SSHC: A Secure and Scalable Hybrid Consensus Protocol for Sharding Blockchains with a Formal Security Framework

Yizhong Liu, *Student Member, IEEE,* Jianwei Liu, Qianhong Wu, *Member, IEEE,* Hui Yu, Yiming Hei, and Ziyu Zhou

Abstract—Sharding blockchains are proposed to solve the scalability problem while maintaining security and decentralization. However, there are still many issues to be solved. First, the member selection and assignment process are not strictly analyzed, which might lead to an increase in the adversary proportion. Second, current intra-shard consensus algorithms are inefficient. Besides, cross-shard transaction processing costs expensive system overhead. Moreover, there is a lack of a formal security framework. In this paper, we propose a secure and scalable hybrid consensus (SSHC). First, we propose a fair sharding selection scheme to select committee members, including mining processes and member lists confirmation by a reference committee. Second, a pipelined Byzantine fault tolerance for intra-shard consensus is designed, combining the pipelined technology with threshold signatures. Third, we propose a responsive sharding transaction batch processing mechanism to handle cross-shard transactions, which reduces the number of calls to Byzantine fault tolerance algorithms. Fourth, a secure committee reconfiguration method is designed to update shard members efficiently. Furthermore, we employ a formal security framework to design and analyze a sharding blockchain. For an adversary whose computational power fraction is less than $1/3$, by reasonably setting a corruption parameter and other related parameters, SSHC is proved to achieve consistency and liveness.

Index Terms—sharding blockchains, scalability, Byzantine fault tolerance, fairness, responsiveness

1 Introduction

Since proposed by Bitcoin [1] in 2008, blockchain technology is developing rapidly, which uses tools such as mathematics and cryptography to establish a trusted distributed ledger in an untrusted environment [2]. The characteristics of data transparency, immutability, and decentralization, make blockchain have great application potentials in various fields, such as finance, supply chain management, and the Internet of Things [3]. However, to achieve secure and large-scale applications, blockchain technology needs to break through some restrictions and solve existing problems, realizing decentralization, security, and scalability [4].

The sharding blockchain is a promising way to achieve scalability while maintaining decentralization and security. The sharding technology is first adopted in the distributed database area [5] and is combined with blockchains first by ELASTICO [6]. In a sharding blockchain, participating nodes are divided into different shards, where each shard creates and maintains its specific blockchain [7]. Besides, transactions are processed by corresponding shards according to some pre-defined rules. When the number of nodes increases, more shards could be added to improve the ability to process transactions, namely, achieve transaction processing scalability or scale-out.

A sharding blockchain usually contains several essential components, such as node selection, node assignment, intro-shard consensus, cross-shard transaction processing, and shard reconfiguration. Each component is required to complete a specific function, while there might be certain problems in the process. In the following, we introduce the functions and problems of each component separately.

Node Selection. In permissionless sharding blockchains, a node selection mechanism, such as proof-of-work (PoW) or proof-of-stake (PoS), is in need to select qualified members. In a PoW-based member selection process, two important factors influence the selection results. One is network latency. An adversary is usually responsible for network transmission who might have a certain advantage in mining since he could acquire a mining puzzle in advance of honest nodes. The other is the selection process. The honest computational power ratio usually drops after a node selection process due to possible attacks. There are currently two selection methods, one of which uses an underlying blockchain where block producers are chosen as members, such as Byzcoin [8]. In this case, chain quality decreases severely due to selfish mining [9]. Another method uses a reference committee and a fixed PoW puzzle [10]. However, they do not consider possible attacks during the process, leading to an increase in malicious nodes proportion and damage to system security.

Node Assignment. The selected new nodes need to be randomly assigned to multiple committees based on a randomness. To ensure honest nodes fraction in each committee

- *Y. Liu is with School of Cyber Science and Technology, Beihang University and Department of Computer Science, University of Copenhagen. E-mail: liuyizhong@buaa.edu.cn*
- *J. Liu, Q. Wu, Y. Hei, and Z. Zhou are with School of Cyber Science and Technology, Beihang University.*
- *H. Yu is with with School of Electronic and Information Engineering, Beihang University.*

Manuscript received June 19, 2020; revised November 8, 2020. (Corresponding author: Qianhong Wu, Email: qianhong.wu@buaa.edu.cn)

after the allocation is greater than or equal to a tolerable ratio Q, which is determined by the intra-shard consensus algorithm, such as $1/3$ in practical Byzantine fault tolerance (PBFT) [11], the honest nodes ratio in all new nodes before allocation needs to be larger than Q.

Intra-shard Consensus. The intra-shard consensus algorithm is used to process transactions, determining system efficiency. The consensus algorithms used in existing blockchains need to be improved. Besides, interfaces between internal algorithms and external protocols need to be more explicit.

Cross-Shard Transaction Processing. To process cross-shard transactions, a two-phase commit (2PC) [12] scheme is usually required. In current methods [13], to process one transaction, input and output shards need to run the intra-shard consensus algorithm multiple times [14].

Committee Reconfiguration. Due to the adversary corruption attack, committees need to be updated after a certain period of time, or an adversary might control a committee. Previous schemes do not give a quantitative analysis of the corruption parameter τ, which denotes the time to complete a corruption process. However, τ is a crucial parameter in committee reconfiguration, determining system security. Besides, efficiently and safely downloading the UTXO data during epoch switching is also a key problem.

Lack of a Formal Security Model for Sharding Blockchains. A formal design and rigorous security analysis of a sharding blockchain are in need. Moreover, a quantitative analysis of the system liveness parameter is required.

1.1 Our Contributions

To solve the above questions, this paper proposes SSHC and makes the following contributions.

A Novel Fair Sharding Selection (FSS) Scheme with Rigorous Proofs. We propose FSS to select committee members for each shard, including the mining processes and new member lists confirmation, without node random allocation. We analyze the impact of network delay on the mining process and use a threshold-vote strategy to prevent a malicious leader from node-censorship attacks. By setting parameters reasonably, the honest node proportion in each selected committee is guaranteed to be greater than $2/3$.

