EquiBFT: A Framework for Achieving Fairness in BFT Consensus

Siwei Cai
Shanghai Jiao Tong University
Shanghai, China
thranna@sjtu.edu.cn

Lei Fan
Shanghai Jiao Tong University
Shanghai, China
fanlei@sjtu.edu.cn

Shengyun Liu
Shanghai Jiao Tong University
Shanghai, China
shengyun.liu@sjtu.edu.cn

Hong-Sheng Zhou
Virginia Commonwealth University
Richmond, United States
hszhou@vcu.edu

***Abstract*—Byzantine Fault-Tolerant (BFT) consensus protocols are increasingly utilized in blockchain environments. In such protocols, the leader node holds the authority to dictate the transaction order, potentially impacting the fairness of decentralized finance (DeFi) applications. For instance, attackers can exploit this to manipulate transaction order and conduct front-running attacks. The concept of *order-fairness*, which recently emerged, has become a critical property for preventing a single node from unilaterally determining transaction order. Protocols designed to uphold *order-fairness* often rely on the sequence in which transactions appear across the network, a factor that can be influenced by the network's topology. However, this approach has inherent limitations, such as challenges in avoiding Condorcet cycles (Kelkar et al., Crypto 2020).**

To address these challenges, we propose a novel definition of fairness that requires concealing transaction content before ordering. Additionally, we extend the definitions of liveness and safety of consensus protocols to cover the transaction decryption process, guaranteeing the successful decryption of transactions. Based on the existing BFT protocol and utilizing threshold encryption algorithms, we designed a framework called EquiBFT which can incorporate fairness to BFT protocols. We have proven that the EquiBFT satisfies fairness while ensuring the liveness and safety. We implemented this framework based on HotStuff (Yin et al., PODC 2019) and validated its feasibility in a real-world network environment.

I. Introduction

Byzantine Fault Tolerant (BFT) protocols are crucial for achieving consensus in decentralized systems with potentially malicious nodes and have become prevalent in blockchain systems.The execution of a BFT protocol typically consists of two stages. First, the leader node collects and orders transactions submitted by clients, packages them into a block, and sends it as a proposal to the consensus nodes. Subsequently, the consensus nodes engage in multiple voting rounds to reach agreement on the proposal. The leader node has the authority to determine the order of transactions within the block. In blockchain-based Decentralized Finance (DeFi) applications, transaction ordering directly impacts profitability. Attackers can exploit this by manipulating transaction ordering through the leader node to gain profit [2, 8, 10]. For example, an attacker might identify a high-value target transaction involving the purchase of a specific asset and insert a front-running transaction to buy the asset first. After the target transaction inflates the asset's price, the attacker then inserts a reverse transaction to sell the asset at a higher price, securing a risk-free profit. This strategy, known as a "sandwich attack", involves placing two opposing transactions around the target transaction to gain profits, significantly undermining the fairness of the DeFi system.

In this paper we propose a novel definition of fairness for consensus protocols, considering it as a fundamental property. This notion of fairness ensures that no participant can discern the content of a transaction before it is definitively ordered, thereby preventing order manipulation attacks. Additionally, we develop a framework capable of integrating fairness into Byzantine Fault Tolerance (BFT) protocols.

A. Background and Related Work

To mitigate sandwich attack, two primary strategies have been proposed. The first strategy is **fair transaction ordering**, which seeks to prevent leader nodes from unilaterally deciding transaction sequences. Introduced by Kelkar et al. [15], the principle of *order-fairness* ensures that transaction order is dictated by a majority of consensus nodes rather than a single leader. Specifically, for any two transactions, a and b, if the majority of nodes receive transaction a before transaction b, then a must precede b in the final order. While this approach has inspired subsequent works [6, 13, 14], they all struggle to handle the cyclic dependencies known as the Condorcet Paradox. It prevents the establishment of a unique and deterministic transaction order. Aequitas [15] accommodates this by achieving *batch-order-fairness*, outputting cyclic transactions as a batch where the internal order is decided by the leader node. To address performance issues, Rashnu [23] proposed a fast fair ordering mechanism for transactions accessing inter-dependent data, while SpeedyFair [22] enhances performance by decoupling the ordering and consensus processes. Furthermore, recent studies [26] highlight that fair ordering can be undermined by sophisticated order attacks, underscoring the complex challenges of maintaining consistently fair transaction orders. To broaden fairness across decentralized networks, several investigations [13, 17, 29] have explored *order-fairness* in permissionless environments.

Achieving strict fair ordering that is unaffected by the Condorcet Paradox is known as strong fair ordering. Protocols such as Pompe [31] and Wendy [18] aim for strong fairness using timestamp-based mechanisms, relying on synchronized clocks to streamline transaction ordering. However, this dependence

on clock synchronization presents a significant assumption. Recently, this limitation was tackled by AOAB [11], which proposed a fully asynchronous protocol utilizing logical timestamps. However, AOAB requires additional interactions, which increases the consensus latency.

The second strategy is **transaction concealing**, using cryptographic algorithms to protect transactions without modifying the consensus protocol. The commit-reveal scheme [9, 10] is designed based on this technical path, but it is vulnerable to denial-of-service attacks from malicious clients, who first observe the decryption results of other transactions after consensus is reached and refrain from sending the decryption keys. To address this issue, TEX [16] employs trusted hardware to create a secure environment, allowing transactions to be encrypted and revealed in a controlled manner, thus preventing unauthorized access to plaintext during ordering. Another approach [19] introduces verifiable fairness, where transaction order can be verified by other consensus nodes, utilizing TEE to ensure a secure transaction processing environment. Similarly, Fairy [25] is a Byzantine fault-tolerant protocol resistant to front-running, using TEE to reveal transaction content upon delivery, aiming to ensure verifiable fairness.

To eliminate the need for additional trusted hardware, some schemes employ verifiable secret sharing scheme. In Helix [27], blocks are distributed in shares among consensus nodes, who can reconstruct the entire block by combining a sufficient number of shares. Fino [20] encrypts transactions, allowing the recovery of the encryption key only after transactions are finalized. Threshold encryption is another technique used to obscure transaction content prior to finalization. F3B [30] implements a verifiable threshold encryption scheme, encrypting each transaction with a key distributed in parts among several nodes.FairBlock [21] utilizes block-level encryption, with transactions encrypted by a block-specific key and decrypted collectively. To our knowledge, Ferveo [4] is the first scheme which tries to provide fairness in consensus protocol. It modifies Tendermint protocol with threshold cryptography to a synchronous BFT protocol with confidentiality. However, it lacks a formal analysis of its guarantees, leaving critical aspects of its design unvalidated.

Based on the above discussions, we have the following research question:

> *Is it possible to achieve complete fairness at the consensus protocol without bringing extra latency and heavy cost?*

B. Our contribution

A novel formal definition of the fairness property in consensus protocols: We define fairness based on the logical relationship between the decryption of encrypted transactions and the completion of transaction ordering, rather than the order in which transactions are observed by participants. A transaction can be decrypted if and only if it has been successfully ordered.

