




Article

Online Voting Scheme Using IBM Cloud-Based Hyperledger Fabric with Privacy-Preservation

Ross Clarke ¹, Luke McGuire ¹, Mohamed Baza ^{1,*}, Amar Rasheed ² and Maazen Alsabaan ³

¹ Department of Computer Science, College of Charleston, Charleston, SC 29424, USA; clarkerp@g.cofc.edu (R.C.); mcguirel@g.cofc.edu (L.M.)

² Department of Computer Science, Sam Houston State University, Huntsville, TX 77341, USA; axr249@shsu.edu

³ Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia; malsabaan@ksu.edu.sa

* Correspondence: bazam@cofc.edu

Abstract: The current traditional paper ballot voting schemes suffer from several limitations such as processing delays due to counting paper ballots, lack of transparency, and manipulation of the ballots. To solve these limitations, an electronic voting (e-voting) scheme has received massive interest from both governments and academia. In e-voting, individuals can cast their vote online using their smartphones without the need to wait in long lines. Additionally, handicapped voters who face limited wheelchair access in many polling centers could now participate in elections hassle-free. The existing e-voting schemes suffer from several limitations as they are either centralized, based on public blockchains, or utilize local private blockchains. This results in privacy issues (using public blockchains) or large financial costs (using local/private blockchains) due to the amount of computing power and technical knowledge needed to host blockchains locally. To address the aforementioned limitations, in this paper, we propose an online voting scheme using IBM cloud-based Hyperledger Fabric. Our scheme allows voters to cast their encrypted votes in a secure manner. Then any participant can obtain the ballot results in a decentralized and transparent manner, without sacrificing the privacy of individual voters. We implement the proposed scheme using IBM cloud-based Hyperledger Fabric. The experimental results identify the performance characteristics of our scheme and demonstrate that it is feasible to run an election consisting of thousands of participants using cloud-based Fabric.

Keywords: electronic voting; blockchain; smart contract; hyperledger fabric; privacy; hyperledger caliper



Citation: Clarke, R.; McGuire, L.; Baza, M.; Alsabaan, M. Online Voting Scheme Using IBM Cloud-Based Hyperledger Fabric with Privacy-Preservation. *Appl. Sci.* **2023**, *13*, 7905. <https://doi.org/10.3390/app13137905>

Academic Editor: Gianluca Lax

Received: 17 May 2023

Revised: 27 June 2023

Accepted: 28 June 2023

Published: 5 July 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The democratic election process is a critical component of governance for systems ranging from small local governments and corporations to major national and international organizations. Democracies rely on the election process to give an equal voice to each of their members or citizens; however, traditional election schemes consisting of paper ballots are flawed in several ways. Firstly, traditional paper ballot elections require voters to visit physical polling centers. Many of these polling locations are not designed to ensure fair and equal access to all. For example, long lines and limited wheelchair access discourage voters from participating [1]. Secondly, traditional paper ballot elections do not scale well for large voter populations so it can take days or weeks for election results to be processed. Thirdly, traditional ballot elections are difficult to audit for manipulation. Traditional elections typically rely on third parties to perform poll-watching or conduct surveys (i.e., exit polling) to identify inconsistencies and increase public trust of the election results. However, these methods fail to produce perfect traceability of elections and can

themselves be corrupted [2]. The difficulty of maintaining trust and accountability in elections increases for large elections where thousands of individuals are relied upon and can act maliciously or make consequential mistakes. As a result the chances of lost or falsified ballots increases [3].

To solve these issues, governments, academic researchers, and private companies have proposed moving to an electronic voting (e-voting) scheme [4]. E-voting is a voting process in which voters can submit their votes using computers or even smart phones without the need of traditional paper ballots [4]. E-voting aims to solve the aforementioned issues by allowing anyone including people who struggle with mobility due to age or disability to vote safely and conveniently from their homes. In addition, e-voting schemes can save time tallying votes [5]. Moreover, an e-voting scheme can provide more secure and accountable elections by introducing logging schemes as a countermeasure to prevent fraudulent behavior and vote manipulations. Unlike paper ballots, e-voting can allow for the automated generation of immutable audit trails that can be leveraged to validate election integrity.

While e-voting schemes have the potential to solve issues associated with traditional paper ballot elections, the majority of the current e-voting schemes rely on a central third party to run the election [6–8]. This makes them vulnerable to single-point-of-failure attacks as well as privacy concerns of submitted votes by both internal and external attackers [9–11]. They are vulnerable to distributed denial of service (DDoS) and Sybil attacks launched by malicious users and external attackers. To address the limitations of centralized e-voting schemes, many researchers have proposed using the blockchain to host an e-voting scheme [12–18]. The blockchain technology, at its core, contains a ledger, or transaction history, that follows an append-only scheme. Due to the immutable structure of the ledger, the use of a blockchain would naturally deter vote tampering and manipulation. This is because previous votes written to the ledger cannot be overwritten or changed. A set-in-stone timeline is created as the election takes place, recording every encrypted vote and action taken on the blockchain.

However, the majority of existing implemented blockchain-based voting schemes are based on public blockchains, which could result in privacy issues as the votes are easily available to the public if additional measures such as vote encryption are not implemented [16]. In [5,17], the Ethereum blockchain is used, which posts votes publicly, providing no election privacy. While [17] utilizes encryption of votes to protect voter privacy, [5] provides no such safeguard. This transparency sacrifices privacy to allow for an easily auditable scheme. In [19], homomorphic encryption is used to protect the ballots. The organizer can verify the ballot when tallying the votes without obtaining any details about the content of the ballot. However, the scheme suffers from high computation and communication overheads due to the time and computational power required to aggregate homomorphic encrypted ballots and to decrypt the large ciphertexts that were produced by the scheme.

In this paper, we propose an electronic voting scheme using IBM Cloud-based Hyperledger Fabric. We opt to develop our e-voting scheme using the Hyperledger Fabric blockchain for two main benefits. Firstly, Hyperledger Fabric is a private blockchain, meaning only identified nodes can participate in the network, and these known participants can access the data stored on each block. Secondly, using IBM cloud-based Hyperledger Fabric removes the burden needed to set up and run a private local blockchain. Self-hosting a private blockchain requires thousands of dollars worth of hardware and a great wealth of technical knowledge to set up and maintain [20]. Our developed scheme is unique, as most existing works depend on using *public or locally-hosted blockchains* to run an e-voting scheme. Moreover, ours is not a costly solution, as elections only run on a scheduled basis, e.g., once a year or once every four years [12,13,15,21–23]. Our proposed e-voting scheme allows small businesses and governments alike to run an election in a decentralized, transparent, and secure manner.

We developed a custom smart contract with four custom attributes to drive all of the functionality required to host an election. The smart contract handles the creation of the election, registering of voters, casting of votes, and tallying of votes. In our voting scheme, the blind signature cryptoscheme is leveraged so that voters can obtain anonymous voting tokens so they can cast their votes to the blockchain while maintaining the privacy of their real identity, which is not shared with the network. To evaluate our proposed scheme, we have used Hyperledger Caliper to measure and evaluate the performance in terms of throughput, latency, and error rate. The results indicate that the use of the IBM Cloud to remotely host a blockchain-based e-voting scheme is feasible and scalable to elections involving 10,000 voters.

The rest of this paper is organized as follows. In Section 2, we perform a literature review and discuss related works. Next, in Section 3, we present the considered scheme model followed by the design goals of our proposed scheme. Section 4 gives an overview of the preliminaries used in our paper. Section 5 discusses the considered blockchain design and our developed smart contract. In Section 6, we present the performance evaluations of our proposed scheme. An analysis of the proposed scheme is given in Section 7. Finally, we give concluding remarks in Section 8.

2. Related Work

In this section, the literature review is provided for e-voting schemes. Table 1 provides a comparison between our scheme and existing e-voting schemes.

Table 1. A summary comparison between our scheme and existing e-voting schemes. Note: ✓ shows a functionality a realized feature; ○ shows a (partially) realized function by relying on a central trust × is an unrealized feature. (Author’s own processing).

