Platform-independent Secure Blockchain-Based Voting System

Bin Yu[1], Joseph Liu[1], Amin Sakzad[1], Surya Nepal[2], Ron Steinfeld[1], Paul Rimba[2], and Man Ho Au[3]

[1] Monash University Australia
[2] CSIRO Australia
[3] The Hong Kong Polytechnic University

Abstract. Cryptographic techniques are employed to ensure the security of voting systems in order to increase its wide adoption. However, in such electronic voting systems, the public bulletin board that is hosted by the third party for publishing and auditing the voting results should be trusted by all participants. Recently a number of blockchain-based solutions have been proposed to address this issue. However, these systems are impractical to use due to the limitations on the voter and candidate numbers supported, and their security framework, which highly depends on the underlying blockchain protocol and suffers from potential attacks (e.g., force-abstention attacks). To deal with two aforementioned issues, we propose a practical platform-independent secure and verifiable voting system that can be deployed on any blockchain that supports an execution of a smart contract. Verifiability is inherently provided by the underlying blockchain platform, whereas cryptographic techniques like Paillier encryption, proof-of-knowledge, and linkable ring signature are employed to provide a framework for system security and user-privacy that are independent from the security and privacy features of the blockchain platform. We analyse the correctness and coercion-resistance of our proposed voting system. We employ Hyperledger Fabric to deploy our voting system and analyse the performance of our deployed scheme numerically.

Keywords: evoting, blockchain, ring signature, homomorphic encryption

1 Introduction

Voting plays a significant role in a democratic society. Almost every local authority allots a significant amount of budget on providing a more robust and trustworthy voting system. Cryptographic techniques like homomorphic encryption and Mix-net [6] are usually applied in contemporary electronic voting systems to achieve the voting result verifiability while preserving voters' secrecy. However, incidents like a security flaw that has erased 197 votes from the computer database in the 2008 United States elections [32] and the compromise of 66,000 electronic votes in the 2015 New South Wales (NSW) state election in Australia [27] raise the public concerns on the security of electronic voting systems. For voting systems based on bulletin (e.g., [1, 7]), one of the major concerns is whether the voting result that is published on the bulletin can be trusted. Blockchain with the growing popularity and remarkable success in cryptocurrency provides a new paradigm to achieve the public verifiability in such electronic voting systems.

In a blockchain-based system, there is no trusted centralised coordinator; instead, each node that is involved in the blockchain system holds the data block locally. Based on the assumption that the decentralised consensus protocol is secure and a sufficiently large proportion of blockchain network nodes are honest, the blockchain can be thought of as a conceptual third party that can be trusted for correctness and availability [18]. The data on the blockchain is append-only and any operation that alters the data in any block violates the blockchain consensus rule and are rejected by the blockchain network.

Recently a number of blockchain-based electronic voting systems have been developed by exploiting its inherent features. These systems can be classified into three broad categories. (1) Cryptocurrency based voting systems (e.g., [20,31,38]). The ballots to a candidate are based on the payment he/she receives from the voters; the problem with such systems are malicious voters may refuse to "pay" the candidates to retain the money. Furthermore, a centralised trusted party, who coordinates the payment between the candidates and voters must exist. (2) Smart contract based voting system [23], which only supports two candidates and the voting is restricted to limited number of participants. Furthermore, it requires all voters to cast their ballots before reaching an agreement on the voting result. (3) Using blockchain as a ballot box to maintain the integrity of the ballots [9,33].

In summary, the security of these blockchain-based systems highly depends on the specific cryptocurrency protocol they employed. Additionally, these voting systems can only work with a specific blockchain platform, and support a limited number of candidates and voters. Based on our observations and studies, we believe that any blockchain-based voting systems should have the following three features: (1) Platform-independence — this means the changes in the underlying blockchain protocols should not affect the voting schemes. (2) Security framework — the voting system should be implemented with comprehensive security features (the detail of security features are discussed in Section 5). The nature of the blockchain allows everyone to obtain the data on it; thus, the comprehensive security features have critical importance to ensure that the ballots are fully secured on the blockchain. (3) Practical — it should be scalable, which means the large amount of ballots casting and tallying can be finished in a reasonable time.

Our Contributions: In this paper, we propose an electronic voting (evoting) system that supports the above identified three features as follows.

1. *Our voting system does not depend on a centralised trusted party for ballots tallying and result publishing.* Compared with traditional voting systems, which highly depend on a centralised trusted party to tally the ballots and publish the result, our voting system takes the advantage of a blockchain protocol to eliminate the need for a centralised trusted party. The details of the blockchain trustworthiness and voting system trust assumptions are discussed in Section 4 and 5, respectively.
2. *Our voting system is platform-independent and provides comprehensive security assurances.* Existing blockchain-based voting systems highly depend on the underlying cryptocurrency protocols. Receipt-freeness [28] and correctness of the voting result are hard to achieve (we analyse the blockchain-based voting system explicitly in Section 2). The security of our voting system is achieved by cryptographic techniques provided by our voting protocol itself, thus, our voting system can be deployed on any blockchain that supports smart contract. To achieve the goal of providing a comprehensive security, we employ Paillier system to enable ballots to be counted without leaking candidature information in the ballots. Proof-of-knowledge is employed to convince the voting system that the ballot casted by a voter is valid without revealing the content of the ballot. Linkable ring signature is employed to ensure that the ballot is from one of the valid voters, while no one can trace the owner of the ballot. The detail of security features that we achieved are discussed in Section 5.

3. *Our voting system is scalable and applicable.* In order to support voting scalability, we propose two optimised short linkable ring signature key accumulation algorithms given in algorithm 1 and algorithm 2 to achieve a reasonable latency in large scale voting. We evaluate our system performance with 1 million voters to show the feasibility of running a large scale voting with the comprehensive security requirements.

The rest of the paper is organised as follows: we discuss the cryptographic techniques applied in voting systems and analyse some typical voting systems in Section 2. Cryptographic primitives and our voting protocol are presented in Section 3 and 4, respectively. We analyse the correctness and security of our voting system in Section 5. In Section 6, we deploy our voting system and analyse its performance.

2 Related work

Secure evoting is considered as one of the most difficult problems in security literature as it involves many security requirements. To satisfy these security requirements, cryptographic techniques are mostly applied in constructing a secure evoting system. In this section, we discuss the existing voting systems based on traditional public bulletin and blockchain technology.

2.1 Public bulletin based voting systems:

In the following, we outline the key cryptographic techniques used in public bulletin based voting systems and how such techniques address the corresponding security requirements.

- **Homomorphic encryption:** Homomorphism feature allows one to operate on ciphertexts without decrypting them [11]. For a voting system, this property allows the encrypted ballots to be counted by any third party without leaking any information in the ballot [8,12,16]. Typical cryptosystems applied in a voting system are Paillier encryption [30, 37] and ElGamal encryption [17, 20].
- **Mix-net:** Mix-net was proposed in 1981 by Chaum [6]. The main idea of mix-net is to perform a re-encryption over a set of ciphertexts and shuffle the order of those ciphertexts. Mix node only knows the node that it immediately received the message from and the immediate destination to send the shuffled messages to. The voting systems proposed in [1,15,19] apply mix-net to shuffle the ballots from different voters, thus the authority cannot relate a ballot to a voter. For the mix-net based voting systems, they need enough amount of mix nodes and ballots to be mixed.
- **Zero-knowledge proof:** Zero-knowledge proof is often employed in a voting system [7, 26, 36] to let the prover to prove that the statement is indeed what it claimed without revealing any additional knowledge of the statement itself. In a voting system, the voter should convince the authority that his ballot is valid by proving that the ballot includes only one legitimate candidate without revealing the candidate information.
- **Blind signature and linkable ring signature:** Voting systems like [10, 14, 21] employ blind signature [10] to convince the tallying centre that the ballot is from a valid voter while not revealing the owner of the ballot. Simultaneously, the authority who signs the ballot learns nothing about the voter's selections. In blind signature, both voters and tallying centre must trust the signer. If the signer is compromised, the signature scheme may stop working. Unlike blind signature, linkable ring signature [22] is proposed to avoid the untrusted signer. Instead, it needs a certain number of voters

to participate in the signing process. By comparing the linkability tag, the authority can easily tell whether this voter has already voted. When the voter signs on the ballot, he/she needs to include other voters' public keys to make his/her signature indistinguishable from other voters' signatures.

2.2 Blockchain-based voting systems

The blockchain-based voting systems can be discussed under three broad categories as follows.

- **Voting systems using cryptocurrency:** In [38], authors propose a voting system based on Bitcoin. In their voting system, the ballot does not need to be encrypted/decrypted as they employ the protocol for lottery. Random numbers are used to hide the ballot that are distributed via zero-knowledge proof. Making deposit before voting may keep the voters to comply with the voting protocol while the malicious voters can still forfeit the voting by refusing to vote. However, only supporting "yes/no" voting may restrict the adoption of this voting system.
  In [31], authors proposed a voting system on the Zcash payment protocol [13] without altering the inner working of Zcash protocol. The voter's anonymity is ensured by the Zcash address schemes. The correctness of the voting is guaranteed by the trusted third-party and the candidates. In this system, the authority, who manages the Zcash and voter status database should be trusted. If the authority is compromised, double-voting or tracing the source of the ballot is possible.
- **Voting systems using smart contract:** In [23], the authors claim that their open voting network is the first implementation of a decentralised and self-tallying Internet voting protocol with maximum voter privacy using Blockchain. They employ smart contract as a public bulletin to achieve self-tallying.[4] However, their voting system can only work with two candidates voting (yes/no voting) and the limitation of 50 voters makes it impractical for a real large scale voting system.
- **Voting systems using blockchain as a ballot box:** Tivi and Followmyvote [9,33] are commercial voting systems which employ the blockchain as a ballot box [5]. They claim to achieve verifiability and accessibility anytime anywhere, while the voters' privacy protection in these systems is hard to evaluate.

To summarize, most traditional voting systems need a centralised trusted party to coordinate the whole voting process. In these systems, the centralised trusted party plays a critical role in storing the ballots, counting the ballots, and publishing the voting result. If it is compromised, the adversary can control the ballot counting and the whole voting result, and there is no efficient approach for participants to detect any compromises. Hence, there is a need of an independent public verifiability feature in such systems. Although the existing blockchain-based voting systems take advantage of blockchain public verifibility, the system security and user privacy of these systems depend on the features provided by the underlying blockchain platform, which are limited and thus make such systems vulnerable to a number of known attacks. Our proposed approach not only takes the benefits of a decentralised trust offered by the blockchain technology to remove the need of a centralised trusted party to do the ballots tallying, voting result decoding and publishing, but also considers key security primitives proposed in the literature including traditional evoting systems to build a practical platform independent secure

[4] Self-tallying means that after the casting phase, voters can count the ballots themselves.