Enhanced definitions of liveness and safety for consensus protocols: The safety property ensures that participants can reach agreement not only on encrypted transactions but also on the decryption key. The liveness property guarantees that when a block is committed, the decryption keys for the transactions within the block will be recovered.

A concrete framework for converting BFT consensus protocols with fairness: Our framework which is named as EquiBFT is compatible with most BFT protocols and does not impose a heavy communication burden. We have shown that BFT protocols adapted through our framework can meet our enhanced definitions of safety and liveness.

It is important to note that integrating encryption algorithms directly into the consensus process is not trivial. For instance, completing the consensus process for encrypted transactions does not ensure the completion of the decryption process. Particularly during view changes stage, the new leader node must ensure that proposals that have already completed consensus also complete the decryption process. This additional requirement may compromise the safety or liveness of the consensus protocol.

Table I presents a comparison of our proposed scheme with major existing cryptographic algorithm-based schemes, excluding those that rely on trusted hardware. In this context, strong fairness refers to schemes unaffected by the Condorcet Paradox. Extra round denotes the number of additional message interaction rounds necessary post-consensus to ensure fairness. Asynchronous communication implies that no strict clock synchronization is needed to assign timestamps.

Protocol	Strong fairness	Extra round	Asynchronous
Aequitas[15]	✗	–	✓
Themis[14]	✗	–	✓
SpeedyFair[22]	✗	–	✓
Pompe[31]	✓	–	✗
Wendy[18]	✓	–	✗
F3B[30]	✓	1	✓
FairBlock[21]	✓	1	✓
EquiBFT	✓	0	✓

TABLE I: Comparison of protocols for achieving fairness.

II. Preliminaries and Model

A. Building Blocks

We will present the basic components needed for the protocol design. Due to space limitations, we will only give a brief description of the security properties of the components and omit rigorous definitions.

1) Hash function: A hash function is a cryptographic algorithm that converts input data of any length into a fixed-length hash value. It is designed to be fast, irreversible, and resistant to collisions as defined in Definition 1.

Definition 1 (Hash Function)**.** A hash function $H(\cdot)$ with output length m is a deterministic algorithm that takes a string $x \in \{0,1\}^*$ as input and outputs a string $h \in \{0,1\}^m$.

2) Blockchain: In blockchain systems, the consensus process facilitates agreement among nodes to append a new block to the chain. Each block contains an array of transactions and

the hash value of the preceding block. Define a block at height h as $B_h(txs, H(B_{h-1}))$, where txs represents the array of transactions included within the block and $H(B_{h-1})$ is the hash value of previous block. In cases where there is no need to highlight the block's height and content, we compactly denote a block as B.

3) Threshold encryption: Threshold public key encryption (TPKE) is an important cryptography primitive. The ciphertext cannot be decrypted unless sufficient number of members participate in the decryption process. This feature effectively mitigates risks associated with single point failures or corruption. It is defined in Definition 2. Please also see [3].

Definition 2 (Threshold Public Key Encryption)**.** A (t, n) threshold encryption scheme is a set of PPT algorithms with these specifications.

- $(\mathsf{pke}, \mathsf{vke}, \mathsf{ske}) \xleftarrow{\$} \mathsf{Setup}(n, t, \lambda)$: The set-up algorithm takes the number of participants n and a threshold t where $1 \leq t \leq n$, and a security parameter λ as inputs. It outputs a triple consisting of $\mathsf{pke}, \mathsf{vke}$ and ske. The public key pke is used for encryption, while the verification key vke is used for validation. The private key share $\mathsf{ske} = (\mathsf{ske}_1, \cdots, \mathsf{ske}_n)$ is a vector of shares distributed among the participants.
- $C \xleftarrow{\$} \mathsf{Encrypt}(\mathsf{pke}, M)$: The encrypt algorithm encrypts the plaintext M with the public key pke and outputs a ciphertext C.
- $\mu_i \leftarrow \mathsf{ShareDecrypt}(\mathsf{pke}, i, \mathsf{ske}_i, C)$: The share-decrypt algorithm takes the pke, the i^{th} private key ske_i and the ciphertext C as inputs and outputs the partial decryption share μ_i.
- $\phi \leftarrow \mathsf{ShareVerify}(\mathsf{pke}, \mathsf{vke}, C, \mu_i)$: The share-verify algorithm takes the public key pke, the verification key vke, and a decryption share μ_i as inputs. If μ_i is the correct share for C, it outputs a boolean value ϕ as `true`. Otherwise it outputs `false`.
- $M \leftarrow \mathsf{Aggregate}(\mathsf{pke}, \mathsf{vke}, C, \{\mu_1, \cdots, \mu_t\})$: The combine algorithm takes pke, vke, and any t valid decryption shares $\{\mu_1, \cdots, \mu_t\}$ as inputs. It outputs the plaintext M or $\perp$.

4) Symmetric Encryption: Symmetric encryption is a type of encryption scheme where the same key is used for both encryption and decryption of data. It is defined in Definition 3. Please also see [12].

Definition 3 (Symmetric Encryption Scheme)**.** A symmetric encryption scheme consists of three PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- $k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$: The key-generation algorithm Gen takes as input 1^λ and outputs a key k.
- $C \xleftarrow{\$} \mathsf{Enc}(k, M)$: The encryption algorithm Enc takes as input a key k and a plaintext message $M \in \{0,1\}^*$, and outputs a ciphertext C.
- $M \leftarrow \mathsf{Dec}(k, C)$: The decryption algorithm Dec takes as input a key k and a ciphertext C, and outputs a message $M \in \{0,1\}^*$ or an error, denoted by the symbol $\perp$.

B. Model Formalism

1) BFT Protocol: Byzantine fault tolerant protocols are becoming more and more important in blockchain [7, 28]. These protocols enable a group of nodes to reach consensus despite the arbitrary or malicious behavior of some nodes. The total number of nodes is denoted as n and the number of Byzantine nodes is denoted as f where $n = 3f + 1$. The node set is represented as $\Phi = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$. Each node processes inputs from the environment and produce a log of decided values.

2) Network: In our paper, consensus nodes leverage a point-to-point network for message exchange, ensuring that all communication channels are both authenticated and reliable. The protocol is designed to function under partially synchronous network conditions, characterized by two key parameters: Global Stabilization Time (GST) and the network delay upper bound Δ. GST is an unknown value that represents the time required for the network to stabilize. The parameter serves as the upper bound on the maximum delay for message transmission between two correct nodes. After the network stabilizes at GST, all transmissions between any two correct replicas are expected to arrive within the time frame Δ.

3) Corruptions: The adversary, denoted as $\mathcal{A}$, controls up to f corrupted nodes within the system, which are identified as Byzantine faulty during initialization, allowing them to exhibit arbitrary behavior. Within the network, $\mathcal{A}$ has the capability to insert, record, and replay messages. Crucially, however, messages exchanged between two honest nodes cannot be tampered with or altered.

