-resistent hash functions $H()$ and $H'()$, a block cipher $E_k()$ (e.g. AES) with key k and a message space Z_L is employed.

2. The talliers randomly choose an integers r from Z_N and a random message m from Z_L. They calculate and conceal

$$z_j = y_j^r \bmod N^2 \quad for \quad j = 1,2,\dots,w.$$

They calculate and publish

$$c_j = E_{H(z_j)}(m) \quad for \quad j = 1,2,\dots,w$$

$$c'_j = E_m(H(z_j)) \quad for \quad j = 1,2,\dots,w$$

$$z = (R_i^N)^r \bmod N^2.$$

3. V_i calculates

$$u = H(z_t) \oplus c_t$$

where $z_t = z^{x_t} \bmod N^2$ and verifies

$$E_{D_u(c'_j)}(u) = c_j \quad for \quad j = 1,2,\dots,w$$

where $D()$ is the decryption function of $E()$. He publishes u if all the w equations hold. Otherwise he gives up his proof as the verifier is detected to be dishonest.

4. The talliers verify

$$u = m.$$

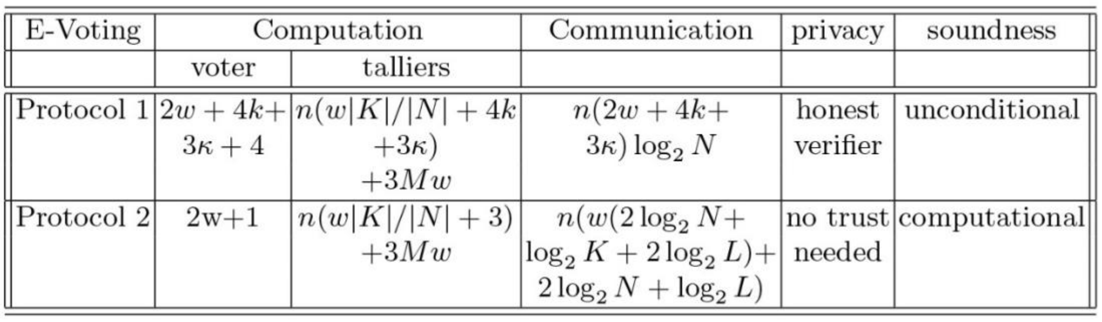
They accept the proof iff this equation holds.

As the employed hash functions are one-way and collision-resistent and it is difficult to find different integers pairs (k_1, m_1) and (k_2, m_2) in Z_L^2 such that $E_{k_1}(m_1) = E_{k_2}(m_2)$ and $E_{m_1}(k_1) = E_{m_2}(k_2)$, the verifier is guaranteed to have committed to the same m in the w instances of commitment (encryption). So it is not necessary to be trusted. Except that the user returns opening of one of $c_1, c_2, \dots, c_n$ in two steps (instead of in one step) and there is a verification against the verifier before the second step, this further optimisation actually employs the same proof principle. So it inherits the security properties proved in Theorem 8 and Theorem 9. Moreover, it optimises its zero knowledge property to achieve stronger privacy without any trust on the verifier. However, its soundness is only computational. The other vote validity proof techniques including Protocol 1 employ standard ZK proof primitives like ZK proof of knowledge of discrete logarithm and ZK proof of partial knowledge, so they can formally build an extractor to extract their claimed knowledge from their proof transcript. To achieve higher efficiency and stronger privacy, this second vote valididty proof protocol bases its soundness on the computational assumption on the Diffie-Hellman problem and the employed hash functions. So it only achieves computational soundness and cannot provide an extractor to show unconditional soundness.

C.8.3 Comparison and summary

The two membership proof techniques are compared in Table 2. In comparison of privacy, the ZK models in different solutions are listed. In estimation of computational efficiency, computational cost of a voter and the talliers is estimated and exponentiations with full length exponents in Z_N are counted. An exponentiation with an exponent in Z_K is regarded as $|K|/|N|$ full length exponents where $||$ stands for bit length as the computational cost of a modulo exponentiation is approximately linear in the bit length of its exponent. As N is usually at least 1024 bits long, $|K|/|N|$ can be quite small while K is still large enough to guarantee that w/K is negligible. In estimation of communicational efficiency, the number of bits transfered between the participants is estimated. Note that N is usually much larger than L and K. For example, it is common to set $\log_2 N = 1024$ and $\log_2 L = \log_2 K = 128$.

Table 2. Comparison of E-Voting Schemes



E-Voting	Computation		Communication	privacy	soundness
	voter	talliers			
Protocol 1	$2w + 4k + 3\kappa + 4$	$n(w\|K\|/\|N\| + 4k + 3\kappa) + 3Mw$	$n(2w + 4k + 3\kappa)\log_2 N$	honest verifier	unconditional
Protocol 2	2w+1	$n(w\|K\|/\|N\| + 3) + 3Mw$	$n(w(2\log_2 N + \log_2 K + 2\log_2 L) + 2\log_2 N + \log_2 L)$	no trust needed	computational

The comparison clearly demonstrates that they are more efficient and can achieve stronger privacy.

D Relation Attacks

As there are many kinds of relation attacks it is not easy to give a simple and formal definition to cover all of them. However, they share a common idea and usually work as follows.

1. To trace a ciphertext c_i, an adversary computes one or more ciphertexts encrypting messages related to the message encrypted in c_i. The ciphertexts are mixed with the normal and valid input ciphertexts and are shuffled together. They may appear as input ciphertexts to a mix network and then be shuffled by all the mixers or be inserted into the mix network by a dishonest mixer and replace some ciphertexts received by the mixer and then be shuffled by the following mixers.

2. The key idea of the attack is that there are some special relations between the message in c_i and the messages in the ciphertexts generated by the adversary. The special relations are chosen by the adversary A and can be recognised by A.

3. After the output messages of the mix network are published, the adversary finds the special relations among them and identifies the message in c_i in polynomial time according to recognised relations.

In summary, a relation attack generates some special relations between some output messages so that an adversary can recognise the relation and trace one of the messages to its corresponding input ciphertext. We have to emphasize that the special relation is not limited to normal algebraic relations and includes a very wide range of relations. Any relation distinguishing the attacked message from other messages can be employed. Actually, a relation useful in a relation attack can be defined as follows.

- A relation is in the form $R(m, m_1, m_2, \dots, m_\delta)$ where m is the attacked and traced message, $\delta \geq 1$ and $m_1, m_2, \dots, m_\delta$ are some messages in the output result of the mix network. The domain of $R()$ is a definite set and if necessary such a set with τ elements can be represented in the form $\{0,1,\dots,\tau-1\}$.

- A relation $R(m, m_1, m_2, \dots, m_\delta)$ is useful in a relation attack if for any m' not equal to m in the output messages $R(m, m_1, m_2, \dots, m_\delta) \neq R(m', m_1, m_2, \dots, m_\delta)$.

A simple example of relation attack is that a corrupted polynomial party (the first mixer or a sender) inserts c^t into the input list where the attacked input is a ciphertext c, t is an integer randomly chosen by the adversary and ElGamal encryption is employed (and such that $c = (a, b)$ is related to $c^t = (a^t, b^t)$). The adversary finds in the output message list two messages m_i and m_j such that $m_i = m_j^t$ and deduces that m_j is the message encrypted in c. A special case of this attack is $t = 1$, when the attacked vote is suspected to be a rare selection.

A security model defining robustness against relation attacks in shuffling-based e-voting is proposed in Definition 4 and Definition 5. The first condition guarantees that in any relation attack any ciphertext is not helpful and the only information to exploit is the decrypted votes. The second condition guarantees that each decryption output a legal vote and the ILLEGAL state cannot be exploited in any relation attack. The third condition further specifies the second one and guarantees that the decrypted votes only reveals the voters' election choices, so that no other information can be used by any relation attack. The fourth condition guarantees that with a very large probability there are multiple instances of each election choice in the recovered votes, such that the votes involved in a relation attack are untraceable. In Theorem 6, it is illustrated that the four conditions are sufficient to prevent relation attack and they guarantee that the probability that a vote is traceable is vanishingly small.

Definition 4 *A vote is traceable if a link between its content and its origin can be found. Suppose a vote appears as an encryption or commitment η_1 when it is cast and its content is published as η_2 in the tallying phase. The vote is traceable if η_2 can be linked to η_1 and thus linked to the corresponding voter.*

Definition 5 *A mix network-based e-voting scheme is robust against relation attacks if the following two conditions are satisfied.*

- The encryption algorithm employed to encrypt the votes is semantically secure.
- An encrypted vote is decrypted only if it contains a legal election choice.
- No other information than the submitted election choices is extracted from the votes. More formally and precisely, the following two vote transcripts are indistinguishable. The first vote transcript contains the decryption results of an instance of the optimised e-voting scheme, where every participant is honest and strictly follows the e-voting protocol and no relation attack is launched. The second vote transcript contains the decryption results of an instance of the optimised e-voting scheme, where the participants are dishonest and may launch any relation attack.
- The vote space is much smaller than the number of voters such that many voters have the same vote.

Thoerem 6. Satisfaction of the the four conditions in Definition 5 is sufficient to prevent any relation attack.

Before Theorem 6 is proved, a lemma is proved first.

Lemma 1. The probability that a vote is traceable is negligible when the number of voters n is much larger than ρ, the number of valid voting choices.