	Architecture	Election Privacy	Identity Privacy	Easy for Voters to Audit Results	Implemented Performance Evaluation	Affordable
Current Paper Ballot scheme ¹	Centralized	×	○	×	✓	×
Hanifatunnisa et. al. [5]	Hybrid	×	○	×	✓	×
Liu and Wang: [12]	Blockchain	×	✓	✓	×	×
Hjálmarsson et al. [13]	Blockchain (public)	×	×	✓	×	×
Seifelnasr et al. [17]	Blockchain (public)	×	✓	✓	✓	×
Our scheme	Blockchain (private)	✓	✓	✓	✓	✓

¹ <https://www.blablacar.com/>.

E-Voting In Industry. In [24], Specter et al. analyze the first blockchain-based e-voting scheme used by a state in its general elections, Voatz. The app used in this scheme was found to contain severe security flaws such as requirements of weak passwords, easily bypassed anti-virus, and an API that can take full control of a user’s device. In addition, voter’s choices could be deduced via a side-channel attack.

Score Voting. In [25], Yang et al. present a distributed scored voting scheme that uses a novel dual zero-knowledge proof to preserve privacy. Each component of the crypto scheme is performance tested locally using Charm, then the scheme is benchmarked locally on the Ethereum blockchain to determine average costs. Unlike our scheme, all scheme testing was conducted locally.

Self-Tallying. In [26], Lin et al. present a complete self-tallying e-voting protocol based on [27]. Minimal performance testing was conducted. The authors present the average time for proving and verifying a single vote, but no testing of scale is shown. In [28], Li et al., propose a modification of [27,29] to deliver a scheme model of self-tallying voting schemes based on blockchain in decentralized IoT. The scheme is performance tested, but scalability is capped to only 12 voters. In [30], Li et al. propose and implement a traceable self-tallying e-voting protocol using a homomorphic time-lock puzzle to provide self-tallying. However, both on-chain and off-chain performance testing was limited to less than 10 voters, showing minimal scale for the scheme.

Blockchain Based E-Voting schemes. In [21], Onur et al. propose a blockchain-based, ranked-choice election protocol on the public Ethereum blockchain. While performance testing was only implemented locally using Hardhat, virtualized scalability was demonstrated up to 10,000 voters. Result verification through Merkle Tree generation was demonstrated up to 100,000 voters, but no election scheme functions were tested at this scale. In [31], Han et al. propose and implement a blockchain-based self-tallying voting scheme with software updates in decentralized IoT. The authors tested the voting scheme using a Raspberry Pi device to simulate an IoT device, but did not test the update integration. In addition, the testing only included up to 100 voters. In [32], Mukherjee et al. present a framework to be used for blockchain-based elections, but no implementation beyond broad software architecture concepts is discussed. Since no implementation is shown, there are also no performance metrics for the proposal.

On-Chain Voting schemes. E-voting schemes that utilize the blockchain for the conduction of the voting process require more resources, but allow for a more open, transparent voting process. In [12,13,15,22,23], the election process logic exists entirely on-chain. In [13,22], no encryption is used to protect voter identities. In addition, the Rinkeby Network, a private but widely used blockchain, is used which poses privacy concerns. While Koç et al. conducted performance testing on their scheme, they failed to show any scalability beyond four voters. In [12,13,15,23], no performance testing or scalability is shown. Without performance testing, it is impossible to conclude if the proposed system is a feasible solution at any scale. In [15], some votes have the chance of going uncounted if they are cast at the same time. Ref. [13] relies on multiple nodes representing each voting district, which establishes a significantly high cost to scale. Ref. [12] allows the viewing of vote totals before casting a ballot, which could lead to voter manipulation and sway.

Blockchain as Vote Storage. To combat the high resource cost of utilizing blockchain as a voting solution, some proposed schemes simply leverage a blockchain as a database to store votes, but these solutions rely on traditional ballot casting, which can lead to vote manipulation as well as other forms of voter fraud. Refs. [5] utilize blockchain to store votes or vote counts, but still rely on traditional polling centers for ballot casting. Refs. [5] rely on traditional paper ballots at polling centers. In addition, no individual votes are stored on the blockchain, only vote totals from each polling center, allowing for fraud and vote manipulation on the local level at polling centers. Ref. [5] only verifies polling locations, and not individual voters. This passes the reliance of voter verification to unreliable traditional methods.

The main advantages to our approach compared to the previous proposals are as follows:

- Our scheme hosts all aspects of the election on the blockchain, allowing for an entirely remote election to take place. This aspect allows for remote voters to still have their voice heard. Since our scheme aims to address the requirement for secure online voting beyond national presidential elections, a fully remote voting system is necessary. This is particularly relevant as numerous companies that wish to conduct internal elections or employee surveys have implemented a 'work-from-home' policy for their employees. Without the need for a polling center, geographically distributed voters who would otherwise be unable to attend an election can have their voices heard.
- Our scheme utilizes the IBM Cloud infrastructure to host the latest version of the Hyperledger Fabric blockchain. By hosting the private blockchain in the cloud, managing and hosting elections requires less physical hardware and resources. In addition, the IBM cloud provides a scalable "pay as you go" platform, allowing elections of any size to be hosted by our scheme.
- Our scheme has been implemented and tested, allowing for the collection of performance metrics. This implementation proves that the conceptual ideas proposed for our blockchain e-voting scheme are able to be put into action. These performance metrics showed us that our scheme, when hosted on the IBM Cloud with minimum

required cost, is easily scalable up to 10,000 voters. In addition, these performance metrics reflect expected results from [33].

3. Scheme Model

In this section we first describe the considered scheme model, followed by the threat model, and finally the design goals that we established.

3.1. Scheme Model

As depicted in Figure 1, the considered network model has the following entities.

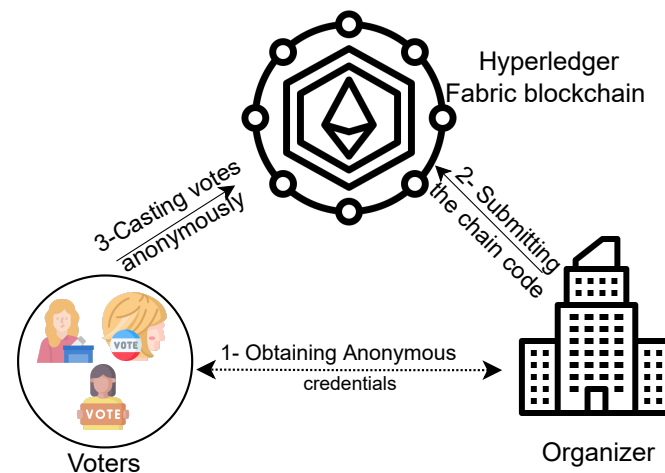


Figure 1. Illustration of our considered scheme model. (Author’s own processing).

- Blockchain Network.** The private Hyperledger Fabric blockchain hosted on IBM’s cloud infrastructure. The Hyperledger Fabric requires the use of a minimum of three peers: an endorser peer, which utilizes the smart contract to write new transactions; an orderer, which verifies the new transaction and writes it to a new block on the blockchain; and the committer, which checks the validity of all transactions on the blockchain and writes valid transactions to the ledger. Our blockchain network utilizes a voter peer, which utilizes our developed smart contract to write our transactions (any action taken within the election), and an orderer, which verifies and writes each transaction. Each of these peers act also as the committers for our network. In a production environment, additional peers can be added to act solely as committers, funds permitting. In accordance with [34], our organization and orderer have separate Certificate Authorities (CA). Hyperledger Fabric utilizes certificate authorities to identify each peer and define each functionality. If additional peers are added solely as committers, a third CA is required; however, all additional committer-only peers can share the third CA.
- Voters.** The voter is an entity that can cast votes to the ballot. Note that voters do not maintain a whole copy of the ledger; but they can run the scheme using lightweight nodes, which lets them communicate with the network, read from the blockchain and submit transactions [35]. In our scheme, we implemented a preferential voting scheme; so, when casting a vote, the voters rank three candidates to receive vote payouts. The candidate selected first receives three votes, the second receives two votes, and the third one vote.
- Organizer.** This is the organization hosting the election, such as small businesses, states and governments, etc. Specifically, the organizer is represented by a person in the IT department who has the technical skills to set up a blockchain environment on the IBM Cloud Platform. The organizer’s main duty is to initiate the election, e.g., defining the candidates’ names and configuring the blockchain on the IBM Cloud by

defining the nodes necessary to be used to carry out the election, creating the channels between the nodes, and uploading/initializing the packaged smart contract.