A Pipelined Byzantine Fault Tolerance (plBFT) Algorithm. We design an ideal functionality $\mathcal{F}_{\text{TSIG}}$ for threshold signatures equipped with signature share generation and verification interfaces. We propose plBFT, which uses a pipelined method to improve processing efficiency and adopts threshold signatures to cut down message complexity. plBFT adopts a stable-leader, realizing a liveness parameter $O(\delta)$ in the optimistic case where δ denotes the actual network latency. plBFT is specially designed to defend against transaction and node censorship.

A Responsive Cross-Shard Transaction Batch Processing Scheme. SSHC designs a scheme to process cross-shard transactions responsively, where each shard packs q_s inputs of multiple transactions and processes them simultaneously. The processing efficiency is improved, and the number of calls to the Byzantine fault tolerance (BFT) algorithm is reduced. Specifically, to process the same number of transactions, the number of times to invoke BFT is reduced by about q_s times compared with classic 2PC methods, where q_s is usually on the order of hundreds or thousands. Furthermore, SSHC reduces the communication complexity among shards to a certain extent.

A Secure Committee Reconfiguration Method for Sharding Blockchains. During epoch switching, SSHC requires each committee to construct a Merkel tree with the current shard's UTXO and use plBFT to commit the root. The committed root serves as a reference for new members to update their UTXO from old members, preventing a malicious node from sending false UTXO data. Besides, the method improves committee reconfiguration efficiency.

A Formal Security Framework for Sharding Blockchains. We extend the security framework in [15] and give a formal treatment of sharding blockchains. The definition of a secure sharding blockchain and security properties that each subprotocol should satisfy are described. Then formal security analysis of the protocol is given. It is proved that by setting the corruption parameter τ and other related parameters reasonably, SSHC satisfies consistency, liveness, and responsiveness, where the confirmation time is $O(\delta)$ in the optimistic case.

1.2 Overview of SSHC

An overview of SSHC is shown in Fig. 1.

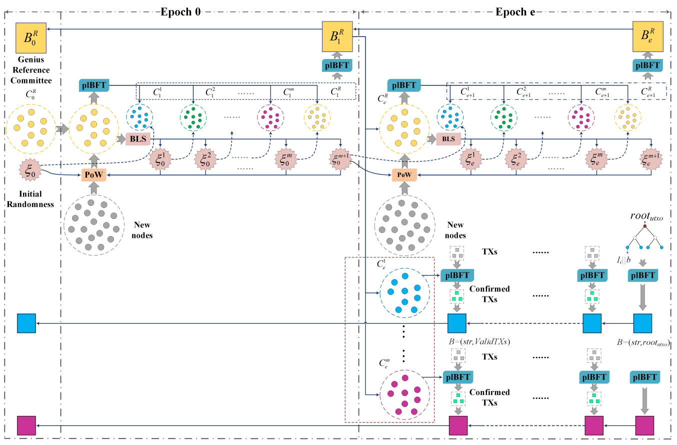

Figure 1. An overview of SSHC.

In epoch 0, SSHC relies on a trusted setup where an initial reference committee C_0^R, a randomness ξ_0, and genius blocks of m chains are embedded into an ideal functionality $\mathcal{F}_{init}$. Then nodes use ξ_0 as a puzzle to mine and submit PoW solutions to C_0^R. When k solutions are found, C_0^R runs plBFT to confirm C_1^1, then generates a randomness ξ_0^1 and broadcasts it. Nodes set ξ_0^1 as a puzzle to mine. The above processes repeat until $m+1$ committees $C_1^1, \cdots, C_1^m, C_1^R$ are confirmed. Then SSHC enters epoch 1. SSHC adopts a sequential committee selection instead of a concurrent one to prevent an adversary from focusing his computational power on one shard. An elaborate mining and member list confirmation design guarantees that each committee's honest fraction is greater than $2/3$. In epoch e ($e \geq 1$), a reference committee C_e^R is responsible for confirming committees of epoch $e+1$. The selection process is similar to epoch 0. Nodes first use ξ_{e-1}^{m+1} as a puzzle to mine. C_e^R runs plBFT to confirm C_{e+1}^1, then generates a randomness ξ_e^1. Based on ξ_e^1, C_{e+1}^2 is confirmed. The above selection is repeated until C_{e+1}^R is confirmed, and ξ_e^{m+1} is generated. Then SSHC enters epoch $e+1$.

Table 1
Comparison of SSHC with other sharding blockchains.

	System	Elastico	Omniledger	RapidChain	Chainspace	SSHC
System Model	Network Model	Partially Sync.	Partially Sync.	Sync.	Partially Sync.	Partially Sync.
	Total Adversary Model	$\leq \frac{1}{4}$	$\leq \frac{1}{4}$	$\leq \frac{1}{3}$	-	Any $\frac{1}{3} - \epsilon$
Intra-Shard Consensus	Adversary Model	$< \frac{1}{3}$	$< \frac{1}{3}$	$< \frac{1}{2}$	$< \frac{1}{3}$	$< \frac{1}{3}$
	Consensus Algorithm	PBFT	Omnicon	Sync BFT	PBFT	plBFT
	Complexity	$O(k^2)$	$O(k)$	$O(k^2)$	$O(k^2)$	$O(k)$
Cross-Shard Scheme	Basic Algorithm	-	2PC	Split	2PC	2PC
	Coordinator	-	Client-Driven	Shard-Driven	Shard-Driven	Shard-Driven
Performance	Responsiveness	✗	✗	✗	✓	✓
	Scalability	✗	✓	✓	✓	✓
Safety	Formal Security Framework	✗	✗	✗	✗	✓
	Corruption Parameter	-	-	-	-	τ

Transactions are processed by m ordinary committees through epoch sharding consensus (ESC). ESC invokes plBFT inside a committee to process proposals and handles cross-shard transactions. Every input shard confirms related input availability states, composes them into a Merkel tree, and uses plBFT to commit the root value. Then, for each shard, its availability certificate ac is constructed and sent. After receiving all input ac of a transaction, a shard could judge its validity and complete the confirmation.

