Hindawi
Security and Communication Networks
Volume 2021, Article ID 8338616, 14 pages
https://doi.org/10.1155/2021/8338616

Research Article

Towards Achieving Personal Privacy Protection and Data Security on Integrated E-Voting Model of Blockchain and Message Queue

Siriboon Chaisawat and Chalee Vorakulpipat

Information Security Research Team, National Electronics and Computer Technology Center, Pathumthani 12120, Thailand

Correspondence should be addressed to Chalee Vorakulpipat; chalee.vorakulpipat@nectec.or.th

Received 25 May 2021; Revised 2 August 2021; Accepted 8 September 2021; Published 29 September 2021

Academic Editor: Jianting Ning

The growing number of e-voting applications indicates the need in resolving issues that exist in the traditional election model. By integrating with blockchain technology, we could extend the model's capabilities by presenting transparency in logic execution and integrity in data storage. Despite these advantages, blockchain brings in new challenges regarding system performance and data privacy. Due to distributed nature of blockchain, any new updating request needs to be reflected in all network's peers before proceeding to the subsequence requests. This process produces delay and possibility in request rejection due to update conflict. In addition, data removal is no longer feasible since each record is protected by immutable hashed link. To overcome these limitations, the integration model of blockchain and message queue is proposed in this paper. The design addresses security concerns in data exchanging patterns, voter anonymization, and proof of system actor's legitimacy. Performance tests are conducted on system prototypes which were deployed on two different settings. The result shows that the system can perform well in production environment, and introduction of message queue handling scheme can cope with blockchain's errors in unexpected scenarios.

1. Introduction

Voting is an act of delegating one's decision-making power. Traditional election relies on marking and counting ballot papers. Even though this model is still widely used in many nations, the overall procedure is time consuming, inefficient, and prone to error and electoral frauds [1]. Online voting is introduced to overcome the limitations, achieve better efficiency as well as provide convenience to the users. The simplest implementation started from a single server where authentication and vote processing are performed. Despite the presence of data encrypting schemes, all cryptographic operations and key storing are done at server side. In sum, overall system operation remains hidden from the users.

Decentralized Application (DApp) is a new programming approach that allows application to operate on the distributed computer system or trusted P2P network like blockchain. Execution of application's logic is moved to the client side without central authority governing. Also, data directly traverse among only trusted app's clients. Every transaction must be validated against the consensus and pre-agreed rulesets. No malicious action beyond logic agreement shall be carried out. With blockchain, data integrity is preserved by block hash which represents the entire chain's state up to that current point and can be computed by taking previous block hash as an input. Merkle tree [2], illustrated in Figure 1, is a data structure for representing structure of the chain. To validate whether a specific transaction i exists, inclusion can be proved by checking if the tree root (R) is equal to hash of the transaction i that concatenates with its sibling and sequences of sibling of all i's ancestors (π). Let ϕ denote position of i node in Merkle tree, verification can be computed within time complexity of $O(\log n)$, and the equation is defined as follows:

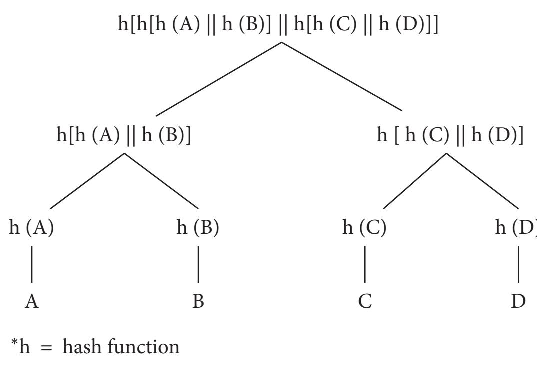

Figure 1: Example of Merkle tree data structure.

$$R = \begin{cases} \text{hash}\left(i \,\|\pi_1 \; \pi_2\| \ldots \pi_{|\pi|}\right), & \text{if } \varnothing < \text{node position of } \pi_1, \\ \text{hash}\left(\pi_1 \, i\|\pi_2\| \ldots \pi_{|\pi|}\right), & \text{otherwise.} \end{cases} \tag{1}$$

Since operations on distributed ledger rely largely on an underlying consensus mechanism, a number of consensus mechanisms have been proposed, e.g., Proof of Work (PoW), Proof of Stake (PoS), and Practical Byzantine Fault Tolerance (PBFT). Differences in mechanisms directly influence security and performance of blockchain. To avoid possibility of double spending attack [3] and damages from block reversion due to chain fork, transaction ordering service based on Raft consensus [4] was chosen for this proposed design. With Raft implemented, transactions processing is ensured to be linearizable as there is only a single leader per term. Its responsibilities are to ensure that new updating requests are committed to replicate log and all followers maintain exactly the same order of log entries. In addition, the system can tolerate up to n servers failure given that there are $2n + 1$ servers in total.

Previous research work on implementation of fault-tolerant e-voting systems [5] addresses system design based on the assumption that each district function requires different control mechanisms and encounters different amount of traffic load. The design enables the system to be scaled at functional level, which promotes efficient use of resources and suitable to serve a large-scale election. Nevertheless, there is still a need for refinement in several issues, such as weakness in anonymization schemes, security protection for data transmission over the public Internet, and handling schemes to cope with unexpected circumstances.

Presenting the queuing mechanism that supports reliable data delivery, message queue is found to be an attractive solution as it could be integrated as a middleware for transaction buffering, error handling, and blockchain's event messages listener. Extending the prior study, this paper proposes an integrated model of e-voting by leveraging a messaging protocol. The goal is to overcome technical challenges appeared in previous work and other existing e-vote models, which are data privacy, security, and a need for performance improvement. Refined architecture design and setups of network components are presented, along with error handling schemes to ensure delivery of data.

The subsequent section presents related studies on the e-voting system and different solution schemes to overcome limitations in blockchain, followed by the proposed system design and operation workflow. To elaborate on the introduced concepts, implementation details are presented along with performance evaluation and security assessment on the developed prototypes. The final section summarizes the research findings and provides recommendations for future study.

2. Related Works

Helios [6] is one of the earliest implementations of e-voting systems in which system transparency is promoted by publicly displaying all votes in encrypted form. Even though the design can eliminate external parties' intervention, voter's privacy cannot be guaranteed since all ballots need to be decrypted by the authority who can access to all voter's private keys during the tallying process. Thus, it is possible to sneak into individual's information. Online voting has begun to adopt in many countries [7–11]. For example, in Estonia [12], the system is developed upon national ID infrastructure. Eligible voters must authenticate themselves by dipping ID card and installing a voting application. Once the vote is casted, personal data will be removed and only the candidate selection data will be encrypted using with Estonian National Electoral Committee (ENEC)'s public key. Nevertheless, the analysis study [13] suggests that the system requires major fixes as multiple security loopholes have been found (e.g., server-side malware injection and client-side vote data sniffing). With an advancement in cryptography, some cryptosystems exhibit homomorphic properties in which the mathematical operation $\varnothing$ on a set of encrypted payloads shall be equivalent to encryption of the result from performing operation θ on plaintexts $E(a)\varnothing E(b) \equiv E(a\theta b)$. Such a scheme is beneficial in ballot tallying without requiring prior payload decryption. The study [14] proposes implementation e-voting based on ElGamal encryption scheme which possesses homomorphic properties. During the voting period, voter must construct table of $R * C$ where R represents a selection array in which each cell could either be 0 or 1 and C represents the list of candidates. Vote tallying can be conducted by performing algebraic multiplication on encrypted ballots. The similar concept applies to an implementation of voting scheme based on Paillier cryptosystem [15]. However, proving whether the ballot format is valid (each cell is marked by only 0 or 1) requires generation of all voting possibilities. Thus, the scheme is applicable to the election where voting options are predefined and unchanged. Even though many studies have put efforts on voting scheme design, overall operation remains a black box from user's perspective. This leads to an introduction of decentralized application (DApp) where operation execution is shifted to the client side. All business logic and permission ruleset must be pre-agreed prior to initialization of system process. In addition, an underlying consensus mechanism ensures that distributed ledgers are synchronized and integrity of each record entry is preserved by the cryptographic algorithm. With the presence of immutable audit trails, it attracts adoption in various business fields. Served as a trusted verification source, blockchain is applied to improve effectiveness in current banking systems [16–18], health insurance [19], supply chain management [20], and right management on digital content sharing platforms [21, 22]. Despite the aforementioned benefits, blockchain encounters two major challenges which are privacy and performance.