3.2. Threat Model

Since attacks may come from both internal and external attackers, this work is following the blockchain threat model given in [36], where the blockchain is trusted for immutability and availability but not designed to maintain privacy. Therefore, we assume that both the organizer and the blockchain are *honest-but-curious*, meaning they correctly run the voting scheme but also aim to infer voters' sensitive information. Therefore, the blockchain shall not know any information about the voters' votes other than the election results, i.e., the total number of votes per candidate.

We consider threats that may come from some of the blockchain validators/miners who could misbehave with the aggregated data. In other words, blockchain validators may try to modify the votes of legitimate voters' votes (i.e., double counting votes twice) before aggregating the total votes by double counting votes to *give a candidate more votes than their actual vote count*.

Finally, existing works depend on voters' anonymity to protect voter's privacy; however, internal attackers can link voters' pseudonyms to their real identities by snooping their messages generated from their IP addresses.

3.3. Design Goals

Based on the considered threat assumptions and to ensure security and privacy against internal and external attackers, our design goals for our scheme are laid out as follows:

- The proposed e-voting scheme should neither rely on a central entity to run the election nor require certain infrastructure to set up a private blockchain network. Central entities are vulnerable to a single-point-of-failure or attack. Our goal is to develop and implement the scheme using an open source blockchain platform and cloud-based infrastructure to avoid hardware costs required to host a private blockchain.
- The proposed scheme should protect voters' privacy by ensuring the following: (i) Only authenticated voters should be able to cast their votes to the ballot while protecting their real identities from being revealed or tracked. (ii) Since anonymity alone is not sufficient to ensure voters' privacy against internal attackers who can link voters' pseudonyms to their real identities by knowing their IP addresses, the proposed scheme should withstand such eavesdropping attacks by making it difficult to determine the exact selection of a specific voter.
- The scheme should also resist any fraud or manipulation attacks that would give preference to specific candidates over others. For example, blockchain nodes or internal attackers may try to replay legitimate voters' votes or recorded transactions on the blockchain to give a candidate more votes than their actual vote count. In other words, authenticated voters can cast their encrypted votes, and the blockchain nodes can only compute the number of votes per candidate *correctly*.

Finally, one primary goal of this work is to *evaluate the efficacy of developing the blockchain-based e-voting scheme on top of IBM cloud-based infrastructure* in terms of throughput and scalability.

4. Preliminary Background

4.1. Blind Elliptic Curve DSA Signatures

In this cryptoscheme, a *user* checks if a signature on a message M is valid from the *signer* while hiding M from the signer. In our voting scheme, the blind signature cryptoscheme in [37] is leveraged so that voters can obtain anonymous voting tokens so they can cast their votes to the blockchain while keeping their real identity private. The aforementioned cryptoscheme is used because it is efficient in terms of computations with shorter signatures and it contains the following steps:

1. All parties are assumed to use an elliptic curve of order n with generator G . $P = d \cdot G$ is the signers' public key, where $d \in \mathbb{Z}_n^*$ is the private key.
2. The signer first selects a random element $k \in \mathbb{Z}_n^*$ and then he/she sends $R = k \cdot G$ to the requester.
3. Then, the requester selects the following random elements $\gamma, \delta \in \mathbb{Z}_n^*$ to compute $A = R + \gamma \cdot G + \delta \cdot P$. Let x be the x -coordinate of point A , and $t = x \bmod n$. The requester then computes $c = H(M||t) \bmod n$ and sends $c' = (c - \delta) \bmod n$ to the signer. $H(\cdot)$ is a cryptographically secure hash function, and $H : \{\cdot\}^* \rightarrow \mathbb{Z}_n^*$.
4. The signer then computes $s' = (k - c' \cdot d) \bmod n$ and sends the result back to the requester.
5. The requester then computes $s = (s' + \gamma) \bmod n$ and the signature of M is stored as (s, c) . Finally, to validate the signature, the verifier calculates $A = c \cdot P + s \cdot G$. Then, $t = x \bmod n$ is computed, where x is the x -coordinate of point A . The verifier verifies if $c \stackrel{?}{=} H(M||t) \bmod n$.

4.2. Verifiable Aggregator Oblivious Encryption

Aggregator oblivious encryption cryptoscheme enables the computation of aggregated data M of n users' individual data in a privacy-preserving way by an untrusted aggregator. The main idea is that a user i encrypts a message m_i and sends the ciphertext c_i to the aggregator. Then, the aggregator computes the sum $M = \sum_{i=1}^n m_i$ from $\{c_i\}_{i \in [1, n]}$. On the other hand, verifiable aggregator oblivious encryption [38] ensures the correctness of the aggregated data. In this cryptoscheme, the user i computes a tag σ_i in addition to c_i , and the aggregator generates a publicly verifiable proof σ from $\{\sigma_i\}_{i \in [1, n]}$ that proves the correctness of M . Note that ensuring the correctness of the aggregated data is very critical in a voting scheme to ensure a fair ballot, and the results of the ballot reflect the voters selections. In our scheme, we employ verifiable aggregator oblivious encryption to make sure that blockchain nodes cannot manipulate the ballot results by, for example, counting multiple votes for the same voter. The verifiable aggregator oblivious encryption consists of the following algorithms:

- $Setup(1^\lambda) \rightarrow \text{param}, sk_A$: Given a security parameter λ as an input, this algorithm generates public parameters PP and a secret key of aggregator sk_A , a set of user secret keys $\{sk_i\}_{i=1}^n$, and the aggregate verification key vk .
- $Enc(\text{param}, t, x_i) \rightarrow c_i, \sigma_i$: Given param , t , a value x_i , and sk_i , this algorithm produces a ciphertext c_i and a tag σ_i .
- $AggrDec(\text{param}, \{(c_i, \sigma_i)\}_{i=1}^n, sk_A) \rightarrow X_t = \sum_{i=1}^n x_{i,t} \bmod M$: This algorithm is considered the aggregation and decryption algorithm and it uses the param , t , the ciphertexts and tags $\{(c_i, \sigma_i)\}_{i=1}^n$, σ_t and sk_A , to produce $\{(c_i, \sigma_i)\}_{i=1}^n$, and the proof σ_t where M is some fixed integer contained in param .
- $VerifySum() \rightarrow :$ This verification algorithm is applied to the aggregation algorithm and it takes param , t , vk_t , and (X_t, σ_t) as input, and outputs 1 or 0.

5. Proposed Solution

This section details our proposed cloud hosted blockchain e-voting scheme, and demonstrates how our scheme meets all our design goals. We will explain our proposed anonymous credential scheme, our blockchain choice and setup, our developed smart contract functionality, and how these work together to accomplish all of our design goals.

5.1. Initialization

In this phase the organizer runs the *Setup* algorithm explained in Section 4.2 to compute public parameter PP and a secret key of aggregator sk_A used by the blockchain validators, a set of user secret keys $\{sk_i\}_{i=1}^n$, and the aggregate verification key vk , as follows. The organizer chooses $(p, e, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ where $\mathbb{G}_1, \mathbb{G}_2$ and \mathbb{G}_T are groups of λ -bit prime order $p = M$, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map.

Let $H_i : \mathbb{Z} \rightarrow \mathbb{G}_1$ ($i = 1, 2, 3, 4, 5$) be hash functions, then choose $\gamma, s_1, \dots, s_n, t_0 \xleftarrow{\$} \mathbb{Z}_p$, set $s_0 = -\sum_{i=1}^n s_i, h = g_1^\gamma$, and $Z = e(h, g_2)$. Finally, the *Setup* algorithm Outputs PP as $PP = ((p, e, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T), Z, H_1, H_2, H_3, H_4, H_5), sk_A = (s_0, t_0), sk_i = (s_i, t_0, h)$, and $vk = \emptyset$.

5.2. Obtaining Anonymous Credentials

Privacy is critical in an e-voting scheme. The scheme shall allow only authenticated voters to cast their votes while protecting their real identities from being tracked by the blockchain nodes, organizers, and other potential eavesdroppers. To address this issue, voters need to obtain untraceable tokens from the organizers. The process of acquiring the tokens is as follows.