4) Environment: To facilitate the execution of the protocol, we employ an environment denoted as $\mathcal{Z}(1^\lambda)$, where λ represents the security parameter. The role of $\mathcal{Z}$ is to drive the consensus among nodes. At the start of the execution, $\mathcal{Z}$ initiates the activation of all nodes, both honest and corrupted, ensuring their participation in the protocol. It then generates encrypted transactions and transmit them to consensus nodes. For a given protocol Π, the notation $\text{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$ represents the random variable encompassing all potential execution traces of Π with respect to the adversary $\mathcal{A}$ and environment $\mathcal{Z}$ and λ is security parameter. A view within the support of $\text{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$ refers to a fully specified instance of an execution trace, including inputs, outputs, random coins, and any other relevant components of the execution.

C. Properties

The concept of *order-fairness* [15] is a well-known abstraction of fairness, defined by the causal order of network transmissions. It asserts that if a majority of nodes receive transaction $\mathtt{tx}_1$ before transaction $\mathtt{tx}_2$, then $\mathtt{tx}_1$ should be output prior to $\mathtt{tx}_2$. This concept, however, faces the challenge of the Condorcet paradox, where cyclic order dependencies among three or more transactions prevent establishing a definitive transaction order. While numerous research efforts address

this issue from different angles, proposed solutions often lead to increased communication complexity.

Our protocol maintains fairness by encrypting the transactions. This encryption provides confidentiality, thereby prohibiting adversaries from manipulating the order of transactions for their benefit. As such, all participants enter the consensus process on equal footing, unaware of the specific transaction details until an ordering agreement has been reached. It is only post this agreement that the content of the transactions is disclosed.

1) Fairness: In EquiBFT, the content of a transaction is revealed only after it has been ordered. First, we define the ordering of a transaction.

Definition 4 (A transaction is ordered)**.** For a view in the support of $\text{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$, let $\mathtt{tx}$ be a transaction generated by $\mathcal{Z}$ and sent to a replica $\mathcal{P}$. We define that transaction $\mathtt{tx}$ is ordered by $\langle i, j \rangle$ if it will be eventually output by every honest node as the j-th transaction of the i-th block.

When the ordering of a transaction satisfies the following relationship with decryption, it is called fair ordering. We use $\widetilde{\mathtt{tx}}$ to denote the encrypted version of transaction $\mathtt{tx}$. That is, $\widetilde{\mathtt{tx}} \xleftarrow{\$} \mathsf{Enc}(k, \mathtt{tx})$ where k is an encryption key.

Definition 5 (Fairness)**.** For a view of $\text{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$, let $\widetilde{\mathtt{tx}}$ be a transaction generated and encrypted by $\mathcal{Z}$. We define that Π satisfies fairness if an encrypted transaction $\widetilde{\mathtt{tx}}$ can be decrypted by any replica $\mathcal{P}$ only if $\mathtt{tx}$ is ordered.

2) Liveness and safety: BFT protocols are designed to ensure that honest nodes generate consistent outputs for user requests, a goal typically defined by the properties of safety and liveness. However, these classical definitions are inadequate for EquiBFT. In EquiBFT, transactions are encrypted when they are sent out. Therefore, the protocol's obligations extend beyond guaranteeing the liveness and safety of encrypted transactions; it must also ensure that all honest nodes can accurately decrypt the transactions and that the decrypted outputs are consistent across nodes. The liveness and safety of EquiBFT are defined as following:

Definition 6 (Liveness)**.** After GST, honest nodes will complete consensus on the new block. Transactions sent by honest clients within the block can be decrypted by honest nodes.

Definition 7 (Safety)**.** If a correct node $\mathcal{P}_1$ has decided on blocks $B_1, B_2, \ldots, B_s$, and another correct node $\mathcal{P}_2$ has decided on $B'_1, B'_2, \ldots, B'_s$, then $B_i = B'_i$ for $1 \leq i \leq s$. If transaction $\mathtt{tx}$ is decrypted from the j-th encrypted transaction in block B_i by $\mathcal{P}_1$, and $\mathtt{tx}'$ is decrypted from the j-th encrypted transaction in block B'_i by $\mathcal{P}_2$, then $\mathtt{tx} = \mathtt{tx}'$.

III. BFT Consensus Protocol with Fairness

In this section, we introduce EquiBFT, a framework designed to enhance BFT consensus protocols with fairness. We achieve fairness by integrating a threshold public key encryption scheme [1] into the consensus protocol.

A. BFT Consensus Protocol Paradigm

From a system perspective, leader-based Byzantine Fault Tolerance (BFT) protocols possess structural similarities in blockchain generation. Within a consensus view, the process encompasses four phases: proposing, voting, execution, and view change, as illustrated in Fig 1.

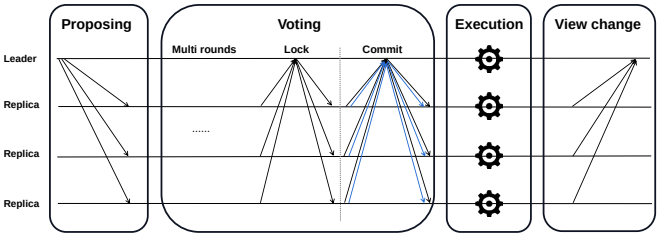


Fig. 1: The structure of leader-based BFT protocols. The blue lines represent the decryption message which will be used in EquiBFT.

- **Proposing:** The leader node packages the received transactions into a proposal block and sends it to consensus nodes. It is important to note that the new proposal block must not conflict with other blocks that have already reached consensus.
- **Voting** Consensus nodes engage in multiple rounds of voting on the proposal block. After each round, nodes update their state to facilitate consensus among honest nodes. The number of voting rounds can vary across protocols. Typically, if a proposed block completes specific rounds of voting, it enters a locked state. A block in this locked state is committed via a final voting round, ensuring consensus among all honest nodes. Importantly, once a block is locked, it will eventually be committed, and subsequent proposal blocks must not conflict with locked blocks.
- **Execution** Consensus nodes execute the transactions contained within the committed block. Generally, this step lies outside the core consensus protocol, focusing instead on processing the consensus data. We deliberately distinguish the execution process here because, within our framework, it is crucial to ensure that consensus nodes can accurately execute encrypted transactions.
- **View Change** This phase facilitates the transition to a new leader node for achieving consensus in the subsequent view. The new leader must generate a proposal block at the next height following the previous decided block. The view change mechanism ensures that the new leader can receive the latest locked block. Various protocols implement the view change process differently, and some may not include it at all. For instance, in the Casper protocol [5], which assumes a synchronous network, the view change is not explicitly defined.

Abstracting away the specifics of individual protocols, BFT consensus protocols can be uniformly represented as shown in Algorithm 1. The function $\texttt{CreateLeaf}(parent, \texttt{TXS})$ in

the proposing phase is used to extend a new block which is defined in Algorithm 2.