Due to immutable property that applies to all record entries, deletion of data is no longer possible once it has been stored on the chain. As stated in the article [23] regarding storing and processing personal data, individual has rights to withdraw consent and request for personal data erasure. Thus, confidential data are recommended to store off-chain as to comply with general data protection regulations (GDPRs) [24]. Nevertheless, many research studies have sought a way to attain on-chain privacy protection for sensitive data [25]. Mixing is one of the approaches, which suggests aggregation of multiple users' transactions into a single transaction in order to prevent attacker from analysing victims' actions. Several implementations of mixing include Mixcoin [26] and CoinJoin [27]. Anonymous signature is another alternative method that presents the concept of using a representative signature for transaction signing instead of the actual performer. Utilizing ring signatures for concealing identity, the scheme [28] ensures that all involved parties can validate the transactions' authenticity without gaining further knowledge of the originating source. Nevertheless, the odds of correct guessing is $1/n$, where n denotes the number of network participants. The probability will be incremented as the number of participants is smaller to 1. Adopting a variant of the TOR and similar to the concept of Mixnet implementation [29], a study [30] introduces an implementation of the Garlic Routing (GOR) on a sidechain in which all transactions created on the main chain are required to route through a sidechain's smart contract mesh in order to conceal creators' identities. The more complexity in sidechain topology, the more often the sender's address is encapsulated and concealed. Nevertheless, the greater complexity of the blockchain logic, the more computational resources are required for the blockchain to validate transactions.

Another key issue is performance. Indeed, there are several factors that contribute to execution competency [31] such as network latency, consensus algorithm, number of participating nodes, and smart contract complexity which are in proportion to the number of read/write operations needed to be executed. Despite the distributed design of blockchain that offers high availability, there exists a limitation in handling large volume of transaction. Common problems that blockchain will encounter in production setup is transaction rejection due to disagreement in transaction generation and processing capacity as well as read-write operation conflicts. To resolve such issues, transaction queuing and error handling mechanism are required.

With lightweight and P2P communication of messaging protocol, it enables emergence of distributed system models, such as device communication in IoT ecosystems. Several protocols serve to standardize message exchange patterns such as MQTT [32], AQMP [33], and ZMTP [34]. With the presence of queueing mechanism, delivery of data is ensured to be in order and take place exactly once. Due to its asynchronous nature and publish-subscribe communication pattern, a message queue has become a popular middleware for synchronizing states among dispersed services. A study [35] leverages message queues for providing consistent updates across databases located in heterogeneous environments. Any change to the database will trigger generation of event messages for acknowledging relevant parties to perform local updates corresponding to the new changes. Apart from data synchronization, message queue is introduced for improving reliability and delivery in data transmission, especially when data production and consumption rate are inharmonious. A study [36] presents use of message queue in replacement of relational database in a mailing system. Traditionally, mail queuing pipeline relies largely on altering rows in relational databases. To prevent the occurrence of operational conflict or bottleneck, the system must avoid large load generation by limiting the number of concurrent active users. With introduction of Apache Kafka [37] as a queuing middleware, tasks beyond capacity limits are added to a queue and held to be processed later without interrupting core operation pipeline.

Multiple studies have proposed integration models of blockchain and message queues. Nowadays, many applications rely heavily on event-driven processing. To avoid alteration or insertion of falsified events into the message stream, blockchain has been introduced for validating authenticity of data exchanged over messaging protocols [38, 39]. On the other way round, message-oriented middleware services have been deployed as blockchain's event listener. Eventeum [40] is one of the implementations for Ethereum Blockchain [41] in which all the blocks and transaction events will be propagated to message bus, and the bus then exposes REST api to application for further processing. Other products of message streaming middleware are OCI Streaming service [42] and Amazon Simple Queue Service (SQS) [43, 44] which offer blockchain event collection and integration with a number of business services, such as user notification (SMS and e-mail) or streaming events directly to business intelligence or analytics engines. However, these products only facilitate the outgoing messages from blockchain which have low traffic density, small chance of bottlenecks, and low error conflicts in contrast to an inward direction which is one of the concerns stressed in this paper.

3. Proposed Integration Design of Blockchain-Based E-Voting Systems

To promote ease of adoption to real-world settings and enhance overall system resilience, the proposed model emphasizes design towards generality while ensuring data protection from end to end. The first part presents system topology design and setup of key components. The later part introduces system operational procedure comprising voter authentication, ballot data transmission, and ballot verification and storing process.

3.1. System Overview. In the architecture design (Figure 2), we assume that voters are not technical experts who can host or run full blockchain nodes. Voters are assumed to reside off-chain and possess personal devices (i.e., smartphone or PC) with Internet connectivity. The authentication process is expected to be performed off the chain with local personnel database managed by the responsible authorities. After a voter is authenticated, he/she is permitted to submit a voting request. To follow the principle of directness stated in system design guidelines [45, 46], point-to-point with no broker messaging protocol is leveraged to ensure that data are transmitted to the chain without passing any intermediaries. In the case that peers are not discoverable within the network (i.e., peers might be located in different networks and do not own public IPs), then messaging brokers are required to be configured as a directory service only for the purpose of facilitating peer discovery [47]. Once data reach the on-chain node, integrity of data and authenticity of the sender will be validated against sets of information records stored on blockchain states. The following display the records which consist of (a) list of eligible token seeds, (b) list of used voting tokens, (c) list of ballots, and (d) lists of nodes' public keys.

(a) Eligible Token Seeds
TS_1
TS_2
....
TS_n

(c) Ballots	
Transaction ID	Selected Candidate
Tx_ID_1	$c_1 \in \{Candidates\}$
Tx_ID_2	$c_2 \in \{Candidates\}$
....	....
Tx_ID_n	$c_n \in \{Candidates\}$

(b) Used Voting Token
TE_1
TE_2
....
TE_n

(d) Nodes' Public Keys	
NODE ID	Public Key
NID_1	PK_{NID1}
NID_2	PK_{NID2}
....	....
NID_n	PK_{NIDn}

Permissioned blockchain is leveraged in order to prevent unnecessary flow of data to irrelevant parties. To ensure that the system can operate with high availability, we leverage the design of a fault-tolerant blockchain network as proposed in previous research work [5].

3.2. Key Components. This proposed design introduces three key actors as defined below. These actors are blockchain client nodes with additional setup of supplementary services. Prior to chain initiation, each node will be assigned to a specific role. A public key for each node ($PK_{<NODE\_ID>}$) is required to be published to the blockchain's shared records to enable key lookup among network components. Two-level role-based permissions (displayed in Table 1) are introduced for defining nodes' accessibility to system resources.

3.2.1. Authenticator Node. This node comprises of two functional components. The first part is an authenticating service, which locally connects to the personal database in order to provide high-security protection on the data. Another part is the blockchain interface, which stores the node's cryptographic credentials and connects authenticator to the running blockchain network.

3.2.2. Proxy Voter Nodes. In order to provide anonymity to the voter, voting token (TE) is proposed to represent voting eligibility instead of referencing to an actual performer. The main tasks of this node are to perform token validation, initiate peer-to-peer connection with trustworthy clients (voters), and trigger submission of voting transactions on behalf of the actual voters. Two main services are implemented in this node: messaging sockets and blockchain interface service. This type of node can be set up as a cluster for load-balancing incoming data packets from clients.

3.2.3. Validator Node. The validator node is responsible for verifying the election results by ensuring that the number of created ballots and used tokens always matches. Also, it facilitates transaction querying in case voters wish to verify their ballots. As displayed in Table 1, the node is permitted only to inspect and query blockchain resources for the purposes of validation. Thus, the node plays no role in modifying the data due to restrictions of the consensus rules.