Proof: A vote can be traced only if no other vote contains the same voting choice with it, which happens with a probability

$$(1-\upsilon)^{n-1}$$

where υ is the probability that the choice in the vote is chosen by a common voter. When n is large and much larger than ρ, it is a very small probability. For example, if $\upsilon = 1/\rho$, the probability is

$$(1-1/\rho)^{n-1} = ((1-1/\rho)^{\rho})^{n-1/\rho}$$

As $(1-1/\rho)^{\rho}$ is a small value asymptotically approaching $1/e$ and n is much larger than ρ, $(1-1/\rho)^{n-1}$ is overwhelmingly small in terms of n/ρ and negligible. Even if the choice in the vote is unpopular and υ is much smaller than $1/\rho$ (e.g. $\upsilon = 0.01$ in a 5-candidate election), in a large scale election (e.g. with millions of voters) the probability is still quite small and negligible. In practice, only one different vote with the same choice may not be enough to satisfactorily hide an attacked vote and thus a vote is regarded as untraceable if and only if there are other votes containing the same choice. Suppose γ is a security parameter and it is required that at least γ other votes contain the same choice. Usually, γ only needs to be large enough to hide the attacked vote among $\gamma+1$ votes with the same choice, so γ can be much smaller than n. The probability that there are less than γ other votes containing the same voting choice is

$$((\rho-1)^{n-1} + n-1_{1}\ (\rho-1)^{n-2} + n-1_{2}\ (\rho-1)^{n-3} + \cdots$$

$$+ n-1_{\gamma-1}(\rho-1)^{n-\gamma})/\rho^{n-1}$$

where a_{b} denotes the number of possible choices of b elements from a candidate elements.

As $\rho^{n-1} = (\rho-1+1)^{n-1} = \sum_{i=0}^{n-1} n-1_{i}\ (\rho-1)^{n-1-i}$ and n is much larger than ρ and γ, the probability that there are less than γ other votes containing the same voting choice is negligible. □

There are four main countermeasures against relation attacks: ZK Proof, Forward and Reverse Shuffling, Encryption with Small Message Space and RCCA Encryption Free of ZK Proof. Obviously, forward and reverse shuffling is the most inefficient and complex countermeasure and is not a good candidate in most applications. ZK proof is the most straightforward countermeasure and when the employed ZK proof primitive is simple, it is a good solution. But when the ZK proof primitive becomes complicated and costly, the other two countermeasures can be considered. In most cases, RCCA encryption free of ZK proof is the most efficient, especially when the vote is not long and thus the two parts of the transformed message can be contained in a single ElGamal encryption.

E Relation Attacks: Countermeasures

There are four main countermeasures against relation attacks and they are described in details in the following.

E.1 ZK Proof

The most common countermeasure is to require every voter to prove his knowledge of his vote encrypted in his ballot. Of course, the vote cannot be revealed in the proof and the proof should be publicly verifiable. This can be implemented by ZK proof of encryption (e.g. ZK proof of equality of discrete logarithms in the case of ElGamal encryption). In some e-voting applications, more complicated ZK proof of certain properties of the encrypted votes may be needed. There are some other countermeasure indirectly employs ZK proof including some non-malleable encryption algorithms based ZK proof, CCA (chosen ciphertext attack) secure encryption algorithms based on the model of CCA security of multiple encryptions. A drawback of this countermeasure is complexity and inefficiency. Even if some CCA secure encryption algorithms do not need zero knowledge proof of knowledge on the encryption performer's (voter's) side, they usually extend the length of ciphertext and greatly increase the cost of shuffling the ciphertexts. Moreover, they usually cannot be used in shuffling, which re-encrypts and randomises some ciphertexts.

E.2 Forward and Reverse Shuffling

Another countermeasure can prevent any relation attack in an e-voting application employing any mix network. A counting center selects different ciphertexts standing for the same possible choices in the election for different voters, shuffles them into a random order and proves their validity using ZK proof primitives. Then the mixers form a reverse-direction mix network, through which the ciphertexts are shuffled and sent to the voters. At the same time, the counting center and the mixers send their permutations on the ciphertexts for each voter to the corresponding voter through an "untapped secure channel". Finally, the voters submit the ciphertexts standing for their choices to the mixers, who then shuffle them as in a normal mix-based e-voting scheme. An obvious drawback of this countermeasure is very low efficiency. Before n votes are mixed, kn ciphertexts must be mixed in a reverse mix network where k is the number of possible choices in the election, not to mention transmission of $n(m+1)$ k-dimension permutations through the additional "untapped secure channel" between the counter center, mixers and the voters must be specified with costly cryptographic operations where m is the number of mixers. Even if the simplest election rule is used and $k = 2$, efficiency is very low. When a more general election is involved and k is larger, cost of this countermeasure is intolerable.

E.3 ElGamal with Small Message Space

Specially designed encryption algorithm is also a solution to relation attacks. We notice that although it is difficult to design a general countermeasure against relation attacks in general-purpose mix networks, it may be easier to design a general countermeasure against relation attacks in e-voting-oriented mix networks as the vote space in e-voting applications is usually small. We specially design a new encryption algorithm in an e-voting-oriented mix network such that any ciphertext is decrypted into a small vote space, each element in which stands for a legal choice in the election.

More precisely,

- if a legal choice is encrypted into a ciphertext and shuffled in the mix network, after being shuffled it will be recovered correctly;
- if a ciphertext encrypting an illegal choice is shuffled in the mix network, no matter it is generated by a malicious voter or inserted into the mix network by a malicious mixer it will either be detected before being decrypted or be recovered as a legal choice by a deterministic function.

So, no matter how the input ciphertexts are generated, modified or how special they are only legal votes are decrypted and published in the end, while the legal votes are in a small set in normal e-voting applications. Therefore, although a very large number (e.g. larger than 2^{1024}) of possible inputs may be submitted or inserted into the mix network and various relations may exist among them most relations will vanish after they are processed in the mix network. An adversary in a relation attack can only base his attack on relations able to survive the mix network: relation between legal votes. As usually many voters have the same choice in the election, many votes satisfy those surviving relations and the attacked vote and the vote related to it cannot be distinguished from other votes equal to them. As a result, the relation attacks in e-voting applications are prevented.

Firstly, a novel encryption algorithm is designed as follows:

- **Parameter setting**

N is a large public composite with unknown factorization and the largest cyclic group in Z_N^* has an order $2q\alpha\beta$ where q is not a divisor of N. α and β are secret large primes with similar size. q is a public prime no smaller than ρ where ρ is the number of possible choices in the election. Note that such parameter setting is easy and practical as:

- in most practical e-voting applications, ρ is not large and so q can be much smaller than α and β;
- generation of N is straightforward by satisfying that $N = PQ$, $2q\alpha$ divides $P - 1$ and 2β divides $Q - 1$.

g is a generator of G_1, the cyclic subgroup with order $q\alpha$. h is a generator of G_2, the cyclic subgroup with order α. $g' = g^{\alpha}$ and thus g' is a generator of G, the cyclic subgroup with order q. The three integers, g, h, g', are publicly known.

- **Setting of keys**

The private key is α and the public key is g'.

- **Key sharing**

The verifiable t-out-of-m secret sharing function is employed to share α among the talliers $A_1, A_2, \ldots, A_m$ where α_j is the share held by A_j. $\theta_j = g^{\alpha_j}$ for $j = 1,2,\ldots,m$ are published.

- **Encryption:**

a message m in Z_q is encrypted into $c = E(m) = g^m h^r \bmod N$ where r is a random integer in Z_l and l is a large security parameter.

- **Re-encryption:**

a ciphertext c in Z_q is re-encrypted into $c' = ch^{r'} \bmod N$ where r' is a random integer in Z_l.

- **Decryption function**

1. The talliers receive a ciphertext c and cooperate to calculate $d = \prod_{j \in S} c_j^{w_j} \bmod N$ where A_j publishes $c_j = c^{\alpha_j} \bmod N$, $w_j = \prod_{k \in S, k \neq j} \frac{k}{k-j}$ and set S contains the indices of $t + 1$ cooperating talliers.
   Each A_j proves correctness of his partial decryption by proving $\log_c c_j = \log_g \theta_j$ using zero knowledge proof of equality of logarithms.
2. Search for $\log_{g'} d$ in Z_q. If $\log_{g'} d$ is found in Z_q, it is output as the decryption result. If $\log_{g'} d$ does not exist, c is declared as an illegal ciphertext and a result "ILLEGAL" is output. When c is declared as illegal, anyone can test whether $d^q \neq 1 \bmod N$ to verify.

The drawback of the third countermeasure is large composite modulus and search of discrete logarithm although in a small group. They may affect efficiency of the e-voting system.

E.4 RCCA Encryption Free of ZK Proof

The last counter measure relies on the idea that a non-malleable encryption algorithm would guarantee that any unauthorized ciphertext manipulation would trigger an alert. However, we still need to be able to re-randomize ciphertexts, in order to be able to perform the mixing operations. This leads to the so called re-randomizable RCCA (replayable security against chosen-ciphertext attack) encryption, which requires the possibility to re-randomize ciphertexts while preventing any other homomorphic transformation. RCCA security of course does not prevent a mixer to make exact ballot copies (possibly re-randomized), which needs a mechanism to detect. Moreover, the required RCCA security cannot reply on ZK proof for the sake of efficiency. One efficient solution for building re-randomizable RCCA encryption consists in applying a transformation, called OAEP 3-Round (OAEP3 for short), to messages before encrypting them with a probabilistic encryption scheme like ElGamal. This transformation assumes the availability of 3 independent random oracles, H_1, H_2 and H_3 (in practice, these can be implemented with a single hash function and 3 distinct prefixes). The OAEP3 transform processes a message m, represented as a bit-string, using a fresh random bit string r as follows:

$$s = m \oplus H_1(r); t = r \oplus H_2(s); u = s \oplus H_3(t)$$

and outputs the pair (t, u). The OAEP3 transformed message can then be encrypted with ElGamal usual, and the resulting scheme is RCCA secure, assuming the hardness of the gap Diffie-Hellman problem, a problem that consists in solving an instance of the computational Diffie-Hellman problem in the presence of a Decisional Diffie-Hellman oracle. The intuition underlying this result is common to the OAEP-style transformations: any change into t or u leads to message changes that are unpredictable without querying the H_i oracles first. So, a modification of a ciphertext can result in a recognizable modification of the corresponding plaintext only if this plaintext is known in the first place.