SSHC achieves an optimal scaling factor $O(\frac{n}{\log n})$ as defined in [16]. A comparison between SSHC and previous works is given in Table 1.

2 Notations and System Model

In this section, we give notations and system models that are adopted in this paper.

2.1 Notations

The notations in the paper are shown in Table 2.

Table 2
Notations

Symbol	Definition
k	the total number of nodes in a committee
f	the number of malicious nodes in a committee
m	the number of ordinary committees
n	the total number of nodes in the protocol
S_i	the i-th shard
C_e^i	the i-th committee of epoch e
C_e^R	the reference committee of epoch e
Δ	the upper bound of the network delay
δ	the actual network delay
tx	a transaction
chain	a blockchain
LOG_i	the output log of the i-th shard's nodes
ρ	the adversary's computational power fraction
τ	the adversary's corruption parameter

2.2 System Model

Formal Framework for Composing Consensus Protocols. Following Hybrid Consensus [15], we adopt a universal composition-like formal framework [17], [18], enabling composition of cryptographic protocols. Formal properties of protocols are defined instead of describing ideal behavioral specifications. A protocol's security requirements are defined based on properties over honest nodes' outputs to an environment. Security properties are preserved when a protocol is composed with other protocols. Besides, we adopt a modular composition approach to present SSHC.

Restrained Execution Environments. Let the adversary and environment $(\mathcal{A}, \mathcal{Z})$ be probabilistic polynomial time (P.P.T.) algorithms, and let $k \in \mathbb{N}$ be the security parameter. The protocol Π is executed under $(\mathcal{A}, \mathcal{Z})$, and we define $\mathsf{EXEC}_\Pi(\mathcal{A}, \mathcal{Z}, k)$ as a random variable denoting nodes' joint view, including the inputs, messages received, and outputs.

$(\mathcal{A}, \mathcal{Z})$ should respect certain restraints to achieve the security goal.

Definition 1 ((n, ρ, Δ, τ)-restrained $(\mathcal{A}, \mathcal{Z})$)**.** *We say that the environment and adversary pair $(\mathcal{A}, \mathcal{Z})$ is (n, ρ, Δ, τ)-restrained w.r.t. the protocol Π if and only if for every $k \in \mathbb{N}$ and every* view *of* $\mathsf{EXEC}_\Pi(\mathcal{A}, \mathcal{Z}, k)$, *it holds that:*

- *The number of live or active nodes at any time is n;*
- *$\mathcal{A}$ controls at most ρ fraction of the computational power;*
- *$\mathcal{A}$ delays messages of honest nodes in at most Δ time steps;*
- *$\mathcal{A}$ corrupts an honest node obeying the following rule: $\mathcal{A}$ first sends a corrupt instruction to an honest node through $\mathcal{Z}$ and waits for τ time before he could control the node.*

Adversary Model. We assume a τ-mildly adaptive adversary. $\mathcal{A}$ has the ability to control all the corrupted nodes perfectly. $\mathcal{A}$ cannot modify any message from honest nodes, nor can he forge any honest node's signature.

Network Model. SSHC assumes a partially synchronous network, where there is an unknown upper bound Δ of the network delay. $\mathcal{A}$ is responsible for messages delivery in the network. Messages might be delayed or reordered by $\mathcal{A}$, yet constraint to the Δ-time limit.

3 Building Blocks

In this section, we design two building blocks that are useful in later protocol construction. One is a threshold signature ideal functionality, and the other is plBFT.

3.1 Threshold Signature

The threshold signature is used in SSHC for intra-shard consensus and randomness generation. Committee members adopt the threshold signature to vote, reducing communication costs. The uniqueness property of some threshold

Functionality $\mathcal{F}_{\text{TSIG}}$

Key Generation: On receiving n messages $(\mathsf{KeyGen}, sid, P_i)_{i=1}^{n}$ which contain at least $t+1$ messages from honest nodes in the same round, hand $(\mathsf{KeyGen}, sid, P_i)_{i=1}^{n}$ to the adversary. On receiving $(\mathsf{VerificationKey}, sid, P_i, v_i, V)_{i=1}^{n}$ from the adversary, output $(\mathsf{VerificationKey}, sid, v_i, V)_{i=1}^{n}$ to all nodes, and record the pair $(sid, P_i, v_i)_{i=1}^{n}$ and (sid, V).

Signature Share Generation: On receiving $(\mathsf{ThSign}, sid, P_i, m)$ from party P_i, verify that (sid, P_i, v_i) is recorded for some sid. If not, ignore the request. Else verify that if there is an entry $(m, \sigma_i, v_i, 1)$ recorded. If it is, then output $(\mathsf{ThSignature}, sid, P_i, m, \sigma_i)$ to P_i. Else, hand $(\mathsf{ThSign}, sid, P_i, m)$ to the adversary. On receiving $(\mathsf{ThSignature}, sid, P_i, m, \sigma_i)$ from the adversary, verify that no entry $(m, \sigma_i, v_i, 0)$ is recorded. If it is, then output an error message to P_i and halt. Else, output $(\mathsf{ThSignature}, sid, P_i, m, \sigma_i)$ to P_i and record the entry $(m, \sigma_i, v_i, 1)$.