[5] The authors call the storage of the ballot as the ballot box.

evoting protocol that can be deployed to any blockchain platforms that support smart contract.

3 Cryptographic Primitives

In this section, we describe the cryptography primitives borrowed from traditional voting systems and apply in our evoting system. Note that the syntaxes, correctness conditions, and security models of a linkable ring signature and a public key encryption are given in Appendix A and Appendix B, respectively.

3.1 Message encode and decode

Before the voting starts, we must encode the candidate ID to make it suitable for vote tallying. The encode/decode functions are defined as follows:

- **Candidate encoding:** We define $\zeta := \mathsf{Encode}(m) \in \mathbb{Z}$ as the candidate encoding function. For ρ candidates, each labeled with an ID from $\mathcal{P} = \{1, 2, \dots, \rho\}$, $\beta = 2^{\rho+1}$ be the basis of encoding operation. We encode the m^{th} candidate as $\zeta = \beta^m$ where $m \in \mathcal{P}$. We choose 2 as the basis of β as the division operation can be replaced by the CPU register right shift instruction to achieve a better performance.
- **Candidate decoding:** Let $k = k_t\beta^t + \cdots + k_n\beta^n + k_{n-1}\beta^{n-1} + \cdots + k_1\beta + k_0$ be the representation of k base β, $k \in \mathbb{Z}$, then we define the right shift k with n positions as $\mathsf{rsh}(k, n) = k_t\beta^{t-n} + k_{t-1}\beta^{t-n-1} + \cdots + k_{n+1}\beta + k_n$. Let $\mathsf{sum} = s_\rho\beta^{\rho-1} + s_{\rho-1}\beta^{\rho-2} + \cdots + s_2\beta + s_1$ be the addition of all the ballots where s_j is the total number of ballots that the candidate j^{th} acquires. We define $s_j := \mathsf{Decode}(\mathsf{sum}, j)$ for $1 \le j \le \rho$ and is computed as $s_j = \mathsf{rsh}(\mathsf{sum}, \beta^{j-1}) \bmod \beta$.

3.2 Paillier Encryption System [34]

Paillier Encryption system is employed to enable our voting system to tally the encrypted ballots. In our voting system, we implement the following functions in Paillier system and the detail of these functions are described in Appendix B.1.

- **Key Generation:** $(\mathsf{sk}_{\mathsf{Paillier}}, \mathsf{pk}_{\mathsf{Paillier}}) := \mathsf{Gen}_{\mathsf{Paillier}}(K_{\mathsf{len}})$ is the function to generate the secret key $\mathsf{sk}_{\mathsf{Paillier}}$ and the corresponding public key $\mathsf{pk}_{\mathsf{Paillier}}$ with the given key length K_{len}.The voting administrator invokes this function to generate the key pair and uploads the public key $\mathsf{pk}_{\mathsf{Paillier}}$ to the blockchain.
- **Encryption:** $C_{\mathsf{Ballot}} \leftarrow \mathsf{Enc}_{\mathsf{Paillier}}(\zeta_{\mathsf{Ballot}}, \mathsf{pk}_{\mathsf{Paillier}})$ where $\zeta_{\mathsf{Ballot}} \in \mathbb{Z}_n$ is the plaintext ballot to be encrypted. Voters download the $\mathsf{pk}_{\mathsf{Paillier}}$ from the blockchain and call this function to encrypt their ballots. This function generates the encrypted ballot C_{Ballot}.
- **Decryption:** $\zeta_{\mathsf{Res}} := \mathsf{Dec}_{\mathsf{Paillier}}(C_{\mathsf{Res}}, \mathsf{sk}_{\mathsf{Paillier}})$ where $C_{\mathsf{Res}} \in \mathbb{Z}_n^*$ is the encrypted voting result; the voting administrator invokes this function to decrypt the voting result.
- **Message Membership Proof of Knowledge [3]:** $\{v_j, e_j, u_j\}_{j\in\mathcal{P}} := \mathsf{PoK}_{\mathsf{mem}}(C_{\mathsf{Ballot}}, \Upsilon)$ where C_{Ballot} is the encrypted ballot, Υ is the set of the encoded candidates. When a voter publishes his/her ballot, he/she invokes this function to generate the proof $\{v_j, e_j, u_j\}_{j\in\mathcal{P}}$ that demonstrates his/her ballot encrypts only one of the encoded candidates in Υ.
- **Decryption Correctness Proof of Knowledge:** $(\zeta_{\mathsf{Res}}, r) := \mathsf{PoK}(C_{\mathsf{Res}}, \mathsf{sk}_{\mathsf{Paillier}})$, where ζ_{Res} is the plaintext and r is the random factor that generate the encrypted voting result C_{Res}. After publishing the voting result, the voting administrator invokes this function to generate a unique value pair $(\zeta_{\mathsf{Res}}, r)$ that constructs the C_{Res} to prove that he/she decrypts the voting result C_{Res} correctly.

3.3 Linkable Ring Signatures

Linkable ring signature (LRS) can be applied in our system to protect the privacy of the voters. In practice, we apply the short linkable ring signature (SLRS) [2] which extends all the SLRS features to make the signature size constant with the growth of voter numbers, it has the following features: (1) every ballot that is accepted by the system is from one of the valid users, (2) the voter can check whether his ballot is counted by the blockchain, (3) the size of the signature is constant to support scalability and (4) double-voting is prevented. In our voting system, we implement the function tuple ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link}$). The details of these functions are explained in Appendix A.2.

- **Setup:** $\mathsf{param} \leftarrow \mathsf{Setup}(\lambda)$ is a function that takes λ as the security parameter and generates system-wide public parameters param such as the group of quadratic residues modulo a safe prime product N (explained in Appendix A.2) denoted as $\mathsf{QR}(N)$, the length of the key, and a random generator $\tilde{g} \in \mathsf{QR}(N)$.
- **Key Generation:** $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ is a function to generate a key pair for each voter i.
- **Signature:** $\sigma \leftarrow \mathsf{Sign}(\mathcal{Y}, \mathsf{sk}, \mathsf{msg})$ is a function to generate the signature σ using all voters' public keys $\mathcal{Y} = \{y_1, y_2, \ldots, y_b\}$, the message msg to be signed, and the voter's secret key sk. Voters should invoke this function to sign on his/her encrypted ballot.
- **Verification:** $\mathsf{accept/reject} \leftarrow \mathsf{Verify}(\sigma, \mathcal{Y}, \mathsf{msg})$; our voting system invokes this function to test the validity of every ballot. Based on the input of the encrypted ballot itself, the voter's signature and all the voters' public keys, the blockchain accepts or rejects this ballot to be put on the chain.
- **Linkability:** $\mathsf{Link}(\sigma_1, \sigma_2) \rightarrow \mathsf{linked/unlinked}$. When a voter casts his/her vote, our voting system invokes this function to check whether this voter has already casted his vote. If this function returns linked, our voting system rejects this ballot; otherwise, the ballot is recorded on the chain.

3.4 Blockchain

Blockchain as a new scheme targets at removing the centralised trusted party or regulatory actors to achieve public verifiability. We employ these time-based blocks to store both ballots and the voting results.

There are two typical approaches to achieve consensus; one is based on proof-of-work (PoW) [25] and the other is based on Byzantine Fault Tolerance (BFT) network. To achieve better network scalability, we can deploy our voting system on the PoW based blockchain; the BFT based blockchain can be deployed to achieve better transaction processing performance. Because of the page limitation, we give a brief introduction of a Practical BFT (PBFT) protocol which is applied in our voting system.

PBFT protocol: PBFT protocol can tolerate any number of faults over the lifetime of the system provided fewer than 1/3 of the replicas become faulty [5]. Compared with the PoW protocol, PBFT based blockchain can achieve better performance (less network latency) while it has the restriction of node scalability [35]. There is a leader node accompanied by some validation nodes in PBFT network, When a transaction is submitted to the leader node, the PBFT protocol does the following:

- The leader orders the transaction candidates that should be included in a block and broadcasts the list of ordered transactions to all the validation nodes.
- When each of the validation node receives the list of transactions, it executes the transactions on that list one by one.

- The validation nodes calculate a hash value for this newly created block (hash value includes hashes for executed transactions and final state of this distributed system).
- Validation node broadcasts its hash value to other nodes in the network and starts listening to responses from them.
- If any node finds that 2/3 of all nodes broadcast the same hash value from the execution of ordered transactions list, it regards this block as a valid block and commits this new block to its local ledger.

3.5 Smart Contract

For the blockchain system, the "smart contract" is widely used as a general purpose computation that takes place on a blockchain. Smart contract enables interactions between end users and a blockchain by allowing end users to create/query data on the blockchain. We adopt Hyperledger Fabric [24] as a smart contract running environment in our voting system. To use the consistent terminologies, we call hyperledger chaincode as smart contract hereafter. We discuss the smart contract roles and deployment scheme as follows.

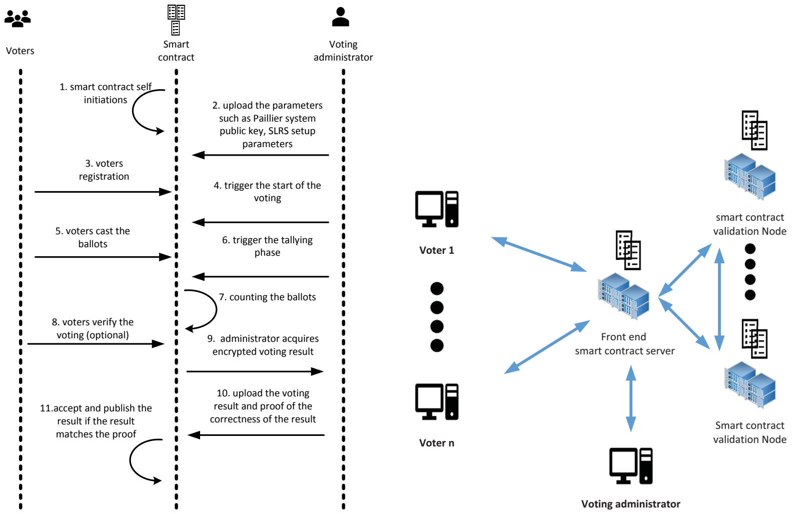
Smart Contract Deployment Scheme: For a practical smart contract, at least three interfaces should be implemented that are **init()**, **invoke()**, and **query()**. **init()** is the interface that is invoked when the smart contract is loaded. **init()** initialises the smart contract system parameters before end-user interacts with the smart contract. **query()** is the interface handling the query request from the blockchain end users. **invoke()** is the interface that is called when the end user wants to put the data on the blockchain. Unlike **query()**, **invoke()** is executed on all validation nodes to ensure the consistency of data on the blockchain.