With the new encryption algorithm, the encryption process is modified by requiring each party willing to submit an encryption of a message m to the first mixer to encrypt the message $OAEP3(m||v||0^{\mu})$ instead, where v is a unique k bit tag randomly chosen by the corresponding voter and μ is a security parameter. After the being shuffled in the mix network and decrypted using the standard ElGamal decryption, a decryption result is processed as follows:

1. If the last μ bits is not zero, the decryption result is discard. Otherwise, the last μ bits are removed and vote opening goes on.
2. If the last v bits is not unique, only the earliest vote with the tag is kept. The last v bits of the kept vote are removed, while all the other votes with the same tag are discarded.
3. The left part of the decryption result is divided into two halves u and t. $s = u \oplus H_3(t)$ and $r = t \oplus H_2(s)$ are calculated and finally $m = s \oplus H_1(r)$ is obtained.

F By-Pass Attacks

Mix network provides privacy and anonymity by employing multiple mixers to shuffle some input ciphertexts such that if at least one of them keeps his shuffling secret the outputs of the mix network cannot be traced back to the inputs to the mix network. By-pass attack breaks this security assumption by by-passing some mixers such that collusion of the left mixers can compromised privacy of the mix network. The strongest by-pass attack is carried out by the last mixer, who cheats as follows.

1. Ciphertexts $c_1, c_2, \ldots, c_n$ are the input ciphertexts to the mix network, which consists of m mixers $P_1, P_2, \ldots, P_m$. As usual, each P_j receives $c_{j-1,1}, c_{j-1,2}, \ldots, c_{j-1,n}$ from P_{j-1} and outputs $c_{j,1}, c_{j,2}, \ldots, c_{j,n}$ to P_{j+1} where $c_{0,i} = c_i$ for $i = 1,2, \ldots, n$.

2. P_m ignores $c_{m-1,1}, c_{m-1,2}, \ldots, c_{m-1,n}$ sent to him and directly permutes and processes $c_1, c_2, \ldots, c_n$. In a re-encryption mix network, his processing is re-encryption of $c_1, c_2, \ldots, c_n$, while in a decryption mix network, his processing is decryption of them.

3. P_m's outputs $c'_1, c'_2, \ldots, c'_n$ is the output of the mix network.

In this attack, P_m by-passes all the other mixers and directly shuffles the input ciphertexts to the mix network such that he monopolizes the permutation shuffling the inputs to the mix network to the outputs of mix network. In a re-encryption mix network, he can carries out the attack alone without any conspirator. In a decryption network, he needs collusion of the other secret/private key holders.

The simplest countermeasures against by-pass attack is to ask every mixer to prove validity of his shuffling from the previous mixer to the next mixer. That is the so-called separate verification mechanism in mix network. Without knowledge of the secret operations of the previous mixers, P_m cannot prove that his shuffling of $c_1, c_2, \ldots, c_n$ to be shuffling of $c_{m-1,1}, c_{m-1,2}, \ldots, c_{m-1,n}$. So, mix networks with separate verification is inherently immune to by-pass attacks.

For sake of high efficiency, there are still some mix networks employing general verification, where no mixer needs to give the costly proof that his shuffling is valid. Instead, a general verification of the whole shuffling of the mix network guarantees that the outputs of the mix network is a permutation of the inputs to the mix network. This general verification mechanism is a vulnerability to by-pass attack.

To prevent by-pass attack in mix networks with general verification, decryption instead of re-encryption must be employed as the randomization operation by every mixer together with his permutation. The best choice is decryption mix network based on symmetric ciphers, namely each message (vote) to be shuffled is encrypted by a cipher chain using every key shared by all the mixers with the voter just like in onion routing or TOR. A merit of this design is that every voter can monitor the routing of his own vote so that any misconducts of any mixer on any vote can be found by the corresponding voter. Moreover, the last mixer does not know the keys of the previous mixer and thus cannot decrypt the initial input ciphertexts to perform a by-pass attack. Therefore, in this design if every voter monitors all the mixers, not only any incorrect shuffling can be detected but also relation attacks can be prevented.

After studying by-pass attacks, we are more confident with our choice of mix networks to by employed in our e-voting system. The first choice is a re-encryption mix network employing ElGamal encryption with separate verification. The second choice is a decryption mix network employing AES encryption without separate verification.

G Adjustments of Parameters of ElGamal Encryption

We have selected ElGamal encryption as the vote-sealing encryption algorithm in our e-voting system. However, we still need to determine more details of its implementation like its concrete parameter setting.

A standard ElGamal encryption (which employs an integer-based cyclic group but can be straightforwardly extended to elliptic-curve-based or other kinds of cyclic groups) to support various vote formats is as follows.

1. **Key generation**
   G with order q is a cyclic subgroup of Z_p^* where q and $p = 2q + 1$ are large primes. Integer g is a generator of G and $x \in Z_q$ is the private key, while g and $y = g^x \bmod p$ is the public key.
2. **Encryption**
   A Message $s \in Z_p^*$ is encrypted to a pair (a, b) where $a = g^r \bmod p$, $b = sy^r \bmod p$ and r is randomly chosen from Z_q.
3. **Decryption**
   A ciphertext $c = (a, b)$ is decrypted to $m = b/a^x \bmod p$. Correctness of the decryption can be verified by using ZK proof to prove $\log_a b/m = \log_g y$.

A merit of this setting is that the message space is Z_p^*, a consecutive range supporting any vote no longer than $\log_2 p$ bits. However, it has a drawback, failure to achieve semantic security. We know that an encryption algorithm with encryption function $E()$ is semantically secure if no polynomial adversary can win the following game with a probability non-negligibly larger than 0.5.

1. The adversary chooses two different messages m_1 and m_2 and sends them to an encryption oracle.
2. The encryption oracle randomly chooses i from $\{1,2\}$ and outputs $c = E(m_i)$.

3. The adversary wins the game if it receives c and finds out i.

With the ElGamal setting, anyone can tell whether the message encrypted in an ElGamal ciphertext is in G and thus a quadratic residue by testing whether the second integer in the ciphertext is a quadratic residue. To test whether an integer in Z_p^* is a quadratic residue is easy: $b^q = 1 \bmod p$ if and only if b is a quadratic residue. That means any encrypted message can be categorized into either of 2 subsets of the messages space since among the p integers in the message space half of them are quadratic residues and the other half are not. So, such an encryption setting may make the e-voting scheme vulnerable to attacks against vote privacy.

A modification to satisfy semantic security is to change the message space to G so that all the messages are quadratic residues. However, this modification imposes a limitation on vote format: a vote cannot just be an integer/string with a certain bit length but has to in a set of non-consecutive integers. If the ballot has a special format (e.g. ending with a certain bit string), usually it is hard for the vote (message) space to be completely contained by G. Although an additional mapping function can be employed to transform a vote into G, common users without absolute confidence and trust in the e-voting technology may be concerned by the transform and actually has to trust a device or program to carry out the transform, which cannot be done by hand.

The dilemma in parameter setting of ElGamal encryption still exists even if the integer-based cyclic group is replace by an ER-based cyclic group. Actually, no matter what kind of cyclic group the discrete logarithm relation is built on, any variant of ElGamal encryption depends on the hardness of the DL (descrete logarithm) problem in the concrete employed DL relation and setting the message space as a super group larger than the cyclic group of the DL relation will compromise semantic security and thus vote privacy. A solution for this dilemma is to modify the ElGamal setting as follows.

1. **Key generation**
   Integer g is a generator of the cyclic group Z_p^* and $x \in Z_{p-1}$ is the private key, while g and $y = g^x \bmod p$ is the public key.
2. **Encryption**
   A Message $s \in Z_p^*$ is encrypted to a pair (a, b) where $a = g^r \bmod p$, $b = sy^r \bmod p$ and r is randomly chosen from Z_{p-1}.
3. **Decryption**
   A ciphertext $c = (a, b)$ is decrypted to $m = b/a^x \bmod p$. Correctness of the decryption can be verified by using ZK proof to prove $\log_a b/m = \log_g y$.

With this modification of the setting of encryption algorithm, called large group setting, any vote in Z_p^* is valid and special formats like certain ending are supported. When more special requirements on vote format need to employ complex ZK proof techniques, additional requirements on the setting of the employed encryption algorithm may be needed. For example, if range proof is needed, for efficiency of the proof and verification it is usually required that the order of the message space is secret. In that case, ElGamal setting has to be something like the following.