Signature Share Verification: On receiving a message $(\mathsf{ThVerify}, sid, m, \sigma_i, v_i')$ from some party P, hand $(\mathsf{ThVerify}, sid, m, \sigma_i, v_i')$ to the adversary. On receiving $(\mathsf{THVerified}, sid, m, \phi)$ from the adversary, do:

1) If there is an entry $(m, \sigma_i, v_i, 1)$ recorded where $v_i' = v_i$, then set $f := 1$.
2) Else, if $v_i' = v_i$, the signer is not corrupted, and no entry $(m, \sigma_i', v_i, 1)$ for any σ_i' is recorded, then set $f := 0$ and record the entry $(m, \sigma_i, v_i, 0)$.
3) Else, if there is an entry (m, σ_i, v_i', f') recorded, then let $f := f'$.
4) Else (i.e. $v' \neq v$ or if (m, σ_i') is recorded for $\sigma_i' \neq \sigma_i$), let $f := \phi$ and record the entry $(m, \sigma_i, v_i', \phi)$.

Output $(\mathsf{ThVerified}, sid, m, f)$ to P.

Signature Combination: On receiving a message $(\mathsf{ThCombine}, sid, m, (v_1, \sigma_1), \cdots, (v_{t+1}, \sigma_{t+1}))$ from some party P, verify that (sid, V) is recorded for some sid. If not, ignore the request. Else verify that each pair $(v_1, \sigma_1), \cdots, (v_{t+1}, \sigma_{t+1})$ is registered. If not, output $(\mathsf{Signature}, sid, m, \perp)$ to P. Else verify that if there is an entry $(m, \sigma, V, 1)$ recorded. If it is, then output $(\mathsf{Signature}, sid, m, \sigma)$ to P. *// (this condition guarantees uniqueness)* Else, hand $(\mathsf{ThCombine}, sid, m, (v_1, \sigma_1), \cdots, (v_{t+1}, \sigma_{t+1}))$ to the adversary. On receiving $(\mathsf{Signature}, sid, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, V, 0)$ is recorded. If it is, then output an error message to P and halt. Else, output $(\mathsf{Signature}, sid, m, \sigma)$ to P, and record the entry $(m, \sigma, V, 1)$.

Signature Verification: On receiving a message $(\mathsf{Verify}, sid, m, \sigma, V')$ from some party P, hand $(\mathsf{Verify}, sid, m, \sigma, V')$ to the adversary. On receiving $(\mathsf{Verified}, sid, m, \phi)$ from the adversary, do:

1) If there is an entry $(m, \sigma, V, 1)$ recorded where $V' = V$, then set $g := 1$. *// (this condition guarantees completeness: if the verification key V' is a registered one and σ is a legitimately generated signature for m, then the verification succeeds)*
2) Else, if $V' = V$, and no entry $(m, \sigma', V, 1)$ for any σ' is recorded, then set $g := 0$ and record the entry $(m, \sigma, V, 0)$. *// (unforgeability: if V' is a registered verification key, the signer is not corrupted, and never signed m, then the verification fails)*
3) Else, if there is an entry (m, σ, V', g') recorded, then let $g := g'$. *// (this condition guarantees consistency: all verification request with identical parameters will result in the same answer)*
4) Else, let $g := \phi$ and record the entry (m, σ, V', ϕ).

Output $(\mathsf{Verified}, sid, m, g)$ to P.

Figure 2. Ideal functionality $\mathcal{F}_{\text{TSIG}}$.

signature schemes such as BLS [19] allows nodes to generate a public verifiable, unbiasable, and unpredictable randomness used as a fresh puzzle for PoW mining. The definition of a (t, n)-threshold signature scheme is referred to [20]. For the modular design of the protocol, we give a novel ideal functionality $\mathcal{F}_{\text{TSIG}}$ in Figure 2 of a secure and unique (t, n)-threshold signature that offers signature share generation and verification interfaces.

3.2 Pipelined BFT

pIBFT is a partially synchronous protocol for state machine replication tolerating up to f malicious nodes. The total number of participating nodes is k, where $k \geq 3f+1$. The flow chart and concrete protocol of pIBFT are shown in Figure 3 and Figure 4, respectively.

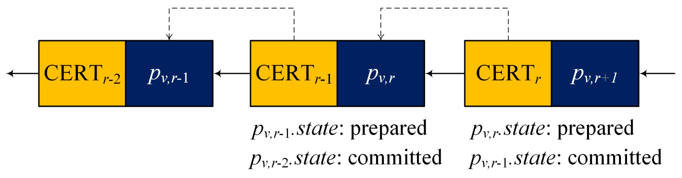
$\mathrm{CERT}_{r\text{-}2}$
$p_{v,r\text{-}1}$
$\mathrm{CERT}_{r\text{-}1}$
$p_{v,r}$
CERT_{r}
$p_{v,r+1}$
$p_{v,r\text{-}1}.state$: prepared
$p_{v,r\text{-}2}.state$: committed
$p_{v,r}.state$: prepared
$p_{v,r\text{-}1}.state$: committed

Figure 3. The flow chart of pIBFT.

v_i is a verification key share, and $C := \{v_i\}_{i=1 \text{ to } k}$ consists of k members. c is the committee sequence number where $c \in [1, m] \cup \{R\}$. The leader of a view v is denoted as L_v. After a start command is input, $\mathcal{Z}$ inputs some values val to honest nodes. p denotes a new proposal. The symbol $\langle \cdot \rangle$ represents that the message is signed by the sender. After val is committed, honest nodes output $\langle \mathsf{val} \rangle$ to $\mathcal{Z}$. The message in a round r contains a new proposal $p_{v,r}$ and a certificate of the previous round $r-1$, denoted as CERT_{r-1}. In each round r, $p_{v,r}$ is set to a null value $\perp$ by default. When CERT_{r-1} is generated, if pIBFT does not receive a new input, the leader will take $\perp$ as the proposal and broadcast it with CERT_{r-1}. This is to ensure that the previous two rounds' proposal can be processed in time.

CERT_{r-1} contains a threshold signature from $2f+1$ valid votes of round $r-1$. We adopt a $(2f, k)$-threshold signature to cut down the message length from $O(k)$ to $O(1)$. The threshold t equals $2f$, while $\mathcal{A}$ controls up to f corrupted nodes. When a leader is honest, pIBFT relies on normal operations. If a leader is malicious or offline, view change is invoked to ensure liveness.