The voter chooses random element $\{s\} \in \mathbb{Z}_n^*$ and then computes the token public keys $\{TP\}$, where $TP = s \cdot G$. Then, the voter and the organizer execute the blind signature protocol, discussed in Section 4, so that the voter can acquire a valid signature $sig_O(TP)$ for the token coin TP without revealing TP to the organizer. The process is as follows:

Step 1: A voter v_i initializes the communication by sending a message msg_1 , as follows:

$$msg_1 = ID_{v_i} \parallel Sig_{v_i}(ID_{v_i})$$

where ID_{v_i} is the voter's real identity, such as his/her registered email address account, and $Sig_v(ID_{v_i})$ is the signature on the whole message using the voter's private key.

Step 2: The organizer chooses a secret element $k \in \mathbb{Z}_n^*$ and computes $R = k \cdot G$ and he/she sends msg_2 back to the voter

$$msg_2 = R \parallel Sig_O(R)$$

where $Sig_O(R)$ is the organizer's signature on the message.

Step 3: The voter computes $A = R + \gamma \cdot G + \delta \cdot P$, where $\gamma, \delta \in \mathbb{Z}_n^*$ are random elements and computes $t = x \bmod n$, where x is the x -coordinate of point A . The voter computes $c' = (c - \delta) \bmod n$, where $c = H(TP \parallel t) \bmod n$. The voter sends msg_3 that includes c' as well as the voter's signature $Sig_{v_i}(c')$ to the organizer:

$$msg_3 = c' \parallel Sig_{v_i}(c')$$

Step 4: The organizer computes $s' = (k - c' \cdot d) \bmod n$ where d is the organizers' secret key and they reply back with msg_4 , where,

$$msg_4 = s' \parallel Sig_O(s')$$

Finally, the voter uses msg_4 to compute $s = (s' + \gamma) \bmod n$ and stores the signature on TP_{v_i} ($Sig_O(TP_{v_i}))$ as (s, c) . The organizer is oblivious to the voters' tokens, i.e., public keys. Note that, although this phase require the voters to use his/her real identity to obtain anonymous voting tokens, and due to the use of the blind signatures, the organizer would not be able to link a specific voting token ($sig_O(TP_{v_i}))$ to the voter v_i when the token is used in the voting process.

5.3. Submitting Anonymous and Encrypted Votes

Each voter encrypts his/her vote using the *Enc* algorithm, discussed in Section 4.2, using PP, t, x_i, sk_i, PP as an input, as follows. The voting ballot is shown in Figure 2. Each ballot B_{v_i} is divided into elements, where each element corresponds to one candidate and each voter selects the candidates by putting 1 in the element corresponding to this candidate. For instance, if the message size is 1000 bits and the number of candidates is five, the message is then divided into five elements, each of them a size of 200 bits. A voter v_i chooses $d_{v_i} \xleftarrow{\$} \mathbb{Z}_p$ and then computes $vk_{v_i} := g_1^{d_{v_i}}$, and computes

$$C_{B_{v_i}} = g_1^{B_{v_i}} H_1(\tau)^{s_i} H_2(\tau)^{t_i}$$

and

$$\sigma_i = h^{B_{v_i}} H_3(\tau)^{s_i} H_4(\tau)^{t_i} H_5(\tau)^{v_i}$$

The smart contract handles all of the necessary election logic and the smart contract voting scheme is summarized in Algorithm 1. To then cast a vote, the voters need to use the anonymous tokens described in Section 5.2 to be able to vote for their specific candidates, as follows. The voter uses the private key that corresponds to the token TP_{v_i} to sign and broadcast the following message to the blockchain:

$$msg_5 = (C_{B_{v_i}}, \sigma_i, vk_i, sig_O(TP_{v_i}), \sigma(v_i)),$$

where $\sigma(v_i)$ is the signature on the whole message. Then, the blockchain should first verify if the voter is authenticated by checking the organizer's signature on the TP_{v_i} ($sig_O(TP)$), as follows. The *blockchain* first computes $A = c \cdot P + s \cdot G$. Then, it computes $t = x \bmod n$, where x is the x -coordinate of point A . The verifier checks if $c \stackrel{?}{=} H(msg_5 || t) \bmod n$. If the verification succeeds, then the blockchain continues to execute the smart contract Algorithm 1. In the example, the vote will only be written as a valid transaction if the voter has not cast a ballot already, the current time lies within the election start and end times, and the voter has their anonymous tokens described in Section 5.2. This logic solely utilizes the smart contract and data stored on the ledger, eliminating the need for external centralized databases during the election process.

Algorithm 1 Pseudocode for *voterContract* (Author's own processing)

```

1 contract voterContract
2   function InitLedger(ctx)
3     WS ← voters // Adds voters to World State
4     WS ← votableItems // Adds votableItems to World State
5     WS ← election // Adds Election to World State
6   function CastVote(ctx, voterID, electionID, votableItems)
7     newBallot(voterID, electionID, votableItems) // generate ballot with choices
8       of voter
9     logBallot() // Link the ballot to the voter
10    // increment count of picked items
11  function GetResults(ctx, electionID)
12    // set results for all candidates to 0
13    for s ← 0 to Candidates.length do
14      | Candidate[s].count = 0
15    end
16    // loop through all voters
17    // for every votableItem on voter's ballot, increase candidate's count
18    for s ← 0 to voters.length do
19      | curBallot = voters[s].ballot for v ← 0 to curBallot.votableItems.length
20      | do
21      | | curBallot.votableItems[v] += len(election.VotableItems) - i
22      | end
23    end

```

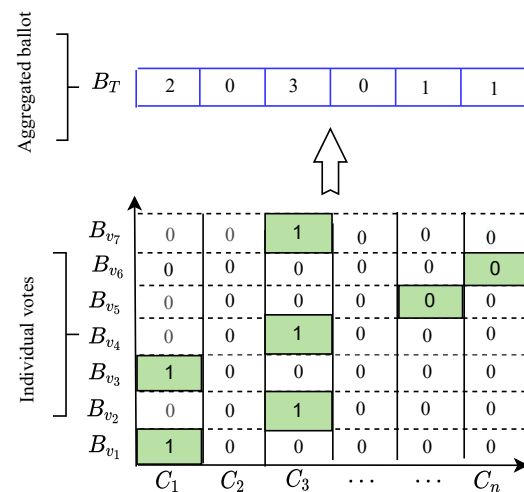


Figure 2. The individual ballot format and the aggregated ballot. The sum of the i -th element in the ballot gives the number of votes for the candidate C_i . (Author’s own processing).

5.4. Obtaining Election Results

This phase starts once the blockchain receives n total votes. The blockchain should run the following algorithm *AggrDec*, discussed before in Section 3, as follows.

The *AggrDec* uses (param, t , $\{(c_{i,t}, \sigma_{i,t}, vk_{i,t})\}_{i=1}^n, sk_A$) : Parse $sk_A = (s_0, t_0)$. Compute

$$V_t = H_1(t)^{s_0} H_2(t)^{t_0} \prod_{i=1}^n c_{i,t} = g_1^{X_t}$$

where $X_t = \sum_{i=1}^n x_{i,t}$, and the discrete logarithm V_t with respect to basis g_1 is solved. Then, the algorithm computes

$$\sigma_t = H_3(t)^{s_0} H_4(t)^{t_0} \prod_{i=1}^n \sigma_{i,t} \text{ and } vk_t = \prod_{i=1}^n vk_{i,t}$$

Output (X_t, σ_t, vk_t) .

Finally, the blockchain broadcasts the results of the ballot as well as the tags for verification. The organizer and the voters can check the *correctness* results by running the *VerifySum*, as follows.

First, the verifier, i.e., the organizer/voters uses the param, t, X_t, σ_t, vk_t to check if

$$\frac{e(\sigma_t, g_2)}{e(H_5(t), vk_t)} = Z^{X_t}$$

holds. If so, this means the ballot was computed correctly by the blockchain.

5.5. Blockchain Design and Methodology

As explained earlier, the voter needs to obtain anonymous tokens from the organizer using blind signatures, then the voters can cast anonymous and encrypted votes to the smart contract. Finally, blockchain nodes aggregate all the votes that have been cast to obtain the result of the election in a privacy-preserving manner.

In our proposed scheme, voters, as seen in Figure 3, interact with the endorser peer through a web application. Their input prompts the endorser peer to utilize the smart contract to write and execute a new transaction. This transaction is simulated by the endorser peer and forwarded to the orderer to verify the transaction. When the orderer verifies the transaction, it writes a new block to be distributed to each peer in the network. Once the transaction has been written as a new block, the network committers, which consist of every peer on the network (including the endorser), verify the integrity of the transaction. If verified, the new block is added to the channel’s ledger.