A smart contract needs to be compiled before being deployed on the blockchain. The Hyperledger administrator is responsible for running the smart contract on every validation node and nominates one node as the front-end server to handle end users requests. The detail of smart contract deployment is discussed in Section 4.

4 The Voting Protocol

In this section, we first provide an overview of the whole voting protocol and then discuss each step in details. The whole voting process can be divided into 11 steps as shown in Fig. 1(a). Except the smart contract administrator who setups the voting smart contract, three entities are involved: voters, smart contract, and voting administrator. We take Bob as a valid voter in this section to show how the voting protocol works. First, the smart contract is initialised to prepare a voting. Then, the voting administrator uploads the voting parameters. After all voters register themselves in this voting and upload their SLRS public key to the blockchain (the SLRS secret keys are kept by voter themselves), the administrator triggers the start of the voting. Bob as a voter casts his ballot before the administrator triggers the tallying phase. It is optional for Bob to verify the tallying result before the administrator acquires the encrypted tallying result. The administrator needs to upload the voting result and the proof to the blockchain to show the correctness of the result to the voters and all the stakeholders. The smart contract verifies whether the result matches the proof uploaded by the administrator and finally publishes the decrypted voting result on the blockchain.

The voting system consists of one front-end smart contract and several validation nodes as shown in Fig. 1(b). The role of a validation node is to replicate the execution of the smart contract codes to ensure its correct execution. The role of the front-end is similar to the validation node except exporting RESTful API interfaces for communication with voters and administrator. For a practical voting, the validation nodes could be held by different entities/stakeholders, thus all ballots on the blockchain have been verified by different entities/stakeholders. As all the entities/stakeholders have the agreement on the data stored on the blockchain, the blockchain built on the servers owned by different entities can be regarded as trustworthy. It is impractical for the attackers to compromise most of the entities/stakeholders[6].


Voters
Smart contract
Voting administrator
1. smart contract self initiations
2. upload the parameters such as Paillier system public key, SLRS setup parameters
3. voters registration
4. trigger the start of the voting
5. voters cast the ballots
6. trigger the tallying phase
7. counting the ballots
8. voters verify the voting (optional)
9. administrator acquires encrypted voting result
10. upload the voting result and proof of the correctness of the result
11.accept and publish the result if the result matches the proof
Voter 1
Voter n
Front end smart contract server
smart contract validation Node
Smart contract validation Node
Voting administrator

(a) The voting protocol diagram. (b) The blockchain evoting system.

Fig. 1: The general voting protocol and how entities are connected.

4.1 Entities in the voting process

Four entities should be involved in our voting system shown in Fig. 1(a), and details are explained as follows:

- **smart contract administrator:** he/she has the ability to access the smart contract platform to deploy/terminate smart contract. In Hyperledger fabric, this account is authorised by the membership service and a permission is granted to deploy/terminate the Smart contract. In our voting system, we need at least one smart contract administrator to deploy the voting smart contract.
- **voting administrator:** The role of voting administrator is to organize the vote by setting up the voting parameters and triggering the the tallying and result publishing phase. Although there are underlying mechanisms in hyperledger to authenticate users, we use SLRS to prevent administrator from linking the ballots with the users.
- **smart contract:** The role of smart contract include 1.) store the encrypted ballots. 2.) verify the validity of the ballots. 3.) count the encrypted ballot. 4.) verify the correctness of the voting result. 5.) publish the voting result and provide the platform for the voters to verify the voting process.

[6] The number of entities to be compromised depends on the underlying consensus protocol.

- **voters:** Voters are the people who have the rights to cast their vote. They need to register into the voting system before they cast their vote.

4.2 Smart contract initiation

The smart contract is initialised by generating an RSA keypair $(\mathsf{pk}_s, \mathsf{sk}_s)$. This keypair is employed to sign/verify every transaction between the smart contract and the end users to avoid man-in-the-middle attacks. [7] Additionally, ensuring all the validation nodes run the identical smart contract is of critical importance; otherwise, validation nodes may tamper the smart contract with back doors and colludes with the adversary to get the ballots' information. We require the smart contract to be deployed as follows: Firstly, the smart contact accompanied with a digital fingerprint is verified by all the voting entities/stakeholders to ensure that there is no backdoor. This eliminates the possibility that an entity/stakeholder colludes with a node to run tampered smart contract. Then, this fingerprint is tagged as verified evoting smart contract fingerprint in blockchain platform. MD5, SHA1, and/or SHA2 hashes are mostly employed to generate the fingerprint for a given file [29]. Secondly, when a new node wants to join the evoting blockchain, the fingerprint of this smart contract is checked, if it is identical to the verified fingerprint, the smart contract is loaded, otherwise, is rejected by the blockchain platform.

4.3 Voting system set up

During the system set up, the voting administrator uploads the following three parameters to the blockchain:

- The public key of the Paillier system $\mathsf{pk}_{\mathsf{Paillier}}$.
- A set of encryptions of zero denoted as T, generated by the administrator's Paillier public key $\mathsf{pk}_{\mathsf{Paillier}}$. For voting with 1 million voters, we suggest the set should contain enough elements to make the randomised pool large enough and the detail of T is discussed in receipt-freeness analysis [8].
- The SLRS scheme parameters, param.

4.4 Voter registration

Bob must register this voting system with his identity. The registration information could be: (1) email address with a desired password, or (2) the identity number with a desired password, or (3) an invitation URL sent by the administrator with a desired password. After Bob passes the identity check conducted by the smart contract, he can login with the desired password to download the SLRS param and the administrator's Paillier public key $\mathsf{pk}_{\mathsf{Paillier}}$, then generates the SLRS key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ by calling $\mathsf{KeyGen}(\mathsf{param})$; Bob then uploads the public key pk_i to the smart contract (Bob's secret key is kept off-chain by himself). If the smart contract accepts his SLRS public key, the smart contract puts his public key pk_i on the blockchain to complete his registration phase.

[7] As we will discuss later, we only accept verified smart contract to run on validation node, thus, avoiding any malicious smart contract exporting the sk_s to anyone.

[8] To avoid requiring the administrator to upload the encryption of zero pool, coin flipping protocol can be applied to generate the consistent encryption of zero on all the validation nodes and this is our future work.

4.5 Vote casting phase

During this phase, Bob casts his vote as follows: (1) Bob chooses one of the candidates $m \in \mathcal{P}$ and encodes it as $\zeta := \mathsf{Encode}(m)$. (2) Bob invokes the Paillier encryption function $C \leftarrow \mathsf{Enc}_{\mathsf{Paillier}}(\zeta, \mathsf{pk}_{\mathsf{Paillier}})$. (3) Bob needs to prove that C is an encryption of an element in $\{\zeta_1, \ldots, \zeta_\rho\}$ (set of all encoded candidates) by calling $\{v_j, e_j, u_j\}_{j \in \mathcal{P}} := \mathsf{PoK}_{\mathsf{mem}}(C, \Upsilon)$; hence he sends $\pi = \{C, \{v_j, e_j, u_j\}_{j \in \mathcal{P}}\}$ to the smart contract.[9] (4) Upon receiving $\{v_j, e_j, u_j\}_{j \in \mathcal{P}}$, the smart contract verifies the validation of the encrypted ballot C. We denote ϕ as a mapping of this transaction's session id to T domain. If C is valid, the smart contract takes an encryption of zero at ϕ^{th} position. Let ϵ be the addition of $T[\phi]$ and C. The smart contract signs on ϵ denoted as s and sends (ϵ, s) back to Bob. (5) If Bob accepts s, he invokes $(v, \tilde{y}, \sigma') := \mathsf{Sign}(\epsilon, (\mathsf{pk}, \mathsf{sk}), \mathcal{Y})$ to generate the $\mathsf{Sign}_{\text{bob}}$ on ϵ and sends $(\epsilon, \mathsf{Sign}_{\text{bob}})$ to the smart contract. (6) If the smart contract detects that $\mathsf{Sign}_{\text{bob}}$ has already been recorded on the blockchain or cached in the memory, it rejects Bob's vote; otherwise, $(\epsilon, \mathsf{Sign}_{\text{bob}})$ is put on the blockchain.

4.6 Ballots tallying and result publishing

Due to the Paillier system's homomorphic feature, counting the vote is as simple as fetching the encrypted ballots from the blockchain and adding them together. The result publishing mechanism is described in the following steps: (1) Let E_{sum} be the sum of all the encrypted votes and Sign_s be the signature signed by the smart contract on E_{sum}. The smart contract sends $(E_{\mathsf{sum}}, \mathsf{Sign}_s)$ to the administrator. (2) The administrator invokes $\mathsf{sum} := \mathsf{Dec}_{\mathsf{Paillier}}(E_{\mathsf{sum}}, \mathsf{sk}_{\mathsf{Paillier}})$ to compute plaintext sum, which encodes the ballots of each candidate. The administrator also invokes $(\mathsf{sum}, r) := \mathsf{PoK}(E_{\mathsf{sum}}, \mathsf{sk}_{\mathsf{Paillier}})$ to calculate the random r that constructs this E_{sum}, and sends (sum, r) to the smart contract. (3) The smart contract verifies the correctness of (sum, r) by checking if $E_{\mathsf{sum}} \overset{?}{=} g^{\mathsf{sum}} r^n$ (g is one of the elements of $\mathsf{pk}_{\mathsf{Paillier}}$). (4) If the smart contract accepts the sum, it iteratively invokes $m := \mathsf{Decode}(\mathsf{sum}, i)$ to compute the ballots for each candidate i. Let Π be the dictionary holding the voting result of all candidates. The smart contract finally puts Π on the blockchain.