A prediction $\mathsf{IsPValid}$, as shown in Algorithm 1, is invoked in pIBFT to verify if an input value is valid. $\mathsf{H}(\cdot)$ is a hash function. $\mathsf{head(chain)}$ returns the last block of a blockchain chain. The input type of $\mathsf{IsPValid}$ could be a root value root, a transaction block B, an update transaction set $\mathsf{UpdateTXs}$, a reference block B^{R}, or a committee list comm. The details of $\mathsf{IsPValid}$ are related to ESC and SSHC. root, B, and $\mathsf{UpdateTXs}$ are input to pIBFT by ESC, and corresponding analysis is given in Lemma 11 and Lemma 12. B^{R} and comm are input to pIBFT by SSHC, and related analysis is given in Lemma 7. It is proved that whatever the type of p is, a p proposed by an honest leader could pass the verification of $\mathsf{IsPValid}$, namely $\mathsf{IsPValid}(p) = 1$, so p will get enough votes from honest members. An invalid p proposed by a malicious leader cannot pass the verification, then honest

Subprotocol pBFT

Inputs. 1) $\mathsf{start}(C_e^c, v_i^{sig})$; 2) Some values val: an input state root $root$, a state set $state$, a transaction block B, a transaction and state set $\mathsf{UpdateTXs}$, a certificate pool AC, a new member list C, or a reference block B^R; 3) stop.
Outputs. Committed values $\langle \mathsf{val} \rangle$.
Initialization. On receiving $\mathsf{start}(C_e^c, v_i^{sig})$, P_i sends $(\mathsf{KeyGen}, sid_c, P_i)$ to $\mathcal{F}_{\mathrm{TSIG}}$, getting $(\mathsf{VerificationKey}, sid_c, v_i^{tsig}, V_c)$ where v_i^{tsig} is the verification key share and V_c is the public key of C_e^c.
Normal Operations.
$\triangleright$ **A Leader.** In round r, set $\mathsf{ValidVotes}_r := \varnothing$. Set $p_{v,r} := \perp$. *(a leader proposes $\perp$ when no input is given)*

- On receiving val from $\mathcal{Z}$, if $\mathsf{val}.type = \mathsf{root}, \mathsf{B}, \mathsf{ValidTXs}, \mathsf{comm}$ or B^R, put val into a proposal queue $\mathcal{P}$; Set $p_{v,r} :=$ the oldest $\mathsf{val} \in \mathcal{P}$. If $\mathsf{val}.type = \mathsf{B} \wedge (B$ contains a $root_{\mathsf{utxo}_c})$, set $p_{v,r} := \mathsf{val}$. // *(B with a $root_{\mathsf{utxo}_c}$ is as a "stop" signal)*
- On receiving a vote message $(m_{\mathsf{vote}}, \sigma_i)$ in some round r: // *(collect valid votes)*
  Send $(\mathsf{ThVerify}, sid, m_{\mathsf{vote}}, \sigma_i, v_i^{tsig})$ to $\mathcal{F}_{\mathrm{TSIG}}$, and receive $(\mathsf{ThVerified}, sid, m_{\mathsf{vote}}, f)$;
  If $f = 1$, then set $\mathsf{ValidVotes}_r := \mathsf{ValidVotes}_r \cup \{(\sigma_i, v_i^{tsig})\}$. Else, ignore the message.
- If $|\mathsf{ValidVotes}_r| = 2f+1$, then L_v: // *(generate a threshold signature using $2f+1$ valid votes, which include its own vote)*
  1) Send $(\mathsf{ThCombine}, sid_c, m_{\mathsf{vote}}, (\sigma_1, v_1^{tsig}), \cdots, (\sigma_{2f+1}, v_{2f+1}^{tsig}))$ to $\mathcal{F}_{\mathrm{TSIG}}$, and receive $(\mathsf{Signature}, sid_c, m_{\mathsf{vote}}, \sigma)$;
  2) Set $\mathsf{CERT}_{r-1} := (\mathsf{H}(m_{\mathsf{vote}}), \sigma)$, and construct $m_{\mathsf{prop}} := (\text{"propose"}, v, r, \mathsf{CERT}_{r-1}, p_{v,r})$;
  3) Send $(\mathsf{Sign}, sid, m_{\mathsf{prop}})$ to $\mathcal{F}_{\mathrm{SIG}}$, receive $(\mathsf{Signature}, sid, m_{\mathsf{prop}}, \sigma_L)$, and broadcast $(m_{\mathsf{prop}}, \sigma_L)$. If $p_{v,r}.type = \mathsf{root}$, broadcast the corresponding $state$. If $p_{v,r}.type = \mathsf{B}$ or $\mathsf{ValidTXs}$, broadcast the corresponding AC.

$\triangleright$ **Honest Nodes.** In round r of view v, set $vcb_{v,r} := \mathsf{false}$.