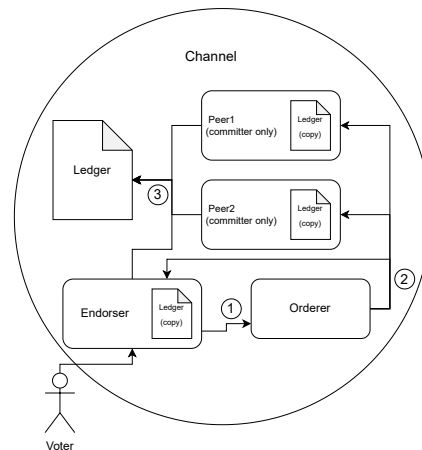


Figure 3. Our considered blockchain network design. In (1), Endorser simulates the transaction from the proposal. In (2), Orderer writes block from the transaction. In (3), Committers validate blocks and append them to the ledger. (Author’s own processing).

Throughout this whole process, Certificate Authority (CA) nodes are used to identify each peer and orderer. In addition, each CA creates and validates each transaction. For example, in our scheme, the Voter CA is used by the Voter Peer to sign, or endorse, the transaction during its creation. Again, after the transaction has been committed to the ledger, the Voter CA is used to verify all endorsements on the transaction to validate it before writing the transaction to the Voter Peer’s copy of the blockchain ledger. In the proposed model, we only include a single voter peer. However, this can be expanded upon to further distribute the endorsement process with additional peers, thus requiring additional signatures, or endorsements, before committing the transaction.

6. Performance Analysis

The performance of our proposed scheme is evaluated in this section. The experiment setup is explained, followed by key metrics used to evaluate the scheme and, finally, the results of these experiments are given.

6.1. Experiment Setup and Methodology

Hyperledger Fabric Platform. To implement our proposed scheme, we leveraged the IBM Cloud infrastructure to host the required hardware. According to the Hyperledger Documentation, this private blockchain is “an open source enterprise-grade permissioned distributed ledger technology (DLT) platform” [20]. This means that a level of trust is required within the network, removing anonymity between nodes, but ensuring only authorized nodes are added to the blockchain. The permissioned nature of the Hyperledger Fabric sets it up as a great tool for companies to utilize internally, without the fear of outside manipulation. In addition, Hyperledger Fabric is a cryptocurrency free blockchain, eliminating some major risk and attack incentives. In order to host our private blockchain network, we first deployed a free Kubernetes cluster consisting of 2 vCPUs with a total of 4 GB of RAM using the IBM Kubernetes Service [39]. On this cluster, we linked an instance of the IBM Blockchain Platform service. This established the environment in which we set up our private Hyperledger Fabric blockchain. Utilizing Fabric version 2, we have established an Endorser peer, a single orderer, and two committer peers. These nodes all share a common channel, on which we instantiated our developed smart contract. Hyperledger is built to be highly configurable through its modular design and use of smart contracts. Hyperledger’s smart contracts, or chaincode as they refer to it, drive the functionality of the blockchain. The chaincode acts as the logic for the blockchain, meaning this is the fundamental aspect of our proposed solution. While many blockchains require their smart contracts to be written in domain-specific languages, Hyperledger allows smart contracts to be written in general-purpose programming languages like Java, Go, and

Node.js. Our developed smart contract is written in Go. In addition, Hyperledger Fabric introduces a new architecture for transactions in which transactions are executed before they are ordered, dramatically reducing the time necessary per transaction. The new execute-order-validate approach first executes a transaction, checking its correctness. Then, an orderer submits the transaction to the rest of the blockchain via a consensus protocol. Finally, a validator validates the transaction. This three-step process allows for multiple peers to execute their transactions before they are committed to the ledger, allowing for theoretically more transactions per second with less cpu usage than other blockchains [40]. Our smart contract defines four objects: an election, a ballot, a voter, and a votable Item. These four objects not only store the data necessary to host an election, but also interact with each other to form the base of our election functionality. Our contract class extends the Hyperledger Fabric base Contract class, allowing for easy import and initialization onto the endorser peer. Our contract class utilizes our four custom attributes to allow users to register to vote, cast votes, and query the election results. To instantiate the smart contract, we utilized the IBM Blockchain web console to propose our chaincode. Then, once each node in the channel signed the proposal, our developed smart contract took effect.

Hyperledger Caliper Setup. We leveraged Hyperledger Caliper version 2.0 to stress test and collect performance metrics on our scheme. Caliper is a blockchain benchmark framework that implements custom-defined use cases to produce performance reports. As seen in Figure 4, Caliper utilizes workers to generate and serve transactions to a blockchain network. These workers can be configured to serve transactions representing the different functions defined by our developed smart contract; `getBallot`, `castVote`, and `getResults`. In addition, the workers can serve these transactions at varying rates. the number of transactions to be served must also be set. To run these tests, we used an Ubuntu 20.04 DigitalOcean droplet with 1 GB memory and 1 shared vCPU as the test harness to remotely connect to our IBM Cloud-based Hyperledger Fabric instance.

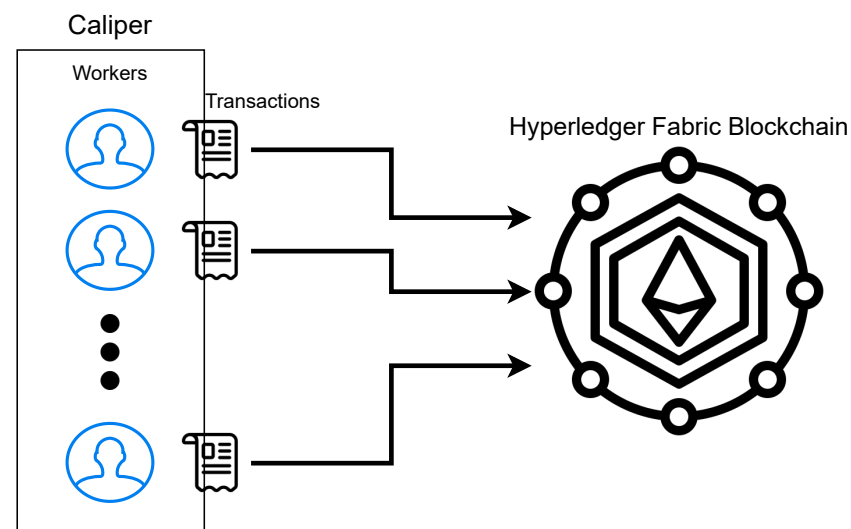


Figure 4. Deployment model of Hyperledger Caliper. Caliper generates and sends a set number of transactions at a defined send rate to the blockchain. In each run of Caliper, these transactions all represent the same type of transaction, as defined by our developed smart contract (read or write). When Caliper finishes one job, another can be started to simulate another type of transaction. (Author’s own processing).

6.2. Performance Metrics and Benchmarks

Performance Metrics. We ran each test experiment independently, utilizing Caliper’s workload modules to construct and send transactions. Three performance metrics of our implemented scheme were then recorded and analyzed to determine its applicability and scalability:

- *Transaction Throughput.* This metric is used to show the rate at which valid transactions are committed by our blockchain network, measured in transactions per second (tps). As emphasized by the Performance and Scalability Working Group (PSWG) in [41], this metric determines the rate at which transactions are committed across the entire blockchain network, not just at a single node. This helps to provide more accurate details of the experiment as, for a transaction to be functional across the network, it must be reflected by every node. A high tps value indicates that the widespread implementation of our proposed scheme is viable, enabling a large-scale participation of voters in the election. Conversely, a low tps would increase the time required for a significant number of voters to participate in the election.
- *Transaction Latency.* This metric, computed as transaction confirmation time at network threshold less the submit time of the transaction, demonstrates the time taken for a transaction to be usable across the network. Measuring transaction latency across all nodes of the network ensures a more realistic timing evaluation that reflects the true latency experienced by all users, rather than just the voter.
- *Error Rate.* This is the rate at which submitted transactions are rejected or dropped. The error rate is computed as the number of failed transactions divided by the number of sent transactions. This metric gives an accurate way to determine the maximum operational stress our scheme can handle without fault. Note that while error rate approaches zero in ideal conditions, we assume that ideal conditions are impracticable at large scales due to the distributed architecture of Hyperledger Fabric.