4.7 Ballot verifying

After tallying ballots and before the voting administrator decrypts the tallying result, the public can verify ballots on the blockchain to make sure the validity of the voting process. We define two roles for people who can verify the voting. The first one is those who have the right to access the data on the blockchain while they do not have the right to vote. The second one is those who have both rights to vote and access the data on the blockchain. The public verifiability to those who have first role is as follows 1) Checking the number of ballots that were counted and the number of people registered for this voting. 2) Checking the correctness of the tallying result by downloading and adding all the encrypted ballots to match with the tallying result published on the blockchain. Compared to those who have the first role, people assigned to the second role can also verify that his/her ballot is recorded on the blockchain by checking whether there exists one ballot that is signed with his/her signature; This ensure his/her vote is recorded and counted.

[9] Bob can choose none of the actual candidates by casting his ballot to a dummy candidate. When the smart contract publishes the voting result, it ignores all the ballots that the dummy candidate gets.

4.8 Blockchain validation nodes and the trustworthiness of blockchain

Under the assumption that the voting administrator will not disclose the Paillier secret key and the encryption of zero (this is discussed in Security and Coercion-Resistance Analysis section later), we discuss the role of the blockchain validation nodes and how they enhance the trustworthiness of the blockchain. The responsibilities of the validation nodes include (1) verify the validity of the ballots (check whether the ballots are from the same voters and/or check whether the given ballot encrypts only one candidate), (2) check the validity of the signature on the given ballot, (3) ballots tallying, and (4) verify the correctness of the voting result. If any interaction between the blockchain and the voting participants does not pass the verification conducted by the validation nodes, this interaction is rejected by the voting system.

In practice, we suggest enhancing the trustworthiness of the blockchain by allowing different political parties/stakeholders host their own validation nodes. The data on the blockchain is verified by different entities/stakeholders, and it is unlikely that these entities/stakeholders collude with each other to publish a false voting result.

4.9 Comparison with other non-blockchain-based voting protocols

Compared with other non-blockchain-based evoting protocols, our voting system can be differentiated using the following three security features. Firstly, there is no need for a centralised trusted party to tally the ballots and publish the result. Our trustworthiness is built on the assumption that it is impossible that most of the voting entities/stakeholders who own the blockchain validation servers collude with each other. Smart contract fingerprint guarantees that the smart contract which is deployed on the validation node is verified by all the voting entities/stakeholders and no one can replace that with a tampered one. Second, the correctness of the ballots processing (in vote casting phase) and tallying is achieved by asking all the validation nodes to verify the validity of the process; the blockchain network rejects the ballot if the validation nodes disagree on the verification of the ballot process. Third, we allow the voters to verify that his/her ballot is recorded and the correctness of the tallying. Additionally, voting entities/stakeholders can also verify the validity of the vote tally and the voting result.

5 Correctness and Security Analysis

5.1 Correctness analysis

The correctness of our voting system is guaranteed by the public verifiability provided by the smart contract and the proof of knowledge provided by the cryptographic schemes. More than that, the smart contract ensures the consistency of a transaction execution. Any inconsistencies generate an error which results in the rejection of the transaction. This means the voting participants can be assured that every transaction on the blockchain is verified and accepted by all participating nodes. This prevents compromised nodes from putting an invalid data on the blockchain unless the adversary can take control of a proportion of the nodes in the whole blockchain network [10].

[10] The number of nodes to be compromised depends on the underlying consensus protocol.

5.2 Security features of our voting system

- **Privacy:** The ballots on the public ledger are encrypted and only the voting administrator who initiates the voting can decrypt the ballots. This ensures that the tallying center can count the ballots without knowing the content of the ballots.
- **Anonymity:** The voters, candidates, or smart contract cannot tell the public key of the signer with a probability larger than $1/b$, where b is the number of the voters. This can be guaranteed by the anonymity property of the linkable ring signature (LRS) scheme. Details are explained in Appendix A.2.
- **Double-voting-avoided:** In our voting system, we take the advantage of linkability provided by the short ring signature scheme. This means it is infeasible for a voter to generate two signatures such that they are determined to be unlinked. Our system can detect whether the signatures are from the same voter. Hence, a voter can only sign on one ballot and cast his/her ballot only once. This can be guaranteed by the linkable property of the LRS scheme. Details are explained in Appendix A.2.
- **Slanderability-avoided:** A voter cannot generate a signature which is determined to be linked with another signature not generated by him/her. In other words, an adversary cannot fake other voters' signature. This can be guaranteed by the non-slanderability property of the LRS scheme. Details are explained in Appendix A.2.
- **Receipt-freeness:** Even if an adversary obtains a voter's secret key, the adversary cannot prove that this voter voted for a specific candidate. This is guaranteed by the addition of encryption of zero which provides additional randomness to the ciphertext which is unknown to the voter. Thus, even if the voter's secret key is disclosed, no one can prove his casted ballot. For our voting system, the security level of the receipt-freeness is affected by the size of the encryption of zero pool, as the voters can collude with each other to reconstruct the encryption of zero pool. One solution is increase the size of zero pool thus more voters is required to reconstruct the pool. Another solution is applying coin flipping protocol on all validation node to work out a consistent randomness encryption of zero for each valid ballot. We have taken the first approach in this paper with 4096 encryptions of zeros.
- **Public verifiability:** Anyone who has the relevant rights to access the blockchain can verify that all the ballots are counted correctly; moreover, voters can also verify whether their ballots have been recorded.
- **Correctness:** Proof-of-knowledge ensures the correctness of the voting process. Voting participants need to prove the correctness of the interactions with the blockchain. Even if some blockchain nodes are compromised, others can still verify whether the proof is correct.
- **Vote-and-Go:** Compared with the voting system proposed in [28], our voting system does not need the voter to trigger the tallying phase. Moreover, in our system, voters can also cast their vote and quit before the voting ends, unlike [23] which needs all voters to finish voting before tallying the ballots.

5.3 Security and Coercion-Resistance Analysis

To address the security and coercion-resistance, we make the following assumptions:

- The trustworthiness of the blockchain platform is achieved by allowing different stakeholders/entities to host the blockchain validation nodes (the details have already been discussed in Section 4.7).
- Voters cast their ballots in a secure terminal, which means it is assumed that no one stand behind a voter or uses digital devices to record the voting process. We do not take the physical voting environment security into our consideration.

- The possibility of an attacker to create a blockchain and apply a social engineering technique to launch a phishing attack is beyond our research scope.
- The administrator should not reveal the Paillier system sercret key and the encryption of zeros to anyone.
- Voters should cast their ballots by themselves. No one else can cast the ballot with a voter's identification except the voter himself.

We demonstrate the robustness of our system under two typical attacks below.

Man-in-the-Middle Attacks: Our voting system has strong resistance to this attack. First, as the voters and the smart contract both sign their messages and the voting data is encrypted, it is impossible for an adversary to forge the signature or alter the data on any parties involved in the transactions. Second, the public keys used for signature verification are all published on the blockchain, preventing the adversary from cheating any parties by replacing the original public key with the adversary's public key. The encryption of the ballot also eliminates the possibility of the ballot leakage.

Denial-of-Service (DoS) Attacks: DoS attack is feasible to launch since the network service is provided in a relatively centralised way. In addition, the servers have relatively limited processing ability for a large number of requests. Distributing the service on different nodes is one of the solutions to DoS attack as it is almost impossible for the adversary to compromise all the servers.

Coercion-Resistance Analysis: Coercion-Resistance means it is infeasible for the adversary to determine whether a coerced voter complies with the demand. Our voting system security features discussed in Section 5.2 make it impossible to launch the Ballots-buyer attack and Double voting attack. Additionally, our voting system is free from randomization attack.
For the Ballots-buyer attack, an attacker coerces a voter by requiring that he submits a randomly composed ballot. Under such circumstances, both the attacker and the voter have no idea about which candidate this voter casts the ballot for. The purpose of this attack is to nullify the ballots. For our system, it is impossible to launch this attack as the voter should prove that the ciphertext is one out of ρ encrypted candidates by calling $\{v_j, e_j, u_j\}_{j \in \mathcal{P}} := \mathsf{PoK}_{\mathsf{mem}}(\mathsf{Enc}_{\mathsf{Paillier}}(\zeta, \mathsf{pk}), \varUpsilon)$. Since $\varUpsilon$ is held by the smart contract, any ballot that is not the encryption of any element in $\varUpsilon$ is rejected and the voter is notified that this transaction is failed.

6 System deployment and Experiments

6.1 System deployment

Our voting system can be deployed in any blockchain platforms with smart contract capability and achieve the same level of security. There might be some other reasons to choose a particular platform such as voting latency and flexibility requirements. Different consensus protocols have significant impact on the blockchain network latency and node scalability [35]. If the ballots' confirmation latency is not a major issue for the voting system, the PoW-based blockchain system could be a good option to achieve maximum node scalability. Otherwise, a BFT-based blockchain platform is a better solution. In our scenario, we employ the BFT-based blockchain platform Hyperledger Fabric and deploy our voting system in a practical scenario.

6.2 Experiments and performance evaluation

We deploy our system in docker containers running on a desktop with 4 cores i5-6500 CPU and 8 GB DDR3 memory. We conduct 1 million voters voting process on the blockchain that consists of 4 validation nodes and 1 PBFT leader node. Each of the validation nodes runs in one dedicated container; thus, we run five docker containers to build our testing blockchain system. We set a voter's public key as 1024 and 2048 bits respectively and the Paillier key pairs as 1024 bits. The deployment pattern is shown in Fig. 1(b). We summarize the time spent on our employed cryptographic processes for 1 million voters' voting in Table 1.

Step	Time
generate T	$13,560$ms
bottom half key accumulation	< 34s
top half key accumulation	< 23ms
download parameters	4ms
upload ballots	≈ 776ms
tally	$3,850$ms
decode and publish	$< 2,000$ms

Table 1: Time consumed on each step.

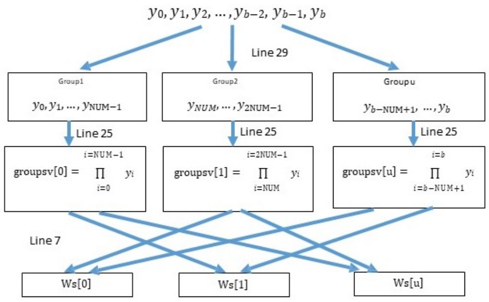

$y_0, y_1, y_2, \dots, y_{b-2}, y_{b-1}, y_b$
Line 29
Group1
$y_0, y_1, \dots, y_{\text{NUM}-1}$
Group2
$y_{NUM}, \dots, y_{2\text{NUM}-1}$
Group u
$y_{b-\text{NUM}+1}, \dots, y_b$
Line 25
Line 25
Line 25
groupsv[0] = $\prod_{i=0}^{i=\text{NUM}-1} y_i$
groupsv[1] = $\prod_{i=\text{NUM}}^{i=2\text{NUM}-1} y_i$
groupsv[u] = $\prod_{i=b-\text{NUM}+1}^{i=b} y_i$
Line 7
Ws[0]
Ws[1]
Ws[u]

Fig. 2: The diagram of Algorithm 1.