Algorithm 1 BFT consensus protocol

▷ **Proposing Phase**
1: **as** a leader
2: wait for $n - f$ NEW-VIEW messages V
3: $lockedB \leftarrow$ extract the highest locked block from V
4: $B \leftarrow \texttt{CreateLeaf}(lockedB, \texttt{TXS})$
5: broadcast B

▷ **Voting Phase (multi rounds)**
6: **while** B is not locked **do**
7: **as** a replica
vote for B if it is correct
8: **as** a leader
push the voting phase for B
9: **end while**
10: $lockedB \leftarrow B$
11:
12: **as** a replica $\mathcal{P}_i$
vote COMMIT for B if it is correct
13: **as** a leader
14: wait for $n - f$ valid COMMIT votes for B
15: broadcast DECIDE message
16:
17: **as** a replica
18: Execute the transactions in B

▷ **View Change** ▷ Goto this line if timeout
19: **as** a replica
20: send NEW-VIEW message to leader of next view

B. Transaction Encryption

Our framework, EquiBFT, utilizes a threshold public key encryption (TPKE) scheme to safeguard user transactions. Consensus nodes are required to pre-share encryption key shares, while clients encrypt their transactions with a symmetric key, which is encapsulated by the corresponding public key when sending them.

The pre-sharing setup phase incurs a one-time cost and need only be executed once. However, it can be refreshed periodically to enhance security or accommodate membership changes. To ensure robust protection, each transaction is encrypted with a unique symmetric key, preventing the compromise of one transaction from jeopardizing the confidentiality of other uncommitted transactions.

1) Consensus Nodes: The consensus nodes initialize the parameters of TPKE scheme as: $\mathsf{pke}, \mathsf{vke}, \mathsf{ske} \xleftarrow{\$} \mathsf{Setup}(n, t, \lambda)$. The generated public key pke and the verification key vke are made publicly available as global parameters. Clients can use pke to encrypt transactions. Each consensus node $\mathcal{P}_i$ obtains a unique private key share ske_i of ske, which remains confidential and is used for threshold decryption. The verification key vke serves to confirm the validity of decryption shares received from consensus nodes.

In EquiBFT, given the number of byzantine nodes f, the total number of nodes $n = 3f + 1$, let $t = 2f + 1$ be the threshold of decryption of TPKE.

2) Clients: A client retrieves the public key pke of TPKE to encrypt transactions using a hybrid encryption scheme. In addition to the TPKE scheme, the client employs a symmetric encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. To encrypt a transaction tx, the client randomly generates a symmetric key $k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ and encrypts tx as $\widetilde{\texttt{tx}} \xleftarrow{\$} \mathsf{Enc}(k, \texttt{tx})$. The symmetric key k is then encrypted using the global TPKE public key pke, resulting in $\widetilde{k} \xleftarrow{\$} \mathsf{Encrypt}(\mathsf{pke}, k)$.

A valid encrypted transaction is then structured as $\texttt{ctx} = \langle \widetilde{\texttt{tx}}, \widetilde{k} \rangle$ and sent to the consensus nodes. These encrypted transactions sent by the clients also include their signatures to verify the validity of the transactions, which is the same as regular transactions. Although consensus nodes cannot validate the content of a encrypted transaction upon reception, the system mitigates abuse by charging a gas fee, which deters malicious clients from arbitrarily flooding the system with invalid encrypted transactions. For simplicity, we omit the details of signature verification in the protocol description. The encryption scheme ensures fairness by maintaining the confidentiality of transactions throughout the subsequent consensus phases.

C. Auxiliary functions

Consensus nodes use the auxiliary functions in Algorithm 2 to process the messages.

1) Blocks extension: The leader node creates a block extending the parent block $parent$ by utilizing the function $\texttt{CreateLeaf}(parent, \texttt{TXS})$ in Line 1 Algorithm 2, where TXS represents a set of transactions selected from the transaction pool.

2) Commit Message Verification: The leader node verifies the decryption shares in the COMMIT messages it receives through the function $\texttt{VerifyCommit}(v, B)$ in Line 5 Algorithm 2. This function checks the correctness of the decryption shares. If the verification is unsuccessful, the COMMIT message is deemed invalid.

3) Keys aggregation: The symmetric keys used for transaction encryption are encrypted with the public key pke of TPKE. Consensus nodes include key recovery shares in their voting messages. When the leader collects $n-f$ votes, denoted as V, they can employ $\texttt{AggregateKey}(V, B)$ in Line 10 Algorithm 2 to recover the symmetric keys for the encrypted transactions in block B. The collected set of votes is appended to the block as proof.

4) Transaction verification and decryption: Upon receiving the DECIDE message, consensus nodes verify the decryption keys $keys$ and decrypt the transactions in block B using $\texttt{VerifyDecryptTXS}(keys, B)$ in Line 17 Algorithm 2. The keys for proof are aggregated from the original set of votes attached to the block to verify the correctness of the leader's key recovery. For each encrypted transaction $\texttt{ctx}_j$ in the block, the decryption key is validated with the recovered keys. If the decryption key matches the recovered one, the encrypted transaction is decrypted using the provided key. If the validation fails, the function returns `false`.

Algorithm 2 Auxiliary functions for a node $\mathcal{P}_i$

1: **Func** `CreateLeaf`($parent$, `TXS`)
2: $B.parent \leftarrow parent$
3: $B.txs \leftarrow$ `TXS`
4: **return** B

5: **Func** `VerifyCommit`(v, B)
6: **for** $\texttt{ctx}_j \in B.txs$ **do**
7: **if** (ShareVerify(pke, vke, $\texttt{ctx}_j.\widetilde{k}, v.\mu_j$) = `false` **then**
8: **return** `false`
9: **return** `true`

10: **Func** `AggregateKey`(B)
11: $V \leftarrow B.proof$
12: **for** $\texttt{ctx}_j \in B.txs$ **do**
13: $\mu \leftarrow \{v_i.\mu_j | v_i \in V\}$
14: $k_j \leftarrow$ Aggregate(pke, vke, $\texttt{ctx}_j.\widetilde{k}, \mu$)
15: $keys \leftarrow keys \cup \{k_j\}$
16: **return** $keys$

17: **Func** `VerifyDecryptTXS`($keys, B$)
18: $V \leftarrow B.proof$
19: **for** $v \in V$ **do**
20: **if** (`VerifyCommit`$(v, B) =$ `false`) **then**
21: **return** `false`
22: $txs \leftarrow \emptyset$
23: $proof\_keys \leftarrow$ `AggregateKey`(B)
24: **for** $\texttt{ctx}_j \in B.txs, k_j \in keys$ **do**
25: **if** $proof\_keys_j \neq k_j$ **then**
26: **return** `false`
27: $\texttt{tx}_j \leftarrow$ Dec($k_j, \texttt{ctx}_j.\widetilde{\texttt{tx}}$)
28: $txs \leftarrow txs \cup \{\texttt{tx}_j\}$
29: $B.txs \leftarrow txs$
30: **return** `true`

D. Framework Incorporating BFT Consensus Protocol with Fairness

Our framework, EquiBFT, is detailed in Algorithm 3. The distinctions from the original BFT protocol, as shown in Algorithm 1, are highlighted in **blue color**.

1) Proposing: In the proposing stage, the leader must ensure that the current highest locked block, $lockedB$, has been successfully decrypted before constructing a new block to extend the blockchain. If $lockedB$ has not been decrypted, the leader is required to re-propose $lockedB$ as specified in Line 4 Algorithm 3. This method differs from traditional BFT protocols such as HotStuff, which merely require that new proposals follow $lockedB$. This strategy mandates that the locked block is decrypted.