Use Cases and benchmarks. Based on the aforementioned metrics, three different experiments are used to evaluate and analyze the efficacy of the proposed scheme. These use cases are shown in Table 2 and they are described in detail as follows:

- *Experiment I* shows varying transaction rates' impact on our scheme's throughput and latency to show the scheme's ability to process transactions when receiving varying amounts of transactions. This simulates the impact of multiple voters using the scheme concurrently.
- *Experiment II* analyzes the varying of total transactions' impact on the throughput and latency at differing transaction send rates to show the scheme's scalability. This demonstrates the scheme's ability to handle elections involving differing scales of total voters.
- *Experiment III* shows the error rate due to varying total transactions and transaction send rates.

For each of the use cases, we tested the main functions of our scheme's functions, namely GetBallot and CastVote. The GetBallot function reads the ledger and returns the ballot associated with a specific voter. The CastVote function generates a new ballot for the voter to use in the election and writes the ballot to our blockchain with the voter's choices.

Table 2. Parameters for Performance Evaluation (Author's own processing).

	Send Rate for GetBallot (tps)	Send Rate for CastVote (tps)	Number of Transactions
<i>Experiment I:</i> Transaction send rates and its impact on scheme throughput and latency	50, 100, 150, 200, 250, 300	5, 25, 50, 75, 100	1000
<i>Experiment II:</i> Varying numbers of transactions impact on scheme throughput and latency	50, 100, 150, 200, 250, 300	5, 25, 50, 75, 100	1000, 2000, 10,000, 20,000
<i>Experiment III:</i> Transaction send rates impact on error rate	50, 100, 150, 200, 250, 300	5, 25, 50, 75, 100	1000, 2000, 10,000, 20,000

6.3. Results and Discussion

In this section, the results of our performance testing are given and analyzed.

6.3.1. Experiment I

Figure 5 shows the average throughput and latency for GetBallot and CastVote functions at varying transaction send rates. The results indicate that as transactions are sent at higher rates, the scheme reaches a peak throughput, at which point latency begins to rise and the throughput is unable to match further increases to the send rate. This maximum throughput also is dependant on the transaction type: around 200 tps for GetBallot, a read function, and around 75 tps for CastVote, a write function. As seen in Figure 5 the average latency does not rise until the send rate eclipses the scheme's peak throughput. These results are generalized as follows:

- A large number of voters, 200 per second, can concurrently fetch ballots without significant delay.
- Fewer voters, 75 per second, can concurrently cast votes without significant delay.
- The type of transaction will affect scheme performance.

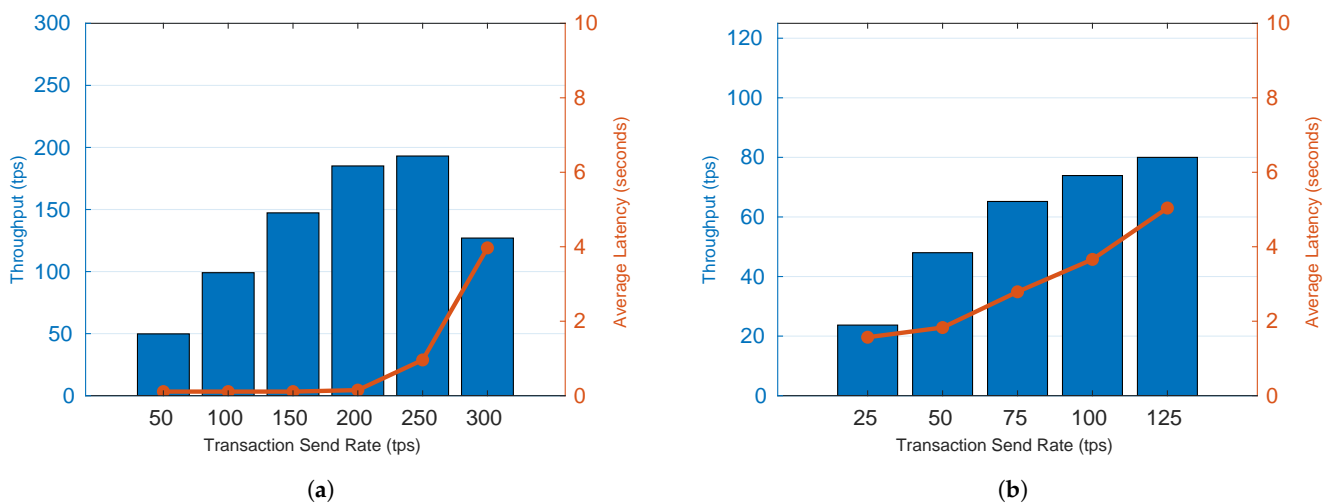


Figure 5. Impact of Transaction Send Rate on Throughput and Latency: 1000 Transactions. (a) GetBallot; (b) CastVote. (Author's own processing).

6.3.2. Experiment II

Figures shown in Figure 6 illustrate the average throughput and latency for GetBallot and CastVote functions for varying total numbers of transactions. The results indicate that there is no significant impact on total number of transactions, as long as the send rate remains below the scheme's peak throughput. As send rates increase beyond the peak throughput, the impact of the number of transactions increases. As seen in Figure 6a,b, an increase to the transaction send rate before reaching the peak throughput of approximately 200 tps for GetBallot does not result in different throughput or latency between different volumes of total transactions. After the peak throughput is surpassed differences in performance emerge depending on the total transaction volume. There is not a clear relationship between transaction count and how throughput degrades after the send rate surpasses optimal throughput. For GetBallot, we observed that the throughput at high send rates was generally better for a total of 1000 tx than 20,000 tx, but this was not the case for CastVote. This could result from inconsistencies that arise in performance when max throughput is exceeded. For average latency, the impact of transaction count is more clear. As seen in Figure 6c,d, there is an increase in latency as the transaction count increased for CastVote. This is also seen in Figure 6a,b for GetBallot, with the exception of the observed latency at 20,000 total transactions, which is lower than the latency at 10,000 tx. These results are generalized as follows:

- When the transaction send rate is lower than the peak throughput, increasing the number of transactions has no impact on throughput or latency.

- After the peak throughput is surpassed, increasing the number of transactions decreases the throughput and increases the average latency.

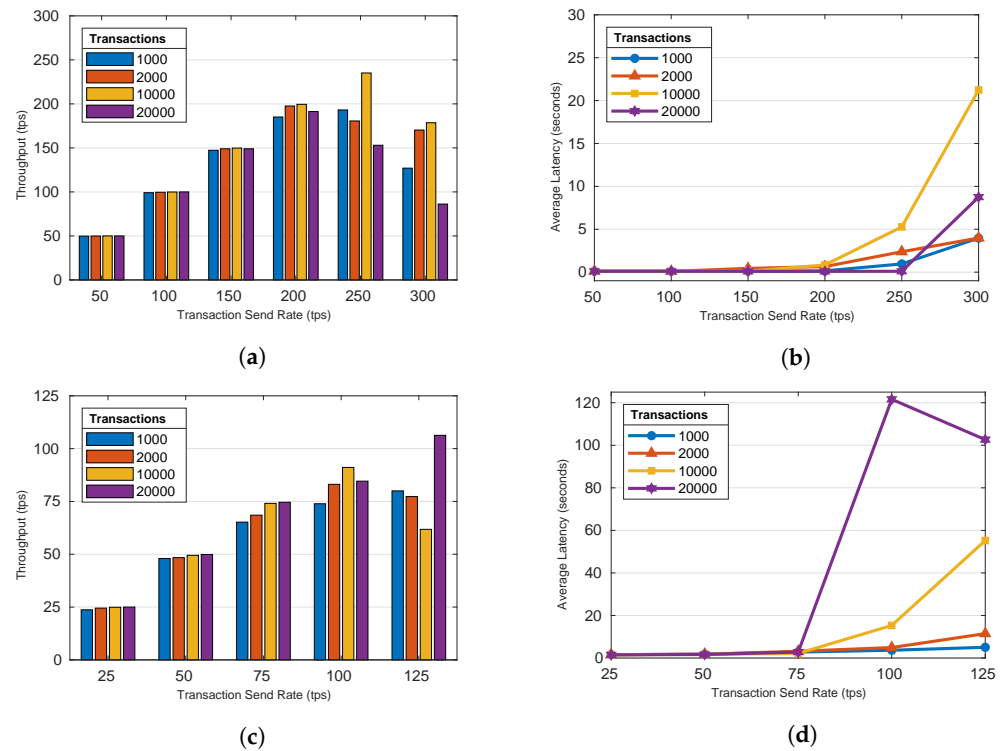


Figure 6. Varying the number of total transactions’ impact on throughput and latency: GetBallot and CastBallot. (a) GetBallot (Throughput); (b) GetBallot (Latency); (c) CastVote (Throughput); (d) CastVote (Latency). (Author’s own processing).