- On receiving val from $\mathcal{Z}$, store it as local $root'$, $state'$, B', $\mathsf{UpdateTXs}'$, AC', C', or $B^{R'}$. // *(used to judge whether to vote later, and as backups if chosen as a leader)*
- On receiving $(m_{\mathsf{prop}}, \sigma_L)$ in some round r:
  Send $(\mathsf{Verify}, sid, m_{\mathsf{prop}}, \sigma_L, v_L^{sig})$ to $\mathcal{F}_{\mathrm{SIG}}$ and receive $(\mathsf{Verified}, sid, m_{\mathsf{prop}}, f)$. If $f = 0$, ignore the message. Else:
  1) Parse m_{prop} as $(\text{"propose"}, v, r, \mathsf{CERT}_{r-1}, p_{v,r})$ and obtain $\mathsf{CERT}_{r-1} = (h, \sigma)$;
  2) Verify if $h = \mathsf{H}(m_{\mathsf{vote}})$. If not, set $\mathsf{blame}_{v,r} := (m_{\mathsf{prop}}, \sigma_L)$, $vcb_{v,r} := \mathsf{ture}$. Else: // *(view-change is triggered)*
     Send $(\mathsf{Verify}, sid, m_{\mathsf{vote}}, \sigma, V)$ to $\mathcal{F}_{\mathrm{TSIG}}$ and receive g. If $g = 0$, set $\mathsf{blame}_{v,r} := (m_{\mathsf{prop}}, \sigma_L)$, $vcb_{v,r} := \mathsf{ture}$. Else:
     a) Consider $p_{v,r-1}.state := \mathsf{prepared}$ and $p_{v,r-2}.state := \mathsf{committed}$, and output $\langle p_{v,r-2} \rangle$ to $\mathcal{Z}$; // *(state change)*
     b) If it has voted for a proposal numbered (v, r), ignore it. Else,
     c) If $\mathsf{IsPValid}(p_{v,r}) = 1$, then: // *(vote if the proposal is valid)*
        i) Construct a vote message $m_{\mathsf{vote}} := (\text{"vote"}, v, r, \mathsf{H}(p_{v,r}), \mathsf{H}(\mathsf{CERT}_{r-1}))$;
        ii) Send $(\mathsf{ThSign}, sid_c, m_{\mathsf{vote}})$ to $\mathcal{F}_{\mathrm{TSIG}}$, and receive $(\mathsf{ThSignature}, sid_c, m_{\mathsf{vote}}, \sigma_i)$;
        iii) Send $(m_{\mathsf{vote}}, \sigma_i)$ to the current leader L_v, and start a new timer t_v. // *(ensure liveness)*
        Else, i.e., $\mathsf{IsPValid}(p_{v,r}) = 0$, set $\mathsf{blame}_{v,r} := (m_{\mathsf{prop}}, \sigma_L)$, $vcb_{v,r} := \mathsf{ture}$. // *(view-change is triggered)*

View-Change.
$\triangleright$ **Honest Nodes.**

- If $(vcb_{v,r} = \mathsf{ture})$ or (P_i receives no proposal for (v, r) when t_v ends): // *(two view-change conditions)*
  1) Set r' to be the maximum round number that is prepared;
  2) If $vcb_{v,r} = \mathsf{ture}$, then: construct $m_{\mathsf{vc}} := (\text{"view-change-blame"}, v+1, r', \mathsf{blame}_{v,r}, p_{v,r'}, \mathsf{CERT}_{r'})$;
     Else, construct $m_{\mathsf{vc}} := (\text{"view-change-timeout"}, v+1, p_{v,r'}, \mathsf{CERT}_{r'})$;
  3) Send $(\mathsf{Sign}, sid, m_{\mathsf{vc}})$ to $\mathcal{F}_{\mathrm{SIG}}$, and receive $(\mathsf{Signature}, sid, m_{\mathsf{vc}}, \sigma_{vc})$. Broadcast $(m_{\mathsf{vc}}, \sigma_{vc})$.
- On receiving a new-view message m_{nv}, if VC_{v+1} contains $2f+1$ valid view-change messages, then:
  1) Set the view number to $v+1$ and $r := 0$; // *(enter a new view)*
  2) Update local state to $p_{v,r^*}.state := \mathsf{prepared}$ and $p_{v,r^*-1}.state := \mathsf{committed}$. Continue processing as in the normal case.

$\triangleright$ **A Leader (L_{v+1}).** Set $p_{v+1,0} := \perp$. On receiving a val from $\mathcal{Z}$, set $p_{v+1,0} := \mathsf{val}$.

- On receiving a $(m_{\mathsf{vc}}, \sigma_{vc})$, send $(\mathsf{Verify}, sid, m_{\mathsf{vc}}, \sigma_{vc}, v_i^{sig})$ to $\mathcal{F}_{\mathrm{SIG}}$ and receive $(\mathsf{Verified}, sid, m_{\mathsf{vc}}, f)$;
  If $f = 0$, ignore the message. Else set $\mathsf{VC}_{v+1} := \mathsf{VC}_{v+1} \cup \{(m_{vc}, \sigma_{vc})\}$. // *(collect valid view-change messages)*
- If $|\mathsf{VC}_{v+1}| = 2f+1$, let Φ denote a tuple in VC_{v+1}, then: // *(launch a view-change)*
  1) Set $r^* := \underset{\Phi \in \mathsf{VC}}{\mathrm{argmax}}(\Phi.r)$, and $p_{v,r^*}, \mathsf{CERT}_{r*}$ is the corresponding proposal and commit certificate;
  2) Constructs a new-view message: $m_{\mathsf{nv}} := (\text{"new-view"}, v+1, 0, p_{v,r^*}, \mathsf{CERT}_{r*}, \mathsf{VC}_{v+1}, p_{v+1,0})$, and broadcast m_{nv}.

Stop. On receiving stop, waits till the last input val is committed. Output $\langle \mathsf{val} \rangle$ to $\mathcal{Z}$, then stop all operations.

Figure 4. Subprotocol pBFT.

members launch a view-change to switch the leader. In this way, consistency and liveness of pBFT are ensured.

Compared with other BFT algorithms such as Hot-Stuff [21], pBFT has the following advantages. First, pBFT is specially designed to be better applicable to sharding blockchains. pBFT supports multiple types of proposals, including transaction blocks, Merkel tree roots, reference blocks, new node lists, etc. There is a special rule for each proposal to judge its validity. Second, pBFT adopts a stable-leader. When a leader is honest, the system stability and processing efficiency are higher. Third, pBFT's design adopts a formal approach consistent with the formal security framework of the whole protocol. The clear interfaces make it more convenient to be invoked by the ESC and SSHC protocols, and a universal composable method can be utilized to prove the system security.