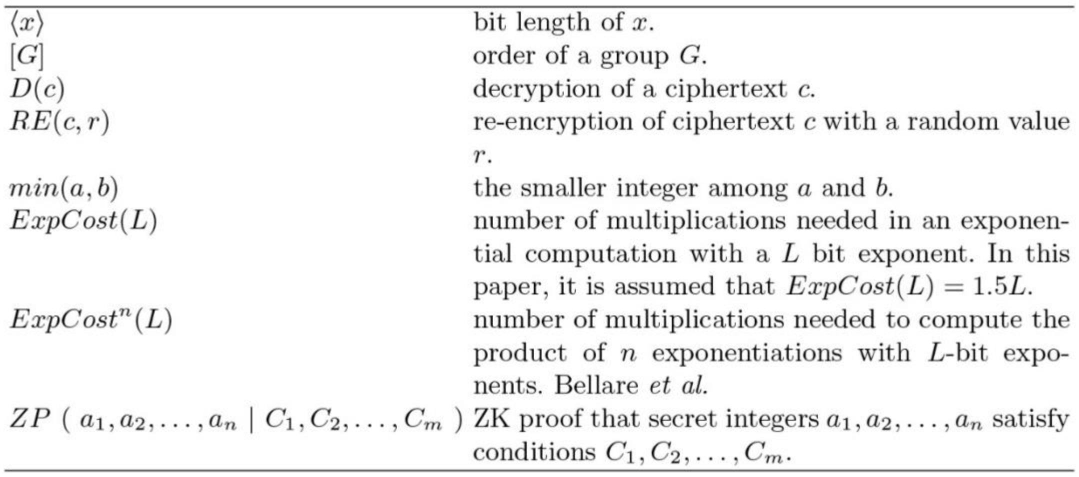
1. **Key generation**
   p is a prime and Z_p^* is a cyclic group. $p-1=q\rho$ and both q and ρ are secret large primes. G with order q is a cyclic subgroup of Z_p^*. Integer g is a generator of G and $x \in Z_q$ is the private key, while g and $y = g^x \bmod p$ is the public key.
2. **Encryption**
   A Message $s \in Z_p^*$ is encrypted to a pair (a, b) where $a = g^r \bmod p$, $b = sy^r \bmod p$ and r is a random integer.
3. **Decryption**
   A ciphertext $c = (a, b)$ is decrypted to $m = b/a^x \bmod p$. Correctness of the decryption can be verified by using ZK proof to prove $\log_a b/m = \log_g y$.

However, as this special setting is not semantically secure and cannot guarantee complete ZK privacy of the votes, we do not suggest to employ it in our e-voting scheme. Our suggestion is to employ the large group setting with the order of the message space published and then use a ZK proof that the same vote is encrypted in the ElGamal ciphertext and committed to in a commitment with a message with an unknown order. Thus efficient range proof can be performed on the commitment to guarantee vote validity.

H Instantiated Batch Proof and Verification

H.1 Parameters and Symbols

Table 3 shows the general parameters and symbols used in this paper. In order to avoid confusion we use separate values for each of the different algorithmic settings as follows.



$\langle x \rangle$	bit length of x.
$[G]$	order of a group G.
$D(c)$	decryption of a ciphertext c.
$RE(c, r)$	re-encryption of ciphertext c with a random value r.
$min(a, b)$	the smaller integer among a and b.
$ExpCost(L)$	number of multiplications needed in an exponential computation with a L bit exponent. In this paper, it is assumed that $ExpCost(L) = 1.5L$.
$ExpCost^n(L)$	number of multiplications needed to compute the product of n exponentiations with L-bit exponents. Bellare *et al.*
$ZP\ (\ a_1, a_2, \ldots, a_n \mid C_1, C_2, \ldots, C_m\ )$	ZK proof that secret integers $a_1, a_2, \ldots, a_n$ satisfy conditions $C_1, C_2, \ldots, C_m$.

Table 3: Notation

ElGamal setting in the new batch proof-verification techniques:
G_1 is a cyclic subgroup of Z_p^* with order q where $p-1=2q$ and p, q are large primes. Let g_1, g_2 be generators of G_1 and g_0 a generator of Z_p^*. Private key X is chosen from Z_q and public key $(g_1, Y=g_1^X)$ is published.

If $c_1=(a_1,b_1)$ and $c_2=(a_2,b_2)$ are ElGamal ciphertexts, then the product ciphertext is $c_1c_2 \overset{def}{=} (a_1a_2, b_1b_2)$ and the quotient ciphertext is $c_1/c_2 \overset{def}{=} (a_1/a_2, b_1/b_2)$.
The absolute-value function from Z_p^* to G_1 defined by

$$|\sigma| = \begin{cases} \sigma & if\ \sigma\ \in\ G_1 \\ -\sigma \bmod p & if\ \sigma\ \in\ Z_p^* \setminus G_1\ where\ -1 = g_0^q \end{cases}$$

For any $\sigma, \sigma_1, \sigma_2, \dots, \sigma_n \in Z_p^*$ we have:

$$|\sigma_1||\sigma_2| \dots |\sigma_n| \bmod p = |\sigma_1\sigma_2 \dots \sigma_n \bmod p| \tag{36}$$

$$|\sigma|^n \bmod p = |\sigma^n \bmod p| \tag{37}$$

H.2 Batch ZK Proof-Verification Techniques

Four batch ZK proof-verification techniques are proposed in this section. Two of them prove equality of logarithms with a common base and two prove equality of logarithms to a common exponent.

H.2.1 Batch ZK Proof-Verification of Equality of Logarithms of Common Base

The techniques in this section are designed to prove $\log_g y_i = \log_h z_i$ for $i = 1,2,\dots,n$ where g and h are generators of a cyclic group G with a prime order. We look first at strict verification and then at loose verification.

Strict Verification of Equality of Logarithms of Common Base

Unless specified, any multiplicative computation in this subsection occurs in G. The following theorem is designed to prove and verify $\log_g y_i = \log_h z_i$ for $i=1,2,\dots,n$ for $i=1,2,\dots,n$ where $y_i \in G$ and $z_i \in G$.

Theorem 10. Suppose $y_i \in G$ and $z_i \in G$ for $i=1,2,\dots,n$. Let L be a security parameter and t_i satisfying $t_i < 2^L < [G]$ for $i=1,2,\dots,n$ be random integers. If there exists integer v, such that $1 \le v \le n$ and $\log_g y_v \neq \log_h z_v$, then $\log_g \prod_{i=1}^n y_i^{t_i} \neq \log_h \prod_{i=1}^n z_i^{t_i}$ with a probability no less than $1-2^{-L}$.

To prove Theorem 10, a lemma is proved first.

Lemma 3. Suppose $y_i \in G$ and $z_i \in G$ for $i = 1,2,\dots,n$ and $t_1,\ t_2,\ ,\ t_{v-1},\ t_{v+1},$ $t_{v+2},,\ t_n$ are constant. If $\log_g y_v \neq \log_h z_v$ and $\log_g \prod_{i=1}^{n} y_i^{t_i} = \log_h \prod_{i=1}^{n} z_i^{t_i}$, then there is at most one possible L-bit solution for t_v.

Proof: If this lemma is not correct, then the following two equations can be satisfied simultaneously where $\log_g y_v \neq \log_h z_v,\ \langle t_v \rangle = \langle \hat{t}_v \rangle = L$ and $t_v \neq \hat{t}_v$.

$$\log_g \prod_{i=1}^{n} y_i^{t_i} = \log_h \prod_{i=1}^{n} z_i^{t_i} \tag{38}$$

$$\log_g\left(\prod_{i=1}^{v-1} y_i^{t_i} \cdot y_v^{\hat{t}_v} \prod_{i=v+1}^{n} y_i^{t_i}\right) = \log_h\left(\prod_{i=1}^{v-1} z_i^{t_i} \cdot z_v^{\hat{t}_v} \prod_{i=v+1}^{n} z_i^{t_i}\right) \tag{39}$$

Subtracting (39) from (38) yields $\log_g y_v^{t_v - \hat{t}_v} = \log_h z_v^{t_v - \hat{t}_v}$ and so $(t_v - \hat{t}_v)\log_g y_v = (t_v - \hat{t}_v)\log_h z_v \bmod [G]$. Note that $t_v - \hat{t}_v \neq 0 \bmod [G]$ because $1 \leq \hat{t}_v, t_v < 2^L < [G]$ and $\hat{t}_v \neq t_v$. Therefore, $\log_g y_v = \log_h z_v \bmod [G]$. Contradiction is found under the assumption that this lemma is not correct, so this lemma proves correct.

□

Proof of Theorem 10: Lemma implies that among the $(2^L)^n$ possible combinations of $\{t_1, t_2, \dots, t_n\}$, at most $(2^L)^{n-1}$ of them can satisfy $\log_g \prod_{i=1}^{n} y_i^{t_i} = \log_h \prod_{i=1}^{n} z_i^{t_i}$ when $y_i \in G,\ z_i \in G$ and $\log_g y_v \neq \log_h z_v$. So if $\log_g y_v \neq \log_h z_v$ and t_i for $i = 1,2,\dots,n$ are randomly chosen, $\log_g \prod_{i=1}^{n} y_i^{t_i} = \log_h \prod_{i=1}^{n} z_i^{t_i}$ is satisfied with probability no more than 2^{-L}. □

According to Theorem 10, verification of $\log_g y_i = \log_h z_i$ for $i = 1,2,\dots,n$ can be batched to $\log_g \prod_{i=1}^{n} y_i^{t_i} = \log_h \prod_{i=1}^{n} z_i^{t_i}$ when $y_i, z_i \in G$ for $i = 1,2,\dots,n$. The probability that $\log_g \prod_{i=1}^{n} y_i^{t_i} = \log_h \prod_{i=1}^{n} z_i^{t_i}$ while $\log_g y_v \neq \log_h z_v$ for some v in $\{1,2,dots,n\}$ is no more than 2^{-L}.