2) Voting: The voting phase adheres to the standard process outlined in the original BFT protocol, culminating in the locking of the proposal with an additional verification step. Each node must verify whether the locked block has been decrypted when evaluating the proposal's validity. If it has not, the current proposal must be the re-proposed locked block; otherwise, the node will refuse to vote on the proposal.

In the final voting round, after the block is locked, each node includes its decryption share for the decryption keys within its voting message. Specifically, node $\mathcal{P}_i$ performs the share decryption for every encrypted transaction $\texttt{ctx}_j$ in the block using its private key ske_i of the TPKE scheme as $\mu_j \leftarrow$ ShareDecrypt(pke, $i, \mathsf{ske}_i, \texttt{ctx}_j.\widetilde{k}$) in Line 16 Algorithm 3. The consensus node $\mathcal{P}_i$ compiles its decryption shares into a vector μ as $\mu \leftarrow [\mu_1, \mu_2, \cdots, \mu_\beta]$. Then, $\mathcal{P}_i$ attaches this vector to the COMMIT message and sends it to the leader node.

The leader will perform additional verification to ensure the correctness of the decryption shares received from nodes using `VerifyCommit`(v, B) in Line 23 Algorithm 3. If the verification fails, the leader will reject the corresponding COMMIT vote. Upon receiving $n - f$ valid COMMIT votes, the leader recover the symmetric keys for the encrypted transactions in B using `AggregateKey`(B) in Line 25. The votes are included in the block B so that all the consensus nodes can verify the keys are recovered properly by the leader node.

3) Execution: During the transaction execution phase, consensus nodes use auxiliary votes included in the DECIDE message to verify the accuracy of the key recovery process in Line 30 Algorithm 3. Subsequently, nodes employ the keys obtained from the leader node to decrypt the transactions. If key recovery verification fails, the block must be re-proposed.

4) View Change: During the view change phase, nodes send the local block that has completed specific rounds of voting. They also indicate the status of the decryption process within the view change messages. If the locked block is successfully decrypted, the associated decryption keys are included with the message. Conversely, if the locked block remains undecrypted, the message will explicitly indicate this status.

Algorithm 3 BFT protocol incorporating fairness for a node $\mathcal{P}_i$ with signature key share sk_i and decryption key share ske_i

▷ **Proposing Phase**
1: **as** a leader
2: wait for $n - f$ NEW-VIEW messages V
3: $lockedB \leftarrow$ extract the highest locked block from V
4: **if** $lockedB$ is decrypted **then**
5: $B \leftarrow$ `CreateLeaf`$(lockedB,$ `TXS`$)$
6: **else**
7: $B \leftarrow lockedB$
8: broadcast B

▷ **Voting Phase**
9: **while** B is not locked **do**
10: **as** a replica
if $lockedB \neq B$ **then**
assert $lockedB$ is decrypted
vote for B (multi rounds) if it is correct
11: **as** a leader
push the voting phase for B (multi rounds)
12: **end while**
13: $lockedB \leftarrow B$
14:
15: **as** a replica $\mathcal{P}_i$
16: **for** $\texttt{ctx}_j \in B.txs$ **do**
17: $\mu_j \leftarrow$ ShareDecrypt(pke, $i, \mathsf{ske}_i, \texttt{ctx}_j.\widetilde{k}$)
18: $\mu \leftarrow \mu \cup \{\mu_j\}$
19: attach μ to COMMIT vote
20: vote COMMIT for B if it is correct
21: **as** a leader
22: wait for $(n - f)$ valid COMMIT votes V for B:
23: $V \leftarrow \{v \mid$ `VerifyCommit`$(v, B) =$ `true`$, v \in V\}$

24: $B.proof \leftarrow V$
25: $keys \leftarrow \texttt{AggregateKey}(B)$
26: attach $keys$ to DECIDE message
27: broadcast DECIDE message
28:
29: **as** a replica
30: **if** $\texttt{VerifyDecryptTXS}(keys, B) = \texttt{true}$ **then**
31: Execute the transactions in B

▷ **View Change** ▷ Goto this line if timeout
32: **as** a replica
33: send NEW-VIEW message to leader of next view

IV. Analysis

In this section, we analyze the liveness, fairness, and safety properties of the framework. We assume that the original BFT protocol, as described in Algorithm 1, ensures the liveness and safety of consensus as defined in Definition 8 and Definition 9.

Definition 8 (Liveness of consensus)**.** After GST, a consensus of block will be decided by correct nodes for a period of time.

Definition 9 (Safety of consensus)**.** If a correct node $\mathcal{P}_1$ has decided blocks $B_1, B_2, \cdots, B_s$ and another correct node $\mathcal{P}_2$ has decided $B'_1, B'_2, \cdots, B'_{s'}$, then $B_i = B'_i$ for $1 \leq i \leq min(s, s')$.

Due to space constraints, we provide only a sketch of the proof.

A. Liveness

We first prove that honest nodes can reach consensus on proposals containing encrypted transactions. Compared to the original BFT, the modified protocol guarantees that locked blocks be decrypted. Therefore, if a locked block has not been decrypted yet, the leader node should repropose this block. If the locked block has been decrypted, the leader node can generate new blocks following it. In either of the above scenarios, the blocks proposed by the leader node do not conflict with those proposed in the original BFT protocol. Therefore, if the leader node is honest and the original BFT protocol possesses liveness, then modified protocol also has consensus liveness.

Lemma 1 (Liveness of consensus)**.** After GST, there is a bounded duration T_f such that if all correct nodes remain in view v during T_f and the leader for view v is correct, then a block is locked.

Proof. The proof can be directly derived from the liveness of the original BFT. □

Next, we prove that if the leader node is honest, the encrypted transactions contained in the committed block can be decrypted correctly. In Lemma 2, it is proved that if the leader node collects enough COMMIT votes, the key used by clients for encrypting transactions will be recovered.

Lemma 2. Let TPKE be a verifiable threshold encryption scheme. If a correct leader gathers $(n-f)$ valid COMMIT votes on a block B, he can recover the keys for the transactions in B such that $\texttt{VerifyDecryptTXS}(keys, B) = \texttt{true}$.

Proof. The validity of a COMMIT vote v is determined by $\texttt{VerifyCommit}(v, B)$ in Line 23 Algorithm 3. The leader checks whether the decryption shares of TPKE in v are correctly decrypted by the replicas.

Consequently, upon receiving $(n-f)$ valid COMMIT votes, the leader can extract $2f+1$ valid decryption shares from these votes for every transaction $\texttt{ctx}$, meeting the TPKE decryption threshold. The leader attaches the votes to $B.proof$. In the $\texttt{AggregateKey}$ function in Line 25 Algorithm 3, the leader can accurately recover the decryption key k_j from these $2f+1$ decryption shares for each encrypted transaction $\texttt{ctx}_j$. The recovered keys form a set denoted as $keys$.