Voting parameters setting up time (administrator side): To initialise the voting, the administrator is responsible for uploading the voting parameters as discussed in Section 3. Let t_{cal} be the time taken to generate T, and t_{upload} be the time spent on uploading T to the blockchain. With 1024 bits key length, the pool size is 1MB. According to our test, t_{upload} is < 1 second and T_{cal} is about 14s. In conclusion, under 100MB bandwidth network, on the smart contract side, the majority of the time is spent on bottom half key accumulation, and on the administrator side, the most time-consuming phase is generating and uploading T (the pool of encryption of zeros).

SLRS parameter setting up time: Compared with LRS, SLRS enables the size of the signature constant no matter how many signers are involved in this signature. This feature is critical important for a large scale voting (i.e. the number of voters $> 100,000$ 1024-bits keys) as the signature should be constant to be suitable for storing on the blockchain. Compared with LRS, SLRS needs an extra step that accumulates all the signers' public keys. Let y_i be the public key of i^{th} voter and ψ be the SLRS public parameter for all voters. We define the key accumulation operation for all the voters' SLRS public keys for the i^{th} voter as $w_i = \psi^{y_1 y_2 \cdots y_{i-1} y_{i+1} y_{i+2} \cdots y_b}$. In order to make the time spent on key accumulation acceptable, we divide the key accumulation into two halves. The bottom half is run on smart contract and the top half is run by each voter.

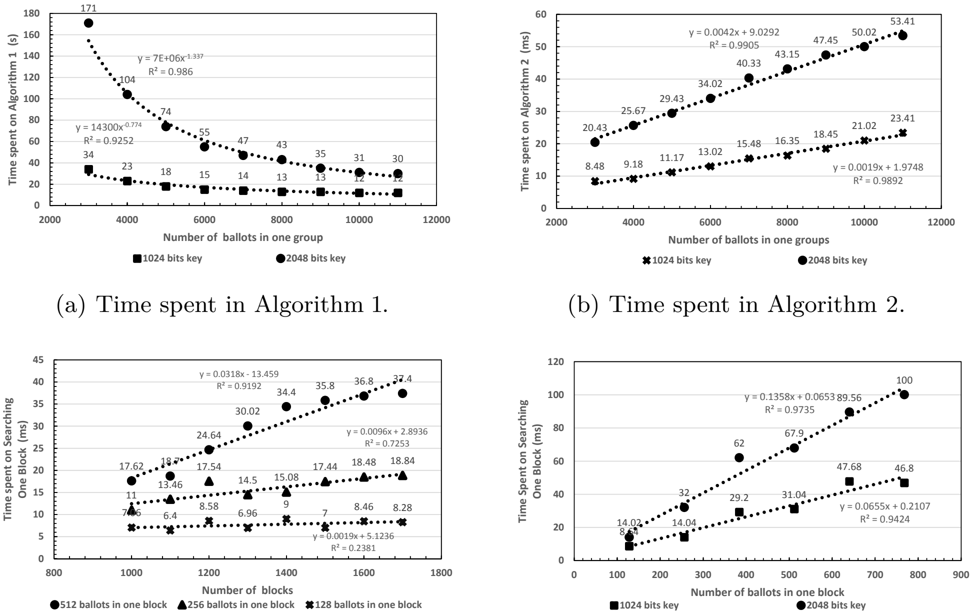
Bottom half time consumption (smart contract side): For the bottom half (shown in Appendix Algorithm 1), on the smart contract, we divide the voter SLRS public keys into m groups and pre-calculate the accumulation of all the public keys except the keys in the given group i and denote this key accumulation as ws_i. A diagram that shows how Algorithm 1 works is also given in Fig. 2. We only discuss a case in which the number of the voters is larger than 500; otherwise, the voters can generate the key accumulation themselves within a reasonable computation time. We denote G as the group that contains the voters' SLRS public key pk and f the public key accumulation function. We invoke an array operation function *append* to add an element into the array. We distribute the voters' SLRS public keys into u groups and each group has Num of keys (except the last group). We denote the array WS to store all ws, and gkeys to store the voters' SLRS public keys groups. The most time-consuming part is the multiplication of public keys for each group. In our implementation, we calculate the WS using four threads to save time. We evaluate the performance for 1 million voters' voting and the result is shown in Fig. 3(a). We find that the time spent on calculating WS decreases with the growth of the voter numbers in one group. For example, for 1024 bits length and 2048 bit length key accumulation, it decreases from 34s and 171s for the group that contains 3000 voters to 13s and 35s for the group that contains $8,000$ voters, respectively. This is due to 1) the time spent on running the exponential computation loop at line 7 in Algorithm 1 that dominates the time spent for the whole algorithm2.) For the 1 million voters' public key accumulation, we decrease the number of groups by increasing the number of the voters in a group to decrease the time spent on the loop at line 7 in Algorithm 1.

Top half time consumption (Voter side key accumulation): As shown in Appendix Algorithm 2, the voter downloads the array WS and the key group that his key belongs to in gkeys from the blockchain. The time spent on downloading these parameters is acceptable because of two reasons 1) the key size and the element size in WS are constant. 2) The number of groups is relatively small. For instance, if we have 1 million voters and we group 5000 voters in one group, and set the public key size as 1024 bits and N as 1024 bits, then the size of all the public keys in this group, denoted as $\mathcal{Y}'$, is approximately 624KB. The size of an element in WS is restricted by SLRS parameter N; thus with the parameters above, WS is 256KB. Therefore, the total size of $\mathcal{Y}'$ and WS is about 880KB. The voter only needs to do one exponential operation, regardless of the voting scale. From Fig. 3(b), it can be seen that with the key size of 1024 bits, it increase from 8.48ms for the group that contains 3000 voters to 16.35ms for the group that contains 8000 voters. The increase of time spent for the key accumulation on the voter's side can be explained as the increase of time spent at line 4 in Algorithm 2, as it dominates the total time spent for the voter side key accumulation.

For 1 million voter's voting, we could set the number of voters in one group smaller to reduce the time spent on the voter side for key accumulation. For the key length of 1024 bits, we recommend to set each group contains about 7000 voters so that it takes 14s and 15.48ms on the smart contract side and the voter side key accumulation, respectively.

The time spent on casting votes can be divided into three parts. The first part is the parameter preparation, that is, downloading the voting parameters from the server, denoted as t_1. The second part is the time spent for calculating the ring signature, denoted as t_2. The last part is the interactions between voters and the smart contract for uploading the ballots and performing the proof of correctness, denoted as t_3. t_1 can be evaluated roughly as the time spent on downloading WS and $\mathcal{Y}'$. For downloading the parameters of 1 million voting, it takes about 4ms under 100MB network (see Table 1). t_2 is approximately 15ms and the average value of t_3 in our test system is 776.60ms. In conclusion, it takes about 1s for a voter to cast his vote in a setting of 1 million voters' voting.

Ballot tallying and result publishing time: As the Paillier system restricts the length of each encrypted ballot within 2 times of the Paillier keypair size, the addition of encrypted ballot can achieve constant and reasonable performance. We test the time spent on an addition operation of encrypted ballots in a docker container. With 2048



(a) Time spent in Algorithm 1.

(b) Time spent in Algorithm 2.

(c) Time versus growth of ballots in a block.

(d) Time versus growth of voter numbers.

Fig. 3: SLRS public key accumulation and searching a block on a given blockchain in 1 million voters' voting

bits length cipher, it takes 3.85s to add 1 million encrypted ballots. The time spent on publishing the result is < 2s as we optimise the decoding algorithm by using shifting operation.

Ballot verification time: Our voting system provides the public verifiability. Compared with the time spent on checking the signature and tallying the result, the most time-consuming part is searching the block that contains a given ballot. The length of the blockchain and the size of each block have great impact on the search performance. We evaluate the time spent on these two factors as follows. For the first case, we set the evoting SLRS key size to 1024 bits length, and let each block contains 128 ballots, 256 blocks, and 512 blocks, respectively. It is observed in Fig. 3(c) that the time spent on these three blockchains increases linearly with the increase on the number of ballots in one block. For the second case, we set the total number of ballots to 1 million and the SLRS key size as 1024 bits long; it is observed in Fig. 3(d) that the time spent on searching one block grows linearly from 8.64ms in the blockchain that each block contains 128 ballots to 46.8ms in the blockchain that each block contains 768 ballots.

Based on the above experiments, it is clear that both the increases of the block size and chain length increase the time spent on searching one given block in the blockchain. For 1 million voters' voting system, the blockchain that consists of smaller blocks has better search performance. However, the drawback of the smaller block size is that it increases the number of searching operations (e.g., if we put all ballots in one block, users only searches once to get them; whereas if we allocate all of them in 10 blocks, users have to search 10 times to get them). Based on the experiment results shown in Fig. 3(c) and

Fig. 3(d), in practice, we recommend to set each block contains 640 ballots to achieve both a reasonable search time latency and the average number of search operations.