3.3. System Flow

3.3.1. Authentication and Token Generation. In the beginning, voters are required to authenticate themselves with an authenticating service. To access to this service, authenticator node must configure a private connection channel and provide the configuration to all intended voters. Authentication mechanism and strictness level can vary according to election regulations, which are usually defined by the election commission of each campaign. Once a voter is authenticated, the node then invokes a smart contract for adding a new voter. The returned transaction ID will be used as a token seed (TS). The seed will be recorded to a blockchain's list of eligible token seeds and will also be used for constructing a voting token (TE). Leveraging asymmetric signature JSON Web Token (JWT) [48] format, TE's structure can be divided into three parts, as displayed in Figure 3. Payload contains two types of data. The first type is system data, for example, token expiration date and time. These data help system prescreen packets in order to reduce unnecessary load. Another type is custom information. This part contains an encrypted TS with proxy voter's public key (PK_{PX}). Extending asymmetric key encryption, authenticator's private key (SKAUTH) is used for signature signing. In order to validate token authenticity, one must decrypt the signature part with an authenticator's public key (PK_{AUTH}), which is retrievable from the blockchain state.

3.3.2. Ballot Data Transmission. Once voters are authenticated and obtain TE from an authenticator, they need to establish secure connections to the proxy voter prior to exchanging confidential data. According to Figure 2, a client

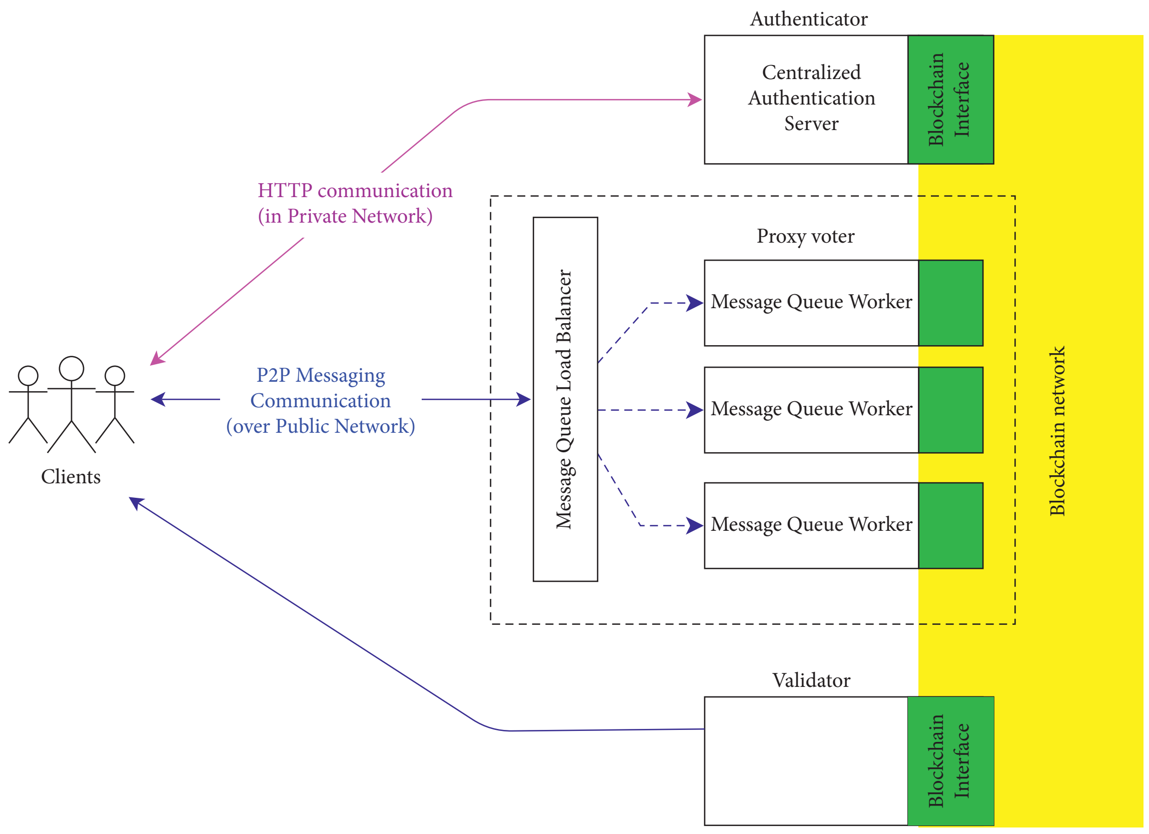

FIGURE 2: System design and components setup.

TABLE 1: Blockchain permissions categorized by node roles and permission types.

Permission types node name	Permission on network resources	Permission on logic			
		List of eligible tokens	List of used token	Ballot list	Nodes' public key
Authenticator	Create and retrieve transaction	*R/W*	—	—	*R*
Proxy voter	Create and retrieve transaction	*R*	*R/W*	*R/W*	*R*
Validator	Retrieve transaction	*R*	*R*	*R*	*R*

∗*R* means Read Permission; *W* means Write Permission.

Section	Content
Header	Base64Encoded({ "alg": <hashing algorithm>, "typ" : <type of token>})

Payload	Base64Encoded({
	//example of system data
	"iss": <issuer>,
	"exp": <expiration time>,
	
	//example of custom data
	"identifier_key" : ENC(PK_{PX}, TS) })

Signature	HashAlg(Base64Encoded(Header)+ " . " + Base64Encoded(Payload), SK_{AUTH})

FIGURE 3: Structure of voting token.

can be seen as a remote peer in distributed network. Since authentication support in messaging protocols is limited to device and service level, a voting token is introduced to offer user-level authentication by embedding in the data frame along with the message stream. To authenticate connecting peers at the device level, we leverage handshake mechanism in CurveCP protocol [49] for exchanging 2 sets of key pairs. The first is the permanent (long-term) key pair, which is used for identifying the data source and for generation of the transient key pair. The second is the transient (short-term) key pair, which is used for encrypting exchanged messages. To prevent any kind of intercepting attacks, the transient keys will be destroyed and recreated every time and a connection session is reestablished. For simplicity in explanation, proxy voter nodes and voters are represented as server and client, respectively. Let permanent public keys of client and server be denoted as C and S, while private keys are denoted as c and s. For transient key pairs, let (S', s') and (C', c') denote pairs of short-term public and private keys of server and client, respectively. All voters in the same campaign are assumed to share a common (C, c) and have initial knowledge of server's public key, S. The server is assumed to know voter common public key, C, and possess its initial key pair (S, s). The communication scheme is illustrated in Figure 4.

(1) Connection Validation. Following the CurveCP protocol, upon connection establishment, each voter must generate his/her own transient key $(C', ')$, encrypted C' with C and send to target end. If the receiver can decrypt the packet, it can then be certain that the connection is from legitimate peer and returns encrypted S' in exchange. The connection is now established.

(2) Eligibility Verification and Data Transmission. Goal of this step (2) is to further assure that connection is initiated from valid entity in an election campaign. Goal of this step is to further perform authentication at user-level to ensure that the connecting clients are eligible to vote. Different messaging patterns are introduced for serving the requirement. Figure 5 displays a socket setup for each end. Server (proxy voter) must implement 1 REP socket, 1 PUSH socket, and a pool of PULL sockets while a client (voter) is required to implement 1 REQ, 1 PUSH, and 1 PULL socket. Prior to ballot submission, client must provide proof of voting eligibility by sending REQ's message that contains TE along with client's PULL socket configuration. On receiving the data, the server then validates integrity of the message and authenticity of TE by verifying TE's signature as well as checking if TE is not expired. If TE is valid, encrypted token seed will be decrypted using its private key (SK_{PX}). The result (TS) will be compared against the blockchain's list of eligible token seeds and made sure that TE itself is not in the list of used tokens. If all conditions are met, the server will find and reserve available address listening by its PULL sockets. The address will be returned to the client for further communication. For ballot data submission, the PUSH-PULL pattern is leveraged since this process requires altering blockchain states which often takes some time for data to be processed. With PUSH-PULL type configured, the message queue ensures that any late responses will be captured and persistently maintained until an intended recipient obtains the data. To submit voting data, the client constructs a message containing a selected candidate choice and encrypts it with the transient key. The message is then pushed to the address listening by the server's provided PULL socket.