6.3.3. Experiment III

Figure 7 shows the measured error rate for the GetBallot and CastVote functions at varying transaction send rates and transaction totals. The results indicate that the number of transactions and transaction send rate impact the error rate in conjunction. For a total number of transactions below 10,000, neither tested function ever dropped or rejected a transaction for every send rate tested. At 10,000 transactions submitted, the error rate for the GetBallot function rose minimally when the send rate passed the scheme’s peak throughput measured in Experiment I. For 10,000 total transactions, the error rate for the CastVote function rose quickly once the send rate passed the scheme’s peak throughput measured in Experiment I. The results are generalized as follows:

- Errors are unlikely to occur when transactions are submitted to the scheme at/or below its peak throughput.
- Error rate follows the same trend as latency, rising only when the transaction send rates eclipse the scheme’s peak throughput.
- Error rate increases when large numbers of transactions are being processed (i.e., 10,000).

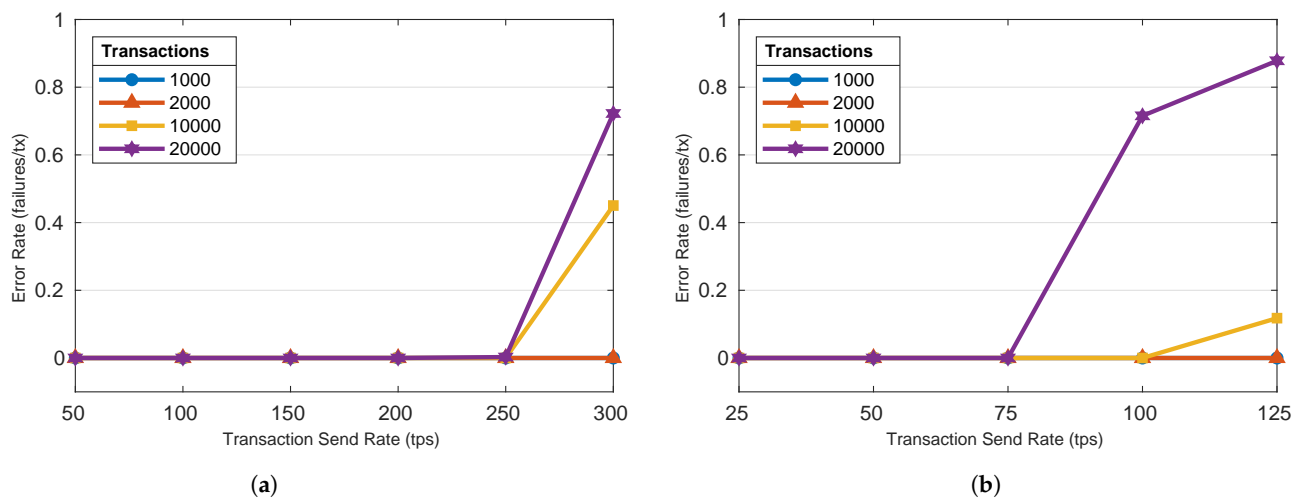


Figure 7. Varying transaction send rates' impact on Transaction Error Rate. (a) GetBallot; (b) CastVote. (Author's own processing).

We have concluded that our scheme is scalable up to 10,000 voters as long as user interaction is capped on a transaction per second basis. For usages such as fetching election results, if user interaction is capped to 200 transactions per second, users will experience little to no latency with our scheme, and no queries are likely to be dropped. For casting votes, if user interaction is capped to 75 transactions per second, users will experience minimal latency, and no votes are likely to be lost. These results also prove the feasibility of using the IBM Cloud to host our Hyperledger Fabric blockchain. As discussed in [33], Hyperledger Fabric itself is scalable, but now we can conclude that hosting Hyperledger Fabric on the IBM Cloud is also scalable, even when constrained by operating cost. As noted previously in this paper, we leveraged the free tier Kubernetes service on the IBM Cloud. This gave us minimal operating capacity, and with a larger budget, more scalability could be expected. To determine the exact scalability of a higher-tier service, however, further performance testing would be required.

7. Security and Privacy Analysis

In this section, security and privacy concerns are discussed, and how the proposed scheme is mitigating them.

Proposition 1. *The proposed scheme protects the privacy of voters.*

Proof. A voter can interact with the blockchain with a *one-request-only* blockchain address obtained using blind signatures. The votes' privacy is protected by encrypting them using the Verifiable Aggregator Oblivious Encryption. Only the voter who submitted his/her vote can know the exact vote and no one including the blockchain nodes can get access to the private information of the voters. □

Proposition 2. *Our scheme ensures election fairness.*

Proof. This is because the votes are encrypted and no one can access the votes that have been cast. Only the election results can be obtained by using the *AggrDec*, as explained in detail in Section 5.4. Because the votes are in an encrypted format, only the tally of the election can be obtained to protect the individual votes of the voters. □

Proposition 3. *The proposed scheme is secure against single-point-of-failure attacks.*

Proof. This is due to the underlying blockchain that is used to run the scheme. Our proposed e-voting scheme allows small businesses and governments alike to run an election in a transparent, decentralized, and secure manner. \square

8. Conclusions

This paper proposed and implemented a decentralized e-voting scheme that offers a robust and accessible alternative to traditional paper ballots. Our scheme provides a scalable and flexible solution to secure online voting. By using the private Hyperledger Fabric blockchain hosted on the IBM cloud, we presented a secure and private scheme that can be implemented with little technical knowledge at a reasonable cost to users. In our scheme, no one can link a vote to a specific voter because the voters interact using random generated addresses with the blockchain. This blockchain address is a one-time *pseudonym* generated by the voters, and it cannot reveal the voters' real identity. Analysis and performance evaluations were taken to prove the feasibility of our scheme. Our testing identified the throughput and election size limitations of our hardware, the impact of throughput and transaction count on latency and error rates, as well as the differing performance characteristics of read and write operations on the network. The results show that even when using minimal resources in the cloud, it is feasible to run an election with thousands of participants with concurrent vote transactions.

In the future, we will consider a bigger scale of elections by simulating a bigger number of transactions at higher transaction rates with increasing the number of blockchain validators. Moreover, we will expand this work to consider score voting. Score-based score voting is a special type of election that enable voters to assign a score to each candidate. Each voter gives each candidate a score within a predetermined range, such as 0 to 5. Then, the sum of the scores is added for each candidate. However, this will require extra computation and communication overheads to perform the aggregation of all scores, thus an efficient aggregation technique suitable for score voting will be further developed.

Author Contributions: Conceptualization, R.C., L.M. and M.B.; formal analysis, A.R. and M.B.; funding acquisition, M.A. and M.B.; methodology, R.C., L.M. and M.B.; resources, M.B.; software, L.M.; validation, M.B. and M.A.; writing—review and editing, R.C., L.M. and A.R.; Resources; M.A., A.R. and M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by Researchers Supporting Project number (RSPD2023R636), King Saud University, Riyadh, Saudi Arabia.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare that there is no conflict of interest.

References

1. Runyan, N.; Tobias, J. *Accessibility Review Report for California Top-to-Bottom Voting Systems Review*; Secretary of State of California: Los Angeles, CA, USA, 2007.
2. Bush, S.S.; Prather, L. Who's There? Election Observer Identity and the Local Credibility of Elections. *Int. Organ.* **2018**, *72*, 659–692.
3. Koven, J.B. Block the Vote: Could Blockchain Technology Cybersecure Elections? *Forbes*, 10 June 2016.
4. Mursi, M.; Assassa, G.M.R.; Abdelhafez, A.; Samra, K. On the Development of Electronic Voting: A Survey. *Int. J. Comput. Appl.* **2013**, *61*, 16. [[CrossRef](#)]
5. Hanifatunnisa, R.; Rahardjo, B. Blockchain Based e-Voting Recording System Design. In Proceedings of the 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), Lombok, Indonesia, 26–27 October 2017; pp. 1–6.
6. Panja, S.; Roy, B. A secure end-to-end verifiable e-voting system using blockchain and cloud server. *J. Inf. Secur. Appl.* **2021**, *59*, 102815. [[CrossRef](#)]
7. Neculache, N.; Petcu, V.A.; Simion, E. An analysis of a scheme proposed for electronic voting systems. *Cryptol. Eprint Arch.* **2023**, preprint.