Upon receiving the $keys$ from the DECIDE message, the replicas verify and decrypt the transactions in the block using the function $\texttt{VerifyDecryptTXS}(keys, B)$. For each encrypted transaction, the votes set V in $B.proof$, provided by the leader, will be used to aggregate the exact same key k_j as recovered by the leader. If the leader is correct, the decryption key must be correctly recovered. Consequently, it holds that $\texttt{VerifyDecryptTXS}(keys, B) = \texttt{true}$. □

In Lemma 3, it is proved that if the leader node is honest, all honest nodes can obtain the correct key to decrypt the transactions.

Lemma 3 (Liveness of decryption)**.** Let TPKE be a verifiable threshold encryption scheme. After GST, there exists a bounded duration T_f during which all correct nodes remain in view v, and the leader for view v is correct. During this period, any committed block will have its encrypted transactions from honest clients successfully decrypted.

Proof. Once an honest node becomes the leader of a view and all correct nodes remain alive after GST for a sufficient period, by Lemma 1, the leader can drives the view through multiple voting phases by collecting votes and broadcasting messages to lock the proposal.

In the COMMIT phase, since the leader is honest, it recovers the keys upon receiving $(n-f)$ valid COMMIT votes and broadcasts the DECIDE message to all nodes. By Lemma 2, the nodes can complete the verification by invoking $\texttt{VerifyDecryptTXS}(keys, B) = \texttt{true}$. For an honest client he must encrypt the correct key using TPKE. The key is then used to decrypt the encrypted transactions. Consequently, all transactions from honest clients in the block are successfully decrypted. □

Lemmas 1 and 3 collectively establish the liveness properties of EquiBFT. Specifically, Lemma 1 ensures that correct nodes will eventually reach a decision on a block, while Lemma 3 guarantees that the decryption of committed blocks proceeds without obstruction. Together, these lemmas demonstrate the liveness of our protocol, as articulated in Theorem 1.

Theorem 1 (Liveness)**.** Let TPKE be a verifiable threshold encryption scheme. Protocol Π_{EquiBFT} satisfies liveness (Definition 6).

Proof. After GST, there is a bounded duration T_f such that if all correct nodes remain in view v during T_f and the leader for view v is correct, then by Lemma 1, a decision of a block is reached. Additionally, by Lemma 3, under such circumstance, the transactions from honest clients in the committed block can be decrypted. □

B. Safety

First, we demonstrate that EquiBFT ensures the safety of consensus, meaning that two conflicting blocks,*i.e.*, blocks at the same height, cannot be committed simultaneously. Within EquiBFT, the voting principles for consensus nodes mirror those in the original BFT protocol. As a result, conflicting blocks cannot simultaneously receive votes from honest nodes, which prevents two honest consensus nodes from finalizing consensus on conflicting blocks.

Lemma 4 proves that if two blocks are conflicting, it is impossible for honest nodes to reach consensus on both blocks, thereby ensuring the safety of block consensus.

Lemma 4. (Safety of consensus) If B_1 and B_2 are two conflicting blocks, they cannot be both committed, each by a correct node.

Proof. The proof can be directly derived from the safety of the original BFT. □

Then we prove the safety of the decryption process for transactions within a committed block in Lemma 5. Any validly decrypted block will contain the same decryption keys.

Lemma 5 (Safety of decryption keys)**.** Let TPKE be a verifiable threshold encryption scheme. Suppose two correct nodes $\mathcal{P}$ and $\mathcal{P}'$ receive sets of decryption keys, denoted as $keys$ and $keys'$, respectively, from a DECIDE message for a block B. If $\texttt{VerifyDecryptTXS}(keys, B) = \texttt{true}$ and $\texttt{VerifyDecryptTXS}(keys', B) = \texttt{true}$, then $keys$ must equal $keys'$.

Proof. Suppose B contains β encrypted transactions, then $keys$ and $keys'$ each contain β decryption keys. If $\texttt{VerifyDecryptTXS}(keys, B) = \texttt{true}$ then each $k_j \in keys$ pass the verification. The $proof\_keys$ are recovered from the shares which are attached in the block B and the shares are verified in in Algorithm 2 Line 20. With the property of TPKE, $\mathcal{P}$ and $\mathcal{P}'$ recover the same $proof\_keys$.

We have $proof\_keys_j = k_j$ and $proof\_keys_j = k'_j$ as in Algorithm 2 Line 25. That is we have $k_j = k'_j$. Therefore, we conclude that $keys = keys'$. □

From Lemma 4 and Lemma 5, we derive the following theorem.

Theorem 2 (Safety)**.** Let TPKE be a verifiable threshold encryption scheme. Protocol Π_{EquiBFT} satisfies safety (Definition 7).

Proof. If B_1 and B_2 are both committed at the same height by any two honest nodes $\mathcal{P}_1, \mathcal{P}_2$, then by Lemma 4, we have $B_1 = B_2$, thereby ensuring the consistency of the ciphertext of the transactions of a committed block for all participants. By Lemma 5, B_1 and B_2 at the same height have identical keys. Therefore, the decrypted transactions of $\mathcal{P}_1$ and $\mathcal{P}_2$ remain the same. □

C. Fairness

Fairness is a critical property of EquiBFT, ensuring resilience against front-running attacks. In EquiBFT, we uphold fairness by demonstrating that every transaction is ordered before decryption, and once decrypted, its order remains immutable. This property safeguards against any participant unfairly influencing transaction order, thereby maintaining an equitable environment for all participants.

In Lemma 6, we prove that if a node can decrypt the transactions in a block, then at least $f + 1$ honest nodes are already locked on that block. In Lemma 7, we prove that if a block has $f + 1$ honest nodes locked on it, then the block will be committed. The statements of these two lemmas are provided as follows.

Lemma 6. Let TPKE be a verifiable threshold encryption scheme. If an encrypted transaction `ctx` in a block B can be decrypted by any party $\mathcal{P}_i$, then at least $f + 1$ honest nodes is locked with $lockedB = B$.

Proof. Since the decryption threshold of TPKE is $2f + 1$, at least $f + 1$ decryption shares for the decrypted transaction come from honest nodes. Furthermore, because honest consensus nodes will only contribute their decryption shares after being locked into the block, it follows that at least $f+1$ honest nodes must be locked into that block. □

Lemma 7. If at least $f+1$ honest nodes are locked on a block B such that $lockedB = B$ and B is uncommitted, then B will be committed as the next block.

Proof. For there are already $f + 1$ honest nodes locked onto block B, then any proposal B' which is conflicting with B will not be voted by these $f + 1$ honest nodes. That is any block $B' \neq B$ will not be committed. We have prove the liveness of EquiBFT in Theorem 1 the next committed block must be B. □

We deduce relation between block decision and transaction ordering as stated in Corollary 1.

Corollary 1. If a block B will be committed as the next block, then the transactions in block B are already ordered.

Proof. Since no block can be committed before the block B, it is evident that the transactions in B will be committed in the exact order as it appear in the block. □

Our protocol guarantees no decryption happens prior to the finalization of a transaction. If a transaction can be decrypted, the consensus system must have reached a status where the transaction is already ordered. Together, these lemmas culminate in Theorem 3, affirming that EquiBFT achieves fairness.

Theorem 3 (Fairness)**.** Let TPKE be a verifiable threshold encryption scheme. Protocol Π_{EquiBFT} satisfies fairness (Definition 5).