3.3.3. Ballot Verification and Storing. On receiving a message, the server decrypts data and passes it to a blockchain interface service to convert into blockchain-compatible transaction format and sign with SK_{PX}. Once the transaction is published to the network, it will be validated against permissions at blockchain network layer and business logic layer. At the network layer (lower level), permission is defined for limiting activities that affect system resources or configurations such as adding new members to the chain or submitting new transactions to the network. Permission at business logic layer (higher level) governs individual rights on invoking specific functions on smart contract. Table 1 displays permissions classified by the node's roles.

Proxy voter is the only type of node that is allowed to append ballot transactions to blockchain state. Once the transaction is recorded to the chain, the corresponding TE will be added to the used token list, and the transaction ID will be directly sent via PUSH socket to the client as verifiable evidence. Until the terminating request is fired, PUSH/PULL sockets ensure that a late blockchain's response of transaction ID is successfully delivered to the client's hands.

4. Implementation

To affirm that the design can satisfy all functional requirements, prototypes were developed by utilizing Hyperledger Fabric [50], an open-source framework for developing permissioned blockchains, and ZeroMQ [51], a messaging library that relies on ZMTP messaging protocol, for facilitating client-to-node communication.

4.1. Blockchain Network Implementation. Hyperledger version 1.4 with Raft ordering service was deployed for development of blockchain network. The network was set up on 2 different environments: a single-host setting and multihost setting. For single-host setup, the network was deployed using Docker [52] which is installed on Ubuntu 16.04 LTS machine with 2 CPU cores 2.80 GHz and 12 GB of RAM. An architecture composes of three Raft ordering nodes and two organizations with single peer and single CA each. LevelDB is set up as peer's state database. For multihost setup (Figure 6), 4 virtual machines (Ubuntu 18.04, 2 CPU cores, and 4 GB of RAM) are set up as Kubernetes [53] cluster (1 master node and 3 worker nodes). An architecture is composed of three Raft ordering nodes and two organizations with two peers and single CA each. Each node is deployed as a pod along with its corresponding Cluster IP service. CouchDB is deployed as peer's state database, and pod affinity was configured to ensure that the database is colocated with its corresponding peer node. Hyperledger

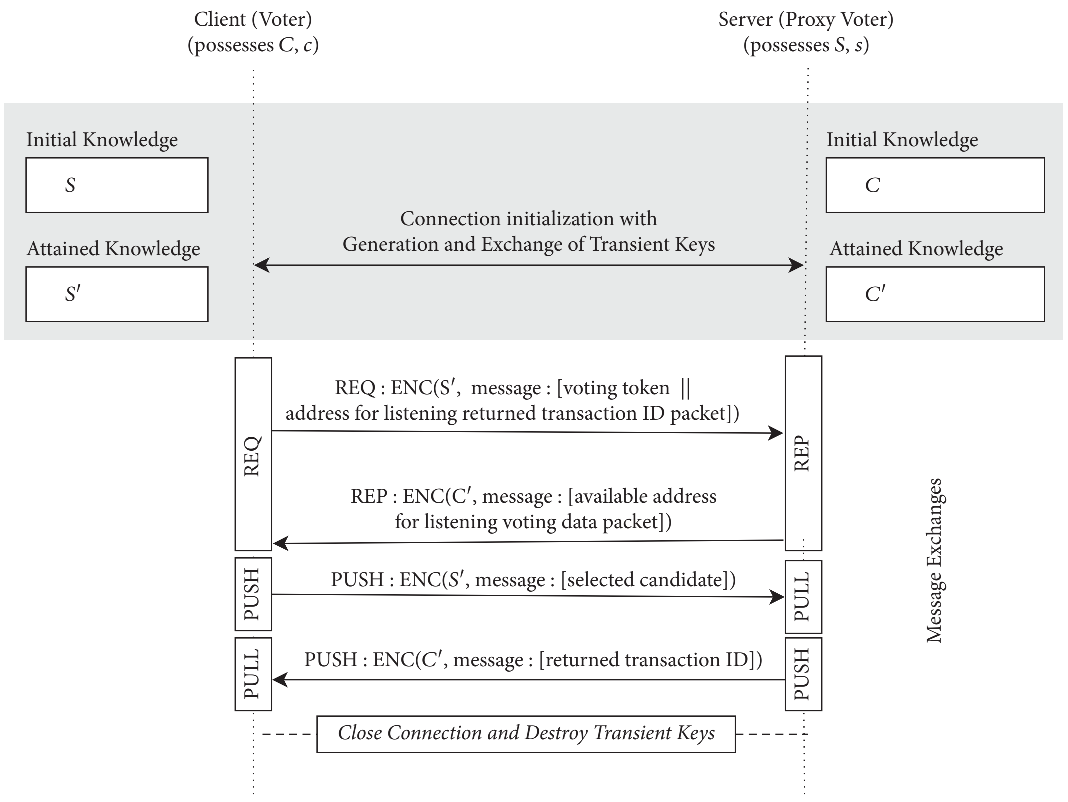
Client (Voter)
(possesses C, c)
Server (Proxy Voter)
(possesses S, s)
Initial Knowledge
S
Attained Knowledge
S′
Connection initialization with
Generation and Exchange of Transient Keys
Initial Knowledge
C
Attained Knowledge
C′
REQ : ENC(S′, message : [voting token || address for listening returned transaction ID packet])
REP : ENC(C′, message : [available address for listening voting data packet])
PUSH : ENC(S′, message : [selected candidate])
PUSH : ENC(C′, message : [returned transaction ID])
REQ
REP
PUSH
PULL
PULL
PUSH
Close Connection and Destroy Transient Keys
Message Exchanges

Figure 4: Message exchanges over messaging protocol.

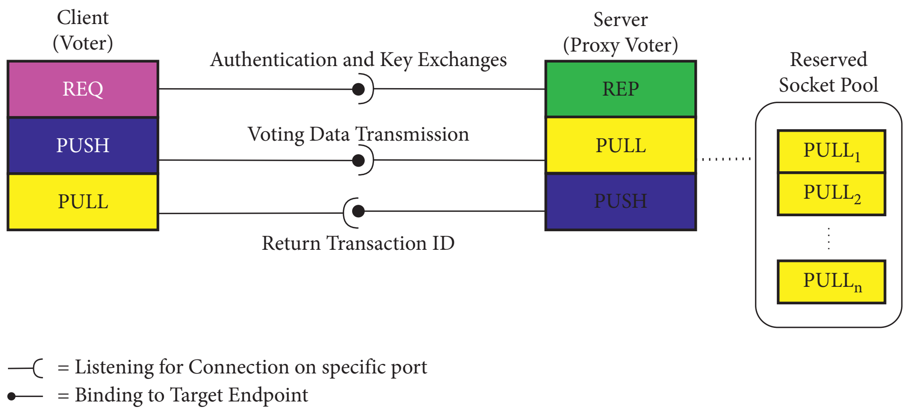

Figure 5: Sockets setup.

Fabric SDK is deployed as a separated service to enable external interaction with the blockchain. The results after conducting a performance test on both environment settings are displayed in the evaluation section.