Definition 2 (Restrained executions for pBFT). *A P.P.T. algorithm pair $(\mathcal{A}, \mathcal{Z})$ is said to be $(n, \rho, \Delta, \tau, Q)$-restrained w.r.t.* pBFT *if and only if the following conditions are satisfied:*

- *$(\mathcal{A}, \mathcal{Z})$ is (n, ρ, Δ, τ)-restrained as defined by Definition 1.*
- *Committee correctness.*
  – *Committee agreement for one shard. For any two honest*

Algorithm 1: IsPValid

```
input : $p$
output: $\{0,1\}$
if $p.type$ = root then
    get $p$'s related $state$ (from the leader);
    for every $(\_, I_i) \in state$, query $I_i$ from local utxo, if
      $\neg$ (utxo$_c[I_i] = 0 \wedge state[(,I_i)] = 0 \vee$
      utxo$_c[I_i] = 2 \wedge state[(,I_i)] = 2 \vee$ utxo$_c[I_i] = 1 \wedge$(one
      and only one (tx.$id, I_i$) satisfies $state[$(tx.$id, I_i)] = 1$,
      and others are $state[$(tx.$id', I_i)] = 2$)) then return 0;
    if ($p \neq$ the root value of $state$) then return 0;
    return 1 // (ordinary committee, input states)
else if $p.type$ = B then
    if $p$ could be parsed as ($str$, ValidTXs) then
        get the related AC (from the leader);
        for every input $I_i$ of every tx in ValidTXs, query
          its state from local utxo for $I_i \in S_c$, or query its
          state from AC for $I_i \notin S_c$ ;
        if (for all $I_i \in$ tx, tx.$state[I_i] = 1$)
          $\wedge(str =$ H(head(chain))) then return 1;
        else return 0 // (ordinary committee, tx block);
    else
        parse $p$ as ($str, root_{\mathsf{utxo}}$), and get local utxo$'$;
        if ($root_{\mathsf{utxo}}$ equals to the hash root value of utxo$'$
          where a leaf node is $I_i||$utxo$'[I_i]$) $\wedge$
          ($str =$ H(head(chain))) then return 1;
        else return 0;
else if $p.type$ = UpdateTXs then
    get the related AC (from the leader);
    for every input $I_i$ of every tx in UpdateTXs, query
      its state from local utxo for $I_i \in S_c$, or query its
      state from AC for $I_i \notin S_c$ ;
    if for every $I_i$ of every tx, tx.$state[I_i]$ is correct then
      return 1;
    else return 0 // (ordinary committee, tx block);
else if $p.type$ = B$^{\mathsf{R}}$ then
    parse $p$ as ($str$, list), and get local list$'$;
    if ($str =$ H(head(chain$^R$))) $\wedge$ (list = list$'$) then return
      1;
    else return 0 // (reference committee, R block);
else if $p.type$ = comm then
    parse $p$ as $C$, and get local $C'$;
    if different nodes between $C$ and $C' \leq k_T$ then return
      1 // (threshold-vote);
    else return 0 // (reference committee, member list);
else return 0;
```

nodes i, j that receive start(C_e^c, v_i) *and* start$(C_e^{c'}, v_j)$ *with identical c and e from $\mathcal{Z}$, it holds that $C_e^c = C_e^{c'}$.*

– *No conflict among shards. For any two honest nodes i, j that receive* start $(C_e^{c_1}, v_i)$ *and* start$(C_e^{c_2}, v_j)$ *from $\mathcal{Z}$ respectively, if $c_1 \neq c_2$, then it holds that there is no v_i that satisfies $(v_i \in C_e^{c_1}) \wedge (v_i \in C_e^{c_2})$.*

- *Close start and stop. Let T_{start} be the time when* start$(C_e^c, -)$ *is first input to an honest node by $\mathcal{Z}$. By time $T_{start} + \delta$, every honest node has received* start$(C_e^{c'}, -)$ *where $c = c'$ or $c \neq c'$. The definition for T_{stop} is the same.*
- *Honest majority. For every* start *command* start$(C_e^c, -)$ *where $c \in [1, m] \cup \{R\}$ and $e \in \mathbb{N}$ issued by $\mathcal{Z}$, it holds that at least $\lceil Q \cdot |C_e^c| \rceil$ number of $v_i \in C_e^c$ must be specified to remain honest until the end of the protocol.*

Definition 3 (Secure plBFT)**.** *Let T_{plbft} be polynomially bounded functions in δ, k and Q. Then we say* plBFT *is secure if and only if the following properties are satisfied.*

- *Agreement. For any two honest nodes i, j of the same committee, if node i outputs a committed value $\langle p_{v,r} \rangle$ and node j outputs $\langle p'_{v,r} \rangle$ for the same v and r, then $p_{v,r} = p'_{v,r}$.*
- *Liveness. If $\mathcal{Z}$ inputs* val *to an honest node at some time $T_{start} \leq t < T_{stop}$, then after time $t' = t + T_{plbft}$,* val *is committed and output by every honest node. T_{plbft} is the liveness parameter for a proposal to be committed.*

Theorem 1 (plBFT is secure)**.** *Assume that $(\mathcal{A}, \mathcal{Z})$ is $(n, \rho, \Delta, \tau, Q)$-restrained w.r.t.* plBFT*. Then* plBFT *satisfies agreement and liveness as per Definition 3 against $(1 - Q) < \frac{1}{3}$ corruption with the liveness parameter T_{plbft} where $T_{plbft} = O(k\delta)$, and $T_{plbft} = O(\delta)$ in the optimistic case.*

4 THE SSHC PROTOCOL

In this section, we describe the concrete construction of SSHC. Figure 5 is used to explain the interaction among SSHC, ESC, and plBFT.