Proof. For a view in the support of $\text{EXEC}^{\Pi_{\text{EquiBFT}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, for every encrypted transaction `ctx` issued by $\mathcal{Z}$, by Lemma 6, once it can be decrypted, at least $f + 1$ honest nodes are locked on the corresponding block. According to Lemma 7, this block will be committed as the next block. By Corollary 1, we deduce that this transaction is already ordered. Therefore, if $\texttt{ctx}.\widetilde{\texttt{tx}}$ can be decrypted, it is already ordered. □

V. Evaluation

In this section we show the implementation of our framework. The experiments are primarily aimed at validating the feasibility of the fair consensus protocol rather than conducting a comprehensive performance test.

A. Setup

1) Implementation: We have implemented our framework EquiBFT based on the open-source library '`libhotstuff`' of HotStuff [28]. We incorporate the threshold encryption scheme from [1] and utilize its practical implementation as detailed in [24]. For transaction encryption, we employ a hybrid approach where each transaction is encrypted using AES-256, and the encryption key is encapsulated with the TPKE scheme. For benchmarking purposes, we also implement F3B [30] that employs threshold encryption at the application layer, using HotStuff as its underlying consensus protocol. The decryption committee is composed of consensus nodes.

2) Experiment settings: Our experiments were conducted on Amazon EC2 using C5.4xlarge instances, each equipped with 16 vCPUs and 32 GB of memory. The performance evaluations were performed in a geo-distributed setting, with consensus nodes evenly distributed across four regions: Virginia (us-east-1), California (us-west-1), Tokyo (ap-northeast-1), and Frankfurt (eu-central-1). Our experiments explored varying block sizes and the number of nodes, with transactions generated and sent to the consensus nodes by clients in a back-loop manner. Each data point captures the system's saturated performance and represents the average of three experimental runs to ensure statistical reliability.

We measured and reported the throughput and latency of EquiBFT while comparing these metrics to those of F3B. Additionally, we implemented HotStuff to evaluate the performance decline margin relative to EquiBFT. Furthermore, we conducted experiments simulating malicious attacks, where clients send invalid transactions. As discussed in Section IV, these attacks do not compromise the safety of the protocol and cause only a similar level of impact as observed in HotStuff. Due to space limitations, we omit graphical representations of these results.

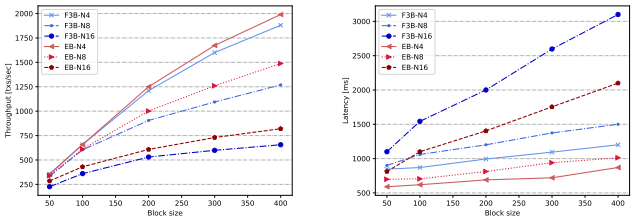
B. Experiment Results

In the legend, we use '`F3B-`' and '`EB-`' to represent the protocol F3B and EquiBFT, respectively. The block sizes which are denoted as '`B50`', '`B100`', '`B200`', and '`B400`' to present the number of transactions in each block. Additionally, experiments were conducted with 4 nodes ('`N4`'), 8 nodes ('`N8`') and 16 nodes ('`N16`') to measure the scalability of protocols.

1) Throughput: We start by analyzing the throughput of the protocols under varying numbers of nodes. Figure 2a illustrates the max throughput as the block size increases from 50 to 400 across configurations with 4, 8, and 16 nodes. Both EquiBFT and F3B exhibit an almost linear growth in throughput as the block size expands. For block sizes below 200, the two protocols demonstrate comparable throughput performance because of the pipeline structure. However, as the block size exceeds 200, EquiBFT achieves slightly higher throughput than F3B, with the performance gap widening as the block size continues to grow. Additionally, as the number of nodes increases, the rate of throughput improvement diminishes. Notably, F3B shows a slower growth rate compared to EquiBFT under these conditions.

2) Latency: We evaluated the latency of the two protocols under varying numbers of nodes and block sizes ranging from 50 to 400. As shown in Figure 2b, EquiBFT achieves latency that is approximately 60-70% of that observed in F3B. While both protocols experience increased latency as the block size grows, EquiBFT consistently outperforms F3B with lower latency, and the gap widens as the block size increases. Notably, the latency increase in EquiBFT is more gradual compared to F3B.

The latency advantage of EquiBFT stems from its design, where the decryption process and block commitment are completed within the same round of message exchange, eliminating additional time delays. In contrast, F3B requires an extra step of decryption message propagation which takes around 100 milliseconds after block consensus is reached, resulting in higher latency. Under transaction saturation, both protocols achieve similar throughput, largely due to their pipeline structures that enable the output of a committed block in every communication round. However, the extra round of decryption message propagation in F3B leads to a higher end-to-end latency.



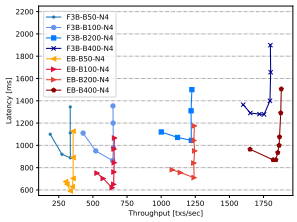
(a) The throughput under different block sizes.

(b) The latency under different block sizes.

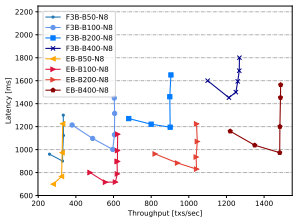
Fig. 2: Throughput and latency comparison between EquiBFT and F3B with different block size.

3) Scalability: We conducted further evaluations to assess the throughput and latency performance of EquiBFT in 4-node, 8-node, and 16-node network configurations. The results in

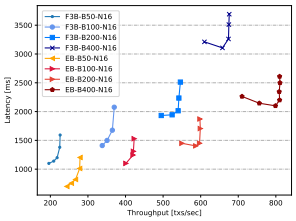
Fig 3 indicate that as the network scale increases, EquiBFT demonstrates better scalability compared to F3B. Specifically, using the 4-node configuration as a baseline, EquiBFT's throughput decreases to 85% of the baseline when the network expands to 8 nodes, with a latency increase of approximately 10%. When the network grows to 16 nodes, its throughput drops to 70% of the baseline, while latency increases by 40%. In contrast, F3B exhibits a more pronounced performance decline as the number of nodes increases; its throughput falls to 70% of the baseline value in the 8-node configuration and approximately 40% in the 16-node configuration. Furthermore, F3B's latency increases by 15% in the 8-node configuration and nearly doubles in the 16-node configuration.



(a) Performance with 4 nodes



(b) Performance with 8 nodes



(c) Performance with 16 nodes

Fig. 3: Throughput and latency comparison between EquiBFT and F3B with different number of nodes.

These results suggest EquiBFT and F3B both encounter a reduction in throughput as the network scales up. This decline is attributed to the increased time ranging from tens to hundreds of milliseconds required for threshold decryption processes, which involve verifying and aggregating decryption shares. Since EquiBFT completes the decryption of transactions simultaneously with the consensus process at the consensus layer without requiring additional time for the decryption process, EquiBFT exhibits better scalability, a conclusion that is also validated by experimental results.