To restrict the capability of each node's role in interacting with network resources, we override default configuration of Hyperledger Fabric's Access Control List (ACL) in order to customize the policy. The following is a code snippet that defines permission on creating (write) and querying (read) transactions in the form of reader/writer set:

Policies:

Writers:

Type: Signature

Rule: "OR("Authenticator.admin," "ProxyVoter.client")"

Readers:

Type: Signature

Rule: "OR("Verifier.admin, "Verifier.peer," " Verifier.client")"

To ensure that business logics on smart contract are invoked by the authorized entities, built-in attribute-based access control (ABAC) is used for checking each performer's credentials and validating against predefined conditions. For the purpose of implementation, we import "ClientIdentity" class from "fabric-shim" library in order to retrieve and

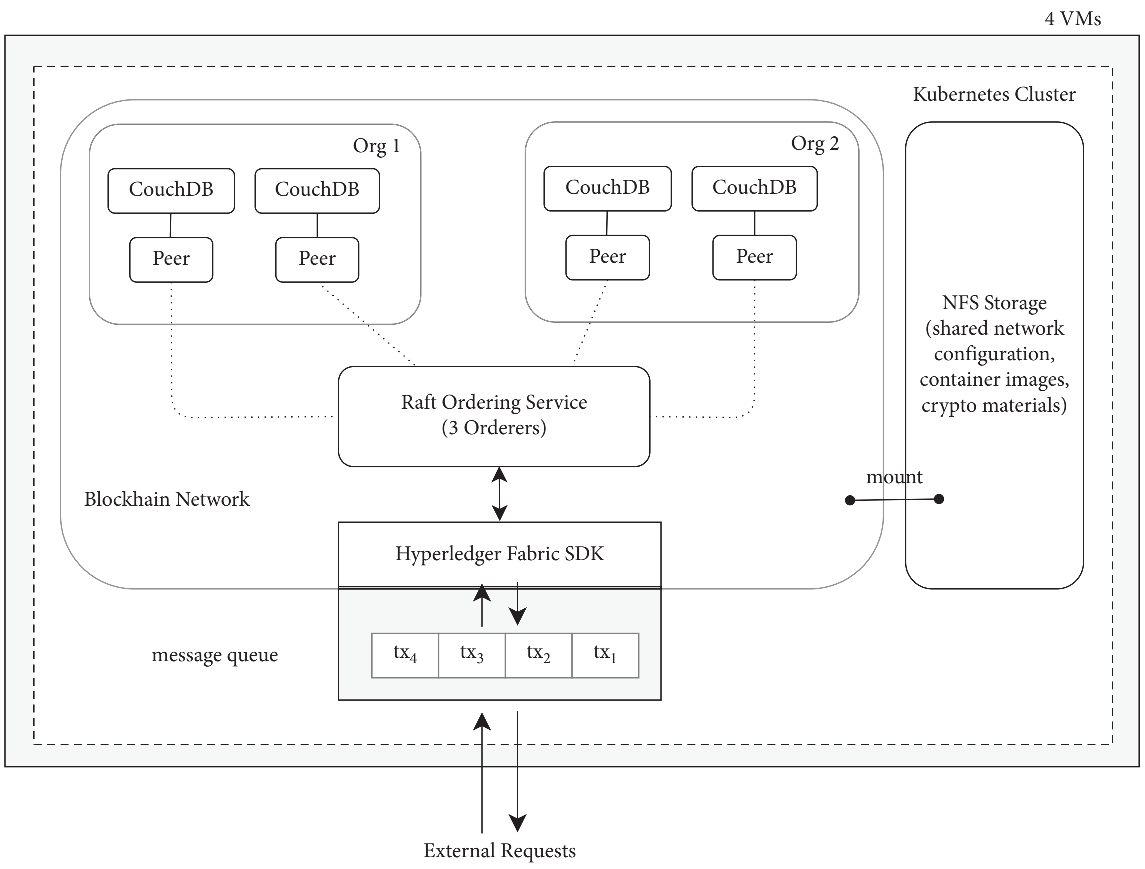

Figure 6: Blockchain network setup on multihost setting.

inspect the credentials of transaction actors. The following is a code snippet of the smart contract's token generation function. Prior to executing the logic, we perform credential checking if the requesting entity is an "authenticator."

```
const ClientIdentity = require("fabric-shim").ClientIdentity;
function addToken(arguments) {
  let cid = new ClientIdentity(stub);
  if (cid.assertAttributeValue("node_name", "authenticator"){
      //add new eligibility token.
}
  else{ throw new Error("Invoking Entity is Unauthorized"); }
}
```

4.2. Message Queue Implementation. ZeroMQ, a brokerless message-oriented networking library based on ZMTP protocol, is leveraged for enabling P2P data exchanges. To make sure that connections are established from/to the legitimated peers, ZeroMQ authentication protocol (ZAP) [54] is used for authenticating connections against a set of known peers' public keys. In order to obtain the keys, CurveZMQ [55], a protocol based on CurveCP and the NaCl cryptographic library, is used for generating 256 bit elliptic curve Curve25519 key pairs.

5. Performance Evaluation and Security Analysis

This section is divided into three parts. The first part presents security analysis on voting tokens, data exchanging schemes over messaging protocol, and state alteration in the blockchain. The second part provides comparative analysis between different designs of blockchain-integrated e-voting models. The final part presents the performance results of overall system operation.

5.1. Security Analysis

5.1.1. Security on Voting Tokens. Design of voting tokens extends the asymmetric signature JWT token format. To verify authenticity and integrity of the token, one must have permission according to ACL and ABAC to access the list of public keys stored on the blockchain. To ensure that voting claim (TS) can be retrieved only by a designated node, the data are encrypted with PK_{PX} to prevent nodes other than proxy voter from accessing. In order to limit the token's usage duration and to reduce unnecessary processing load, "exp" field is used for prescreening packets.

5.1.2. Security on Data Exchanges over Message Queue. Off-chain data exchange leverages CurveZMQ and ZAP which present secure data encryption scheme and authentication mechanism. Introduction of 2 key pairs (permanent and transient) allows each peer to verify if the connection is initiated by a known source and to be certain that exchanging data remain original. Since the transient key is destroyed and recreated for every new connection, analysing user behaviour through intercepting and monitoring packets is no longer feasible. Furthermore, nonce, a random array of arbitrary numbers, is embedded in every outgoing packet. Therefore, messages are ensured to be non-replayable. Results from conducting packet analysis with Wireshark show that the tool cannot extract any information from captured packets.

5.1.3. Security on Blockchain Network. Supported in Hyperledger Fabric, Access Control List (ACL) is utilized for defining permissions on network entities at the system level. The test was conducted by switching between different roles and performing transaction submission as well as attempting to query data from blockchain state. The result of mismatch entities to the ACL policy (defined in Table 1) appears as the following system error.

Error: failed evaluating policy on signed data during check policy [signature set did not satisfy policy].

In order to validate permission at business logic level, Attribute-Based Access Control (ABAC) is evaluated with scenario-based testing, such as letting the proxy voter, who is allowed to submit transactions according to ACL, and invokes a smart contract function token generation (code snippet is displayed in the implementation section). The result displays a custom error, as follows, since calling this function is restricted to the authenticator.

Error: Invoking Entity is Unauthorized.

To confirm that Raft ordering service presents linearizability to blockchain's state transitions, we deployed Hyperledger Explorer [56] for monitoring activities in network (e.g., number of block/transaction creation and size of local ledger maintained by each peer node). By observing through each peer's historical records, the results confirm that transactions are processed in consecutive manner, there is no evidence of cyclic behaviour, and the same copy of ledger is maintained in all peers.

5.2. Comparative Analysis on Blockchain-Integrated E-Voting Models. An integrated model of blockchain and web service is found to be the most popular e-voting model at present. One of the major concerns is an inconsistency in speed of data production and consumption. Message queues and HTTP web services are capable of generating requests at high frequency. By contrast, the capability and speed of processing transactions by blockchain are limited by the consensus mechanism. Comparative analysis was conducted on three different scenarios in order to observe data handling mechanism presented in each design.

Assume that Joe and Mary are eligible voters. Joe uses an e-voting system that connects to web services, while Mary's system is connected to a message queue.

Situation 1. Once users had submitted their votes, the requests were rejected due to unavailability of proxy voter.