Loose Verification of Equality of Logarithms of Common Base

Unless specified, any multiplicative computation in this subsection occurs in G_1 with modulus p. The following theorem is used to prove $\log_{g_1}|y_i| = \log_{g_2}|z_i|$ where $y_i \in Z_p^*$ and $z_i \in Z_p^*$ for $i = 1,2,\dots,n$.

Theorem 11. Suppose $y_i \in Z_p^*$ and $z_i \in Z_p^*$ for $i = 1,2,\dots,n$. Let L be a security parameter and t_i satisfying $t_i < 2^L < q$ for $i = 1,2,\dots,n$ be random integers. If there exists integer v, such that $1 \le v \le n$ and $\log_{g_1}|y_v| \ne \log_{g_2}|z_v|$, then $\log_{g_1}\prod_{i=1}^{n} y_i^{t_i} \ne \log_{g_2}\prod_{i=1}^{n} z_i^{t_i}$ with a probability no less than $1 - 2^{-L}$.

To prove Theorem 11, a lemma is proved first.

Lemma 4. Suppose $y_i \in Z_p^*$ and $z_i \in Z_p^*$ for $i = 1,2,\dots,n$ and $t_1,\ t_2,,\ t_{v-1},\ t_{v+1},\ t_{v+2}\ ,\ ,\ t_n$ are constant. If $\log_{g_1}|y_v| \ne \log_{g_2}|z_v|$ with $1 \le v \le n$ and $\log_{g_1}\prod_{i=1}^{n} y_i^{t_i} = \log_{g_2}\prod_{i=1}^{n} z_i^{t_i}$, then there is at most one possible L-bit solution for t_v.

Proof: If this lemma is not correct. Then the following two equations are satisfied simultaneously where $\log_{g_1}|y_v| \ne \log_{g_2}|z_v|$, $\langle t_v\rangle = \langle \hat{t}_v\rangle = L$ and $t_v \ne \hat{t}_v$.

$$\log_{g_1}\prod_{i=1}^{n} y_i^{t_i} = \log_{g_2}\prod_{i=1}^{n} z_i^{t_i} \tag{40}$$

$$\log_{g_1}\prod_{i=1}^{v-1} y_i^{t_i}\cdot y_v^{\hat{t}_v}\prod_{i=v+1}^{n} y_i^{t_i} = \log_{g_2}\prod_{i=1}^{v-1} z_i^{t_i}\cdot z_v^{\hat{t}_v}\prod_{i=v+1}^{n} z_i^{t_i} \tag{41}$$

Subtracting (41) from (40) yields $\log_{g_1} y_v^{t_v-\hat{t}_v} = \log_{g_2} z_v^{t_v-\hat{t}_v}$. According to Formula-2,

$$\log_{g_1}|y_v|^{t_v-\hat{t}_v} = \log_{g_2}|z_v|^{t_v-\hat{t}_v} \text{ and}$$

$$\text{so } (t_v - \hat{t}_v)\log_{g_1}|y_v| = (t_v - \hat{t}_v)\log_{g_2}|z_v| \bmod q.$$

Note that $t_v - \hat{t}_v \ne 0 \bmod q$ because $1 \le \hat{t}_v, t_v < 2^L < q$ and $\hat{t}_v \ne t_v$.
Therefore, $\log_{g_1}|y_v| = \log_{g_2}|z_v|$. Contradiction is found under the assumption that this lemma is not correct, so this lemma proves correct. □

Proof of Theorem 11: Lemma 4 implies that among the $(2^L)^n$ possible combinations of $\{t_1, t_2, \dots, t_n\}$, at most $(2^L)^{n-1}$ of them can satisfy $\log_{g_1}\prod_{i=1}^{n} y_i^{t_i} = \log_{g_2}\prod_{i=1}^{n} z_i^{t_i}$ when $\log_{g_1}|y_v| \ne \log_{g_2}|z_v|$. So if $\log_{g_1}|y_v| \ne \log_{g_2}|z_v|$ and t_i for $i = 1,2,\dots,n$ are randomly chosen, $\log_{g_1}\prod_{i=1}^{n} y_i^{t_i} = \log_{g_2}\prod_{i=1}^{n} z_i^{t_i}$ is satisfied with probability no more than 2^{-L}. □

According to Theorem 11, verification of $\log_{g_1}|y_i| = \log_{g_2}|z_i|$ for $i = 1,2,\dots,n$ can be batched to verification of $\log_{g_1}\prod_{i=1}^{n} y_i^{t_i} = \log_{g_2}\prod_{i=1}^{n} z_i^{t_i}$. The probability that

$\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{n} z_i^{t_i}$ is correct while $\log_{g_1}|y_v| \neq \log_{g_2}|z_v|$ for some v in $\{1,2,\ldots,n\}$ is no more than 2^{-L}.

H.2.2 Batch ZK Proof-Verification of Equality of Logarithms with Common Exponent

The batch techniques in this subsection are designed to prove $\log_g y = \log_{a_i} d_i$ for $i = 1,2,\ldots,n$ where g is a generator of a cyclic group G and $y \in G$. In some applications $a_i \in G$ and $d_i \in G$ may be assumed in which case strict batch verification is applied. If $a_i \notin G$ or $d_i \notin G$, batch ZK proof-verification with loose batch verification is applied.

Strict Verification of Equality of Logarithms with Common Exponent

Unless specified, any multiplicative computation in this subsection occurs in G. The following theorem is designed to prove and verify $\log_g y = \log_{a_i} d_i$ for $i = 1,2,\ldots,n$ for $i = 1,2,\ldots,n$ where $a_i \in G$ and $d_i \in G$.

Theorem 11. Suppose $a_i \in G$ and $d_i \in G$ for $i = 1,2,\ldots,n$. Let L be a security parameter, t_i for $i = 1,2,\ldots,n$ be random integers where $t_i < 2^L < [G]$. If $\log_g y \neq \log_{a_v} d_v$ where $1 \leq v \leq n$, then $\log_g y = \log_{\prod_{i=1}^{n} a_i^{t_i}} \prod_{i=1}^{n} d_i^{t_i}$ is satisfied with a probability no more than 2^{-L}.
To prove Theorem 11, a lemma is proved first.

Lemma 5. If $\log_g y \neq \log_{a_v} d_v$ where $1 \leq v \leq n$, t_1, t_2, , t_{v-1}, t_{v+1}, t_{v+2}, , t_n are constant and $\log_g y = \log_{\prod_{i=1}^{n} a_i^{t_i}} \prod_{i=1}^{n} d_i^{t_i}$, then there is at most one L-bit solution to t_v.

Proof: If the lemma is not correct, the following two equations are satisfied simultaneously where $\log_g y \neq \log_{a_v} d_v$, $\langle t_v \rangle = \langle \hat{t}_v \rangle = L$ and $t_v \neq t'_v$.

$$\log_g y = \log_{\prod_{i=1}^{n} a_i^{t_i}} \prod_{i=1}^{n} d_i^{t_i} \tag{42}$$

$$\log_g y = \log_{\prod_{i=1}^{v-1} a_i^{t_i} a_v^{t'_v} \prod_{i=v+1}^{n} a_i^{t_i}} (\prod_{i=1}^{v-1} d_i^{t_i}) d_v^{t'_v} \prod_{i=v+1}^{n} d_i^{t_i} \tag{43}$$

Suppose $\log_g y = x$. Therefore,

$$(42) \Rightarrow (\prod_{i=1}^{n} a_i^{t_i})^x = \prod_{i=1}^{n} d_i^{t_i} \tag{44}$$

$$(43) \Rightarrow (\prod_{i=1}^{v-1} a_i^{t_i} \cdot a_v^{t'_v} \prod_{i=v+1}^{n} a_i^{t_i})^x = (\prod_{i=1}^{v-1} d_i^{t_i}) d_v^{t'_v} \prod_{i=v+1}^{n} d_i^{t_i} \qquad (45)$$

Dividing (45) by (44) yields $a_v^{x(t'_v - t_v)} = d_v^{t'_v - t_v}$ so that $(a_v^x/d_v)^{t'_v - t_v} = 1$. $a_v^x/d_v \in G$ as $a_v \in G$ and $d_v \in G$. Note that $t'_v \neq t_v$ and $t'_v, t_v < 2^L < [G]$, so $t'_v - t_v \neq 0 \text{mod}[G]$. So, $a_v^x/d_v = 1$. Therefore, $a_v^x = d_v$. Contradiction is found under the assumption that this lemma is not correct, so this lemma proves correct.

□

Proof of Theorem 12: Lemma 5 implis that among the $(2^L)^n$ possible combinations of t_i for $i = 1,2,\ldots,n$, at most $(2^L)^{n-1}$ of them can satisfy $\log_g y = \log_{\prod_{i=1}^n a_i^{t_i}} \prod_{i=1}^n d_i^{t_i}$ when $\log_g y \neq \log_{a_v} d_v$. So if $\log_g y \neq \log_{a_v} d_v$ and t_i for $i = 1,2,\ldots,n$ are randomly chosen, $\log_g y = \log_{\prod_{i=1}^n a_i^{t_i}} \prod_{i=1}^n d_i^{t_i}$ is satisfied with a probability no more than 2^{-L}. □

According to Theorem 12, verification of $\log_g y = \log_{a_i} d_i$ for $i = 1,2,\ldots,n$ can be batched to $\log_g y = \log_{\prod_{i=1}^n a_i^{t_i}} \prod_{i=1}^n d_i^{t_i}$ when $a_i, d_i \in G$ for $i = 1,2,\ldots,n$. The probability that $\log_g y = \log_{\prod_{i=1}^n a_i^{t_i}} \prod_{i=1}^n d_i^{t_i}$ while $\log_g y \neq \log_{a_v} d_v$ and $1 \leq v \leq n$ for some v in $\{1,2,\ldots,n\}$ is no more than 2^{-L}.