7 Conclusion

To solve the problems that the current blockchain voting system cannot provide the comprehensive security features, and most of them are platform dependent, we have proposed a blockchain-based voting system that the voters' privacy and voting correctness are guaranteed by homomorphic encryption, linkable ring signature, and PoKs between the voter and blockchain. We analyse the correctness and the security of our voting system. The experimental results show that our voting system achieves a reasonable performance even in a large scale voting.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: USENIX security symposium. vol. 17, pp. 335–348 (2008)
2. Au, M.H., Chow, S.S., Susilo, W., Tsang, P.P.: Short linkable ring signatures revisited. In: European Public Key Infrastructure Workshop. pp. 101–115. Springer (2006)
3. Baudron, O., Fouque, P.A., Pointcheval, D., Stern, J., Poupard, G.: Practical multi-candidate election system. In: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing. pp. 274–283. ACM (2001)
4. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: International Conference on Security in Communication Networks. pp. 268–289. Springer (2002)
5. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS) 20(4), 398–461 (2002)
6. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), 84–90 (1981)
7. Chow, S.S., Liu, J.K., Wong, D.S.: Robust receipt-free election system with ballot secrecy and verifiability. In: NDSS. vol. 8, pp. 81–94 (2008)
8. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. Transactions on Emerging Telecommunications Technologies 8(5), 481–490 (1997)
9. follow my vote. `https://followmyvote.com/`, accessed on June 24, 2017
10. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: International Workshop on the Theory and Application of Cryptographic Techniques. pp. 244–251. Springer (1992)
11. Gentry, C.: A fully homomorphic encryption scheme. Stanford University (2009)
12. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: Advances in CryptologyâĂŤEUROCRYPT 2000. pp. 539–556. Springer (2000)
13. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification. Tech. rep., Tech. rep. 2016-1.10. Zerocoin Electric Coin Company (2016)
14. Joaquim, R., Zúquete, A., Ferreira, P.: Revs–a robust electronic voting system. IADIS International Journal of WWW/Internet 1(2), 47–63 (2003)
15. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Proceedings of the 2005 ACM workshop on Privacy in the electronic society. pp. 61–70. ACM (2005)
16. Katz, J., Myers, S., Ostrovsky, R.: Cryptographic counters and applications to electronic voting. Advances in Cryptology Eurocrypt 2001 pp. 78–92 (2001)
17. Kiayias, A., Yung, M.: Self-tallying elections and perfect ballot secrecy. In: Public Key Cryptography. vol. 2274, pp. 141–158. Springer (2002)
18. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: Security and Privacy (SP), 2016 IEEE Symposium on. pp. 839–858. IEEE (2016)

19. Laskowski, S.J., Autry, M., Cugini, J., Killam, W., Yen, J.: Improving the usability and accessibility of voting systems and products. NIST Special Publication pp. 256,500 (2004)
20. Lee, B., Kim, K.: Receipt-free electronic voting scheme with a tamper-resistant randomizer. In: ICISC. vol. 2587, pp. 389–406. Springer (2002)
21. Li, C.T., Hwang, M.S., Lai, Y.C.: A verifiable electronic voting scheme over the internet. In: Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on. pp. 449–454. IEEE (2009)
22. Liu, J.K., Wong, D.S.: Linkable ring signatures: Security models and new schemes. In: International Conference on Computational Science and Its Applications. pp. 614–623. Springer (2005)
23. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. IACR Cryptology ePrint Archive 2017, 110 (2017)
24. Murphy, T.I.: hyperledger whitepaper, https://docs.google.com/document/d/1Z4M_qwILLRehPbVRUsJ3OF8Iir-gqS-ZYe7W-LE9gnE/edit#heading=h.m6iml6hqrnm2
25. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
26. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: Proceedings of the 8th ACM conference on Computer and Communications Security. pp. 116–125. ACM (2001)
27. Nsw election result could be challenged over ivote security flaw (2015), https://www.theguardian.com/australia-news/2015/mar/23/nsw-election-result-could-be-challenged-over-ivote-security-flaw
28. Okamoto, T.: Receipt-free electronic voting schemes for large scale elections. In: International Workshop on Security Protocols. pp. 25–35. Springer (1997)
29. Perrin, C.: Use md5 hashes to verify software downloads (2007), https://www.techrepublic.com/blog/it-security/use-md5-hashes-to-verify-software-downloads/
30. Ryan, P.Y.: Prêt à voter with paillier encryption. Mathematical and Computer Modelling 48(9), 1646–1662 (2008)
31. Tarasov, P., Tewari, H.: Internet voting using zcash. Cryptology ePrint Archive, Report 2017/585 (2017), http://eprint.iacr.org/2017/585
32. Theguardian: Why machines are bad at counting votes (2009), https://www.theguardian.com/technology/2009/apr/30/e-voting-electronic-polling-systems
33. Tivi voting. https://tivi.io/, accessed on June 24, 2017
34. Volkhausen, T.: Paillier cryptosystem: A mathematical introduction. In: Seminar Public-Key Kryptographie (WS 05/06) bei Prof. Dr. J. Blömer (2006)
35. Vukolić, M.: The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In: International Workshop on Open Problems in Network Security. pp. 112–125. Springer (2015)
36. Weber, S.: A coercion-resistant cryptographic voting protocol-evaluation and prototype implementation. Darmstadt University of Technology, http://www. cdc. informatik. tudarmstadt. de/reports/reports/StefanWeber. diplom. pdf (2006)
37. Xia, Z., Schneider, S.A., Heather, J., Traoré, J.: Analysis, improvement, and simplification of prêt à voter with paillier encryption. In: EVT'08 Proceedings of the Conference on Electronic Voting Technology (2008)
38. Zhao, Z., Chan, T.H.H.: How to vote privately using bitcoin. In: International Conference on Information and Communications Security. pp. 82–96. Springer (2015)

A Linkable Ring Signature

A.1 Syntax of Linkable Ring Signature

- $\mathsf{param} \leftarrow \mathsf{Setup}(\lambda)$ is a probabilistic polynomial time (PPT) algorithm which, on input a security parameter λ, outputs the set of security parameters param which includes λ. We denote by $\mathcal{EID}$, $\mathcal{M}$ and Σ the domains of event-id, messages and signatures, respectively.
- $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ is a PPT algorithm which, on input a security parameter $\lambda \in \mathbb{N}$, outputs a private/public key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. We denote by $\mathcal{SK}$ and $\mathcal{PK}$ the domains of possible private keys and public keys, respectively.
- $\sigma \leftarrow \mathsf{Sign}(e, n, \mathcal{Y}, \mathsf{pk}, M)$ which, on input event-id e, group size n, a set $\mathcal{Y}$ of n public keys in $\mathcal{PK}$, a private key whose corresponding public key is contained in $\mathcal{Y}$, and a message M, produces a signature σ.
- $\mathsf{accept}/\mathsf{reject} \leftarrow \mathsf{Verify}(e, n, \mathcal{Y}, M, \sigma)$ which, on input event-id e, group size n, a set $\mathcal{Y}$ of n public keys in $\mathcal{PK}$, a message-signature pair (M,σ) returns accept or reject. If accept, the message-signature pair is *valid.*
- $\mathsf{linked}/\mathsf{unlinked} \leftarrow \mathsf{Link}\ (e, n_1, n_2, \mathcal{Y}_1, \mathcal{Y}_2, M_1, M_2,, \sigma_1, \sigma_2)$ which, on input event-id e, group size n_1, n_2, two sets $\mathcal{Y}_1, \mathcal{Y}_2$ of n_1, n_2 public keys respectively, two valid signature and message pairs $(M_1, \sigma_1, M_2, \sigma_2)$, outputs linked or $\mathsf{unlinked}$.

Correctness. LRS schemes must satisfy:

- (Verification Correctness.) Signatures signed according to specification are accepted during verification.
- (Linking Correctness.) If two signatures are signed for the same event according to specification, then they are linked if and only if the two signatures share a common signer.

A.2 Notions of Security of Linkable Ring Signature

Security of LRS schemes has four aspects: unforgeability, anonymity, linkability and non-slanderability. Before giving their definition, we consider the following oracles which together model the ability of the adversaries in breaking the security of the schemes.

- $pk_i \leftarrow \mathcal{JO}(\bot)$. The *Joining Oracle*, on request, adds a new user to the system. It returns the public key $\mathsf{pk} \in \mathcal{PK}$ of the new user.
- $\mathsf{sk}_i \leftarrow \mathcal{CO}(\mathsf{pk}_i)$. The *Corruption Oracle*, on input a public key $\mathsf{pk}_i \in \mathcal{PK}$ that is a query output of $\mathcal{JO}$, returns the corresponding private key $\mathsf{sk}_i \in \mathcal{SK}$.
- $\sigma' \leftarrow \mathcal{SO}(e, n, \mathcal{Y}, pk_\pi, M)$. The *Signing Oracle*, on input an event-id e, a group size n, a set $\mathcal{Y}$ of n public keys, the public key of the signer $\mathsf{pk}_\pi \in \mathcal{Y}$, and a message M, returns a valid signature σ'.

If the scheme is proven in random oracle model, a random oracle is simulated.

1. Unforgeability. Unforgeability for LRS schemes is defined in the following game between the Simulator $\mathcal{S}$ and the Adversary $\mathcal{A}$ in which $\mathcal{A}$ is given access to oracles $\mathcal{JO}$, $\mathcal{CO}$, $\mathcal{SO}$ and the random oracle:
   (a) $\mathcal{S}$ generates and gives $\mathcal{A}$ the system parameters param.
   (b) $\mathcal{A}$ may query the oracles according to any adaptive strategy.
   (c) $\mathcal{A}$ gives $\mathcal{S}$ an event-id $e \in \mathcal{EID}$, a group size $n \in \mathbb{N}$, a set $\mathcal{Y}$ of n public keys in $\mathcal{PK}$, a message $M \in \mathcal{M}$ and a signature $\sigma \in \Sigma$.

$\mathcal{A}$ wins the game if:
(1) $\mathsf{Verify}(e, n, \mathcal{Y}, M, \sigma) = \mathsf{accept}$;
(2) All of the public keys in $\mathcal{Y}$ are query outputs of $\mathcal{JO}$;
(3) No public keys in $\mathcal{Y}$ have been input to $\mathcal{CO}$; and
(4) σ is not a query output of $\mathcal{SO}$.
We denote by

$$\mathbf{Adv}_{\mathcal{A}}^{unf}(\lambda) = \Pr[\mathcal{A} \text{ wins the game }]$$

Definition 1 (unforgeability). *A LRS scheme is unforgeable if for all PPT adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{unf}(\lambda)$ is negligible.*

2. Anonymity. It should not be possible for an adversary $\mathcal{A}$ to tell the public key of the signer with a probability larger than $1/n$, where n is the cardinality of the ring, even assuming that the adversary has unlimited computing resources.
Specifically, anonymity for LRS schemes is defined in the following game between the Simulator $\mathcal{S}$ and the unbounded Adversary $\mathcal{A}$ in which $\mathcal{A}$ is given access to oracle $\mathcal{JO}$.
(a) $\mathcal{S}$ generates and gives $\mathcal{A}$ the system parameters param.
(b) $\mathcal{A}$ may query $\mathcal{JO}$ according to any adaptive strategy.
(c) $\mathcal{A}$ gives $\mathcal{S}$ an event-id $e \in \mathcal{EID}$, a group size $n \in \mathbb{N}$, a set $\mathcal{Y}$ of n public keys in $\mathcal{PK}$ such that all of the public keys in $\mathcal{Y}$ are query outputs of $\mathcal{JO}$, a message $M \in \mathcal{M}$. Parse the set $\mathcal{Y}$ as $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$. $\mathcal{S}$ randomly picks $\pi_R \in \{1, \ldots, n\}$ and computes $\sigma_\pi = \mathsf{Sign}(e, n, \mathcal{Y}, sk_\pi, M)$, where sk_π is a corresponding private key of pk_π. σ_π is given to $\mathcal{A}$.
(d) $\mathcal{A}$ outputs a guess $\pi' \in \{1, \ldots, n\}$.
We denote by

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Anon}}(\lambda) = \left| \Pr[\pi' = \pi] - \frac{1}{n} \right|$$

Definition 2 (Anonymity). *A LRS scheme is anonymous if for any adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Anon}}(\lambda)$ is zero.*

3. Linkability.
Linkability for LRS schemes is mandatory, that is, it should be infeasible for a signer to generate two signatures such that they are determined to be unlinked using LRS.Link. The following definition/game essentially captures a scenario that an adversary tries to generate two LRS signatures, using strictly fewer than 2 user private keys, so that these two signatures are determined to be unlinked using LRS.Link. If the LRS scheme is unforgeable (as defined above), then these signatures can only be generated if at least 2 user private keys are known. If less than 2 user private keys are known, then there must be one common signer to both of the signatures. Therefore, this model can effectively capture the definition of linkability.