In Joe's case, proxy voter is a web server while Mary's is implemented as message queue server. Joe will receive a response notifying of server's unavailability. Sometime later, Joe needs to retry his submission. As a result, Joe will unexpectedly obtain privilege in reconsidering his voting choice. For Mary, her message will remain in the queue on the client side. Once the server becomes available, her message will be pulled out from queue for processing. She will receive an acknowledgement once her message has already been processed.

Situation 2. Clients are unavailable after requests have been submitted.

In this case, Joe and Mary may lose their Internet connection or encounter unexpected system failure after they have already submitted their votes. Since the response from the blockchain is directly sent to Joe's web browser, the message will be disregarded due to unavailability of the web client. Without catching mechanism presented, Joe will not receive any notification indicating transaction success or failure. On the contrary, Mary's response remains in the queue on the server side. As soon as she becomes available, her socket will automatically retrieve data (transaction response) from the binding socket.

Situation 3. A large number of users are using the system and generating a large amount of transactions that are beyond blockchain's capacity limit. Due to network congestion, two possible cases can occur.

Case 1. Transaction is processed with an unpredictable delay in returning response.

Due to the synchronous nature of web services, once request timeout is reached, the web server will terminate connection with the blockchain and disregard further messages even though later the blockchain might return a successful status (Figure 7). From Joe's perspective, he has no idea whether his data have been recorded. As a result, he may retry submission without noticing that his transactions are now doubling in record. In addition, there is a possibility that his second request gains an opportunity to be processed and recorded before his previous request. For Mary, a message queue acts as data buffer in order to release data to the chain at a steady rate and in chronological order. By relieving the amount of blockchain workload, processing delay and odds of transaction rejection (Case 2) are expected to be reduced.

Case 2. Transaction is rejected due to blockchain's processing limitation or conflict in altering data state (multiple transactions try to update on same key address).

In this case, web server will notify Joe with an error message (Figure 8). In order for his data to be recorded, Joe needs to compete with others by retrying the submission until he receives a success response. For Mary in this case, a message queue will act as a cache which will automatically resubmit the request until transaction ID that indicates success is returned (Figure 9). Otherwise, the request message will not be removed out from queue.

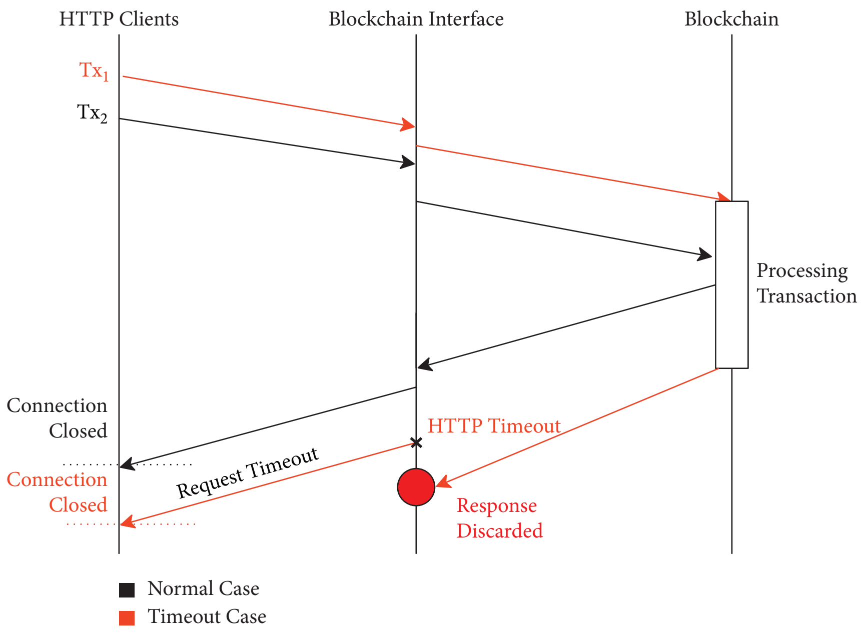

FIGURE 7: Http request handling scheme in the case of connection timeout.

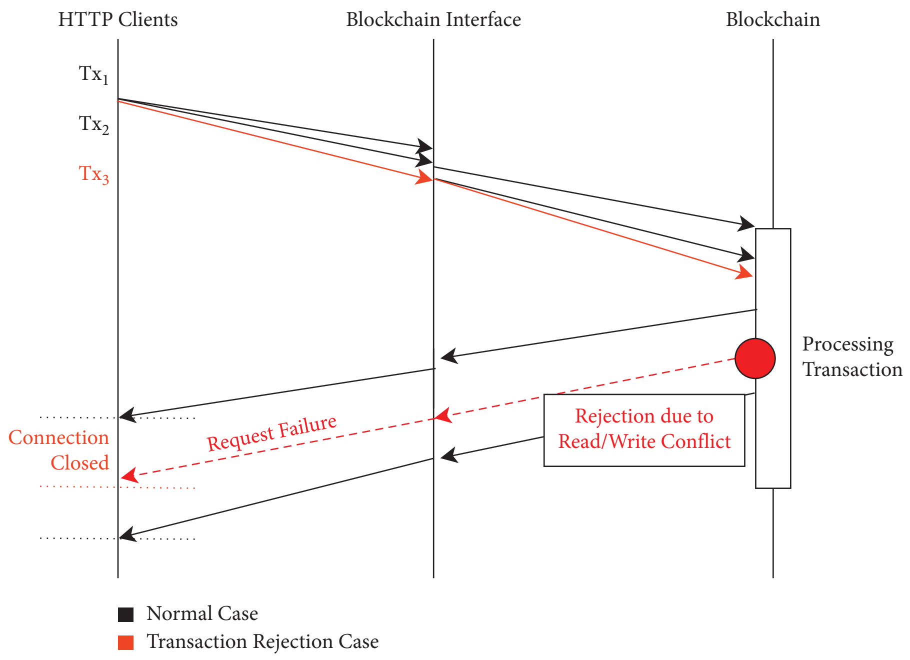

FIGURE 8: Http request handling scheme in the case of transaction rejection.

Enhancing capabilities of blockchain by providing it with an error handling mechanism, an integrated model of message queue guarantees that all data packets are delivered even in the unexpected circumstances.

5.3. Performance Evaluation

5.3.1. Message Queue. Latency and throughput testing was conducted on a single machine of 4 CPU cores 2.90 GHz with 16 GB of memory. Different communication patterns and socket configurations are deployed to fit different operational requirements. For voting token validation, we leverage a Request-Reply pattern in which REQ and REP sockets were configured at the client and server, respectively. Displayed as follows, the multipart message is constructed and sent over tcp://127.0.0.1 : 3000, which is reserved for the socket's listening address.