8. Shanthinii, S.; Usha, M.; Prittopaul, P. A Survey Based on Online Voting System Using Blockchain Technology. In *Computer Vision and Machine Intelligence Paradigms for SDGs: Select Proceedings of ICRTAC-CVMIP 2021*; Springer: Berlin/Heidelberg, Germany, 2023; pp 209–216.
9. Jafar, U.; Ab Aziz, M.J.; Shukur, Z.; Hussain, H.A. A Systematic Literature Review and Meta-Analysis on Scalable Blockchain-Based Electronic Voting Systems. *Sensors* **2022**, *22*, 7585. [PubMed]
10. Denis González, C.; Frias Mena, D.; Massó Muñoz, A.; Rojas, O.; Sosa-Gómez, G. Electronic voting system using an enterprise blockchain. *Appl. Sci.* **2022**, *12*, 531. [CrossRef]
11. Sallal, M.; de Fréin, R.; Malik, A. PVPBC: Privacy and Verifiability Preserving E-Voting Based on Permissioned Blockchain. *Future Internet* **2023**, *15*, 121. [CrossRef]
12. Liu, Y.; Wang, Q. An E-voting Protocol Based on Blockchain. *IACR Cryptol. ePrint Arch.* **2017**, *2017*, 10–43.
13. Tanwar, S.; Gupta, N.; Kumar, P.; Hu, Y.C. Implementation of blockchain-based e-voting system. *Multimed. Tools Appl.* **2023**, 1–32. [CrossRef]
14. Kshetri, N.; Voas, J. Blockchain-enabled e-voting. *IEEE Software* **2018**, *35*, 95–99. [CrossRef]
15. Ayed, A.B. A conceptual secure blockchain-based electronic voting system. *Int. J. Netw. Secur. Appl.* **2017**, *9*, 1–9.
16. Rao, V.; Singh, A.; Rudra, B. Ethereum Blockchain Enabled Secure and Transparent E-Voting. In *Proceedings of the Future Technologies Conference*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 683–702.
17. Seifelnasr, M.; Galal, H.S.; Youssef, A.M. Scalable Open-Vote Network on Ethereum. In *Proceedings of the Financial Cryptography and Data Security*; Bernhard, M., Bracciali, A., Camp, L.J., Matsuo, S., Maurushat, A., Rønne, P.B., Sala, M., Eds.; Springer: Cham, Switzerland, 2020; pp. 436–450.
18. Khoury, D.; Kfoury, E.F.; Kassem, A.; Harb, H. Decentralized Voting Platform Based on Ethereum Blockchain. In *Proceedings of the 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, Bhilai, India, 19–21 February 2021; pp. 1–4. [CrossRef]
19. Meter, C. Design of distributed voting systems. *arXiv* **2017**, arXiv:1702.02566.
20. Hyperledger Fabric Official Documentation. A Blockchain Platform for the Enterprise. 2020. Available online: <https://hyperledger-fabric.readthedocs.io> (accessed on 1 May 2023).
21. Onur, C.; Yurdakul, A. ElectAnon: A Blockchain-Based, Anonymous, Robust and Scalable Ranked-Choice Voting Protocol. *arXiv* **2022**, arXiv:2204.00057.
22. Yavuz, E.; Koç, A.K.; Çabuk, U.C.; Dalkılıç, G. Towards Secure e-Voting Using Ethereum Blockchain. In *Proceedings of the 2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, Antalya, Turkey, 22–25 March 2018; pp. 1–7.
23. Kirillov, D.; Korkhov, V.; Petrunin, V.; Makarov, M.; Khamitov, I.M.; Dostov, V. Implementation of an e-Voting Scheme Using Hyperledger Fabric Permissioned Blockchain. In *Proceedings of the Computational Science and Its Applications—ICCSA 2019, 19th International Conference*, Saint Petersburg, Russia, 1–4 July 2019; Springer: Berlin/Heidelberg, Germany, 2019; Part II, pp. 509–521.
24. Specter, M.A.; Koppel, J.; Weitzner, D. The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in U.S. Federal Elections. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*, Berkeley, CA, USA, 12–14 August 2020; pp. 1535–1553.
25. Yang, Y.; Guan, Z.; Wan, Z.; Weng, J.; Pang, H.H.; Deng, R.H. PriScore: Blockchain-Based Self-Tallying Election System Supporting Score Voting. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 4705–4720. [CrossRef]
26. Lin, Y.; Zhang, P. Blockchain-Based Complete Self-Tallying E-Voting Protocol. In *Proceedings of the 2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Lanzhou, China, 18–21 November 2019; pp. 47–52. [CrossRef]
27. McCorry, P.; Shahandashti, S.F.; Hao, F. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In *Financial Cryptography and Data Security*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 357–375.
28. Li, Y.; Susilo, W.; Yang, G.; Yu, Y.; Liu, D.; Du, X.; Guizani, M. A blockchain-based self-tallying voting protocol in decentralized IoT. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 119–130. [CrossRef]
29. Khader, D.; Smyth, B.; Ryan, P.; Hao, F. A Fair and Robust Voting System by Broadcast. In *Proceedings of the 5th International Conference on Electronic Voting*, Bregenz, Austria, 11–14 July 2012; pp. 285–299.
30. Li, H.; Li, Y.; Yu, Y.; Wang, B.; Chen, K. A Blockchain-Based Traceable Self-Tallying E-Voting Protocol in AI Era. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 1019–1032. [CrossRef]
31. Han, G.; Li, Y.; Yu, Y.; Choo, K.K.R.; Guizani, N. Blockchain-Based Self-Tallying Voting System with Software Updates in Decentralized IoT. *IEEE Network* **2020**, *34*, 166–172. [CrossRef]
32. Mukherjee, P.P.; Boshra, A.A.; Ashraf, M.M.; Biswas, M. A Hyper-Ledger Fabric Framework as a Service for Improved Quality e-Voting System. In *Proceedings of the 2020 IEEE Region 10 Symposium (TENSYP)*, Dhaka, Bangladesh, 5–7 June 2020; pp. 394–397.
33. Kuzlu, M.; Pipattanasomporn, M.; Gurses, L.; Rahman, S. Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability. In *Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain)*, Atlanta, GA, USA, 14–17 July 2019; pp. 536–540. [CrossRef]
34. IBM Blockchain Platform. 2022. Available online: <https://cloud.ibm.com/docs/blockchain> (accessed on 1 May 2023).

35. Lu, Y.; Tang, Q.; Wang, G. Zebalancer: Private and Anonymous Crowdsourcing System Atop Open Blockchain. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 853–865.
36. Kosba, A.; Miller, A.; Shi, E.; Wen, Z.; Papamanthou, C. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016; pp. 839–858.
37. ShenTu, Q.; Yu, J. A Blind-Mixing Scheme for Bitcoin based on an Elliptic Curve Cryptography Blind Digital Signature Algorithm. *arXiv* **2015**, arXiv:1510.05833.
38. Wang, Z. Identity-based verifiable aggregator oblivious encryption and its applications in smart grids. *IEEE Trans. Sustain. Comput.* **2019**, *6*, 80–89. [[CrossRef](#)]
39. Kubernetes Service API Docs. 2020. Available online: <https://cloud.ibm.com/docs/containers> (accessed on 1 May 2023).
40. Dabbagh, M.; Kakavand, M.; Tahir, M.; Amphawan, A. Performance Analysis of Blockchain Platforms: Empirical Evaluation of Hyperledger Fabric and Ethereum. In Proceedings of the 2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAET), Kota Kinabalu, Malaysia, 26–27 September 2020; pp. 1–6.
41. Hyperledger Blockchain Performance Metrics. 2018. Available online: <https://www.hyperledger.org/learn/publications/blockchain-performance-metrics> (accessed on 1 May 2023).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.