Loose Verification of Equality of Logarithms with Common Exponent

Unless specified, any multiplicative computation in this subsection occurs in G_1 with a modulus p. The following theorem is designed to prove and verify $a_i^{\log_{g_1} y} = \pm d_i$ for $i = 1,2,\ldots,n$.

Theorem 13. Suppose $a_i \in Z_p^*$ and $d_i \in Z_p^*$ for $i = 1,2,\ldots,n$. Let L be a security parameter, t_i for $i = 1,2,\ldots,n$ be random integers where $t_i < 2^L < q$. If $a_v^{\log_{g_1} y} \neq \pm d_v$ where $1 \leq v \leq n$, then $(\prod_{i=1}^n a_i^{t_i})^{\log_{g_1} y} = \prod_{i=1}^n d_i^{t_i}$ with a probability no more than 2^{-L}.

Theorem 13 can be proved in a way similar to the proof of Theorem 12. Extending the proof of Theorem 12 to the proof of Theorem 13 is just like extending the proof of

Theorem 10 to the proof of Theorem 11. Only an additional discussion of group membership is needed in the extension. According to Theorem 13, verification of $a_i^{\log_{g_1} y} = \pm d_i$ for $i = 1,2,\ldots,n$ can be batched to $(\prod_{i=1}^{n} a_i^{t_i})^{\log_{g_1} y} = \prod_{i=1}^{n} d_i^{t_i}$. The probability that $(\prod_{i=1}^{n} a_i^{t_i})^{\log_{g_1} y} = \prod_{i=1}^{n} d_i^{t_i}$ while $a_v^{\log_{g_1} y} = \pm d_v$ and $1 \leq v \leq n$ is no more than 2^{-L}

H.3 Summary

After batching, the four proof and verification protocols in this section are still implemented using standard three-move ZK proof-verification protocols. So they are still correct, specially sound and honest-verifier ZK.

I Discussion: Privacy Through Anonymity In Blockchains

Some techniques like group signature and ring signature are proposed to achieve anonymity in a certain set (also called set anonymity) and separate the transactions from their users. Those special digital signature techniques allow each member of a group (set) of users to sign a transaction-carrying message (e.g. a ballot in e-voting schemes) on behalf of the group such that regarding the identity of the signer the only available information is that it is a member of the group. In recent years, more and more anonymous applications are built in the blockchain networks, which start to benefit from those special signature protocols.

Group signatures were conceived to allow any member of a group to produce a signature on behalf of the group, enabling users to sign with the authority of the group, without revealing the specific signer's identity. The concept of a group signature is that a trusted group master or manager is responsible for setting up a 'group' of users, who can then each sign messages on behalf of the whole group, without revealing their individual identity. The group master holds a master key with the ability to reveal the signer of any signature generated by a group member in the past. As a result of this, group signatures offer the participants anonymity only under the condition that the group master does not choose to reveal the signer's identity.

To remove the trust on a central authority (namely the group master) and strengthen anonymity and flexibility, ring signatures were created to offer honestly participating users with 'unconditional anonymity' and are formed without a complex setup procedure or the requirement for a group manager. They simply require users to be part of an existing public key infrastructure. Ring signature based anonymous applications allow different sets of blockchain users to generate groups and signatures on the fly, without requiring any additional trust, at the cost of little added computational time. Ring signatures are constructed in a way that the ring can only be 'completed' and therefore verify correctly, if the signer has knowledge of some secret information, most commonly a private key corresponding to one of the public keys in the ring. In the signature generation algorithm, a number is generated at random for each of the other public keys in the ring, and then the signer uses the knowledge of their own private key, or some other 'trapdoor information' to close the ring. Ring signatures offer users a type of anonymity by hiding transactions within a set of others' transactions. If there are many users contributing very similar amounts to the ring, then the ring is said to have good liquidity, meaning the transactions can occur quickly, and also that transactions can be effectively mixed, with a strong resistance to attempted mixing analysis attacks.

Regarding applications like e-voting, group signature and ring signature have a limitation: it is impossible for third parties to know whether or not a series of messages have been signed by the same person --- the only way to link messages would be through the group manager, who would have to revoke the anonymity of the signer in the process. The linking of messages is an essential property in e-voting applications as any voter must be prevented from voting more than one time. Just imagine that voter in a voting group containing all the qualified voters may cast a vote for every voter and control the election. So, some variants of ring signatures have been proposed to address this limitation.

Linkable ring signature algorithms provide a scheme that allows users to sign on behalf of a group, still without revealing the individual signer's identity, but with the additional property that any signatures produced by the same signer, whether signing the same message or different messages, have an identifier, called a tag, linking the signatures. With this tag, third parties can efficiently verify that the signatures were produced by the same signer, without learning who that signer is.

Unique ring signatures have a tag that links signatures if and only if the signer, message, and ring are the same across the two signatures. This tag is constructed using the signer's private key, message, and description of the ring (most commonly a list of public keys), and enables both other ring members and third parties to observe whether or not two identical messages have been signed by the same ring member.

Possible applications of linkable ring signatures are applications restricting the number of times a user can access a resource or service. For example, in e-voting applications every voter can only vote once.

The centralized control mechanism in group signature is not consistent with the decentralized structure of blockchains, but ring signature may be useful in achieving voters' anonymity and privacy in blockchain e-voting schemes. Although some kinds of ring signature techniques like linkable rang signature and unique ring signatures can be employed to implement anonymous voting, the following drawbacks limit their applications in practice. Firstly, a preliminary setting up phase is needed to organise the group (set). There is also a large overhead when attempting to add or remove members from an established group, with the group manager or all group members required to perform a computationally expensive task.

Secondly, those special signature protocols rely on honesty of the members, in order to offer a desirable level of anonymity, which may fail when being attacked by dishonest members. Very often a large number of honestly participating users are needed. (Note that the larger the group is the stronger the achieved anonymity is.) Although ring signatures offer an acceptable level of anonymity to users, dishonest participants can remove themselves from the set of possible signers of a formerly signed message by simply revealing their own private key or tag based on the specific message and ring in question. This would allow third parties to trivially observe that the given user is not the signer of the original message. If all but one users in a ring choose to act adversarially and reveal their private keys or tags, then the remaining user is left only with the 'pseudonymity' of the employed blockchain. This is the so-called Sybil attack. To counteract Sybil attack and prevent an adversarial ring from revoking the exculpability of any honest member, we may have to implement a scheme with blinded signatures, whose side effect causing serious security concerns has been discussed in the previous reports.

Thirdly, ring signature needs expensive computational operations. A naive implementation of it is to employ ZK proof of partial knowledge, to prove the signature may be signed by the first member's private key or the second member's private key, …. or the last member's private key. The cost of this proof is linear in the number of the members and is intolerable in large groups necessary for strong anonymity. Although opimisations have been proposed to improve efficiency of ring signature, its cost in practical applications is still very high. For example, many groups, each only containing two members or subgroups can be arranged in a binary tree, so that the computational cost can be reduced to linear in the logarithm of the number of users. The optimised cost is still high in applications with a large number of users, especially when special additional operations are employed to achieve the linkability between the transactions of the same user.

Therefore, ring signature cannot replace mix network in our e-voting scheme as the latter is simpler, more robust and more efficient. Finally, even if armed with ring signature, a user (e.g. voter) still has to submit his transaction (e.g. vote) through a communication network. So, as emphasized before, to counteract the attack against anonymity by traffic analysis of the communication network, the users still need a mix network to shuffle their submissions no matter whether ring signature has been employed or not.

J Computational Cost and Efficiency Improvement

Computational overhead is an important issue we cannot ignore in any IT system, especially for a system employing a large number of cryptographic operations, whose computational cost affect efficiency of the whole system in a decisive way. There are usually the following three kinds of cryptographic operations.

- Hash function
- symmetric cipher
- public key cryptographic operations including encryption, digital signature and their derivatives

The first two kinds are usually efficient as they employ simple calculations like bit mapping, bit permutation, bit addition and bit shift. In comparison, public key crypto employs addition, subtraction, multiplication and exponentiations of integers hundreds of bits long, and so is much more costly. Therefore, efficiency analysis of crypto systems always focus on public key cryptographic computations. Among public key cryptographic computations, addition and subtraction are relatively efficient and their expenses are to some extend comparable to those of hash functions and symmetric ciphers. Multiplication is hundreds of times more costly than addition and subtraction even if a modulus is employed to control the size of integers and limit the increase of cost. When talking about multiplications of large integers we assume that a modulus is used unless specifically stated otherwise. Exponentiation is the most expensive and without a modulus to limit integer size it cannot be practically computed.