Frame1	Voting Token (T_E)
Frame2	Client's PULL Socket Listening Address

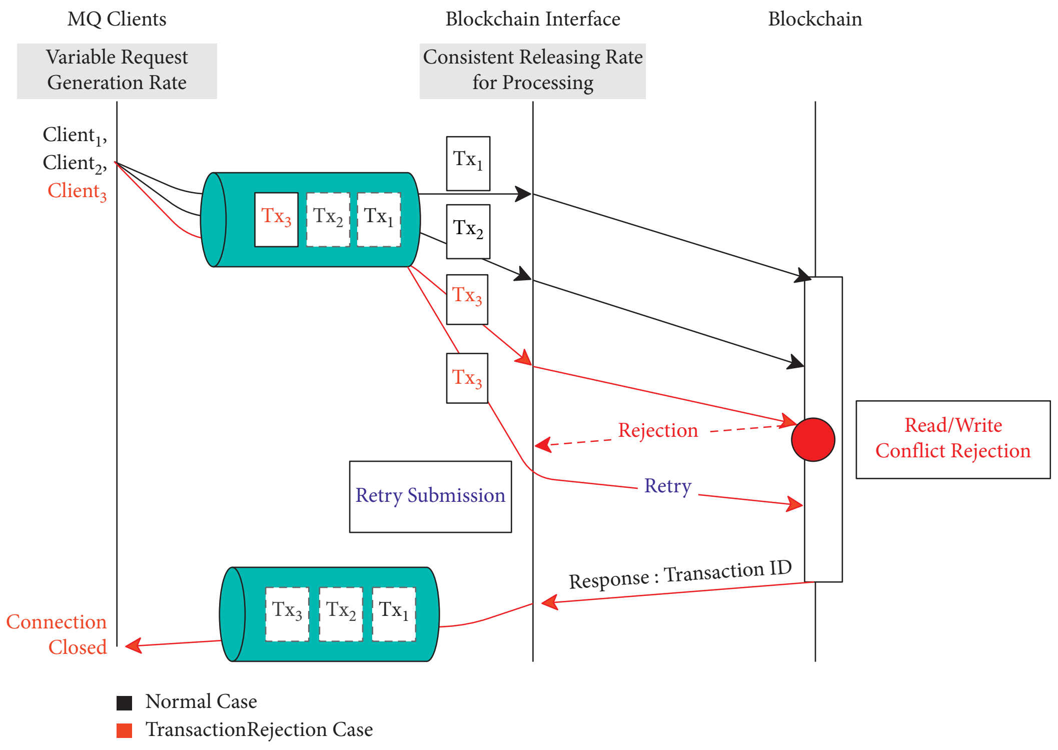

Figure 9: Data handling scheme in message queue (proposed model) in the case of transaction rejection.

For ballot data submission, we leverage a Push-Pull pattern in which PUSH and PULL sockets are configured on the client and server, respectively. A single-framed message containing voter's selected candidate ID is sent over the pre-agreed TCP port.

Frame1	Selected Candidate

For the REQ-REP socket pair, delay is measured in terms of Round-Trip Time (RTT). The equation can be written as $\mathrm{RTT} = t_1 + t_2 + P$, where t_1 and t_2 are the message transmission time from sender to receiver and from receiver back to sender, respectively. P denotes the server processing time, which in this test is set to be close to zero. The measuring result shows an average round-trip time of approximately 168.2917 milliseconds. For the PUSH-PULL socket pair, delay is measured in terms of latency (L). The equation can be written as $L_n = I_n - O_n$, where O_n denotes the time when n-th message is emitted from PUSH socket and I_n denotes the time when the receiving peer obtains the complete message data. Average latency in data transmission between this socket pair is 6.642857 milliseconds.

Message throughput can be measured by counting the number of messages processed within a unit of time. Figure 10 displays the results after conducting 10 rounds of test on each socket pair. An average throughput over the REQ-REP socket is 4,297 messages per second, while the average of the PUSH-PULL socket is 5,589 messages per second.

5.3.2. Blockchain Network. To measure performance of Raft in comparison with other types of ordering services supported in Hyperledger Fabric 1.4, an architecture composing

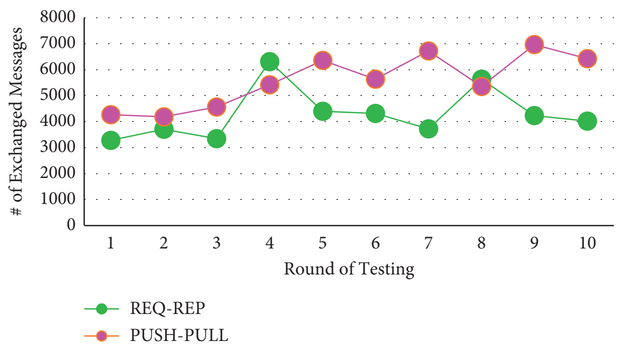

Figure 10: Throughput results on message queue data exchanges over REQ-REP and PUSH-PULL socket pairs.

of 2 organizations with a single peer each was set up. LevelDB is used as peer's state database. The experiment was conducted on Ubuntu 16.04 LTS with 2 CPU cores 2.80 GHz and 12,288 MB of memory. Hyperledger Caliper [57] is deployed as a performance measuring tool. As displayed in Table 2, latency in transaction querying of Raft is close to Solo, a single node ordering service recommended using in only development environment. For production setting, the crash fault-tolerant (CFT) multiordering services such as Kafka and Raft are recommended. According to the results, Kafka exhibits higher latency due to overhead resulted from complex Zookeeper ensembles setup. Raft, on the other hand, outperforms Kafka in both transaction types.

As mentioned in Section 4, two prototyped systems were developed and deployed on two different settings: on single-host docker network machine and on multihost Kubernetes cluster. Hyperledger Caliper is deployed in a separated

Table 2: Average latency in transaction processing.

Transaction types	Ordering service type (total transactions submitted = 300; rate = 100 TPS*)		
	Solo (s)	Kafka (s)	Raft (s)
Query	0.01	0.05	0.01
Invoke	0.33	0.75	0.64

*TPS means transactions per second.

Table 3: Performance of Raft network on different settings.

Deployment settings	Query transaction latency			Invoke transaction latency		
	Min (s)	Max (s)	Avg. (s)	Min (s)	Max (s)	Avg. (s)
Single host (docker)	0.01	0.05	0.01	0.59	0.88	0.68
Multihosts (kubernetes)	0.08	0.36	0.18	0.61	1.96	1.24

container/pod located within the blockchain network. Table 3 displays transaction processing latency of Raft network in both settings. The results show that multihost setting exhibits larger delay on both transaction types. Several external factors contribute to the delay in multihost include location of each server in the cluster, network transmission delay, differences in hardware of underlying physical machines, and so on. Nevertheless, overall performance is in acceptable level and can be improved by adjusting network configuration and/or hardware spec.

6. Conclusion

Exploiting distinct properties of blockchain and message queue, an integrated model for e-voting is proposed with the goal to protect voter's privacy and present transparency and efficiency in overall procedure. In order to preserve voter anonymity, a single-use token is introduced for representing one's eligibility to vote within a specified time period. In order to distribute the tokens, generation of short-term and long-term key pairs following CurveCP protocol allows off-chained data to be transferred securely to the designated receivers. Once ballot packets reach blockchain nodes, transactional actions will proceed corresponding with the pre-agreed consensus. Role-based permissions are defined to restrict nodes' capabilities in accessing and altering blockchain states.

Results after performing scenario-based analysis and performance testing on the prototypes show that the system can perform well in production environment. In addition, introduction of message queue as a data buffering and error handler exhibits competitive advantages over other blockchain-integrated patterns. Recently, many countries start digitalizing their core business processes, and global-wide development of digital identity infrastructure is expected to be put into practice and officially accepted as identity representation. With the use of digital ID in replacement of central authentication server, reliability in voter authentication processes will be enhanced as inputting data from corrupted sources is no longer permitted.

Data Availability

The data used to support this study are available from the first author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] F. Lehoucq, "Electoral fraud: causes, types, and consequences," *Annual Review of Political Science*, vol. 6, pp. 233–256, 2003.

[2] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," *Certificate Transparency*, 2013.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, http://bitcoin.org/bitcoin.pdf.

[4] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the USENIX Annual Technical Conference'14*, pp. 305–320, June 2014, https://raft.github.io/raft.pdf.

[5] S. Chaisawat and C. Vorakulpipat, "fault-tolerant architecture design for blockchain-based electronics voting system," in *Proceedings of the 17th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 116–121, Bangkok, Thailand, November 2020.

[6] B. Adida, "Helios: web-based open-audit voting," in *Proceedings of the 17th Conference on Security Symposium*, pp. 335–348, Berkeley, CA, USA, July 2008.

[7] B. Ian, C. Jordi, G. David, and S. Guasch, "An overview of the ivote 2015 voting system," 2015, https://www.elections.nsw.gov.au/about_us/plans_and_reports/ivote_reports.

[8] M. Epp, "Towards remote e-voting: estonian case," in *Proceedings of the Electronic Voting in Europe-Technology, Law, Politics and Society, Workshop of the ESF TED Programme together with GI and OCG*, pp. 83–100, Schloß Hofen/Bregenz, Lake of Constance, Austria, July 2004.

[9] G. Kristian, "The Norwegian Internet voting protocol," in *Proceedings of the International Conference on E-Voting and Identity*, pp. 1–18, Springer, Tallinn, Estonia, September 2011.