Linkability for LRS scheme is defined in the following game between the Simulator $\mathcal{S}$ and the Adversary $\mathcal{A}$ in which $\mathcal{A}$ is given access to oracles $\mathcal{JO}$, $\mathcal{CO}$, $\mathcal{SO}$ and the random oracle:
(a) $\mathcal{S}$ generates and gives $\mathcal{A}$ the system parameters param.
(b) $\mathcal{A}$ may query the oracles according to any adaptive strategy.

(c) $\mathcal{A}$ gives $\mathcal{S}$ an event-id $e \in \mathcal{EID}$, group sizes $n_1, n_2 \in \mathbb{N}$ (w.l.o.g. we assume $n_1 \leq n_2$), sets $\mathcal{Y}_1$ and $\mathcal{Y}_2$ of public keys in pk of sizes n_1 and n_2 resp., messages $M_1, M_2 \in \mathcal{M}$ and signatures $\sigma_1, \sigma_2 \in \Sigma$.

$\mathcal{A}$ wins the game if

(1) All public keys in $\mathcal{Y}_1 \cup \mathcal{Y}_2$ are query outputs of $\mathcal{JO}$;
(2) $\mathsf{Verify}(e, n_i, \mathcal{Y}_i, M_i, \sigma_i) = \mathsf{accept}$ for $i = 1, 2$ such that σ_i are not outputs of $\mathcal{SO}$;
(3) $\mathcal{CO}$ has been queried less than 2 times (that is, $\mathcal{A}$ can only have at most 1 user private key); and
(4) $\mathsf{Link}(\sigma_1,\ \sigma_2) = \mathsf{unlinked}$.

We denote by

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Link}}(\lambda) = \Pr[\mathcal{A} \text{ wins the game}].$$

Definition 3 (Linkability). *A LRS scheme is linkable if for all PPT adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{Link}$ is negligible.*

4. <u>Non-Slanderability.</u>

Non-slanderability ensures that no signer can generate a signature which is determined to be linked by LRS.Link with another signature which is not generated by the signer. In other words, it prevents adversaries from framing honest users.

Non-Slanderability for LRS schemes is defined in the following game between the Simulator $\mathcal{S}$ and the Adversary $\mathcal{A}$ in which $\mathcal{A}$ is given access to oracles $\mathcal{JO}$, $\mathcal{CO}$, $\mathcal{SO}$ and the random oracle:

(a) $\mathcal{S}$ generates and gives $\mathcal{A}$ the system parameters param.
(b) $\mathcal{A}$ may query the oracles according to any adaptive strategy.
(c) $\mathcal{A}$ gives $\mathcal{S}$ an event e, group size n, a message M, a set of n public keys $\mathcal{Y}$, the public key of an insider $\mathsf{pk}_\pi \in \mathcal{Y}$ such that pk_π has not been queried to $\mathcal{CO}$ or has not been included as the insider public key of any query to $\mathcal{SO}$. $\mathcal{S}$ uses the private key sk_π corresponding to pk_π to run $\mathsf{Sign}(e, n, \mathcal{Y}, sk_\pi, M)$ and to produce a signatures σ' given to $\mathcal{A}$.
(d) $\mathcal{A}$ queries oracles with arbitrary interleaving. Except pk_π cannot be queries to $\mathcal{CO}$, or included as the insider public key of any query to $\mathcal{SO}$. In particular, $\mathcal{A}$ is allowed to query any public key which is not pk_π to $\mathcal{CO}$.
(e) $\mathcal{A}$ delivers group size n^*, a set of n^* public keys $\mathcal{Y}^*$, a message M^* and a signature $\sigma^* \neq \sigma'$.

$\mathcal{A}$ wins the game if

(1) $\mathsf{Verify}(e, n^*, \mathcal{Y}^*, M^*, \sigma^*) = \mathsf{accept}$;
(2) σ^* is not an output of $\mathcal{SO}$;
(3) All of the public keys in $\mathcal{Y}^*, \mathcal{Y}$ are query outputs of $\mathcal{JO}$;
(4) pk_π has not been queried to $\mathcal{CO}$; and
(5) $\mathsf{Link}(\sigma^*,\ \sigma') = \mathsf{linked}$.

We denote by

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{NS}}(\lambda) = \Pr[\mathcal{A} \text{ wins the game}].$$

Definition 4 (Non-Slanderability). *A LRS scheme is* non-slanderable *if for all PPT adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{NS}$ is negligible.*

A.3 Short Linkable Ring Signatures [2]

SLRS schemes are described by the tuple $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link})$. For our SLRS, we define N as a safe prime product if $N = pq = (2p' + 1)(2q' + 1)$ for some primes p, q, p', and q' such that p' and q' are of the same length. We further denote by $\mathsf{QR}(N)$ the group of quadratic residues modulo a safe prime product N.

- **Setup:** $\mathsf{param} \leftarrow \mathsf{Setup}(\lambda)$ is a function that takes λ as the security parameter and generates system-wide public parameters param such as the group $\mathsf{QR}(N)$, the length of the key, and a random generator $\tilde{g} \in \mathsf{QR}(N)$.
- **Key Generation:** $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ is a function to generate key pair for each voter i. This function generates $(\mathsf{sk}_i, \mathsf{pk}_i) = ((p_i, q_i), y_i \in \mathcal{Y})$ for voter i where $y_i = 2p_iq_i + 1$.
- **Signature:** $\sigma \leftarrow \mathsf{Sign}(\mathcal{Y}, \mathsf{sk}, \mathsf{msg})$ is a function to generate the signature σ using all voters' public keys $\mathcal{Y} = \{y_1, y_2, \ldots, y_b\}$, the message to be signed, $\mathsf{msg} \in \{0,1\}^*$, and the voter's secret key sk. The output of this function is the accumulation of public keys v, the linkable tag $\tilde{y}$ and the signature σ'. For a voter i, the generation of SLRS is described below:
  1. Let $\psi \in \mathsf{QR}(N)$ be the public SLRS parameter that is generated once for all voters. We compute the witness w_i for voter i according to Algorithm 1 and Algorithm 2 in Section VI .
  2. We select parameters ℓ and u according to methods in [4]. Let $y \in S(2^\ell, 2^u)$ denote $|y - 2^\ell| < 2^\mu$. We compute the signature of message $\mathsf{msg} \in \{0,1\}^*$ for
  $$\mathsf{SPK}\left\{\begin{pmatrix} w & y \\ p & q \end{pmatrix} : \begin{array}{l} (w^y = v \bmod N) \wedge (y = 2pq + 1) \wedge \\ (y \in S(2^\ell, 2^u)) \wedge (q \in S(2^{\ell/2}, 2^u)) \wedge \\ (\tilde{y} = \tilde{g}^{p+q} \bmod N) \end{array}\right\}(\mathsf{msg}) \tag{1}$$
  3. Let σ' be the result of signatures based on proofs of knowledge (SPK). Note that the voter's linkability tag $\tilde{y}$ is uniquely determined by the voter's secret key. The result of the Sign function is $\sigma = (v, \tilde{y}, \sigma')$.
- **Verification:** $\mathsf{accept/reject} \leftarrow \mathsf{Verify}(\sigma, \mathcal{Y}, \mathsf{msg})$ is a function to verify that the signature of $\mathsf{msg} \in \{0,1\}^*$ is from one of the valid voters and this voter has only voted once. With the given key set $\mathcal{Y}$, message msg, and signature σ, the verifier checks whether
  $$v \stackrel{?}{=} \psi^{\prod_{y \in \mathcal{Y}} y} \bmod N$$
  and the validity of σ' respect to the SPK represented in Equation (1). If these two conditions hold, the Verify function returns accept otherwise reject.
- **Linkability:** $\mathsf{Link}(\sigma_1, \sigma_2) \rightarrow \mathsf{linked/unlinked}$ is a function to test the linkability of the given signatures σ_1 and σ_2. It extracts their respective linkability tags $\tilde{y_1}$ and $\tilde{y_2}$ from σ_1 and σ_2, respectively and returns linked if they are signed by the same person or $\mathsf{unlinked}$ otherwise.

Theorem 1. *Under the assumptions that Decisional Diffie-Hellman (DDH) problem over* $\mathsf{QR}(N)$*, the Link Decisional RSA (LD-RSA) problem, and the strong RSA (SRSA) are hard, the SLRS construction of A.3 is not only unforgeable in the random oracle model but also linkably-anonymous and non-slanderable w.r.t. the definition 1 to definition 4.*

Proof. The proof is given in [2].

B Public-Key Encryption

Syntax: A public-key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms and a finite message space $\mathcal{M}$ (which we assume to be efficiently recognizable). The key generation algorithm Gen outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where pk also defines a randomness space $\mathcal{R} = \mathcal{R}(\mathsf{pk})$. The encryption algorithm Enc, on input pk and a message $m \in \mathcal{M}$, outputs an encryption $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ of m under the public key pk. If necessary, we make the used randomness of encryption explicit by writing $c := \mathsf{Enc}(\mathsf{pk}, m; r)$, where $r \xleftarrow{\$} \mathcal{R}$ and $\mathcal{R}$ is the randomness space. The decryption algorithm Dec, on input sk and a ciphertext c, outputs either a message $m = \mathsf{Dec}(\mathsf{sk}, c) \in \mathcal{M}$ or a special symbol $\perp \notin \mathcal{M}$ to indicate that c is not a valid ciphertext.