Usually, cryptographers measure cost and efficiency of a crypto system in terms of the number of modular multiplications and each modular exponentiation is regarded as a number of modular multiplications. Depending on how modular exponentiations are implemented by the employed hardwares and softwares, estimation of their cost varies. About 20 years ago, a modular exponentiation with an L-bit exponent is roughly counted as $1.5L$ multiplications according to the popular implementations at that time. Recently, with development of algorithms and hardwares, an exponentiation with an L-bit exponent is regarded as $2^{3-1} + L + L/(3+1)$ multiplications. Anyway, cost of an exponentiation is linear in the length of the exponent.

In complex cryptographic operations, exponentiations are often calculated aggregationally in the form of a product of a few exponentitations instead of separately. The cost of a product of n exponentiations with L-bit exponents is denoted as $ExpCost^n(L)$, which is usually implemented by special algorithms much more efficient than n separate exponentiations. In the beginning, $ExpCost^n(L)$ is roughtly estimated as $n + 0.5nL$ multiplications. Recently, it is implemented more efficiently by $2^{3-1}n + L + nL/(3+1)$ multiplications.

In practice, depending on the employed hardware and software, more efficient or less efficient performance may exist although the same order of magnitude is seldom changed. However, the difference within the same magnitude still may contain a large space and is sometimes non-negligible. That is why a test or experiment is needed to experience the actually cost.

Most efforts to improve efficiency of public key crypto operations focus on the length of exponents, the most decisive factor. Can the exponents be minimized to increase efficiency? The answer is that it depends as the functions of an exponent in a crypto system vary in different applications.

In some applications, the purpose of a long exponent is to exist as a secret and prevent an adversary from finding it by a search. (e.g. a secret exponent to resist the search for a discrete logarithm, a very common phenomenon in DL-based crypto systems) In this case, the exponent cannot be smaller, otherwise the DL assumption may fail and security of the whole system may be compromised. Moreover, due to the increasing speed of the search technologies, instead of becoming smaller, those secret exponents have to become larger and larger to countermeasure the searching attacks. Although 512-bit secret logarithms are accepable more than 10 years ago, at present a standard secret exponent to guarantee robustness of public key ciphers like an ElGamal private key is several thousand bits long.

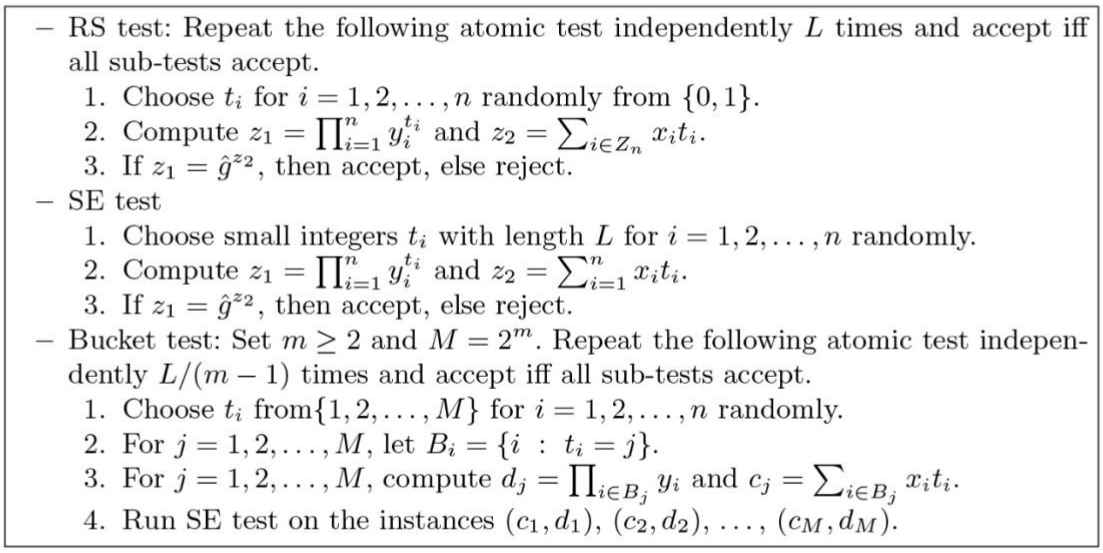
In other applications, the role of an exponent is to guarantee soundness of verification instead of robustness of public key ciphers. Such an exponent is L bits long such that an invalid operation can only pass the verification with a probability no larger than $1/2^L$. Although the larger L is the more sound the verification is, in practice L is not necessary to be as large as several hundred or even several thousand. When L=20, the probability of incorrect verification falls within one out of one million. When L=40, the probability of incorrect verification is smaller than one out of one billion and negligible for many real-world applications. In any application, an exponent scores of bits long is large enough as a soundness guarantee.

Bellare, Garay Rabin noticed the possibility of small exponents in verification operations for crypto computations and proposed three efficient verification protocols using small exponents in his paper 'Fast Batch Verification for Modular Exponentiation and Digital Signatures' at Eurocrypt 1998. This started the adoption of batch verification. Batch cryptology is a cryptographic technique, by which many instances of the same cryptographic operation can be performed in a batch (a single instance of operation achieving the same effect as many instances of separate operations), so that computational cost can be saved. The first practical batch cryptology scheme was proposed so that RSA encryption, decryption, signature generation and signature verification can be batched. For our e-voting scheme, batching ElGamal based cryptographic computations are more useful, especially for the costly verification operations.

Verification is a frequently applied operation in cryptographic applications and very often many instances of the same verification function appear simultaneously, so batch verification technology has a wide range of applications. Bellare gave the first definition of batch verification and proposed three batch protocols for verification of common-base exponentiations commonly used in ElGamal based cryptographic operations, which usually employs the following parameter setting. Let $\hat{p}$ be a prime, $\hat{q}$ a prime factor of $\hat{p}-1$ and cyclic group $\hat{G}$ with order $\hat{q}$ a subgroup of $Z_{\hat{p}}^*$. Let $\hat{g}_0$ be a generator of $Z_{\hat{p}}^*$ and $\hat{g}_1$ and $\hat{g}_2$ are generators of $\hat{G}$.

The target is to verify $y_i = \hat{g}^{x_i} \text{mod} \hat{p}$ where $x_i \in Z_{\hat{q}}$, $y_i \in Z_{\hat{p}}^*$ for $i = 1,2, \dots, n$. The normal solution is to calculate $\hat{g}^{x_i}$ for $i = 1,2, \dots, n$ and compare the results with y_i for $i = 1,2, \dots, n$, the computational cost of which is $nExpCost(\langle \hat{q} \rangle)$.

An intuitive idea to batch verify the n equations is to test $\prod_{i=1}^{n} y_i = \hat{g}^{\sum_{i=1}^{n} x_i}$. However this method is not sound since it is easy to pass the verification for an input containing a pair (x_i, y_i) where $y_i \neq \hat{g}^{x_i}$. For example, $y_1 = z\hat{g}^{x_1}$ and $y_2 = z^{-1}\hat{g}^{x_2}$ with any random z can pass the verification. This scheme is called *naive batch verification*. There are three batch protocols for common base verification: RS (random subset) test, SE (small exponent) test and bucket test. These three tests are illustrated in Figure 15. Unless specified, multiplication in the figure is computed modulo $\hat{p}$.



- RS test: Repeat the following atomic test independently L times and accept iff all sub-tests accept.
  1. Choose t_i for $i = 1, 2, \dots, n$ randomly from $\{0, 1\}$.
  2. Compute $z_1 = \prod_{i=1}^{n} y_i^{t_i}$ and $z_2 = \sum_{i \in Z_n} x_i t_i$.
  3. If $z_1 = \hat{g}^{z_2}$, then accept, else reject.
- SE test
  1. Choose small integers t_i with length L for $i = 1, 2, \dots, n$ randomly.
  2. Compute $z_1 = \prod_{i=1}^{n} y_i^{t_i}$ and $z_2 = \sum_{i=1}^{n} x_i t_i$.
  3. If $z_1 = \hat{g}^{z_2}$, then accept, else reject.
- Bucket test: Set $m \geq 2$ and $M = 2^m$. Repeat the following atomic test independently $L/(m-1)$ times and accept iff all sub-tests accept.
  1. Choose t_i from$\{1, 2, \dots, M\}$ for $i = 1, 2, \dots, n$ randomly.
  2. For $j = 1, 2, \dots, M$, let $B_i = \{i \ : \ t_i = j\}$.
  3. For $j = 1, 2, \dots, M$, compute $d_j = \prod_{i \in B_j} y_i$ and $c_j = \sum_{i \in B_j} x_i t_i$.
  4. Run SE test on the instances (c_1, d_1), (c_2, d_2), $\dots$, (c_M, d_M).

Figure 15. Bellare's Batch Verification of Exponentiations with a Common Base

The RS test does not improve efficiency as much as the other two. The bucket test is a variation of the SE test and more efficient when the batch is of greater size (all the pairs (x_i, y_i) for $i = 1,2, \dots, n$ are divided into buckets and SE test is performed in each bucket). We focus on the SE test but it can be replaced by the bucket test in all the theorems and applications.

It can be formally proved in a security theorem that if $y_i \in \hat{G}$ for $i = 1,2,\dots,n$, the SE test costs $n + L + nL/2 + ExpCost(\langle \hat{q} \rangle)$ multiplications and the probability that incorrect inputs can pass the verification is no more than 2^{-L}. Thus when L is 20, the failure probability is smaller than one in a million. Efficiency can be improved greatly as L is much smaller than the length of $\hat{q}$. A similar efficiency improvement can be achieved with the bucket test. We can use the SE test or bucket test together with a slightly modified DSS scheme to achieve efficient batch signature verification.