[10] S. Wolchok, E. Wustrow, D. Isabel, and J. A. Halderman, "Attacking the Washington, D.C. Internet voting system," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, pp. 114–128, Springer, Berlin, Germany, February 2012.

[11] J. Gerlach and U. Gasser, *Three Case Studies from Switzerland: E-Voting*Berkman Center Research Publication, Cambridge, MA, USA, 2009, https://cyber.harvard.edu/sites/cyber.law.harvard.edu/files/Gerlach-Gasser_SwissCases_Evoting.pdf.

[12] R. M. Alvarez, T. E. Hall, and A. H. Trechsel, "Internet voting in comparative perspective: the case of Estonia," *PS: Political Science & Politics*, vol. 42, no. 03, pp. 497–505, 2009.

[13] D. Springall, T. Finkenauer, Z. Durumeric et al., "Security analysis of the Estonian Internet voting system," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 703–715, Scottsdale, AZ, USA, November 2014.

[14] L. Li, "An electronic voting scheme based on ElGamal homomorphic encryption for privacy protection," *Journal of Physics: Conference Series*, vol. 1544, 2020.

[15] M. A. Will, B. Nicholson, M. Tiehuis, and R. K. L. Ko, "Secure voting in the cloud using homomorphic encryption and mobile agents," in *Proceedings of the 2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, pp. 173–184, Singapore, October 2015.

[16] Q. K. Nguyen, "Blockchain - a financial technology for future sustainable development," in *Proceedings of the 3rd International Conference on Green Technology and Sustainable Development (GTSD)*, pp. 51–54, Kaohsiung, Taiwan, November 2016.

[17] K. Fanning and D. P. Centers, "Blockchain and its coming impact on financial services," *Journal of Corporate Accounting & Finance*, vol. 27, no. 5, pp. 53–57, 2016.

[18] H. Lycklama à Nijeholt, J. Oudejans, and Z. Erkin, "DecReg," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts BCC'17*, pp. 29–34, Abu Dhabi, UAE, April 2017.

[19] Deloitte, "Blockchain applications in insurance," 2016, https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/innovation/ch-en-innovation-deloitte-blockchain-app-in-insurance.pdf.

[20] V. Paliwal, S. Chandra, and S. Sharma, "Blockchain technology for sustainable supply chain management: a systematic literature review and a classification framework," *Sustainability*, vol. 12, no. 18, 2020.

[21] J. Kishigami, S. Fujimura, H. Watanabe, A. Nakadaira, and A. Akutsu, "The blockchain-based digital content distribution system," in *Proceedings of the 2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, pp. 187–190, Dalian, China, August 2015.

[22] S. Fujimura, H. Watanabe, A. Nakadaira, T. Yamada, A. Akutsu, and J. J. Kishigami, "BRIGHT: a concept for a decentralized rights management system based on blockchain," in *Proceedings of the IEEE 5th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*, pp. 345-346, Berlin, Germany, September 2015.

[23] J. Benet, "IPFS—content addressed, versioned, p2p file system," 2014, https://arxiv.org/abs/1407.3561.

[24] European Parliament, "Council, Regulations Council (E. U) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)," *Official Journal of the European Union*, vol. 59, 2016, http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC.

[25] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *ACM Computing Surveys*, vol. 52, no. 3, 2019.

[26] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: anonymity for bitcoin with accountable mixes," *Financial Cryptography and Data Security*, vol. 8437, pp. 486–504, 2014.

[27] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," 2013, https://bitcointalk.org/?topic=279249.

[28] O. Kurbatov, P. Kravchenko, O. Shapoval et al., "Anonymous decentralized e-voting system," in *Proceedings of the International Workshop on Conflict Management in Global Information Networks (CMiGIN 2019)*, Lviv, Ukraine, 2019.

[29] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[30] R. Parizi, S. Homayoun, A. Yazdinejad, A. Dehghantanha, and K. K. R. Choo, "Integrating privacy enhancing techniques into blockchains using sidechains," in *Proceedings of the IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pp. 1–4, Edmonton, Canada, May 2019.

[31] H. Buch, "Improving performance and scalability of blockchain networks," 2019, https://www.wipro.com/blogs/hitarshi-buch/improving-performance-and-scalability-of-blockchain-networks/.

[32] Organization for the Advancement of Structured Information Standards, MQTT, "The standard for IoT messaging," 2020, https://mqtt.org/.

[33] OASIS, "Advanced message queuing protocol (AMQP) overview," 2012, http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html.

[34] iMatix Corporation & Contributors, "ZeroMQ message transport protocol," 2021, Accessed on: Jan. 8, 2021. [Online]. Available: https://rfc.zeromq.org/spec/23/.

[35] Z. Y. Lu and Z. B. Guo, "A method of data synchronization based on message oriented middleware and xml in distributed heterogeneous environments," in *Proceedings of the International Conference on Artificial Intelligence and Industrial Engineering*, pp. 210–212, Phuket, Thailand, July 2015.

[36] A. Cansever, U. Özel, O. Akin et al., "A distributed message queuing mechanism for a mailing system with high performance and high availability," in *Proceedings of the 2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, pp. 1–5, Istanbul, Turkey, October 2018.

[37] Apache Software Foundation, "Apache Kafka," 2017, https://kafka.apache.org/documentation.

[38] Y. Kim and J. Park, "Hybrid decentralized PBFT blockchain framework for OpenStack message queue," *Human-centric Computing and Information Sciences*, vol. 10, no. 1, 2020.

[39] C. Esposito, F. Palmieri, and K.-K. R. Choo, "Cloud message queueing and notification: challenges and opportunities," *IEEE Cloud Computing*, vol. 5, no. 2, pp. 11–16, 2018.

[40] Kauri Team, "Eventeum," 2021, https://github.com/eventeum/eventeum.

[41] G. Wood, "ETH EREUM: A secure decentralised generalised transaction ledger," 2014, https://gavwood.com/paper.pdf.

[42] O. Corporation, "Oracle cloud infrastructure streaming service," 2021, https://docs.oracle.com/en-us/iaas/Content/Streaming/Concepts/streamingoverview.html.

[43] Amazon Web Services, "Inc, Amazon Simple queue service," 2021, https://aws.amazon.com/sqs/.

[44] E. Baizel, "Building an event-based application with amazon managed blockchain," 2020, https://aws.amazon.com/blogs/database/building-an-event-based-application-with-amazon-managed-blockchain/.

[45] D. A. Gritzalis, "Principles and requirements for a secure E-voting system," *Computers & Security*, vol. 21, no. 6, pp. 539–556, 2002.

[46] M. Christian, "Design of distributed voting systems," 2017, https://arxiv.org/abs/1702.02566.

[47] iMatix Corporation & Contributors, "Discussion on broker vs brokerless messaging models," 2017, http://wiki.zeromq.org/whitepapers:brokerless.

[48] AuthO Inc, "Introduction to JSON web tokens," 2021, https://jwt.io/introduction.

[49] D. J. Bernstein, “Curvecp: usable security for the internet,” 2010, http://curvecp.org.

[50] E. Androulaki, A. Barger, V. Bortnikov et al., “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in *Proceedings of 13th EuroSys Conference*, Porto, Portugal, April 2018, https://arxiv.org/pdf/1801.10228.

[51] “ZeroMQ - an open-source universal messaging library,” 2021, https://zeromq.org/.

[52] M. Dirk, “Docker: lightweight linux containers for consistent development and deployment,” *Linux Journal*, vol. 2014, no. 239, 2014.

[53] The Linux Foundation, “Kubernetes,” 2021, https://kubernetes.io/.

[54] iMatix Corporation & Contributors, “ZeroMQ Authentication Protocol (ZAP),” 2013.

[55] iMatix Corporation & Contributors, “CurveZMQ,” 2013, http://curvezmq.org.

[56] The Linux Foundation, “Hyperledger explorer,” 2021, https://www.hyperledger.org/use/explorer.

[57] The Linux Foundation, “Hyperledger caliper,” 2020, https://www.hyperledger.org/use/caliper.