VI. Conclusion

In this paper, we investigate the fairness problem of Byzantine consensus protocols in blockchain. We define the fairness of consensus protocols based on the logical relationship between the transactions ordering and decryption, introducing it as a new property of consensus protocols alongside liveness and safety. We extend the properties of liveness and safety in consensus protocols to encompass not only the consensus process of encrypted transactions but also the decryption process. Based on the existing BFT protocol and utilizing threshold encryption algorithms, we designed a framework called EquiBFT which can incorporate fairness to BFT protocols. We prove that EquiBFT satisfies fairness, liveness, and safety. Based on the well-known HotStuff protocol we implement the framework and evaluate the effectiveness of the protocol through experiments.

Acknowledgment

Lei Fan and Shengyun Liu were supported by the National Natural Science Foundation of China (grant no. 62372293) and the Natural Science Foundation of Shanghai (grant no. 24BC3201300). Hong-Sheng Zhou was supported in part by NSF grant CNS-1801470 and a VCU Research Quest grant.

References

[1] J. Baek and Y. Zheng, "Simple and efficient threshold cryptosystem from the gap diffie-hellman group," in *Proceedings of the Global Telecommunications Conference, 2003. GLOBECOM '03, San Francisco, CA, USA, 1-5 December 2003*. IEEE, 2003, pp. 1491–1495.

[2] M. Bartoletti, J. H. yu Chiang, and A. Lluch-Lafuente, "Maximizing extractable value from automated market makers," in *FC 2022: 26th International Conference on Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, I. Eyal and J. A. Garay, Eds., vol. 13411. Grenada: Springer, Cham, Switzerland, May 2–6, 2022, pp. 3–19.

[3] D. Boneh, X. Boyen, and S. Halevi, "Chosen ciphertext secure public key threshold encryption without random oracles," in *Topics in Cryptology – CT-RSA 2006*, ser. Lecture Notes in Computer Science, D. Pointcheval, Ed., vol. 3860. San Jose, CA, USA: Springer Berlin Heidelberg, Germany, Feb. 13–17, 2006, pp. 226–243.

[4] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[5] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *CoRR*, vol. abs/1710.09437, 2017. [Online]. Available: http://arxiv.org/abs/1710.09437

[6] C. Cachin and J. Micic, "Quick order fairness: Implementation and evaluation," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2024, Dublin, Ireland, May 27-31, 2024*. IEEE, 2024, pp. 230–234.

[7] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, M. I. Seltzer and P. J. Leach, Eds. USENIX Association, 1999, pp. 173–186.

[8] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 18–21, 2020, pp. 910–927.

[9] S. Duan, M. K. Reiter, and H. Zhang, "Secure causal atomic broadcast, revisited," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 61–72.

[10] S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: front-running attacks on blockchain," in *Financial Cryptography and Data Security: FC 2019 International Workshops, Revised Selected Papers 23*. Springer, 2020, pp. 170–189.

[11] V. Gramoli, Z. Lu, Q. Tang, and P. Zarbafian, "AOAB: optimal and fair ordering of financial transactions," in *54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2024*, pp. 377–388.

[12] J. Katz and Y. Lindell, *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[13] M. Kelkar, S. Deb, and S. Kannan, "Order-fair consensus in the permissionless setting," in *APKC '22: Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop, APKC@AsiaCCS 2022, Nagasaki, Japan, 30 May 2022*, J. P. Cruz and N. Yanai, Eds. ACM, 2022, pp. 3–14.

[14] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, "Themis: Fast, strong order-fairness in byzantine consensus," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 475–489.

[15] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, "Order-fairness for byzantine consensus," in *Advances in Cryptology – CRYPTO 2020, Part III*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12172. Santa Barbara, CA, USA: Springer, Cham, Switzerland, Aug. 17–21, 2020, pp. 451–480.

[16] R. Khalil, A. Gervais, and G. Felley, "TEX - A securely scalable trustless exchange," Cryptology ePrint Archive, Report 2019/265, 2019, https://eprint.iacr.org/2019/265.

[17] A. Kiayias, N. Leonardos, and Y. Shen, "Ordering transactions with bounded unfairness: Definitions, complexity and constructions," in *Advances in Cryptology – EUROCRYPT 2024, Part III*, ser. Lecture Notes in Computer Science, M. Joye and G. Leander, Eds., vol. 14653. Zurich, Switzerland: Springer, Cham, Switzerland, May 26–30, 2024, pp. 34–63.

[18] K. Kursawe, "Wendy, the good little fairness widget: Achieving order fairness for blockchains," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 25–36.

[19] R. Li, X. Hu, Q. Wang, S. Duan, and Q. Wang, "Transaction fairness in blockchains, revisited," Cryptology ePrint Archive, Report 2023/1034, 2023. [Online]. Available: https://eprint.iacr.org/2023/1034

[20] D. Malkhi and P. Szalachowski, "Maximal extractable value (MEV) protection on a DAG," *CoRR*, vol. abs/2208.00940, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2208.00940

[21] P. Momeni, S. Gorbunov, and B. Zhang, "Fairblock: Preventing blockchain front-running with minimal overheads," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2022, pp. 250–271.

[22] K. Mu, B. Yin, A. Asheralieva, and X. Wei, "Separation is good: A faster order-fairness byzantine consensus," in *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024.

[23] H. Nagda, S. P. Singhal, M. J. Amiri, and B. T. Loo, "Rashnu: Data-dependent order-fairness," *Proc. VLDB Endow.*, vol. 17, no. 9, pp. 2335–2348, 2024.

[24] Skalenetwork/libBLS, https://github.com/skalenetwork/libBLS/.

[25] C. Stathakopoulou, S. Rüsch, M. Brandenburger, and M. Vukolić, "Adding fairness to order: Preventing front-running attacks in bft protocols using tees," in *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2021, pp. 34–45.

[26] M. A. Vafadar and M. Khabbazian, "Condorcet attack against fair transaction ordering," in *5th Conference on Advances in Financial Technologies, AFT 2023, October 23-25, 2023, Princeton, NJ, USA*, ser. LIPIcs, J. Bonneau and S. M. Weinberg, Eds., vol. 282. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 15:1–15:21.

[27] D. Yakira, A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, and R. Tamari, "Helix: A fair blockchain consensus protocol resistant to ordering manipulation," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1584–1597, 2021.

[28] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[29] J. H. yu Chiang, B. David, I. Eyal, and T. Gong, "FairPoS: Input fairness in proof-of-stake with adaptive security," Cryptology ePrint Archive, Report 2022/1442, 2022. [Online]. Available: https://eprint.iacr.org/2022/1442

[30] H. Zhang, L.-H. Merino, Z. Qu, M. Bastankhah, V. Estrada-Galiñanes, and B. Ford, "F3B: A Low-Overhead Blockchain Architecture with Per-Transaction Front-Running Protection," in *5th Conference on Advances in Financial Technologies (AFT 2023)*, vol. 282, Dagstuhl, Germany, 2023, pp. 3:1–3:23.

[31] Y. Zhang, S. Setty, Q. Chen, L. Zhou, and L. Alvisi, "Byzantine ordered consensus without byzantine oligarchy," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 633–649.