Although the SE test and bucket test are more secure than naive batch verification through use of the random t_i for $i = 1,2,\dots,n$, they are sound only under the assumption that $y_i \in \hat{G}$ for $i \in 1,2,\dots,n$. Otherwise an input containing a false pair (x_i, y_i) with $y_i \neq \hat{g}^{x_i}$ can still be generated to pass the batch verification. Careless users may ignore the impact of this assumption. Although the security theorem for the SE test is correct, its application to DSS verification is limited because there is no efficient way to verify $y_i \in \hat{G}$ (one exponentiation is needed). When $y_i \notin \hat{G}$, the probability for a false batch to pass the verification can be much greater than 2^{-L}. (When $y_i = \hat{g}_0^{(\hat{p}-1)/2} \hat{g}^{x_i}$, the probability is at least 1/2.) This shows that soundness and high efficiency cannot be achieved simultaneously in this application.

When the application can tolerate the limitation the idea can be extended to verify

$$y_i = \hat{g}_1^{x_{i,1}} \hat{g}_2^{x_{i,2}} \dots \hat{g}_k^{x_{i,k}} \mathrm{mod}\hat{p} \quad where \quad x_i \in Z_{\hat{q}}, y_i \in Z_{\hat{p}}^*, \quad for \quad i \in 1,2,\dots,n$$

by testing

$$\prod_{i=1}^{n} y_i^{t_i} = \hat{g}_1^{\sum_{i=1}^{n} x_{i,1} t_i} \hat{g}_2^{\sum_{i=1}^{n} x_{i,2} t_i} \dots \hat{g}_k^{\sum_{i=1}^{n} x_{i,k} t_i}$$

where $\hat{g}_1, \hat{g}_2, \dots, \hat{g}_k$ are generators of $\hat{G}$. When $k = 1$, this batch verification becomes the SE test. We notice that when more common bases are involved, efficiency can still be gained even though y_i must be verified to be in $\hat{G}$. As the number of common bases increases, the efficiency gained from their batch verification increases.

A more wide range of batch cryptology is to extend batch verification to batch ZK proof-verification. In the traditional batch verification schemes, there is only one player, the verifier, while no secret information is involved in the verification. In the extension, ZK proof and the corresponding verification are batched. In a ZK proof, a prover demonstrates to a verifier his knowledge of a secret value satisfying a certain relation. The verifier does not have knowledge of the secret. So there are two players, the prover and the verifier, and two operations, proof and verification. In the extended batching, both operations are batched.

First we explain the definition of batch ZK proof-verification. Suppose R is a Boolean relation with public inputs $x_1,\ x_2,,\ x_l$. For example, $R(x_1, x_2, x_3, x_4) = \text{TRUE}$ iff $\log_{x_1} x_3 = \log_{x_2} x_4$. As a hard problem (discrete logarithm) is included in function $R()$, it is infeasible for a verifier to compute $R()$ in polynomial time. A prover with knowledge of secret integers $z_1, z_2, \dots, z_m$ can help the verifier to test $R()$ efficiently. The prover P and the verifier V run an interactive polynomial-time proof-verification protocol T between prover P with input $x_{1,1}, x_{1,2}, \dots, x_{1,m_1}$ and verifier V with input $x_{2,1}, x_{2,2}, \dots, x_{2,m_2}$.

The protocol must be correct and sound, namely T outputs TRUE if and only if $R(x_1, x_2, \dots, x_l) = \text{TRUE}$. If V acts honestly in the protocol, V gets no help in computing the secrets $z_1, z_2, \dots, z_m$. This property is called *honest-verifier ZK*. If P has to prove to V that $R(x_{j,1}, x_{j,2}, \dots, x_{j,l}) = \text{TRUE}$ using secret integers $z_{j,1}, z_{j,2}, \dots, z_{j,m}$ for $j = 1,2,\dots,n$, normally protocol T must be run n times. To save computational cost, a batch proof-verification protocol BT between P with input $x_{j,1}, x_{j,2}, \dots, x_{j,l}, z_{j,1}, z_{j,2}, \dots, z_{j,m}$ for $j = 1,2,\dots,n$ and V with input $x_{j,1}, x_{j,2}, \dots, x_{j,l}$ for $j = 1,2,\dots,n$ can be employed.

Definition 3 *We say that BT defined as above is a batch proof-verification protocol if the following properties are satisfied.*

1. Correctness: if $R(x_{j,1}, x_{j,2}, \dots, x_{j,l}) = \text{TRUE}$ for $j = 1,2,\dots,n$, then BT outputs TRUE.
2. Soundness: if $R(x_{j,1}, x_{j,2}, \dots, x_{j,l}) = \text{FALSE}$ for any integer j in $\{1,2,\dots,n\}$, then BT outputs TRUE with a negligible probability.
3. Honest-verifier ZK: if V acts honestly in the protocol, V gets no help in computing the secrets $z_{j,1}, z_{j,2}, \dots, z_{j,m}$ for $j = 1,2,\dots,n$.

The essential nature of batch proof and verification is to replace full-length exponentiations with short-length exponentiations. Encryption and signature schemes must resist message revealment and signature forgery through brute-force attacks and so full-length (e.g. 4096 bits) exponentiations must be employed. Consequently, validity proof and verification of encryption (or re-encryption), decryption and signature also involve full-length exponentiations.

An example of batch proof and verication is batched proof and verification of equality of logarithms of common Base. G_1 is a cyclic subgroup of Z_p^* with order q where $p - 1 = 2q$ and $p,\ q$ are large primes. Let $g_1,\ g_2$ be generators of G_1 and g_0 a generator of Z_p^*. Unless specified, any multiplicative computation in this subsection occurs in G_1 with modulus p. The following theorem is used to prove $\log_{g_1} y_i = \log_{g_2} z_i$ where $y_i \in Z_p^*$ and $z_i \in Z_p^*$ for $i = 1,2,\dots,n$.

Theorem 5. Suppose $y_i \in Z_p^*$ and $z_i \in Z_p^*$ for $i = 1,2,\dots,n$. Let L be a security parameter and t_i satisfying $t_i < 2^L < q$ for $i = 1,2,\dots,n$ be random integers. If there exists integer v, such that $1 \le v \le n$ and $\log_{g_1} y_v \ne \log_{g_2} z_v$, then $\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} \ne \log_{g_2} \prod_{i=1}^{n} z_i^{t_i}$ with a probability no less than $1 - 2^{-L}$.

According to Theorem 5, verification of $\log_{g_1} y_i = \log_{g_2} z_i$ for $i = 1,2,\dots,n$ can be batched to verification of $\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{n} z_i^{t_i}$. The probability that $\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{n} z_i^{t_i}$ is correct while $\log_{g_1} y_v \neq \log_{g_2} z_v$ for some v in $\{1,2,dots,n\}$ is no more than 2^{-L}.

J.1 Proof of Theorem 5

To prove Theorem 5, a lemma is proved first.

Lemma 2 Suppose $y_i \in Z_p^*$ and $z_i \in Z_p^*$ for $i = 1,2,\dots,n$ and t_1, t_2, , t_{v-1}, t_{v+1}, t_{v+2} , … , t_n are constant. If $\log_{g_1} y_v \neq \log_{g_2} z_v$ with $1 \le v \le n$ and $\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{n} z_i^{t_i}$, then there is at most one possible L-bit solution for t_v.

Proof: If this lemma is not correct. Then the following two equations are satisfied simultaneously where $\log_{g_1} y_v \neq \log_{g_2} z_v$, $\langle t_v \rangle = \langle \hat{t}_v \rangle = L$ and $t_v \neq \hat{t}_v$.

$$\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{n} z_i^{t_i} \tag{34}$$

$$\log_{g_1} \prod_{i=1}^{v-1} y_i^{t_i} \cdot y_v^{\hat{t}_v} \prod_{i=v+1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{v-1} z_i^{t_i} \cdot z_v^{\hat{t}_v} \prod_{i=v+1}^{n} z_i^{t_i} \tag{35}$$

Subtracting (35) from (34) yields $\log_{g_1} y_v^{t_v - \hat{t}_v} = \log_{g_2} z_v^{t_v - \hat{t}_v}$. According to Formula-2,

$\log_{g_1} y_v^{t_v - \hat{t}_v} = \log_{g_2} z_v^{t_v - \hat{t}_v}$ and

so $(t_v - \hat{t}_v)\log_{g_1} y_v = (t_v - \hat{t}_v)\log_{g_2} z_v \bmod q$.

Note that $t_v - \hat{t}_v \neq 0 \bmod q$ because $1 \le \hat{t}_v, t_v < 2^L < q$ and $\hat{t}_v \neq t_v$.
Therefore, $\log_{g_1} y_v = \log_{g_2} z_v$. Contradiction is found under the assumption that this lemma is not correct, so this lemma proves correct. □

Proof of Theorem 5: Lemma 2 implies that among the $(2^L)^n$ possible combinations of $\{t_1, t_2, \dots, t_n\}$, at most $(2^L)^{n-1}$ of them can satisfy $\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{n} z_i^{t_i}$ when $\log_{g_1} y_v \neq \log_{g_2} z_v$. So if $\log_{g_1} y_v \neq \log_{g_2} z_v$ and t_i for $i = 1,2,\dots,n$ are randomly chosen, $\log_{g_1} \prod_{i=1}^{n} y_i^{t_i} = \log_{g_2} \prod_{i=1}^{n} z_i^{t_i}$ is satisfied with probability no more than 2^{-L}.