Correctness: We call a public-key encryption scheme is correct if

$$\mathsf{E}[\max_{m \in \mathcal{M}} \Pr[\mathsf{Dec}(\mathsf{sk}, c) \neq m | c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)]] \leq \sigma,$$

where the expectation is taken over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$.

Security: Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with message space $\mathcal{M}$. We define the indistinguishable against Chosen-Plaintext Attacks (IND-CPA) game is shown as below, and the IND-CPA advantage function of an adversary $A = (A_1, A_2)$ against PKE (such that A_2 has binary output) as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CAP}}_{\mathsf{PKE}}(A) := \left| \Pr[\mathsf{IND\text{-}CAP}^{A} \Rightarrow 1] - \frac{1}{2} \right|.$$

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
2. $b \xleftarrow{\$} \{0, 1\}$
3. $(m_0^*, m_0^*, st) \leftarrow A_1(\mathsf{pk})$
4. $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b^*)$
5. $b^{'} \leftarrow A_2(\mathsf{pk}, c^*, st)$
6. **return** $[[b^{'} = b]]$

B.1 Paillier Encryption System [34]

Let $G = \mathbb{Z}^*_{n^2}$ and g be a random element from G, the Paillier encryption system is a randomised encryption scheme that encrypts the message msg by raising basis g to the power of msg and randomises it by a random factor. Given public key and the encryptions of msg_1 and msg_2, one can compute the encryption of $msg_1 + msg_2$ without knowing msg_1 and msg_2. We use $\gcd(v, w)$ and $\mathrm{lcm}(v, w)$ to denote the greatest common divisor and least common multiple of two values v and w, respectively. The quotient of a divided by b is denoted by $a \div b$.

- **Key Generation:** $(\mathsf{sk}_{\mathsf{Paillier}}, \mathsf{pk}_{\mathsf{Paillier}}) := \mathsf{Gen}_{\mathsf{Paillier}}(K_{\mathsf{len}})$ is the function to generate the secret key $\mathsf{sk}_{\mathsf{Paillier}}$ and the corresponding public key $\mathsf{pk}_{\mathsf{Paillier}}$ with the given key length K_{len}. We choose two large prime numbers p and q randomly and independently of each other and make sure $\gcd(p, q-1) = \gcd(p-1, q) = 1$. Let $\lambda = \mathrm{lcm}(p-1, q-1)$ and $L(b) = \frac{b-1}{n}$, where $b \in \mathbb{Z}^*_{n^2}$ and $n = p \cdot q$. Select random integer g where $g \in \mathbb{Z}^*_{n^2}$ and compute $\mu = (L(g^{\lambda} \bmod n^2))^{-1} \bmod n$. The public key is $\mathsf{pk}_{\mathsf{Paillier}} = (n, g)$ and the secret key is $\mathsf{sk}_{\mathsf{Paillier}} = (\lambda, \mu, p, q)$. We store p and q in our secret key as we will use these parameters to prove the correctness of the decryption in Paillier cryptosystem.

- **Encryption:** $C \leftarrow \mathsf{Enc}_{\mathsf{Paillier}}(\zeta, \mathsf{pk}_{\mathsf{Paillier}})$ Let $\zeta \in \mathbb{Z}_n$ be the plaintext to be encrypted. Select random $r \in \mathbb{Z}_n^*$ and compute $C = g^{\zeta} r^n \bmod n^2$.
- **Decryption:** $\zeta := \mathsf{Dec}_{\mathsf{Paillier}}(C, \mathsf{sk}_{\mathsf{Paillier}})$ Let $C \in \mathbb{Z}_n^*$ be the ciphertext, compute the plaintext as $\zeta = (L(C^{\lambda} \bmod n^2) \cdot \mu) \bmod n$.
- **Message Membership Proof of Knowledge [3]:** $\{v_j, e_j, u_j\}_{j \in \mathcal{P}} := \mathsf{PoK}_{\mathsf{mem}}(C, \Upsilon)$. In [3], the authors propose an efficient method to prove that a given encrypted message is 1 out of n messages in a set. The non-interactive version of this proof of knowledge is described as follows. Let n be the RSA modulus from Paillier system, $\Upsilon = \{\zeta_1, \zeta_2, \ldots, \zeta_\rho\}$ the set of ρ encoded candidates, $\mathcal{P}$ be the set of n messages, and C the encryption of one encoded candidate. In this proof, the prover P convinces the verifier V that C encrypts the i^{th} message in Υ:

  1. P picks κ randomly from $\mathbb{Z}_n^*$, $\rho - 1$ values $\{e_j\}_{j \neq i}$ in $\mathbb{Z}_n$, and $\rho - 1$ values $\{v_j\}_{j \neq i}$ in $\mathbb{Z}_n^*$. Then, (s)he computes $u_i = \kappa^n \bmod n^2$ and

     $$\{u_j = v_j^n (g^{\zeta_j}/C)^{e_j} \bmod n^2\}_{j \neq i}$$

     The prover sets $e \in \{0,1\}^L$ as the hash value of $\sum_{k=1, k \neq i}^{n} u_k$ (in our system, we set L as 80). The prover further lets $e_i = e - \sum_{k \neq i} e_k \bmod n$ and calculates

     $$v_i = \rho \cdot r^{e_i} \cdot g^{\left(e - \sum_{j \neq i} e_j\right) \div n} \bmod n.$$

     Finally, the prover sends $\{v_j, e_j, u_j\}_{j \in \mathcal{P}}$ to the verifier.
  2. The verifier sets $e \in \{0,1\}^L$ as the hash value of $\sum_{k=1, k \neq i}^{n} u_k$ and checks whether $e \stackrel{?}{=} \sum_{k=1}^{p} e_k$ and that $v_j^n \stackrel{?}{=} u_j (C/g^{\zeta_j})^{e_j} \bmod n^2$ for each $j \in \mathcal{P}$.

- **Decryption Correctness Proof of Knowledge:** We define $(\delta, r) := \mathsf{PoK}(C, \mathsf{sk}_{\mathsf{Paillier}})$, which is the function to compute the plaintext δ and the random factor r to the ciphertext C with the given Paillier system secret key $\mathsf{sk}_{\mathsf{Paillier}} = (n, g, p, q)$. As Paillier system is bijective [34] meaning $\mathsf{Enc}_{\mathsf{Paillier}} : \mathbb{Z}_n^* \times \mathbb{Z}_n \to \mathbb{Z}_{n^2}$ is both one-to-one and onto. The prover sends δ and r to the verifier to prove that (δ, r) is the only pair to construct C.

  The main idea to compute $r \in \mathbb{Z}_n^*$ is described as follows: we denote $g(r) = r^n \bmod n^2$. The $g(r)$ can be calculated by $c \cdot g^{-m}$. Then, based on the Little Fermat theorem, $r^p = r \bmod p$, thus, $g(r) = r^n \equiv r^q \bmod p$. Since $\gcd(q, p-1) = 1$, there exists i_1 such that $q \cdot i_1 = 1 + k(p-1)$ and we get

  $$g(r)^{i_1} = (r^q)^{i_1} = r^{1+k(p-1)} = r \cdot r^{k(p-1)} = r \bmod p$$

  Similarly, we denote i_2 as the modular inverse of p modulo $q-1$ and we get $g(r)^{i_2} = r \bmod q$. We finally get $r \bmod p$ and $r \bmod q$ respectively and apply the Chinese Remainder Theorem to obtain $r \bmod n$.

C Public key Accumulation algorithms

Algorithm 1 Bottom half: Server side WS generation.

```
Input: $\mathcal{Y}$: voter's SLRS public key set with $|\mathcal{Y}| = b$.
Input: Num: number of keys in one group.
Input: $\psi$, $N$, and $\phi(N)$: SLRS parameters.
Input: thisgroup :: temporary set containing SLRS keys belonging to a group.
Output: WS, gkeys.
function GenWs($\mathsf{Num}, \mathcal{Y}, b$)
    groupsv ← GenGroupsv($\mathcal{Y}, b, \mathsf{Num}$)
    for $j = 0 : 1 :$ len(groupsv) do
        val ← $\psi$
        for $i = 0 : 1 :$ len(groupsv) do
            if $j \neq i$ then
                val ← $\mathsf{val}^{\mathsf{groupsv}[i]} \bmod N$
            end if
        end for
        WS[$j$] ← val
    end for
    return WS
end function
function GenGroupsv($\mathcal{Y}, b, \mathsf{Num}$)
    if $b <= \mathsf{Num}$ then
        localv ← 1
        for $i = 0 : 1 : b$ do
            $\mathsf{localv} = \mathsf{localv} \cdot \mathcal{Y}[i]$
        end for
        $\mathsf{groupsv}[0] = \mathsf{localv}$
        return groupsv
    else
        thisgroup ← 0
        for $i = 0 : 1 : b - 1$ do
            $\mathsf{localv} = (\mathsf{localv} \cdot \mathcal{Y}[i]) \bmod (\phi(N))$
            $\mathsf{thisgroup} = \mathsf{append}(\mathsf{kgroup}, \mathcal{Y}[i])$
            if $(i \bmod \mathsf{Num}) == \mathsf{Num} - 1$ then
                $\mathsf{groupsv} = \mathsf{append}(\mathsf{groupsv}, \mathsf{localv})$ and $\mathsf{gkeys} = \mathsf{append}(\mathsf{gkeys}, \mathsf{thisgroup})$
                thisgroup ← 0 and $\mathsf{localv} = 1$
            end if
        end for
        $\mathsf{groupsv} = \mathsf{append}(\mathsf{groupsv}, \mathsf{localv})$
    end if
    return groupsv, gkeys
end function

```

Algorithm 2 Top half: Voter computes the key accumulation.

```
Input: WS: array contains key accumulation from bottom half.
Input: $Y'$: public keys array that this voter's key belongs to.
Input: $N$ and $\phi(N)$: SLRS parameters.
Input: $j$: the index of this voter.
Input: idx: the index of this voter's public key in WS.
Output: ret: the result of key accumulation for this voter.
  function GenKeyAcc(WS, $Y$, $j$, idx, Num)
      $x \leftarrow 1$
      for $i = 0 : 1 :$ Num $- 1$ do
          if then$j \neq i$
              $x = (x \cdot Y'[i]) \bmod \phi(N)$
          end if
      end for
      ret $\leftarrow$ WS[idx]$^x \bmod N$
      return ret
  end function
```