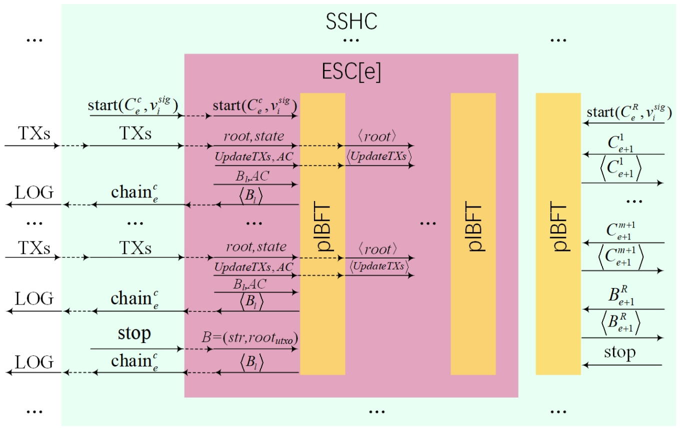

Figure 5. The interaction among SSHC, ESC, and plBFT.

4.1 System Goal

In every shard, nodes maintain a transaction log LOG. Our definition of sharding consensus is an extension of the traditional abstraction proposed in the Bitcoin backbone protocol by Garay et al. [22].

Definition 4 (A Secure Sharding Consensus)**.** *Let $(\mathcal{A}, \mathcal{Z})$ be (n, ρ, Δ, τ)-restrained w.r.t. a sharding consensus protocol Π. Let $T_{initial}, T_{plbft}$ be polynomial functions in n, ρ, δ, τ, k. We say Π is secure w.r.t. $(\mathcal{A}, \mathcal{Z})$ with parameters $T_{initial}, T_{plbft}$ if there exists a negligible function $\mu(k)$ such that the following properties hold for* $\mathsf{EXEC}_\Pi(\mathcal{A}, \mathcal{Z}, k)$ *with probability $1 - \mu(k)$:*

- *Consistency:*
  – *Common prefix in shard: For any two honest nodes $i, j \in S_c$ where $c \in [1, m]$, node i outputs* LOG$_i$ *to $\mathcal{Z}$ at time t, and node j outputs* LOG$_j$ *to $\mathcal{Z}$ at time t', it holds that either* LOG$_i \prec$ LOG$_j$ *or* LOG$_j \prec$ LOG$_i$*. The notation "$\prec$" means "is a prefix of".*
  – *No conflict between shards: For any two honest nodes $i \in S_c, j \in S_{c'}$ where $c, c' \in [1, m]$ and $c \neq c'$, node i outputs* LOG$_i$ *to $\mathcal{Z}$ at time t, and node j outputs* LOG$_j$ *to $\mathcal{Z}$ at time t'. For any two transactions* tx$_1 \in$ LOG$_i$ *and* tx$_2 \in$ LOG$_j$ *where* tx$_1 \neq$ tx$_2$*, it holds that* tx$_1$ *and* tx$_2$ *do not conflict with each other, i.e., no input belongs to* tx$_1$ *and* tx$_2$ *simultaneously.*
- *$T_{liveness}$-Liveness: For any honest node from any shard, if it receives a transaction* tx *at time $t_0 \geq T_{initial}$ from $\mathcal{Z}$, then at time $t_0 + T_{liveness}$,* tx *must be accepted or rejected.*

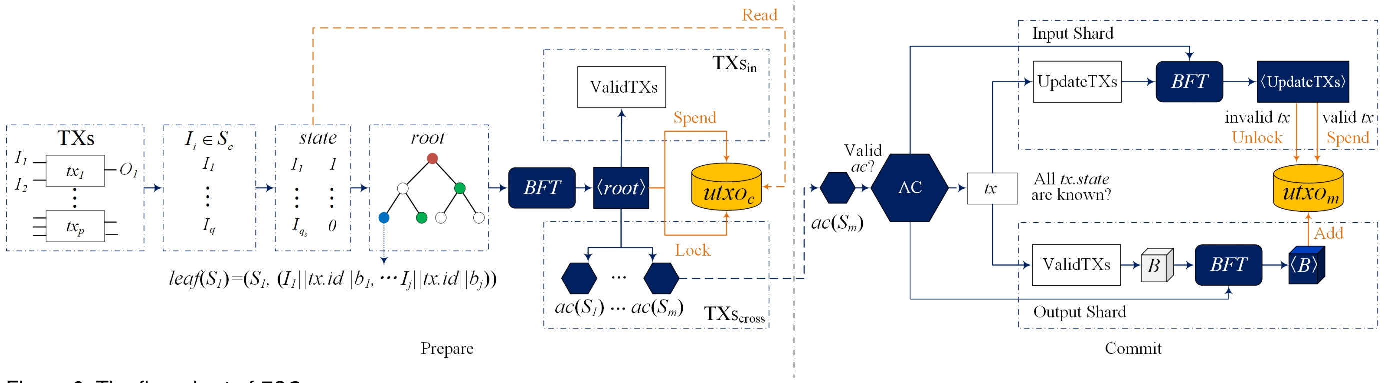

Figure 6. The flow chart of ESC.

4.2 Epoch Sharding Consensus Protocol

ESC is run by m shards. In every shard S_c, there is a committee C_e^c responsible for processing transactions in epoch e. The flow chart and concrete protocol of ESC are described in Figure 6 and Figure 7, respectively.

ESC's cross-shard transaction processing is based on the 2PC protocol, including the prepare and commit phase. To make our description clearer, we give the definition of an availability certificate in the following.

Definition 5 (Availability Certificate)**.** *An availability certificate refers to a proof (one or multiple signatures) in the prepare phase that each shard provides for one or multiple transaction inputs, to prove that the input is available (ready to be spent) or unavailable.*

To verify if a tx is valid, members need to confirm tx's every input state [23]. Only if all its inputs are available, tx is treated valid. A valid tx is only written to ValidTXs in its output shard. If the current shard is an input shard of tx, then tx is put into UpdateTXs, which will be committed by pIBFT. ⟨UpdateTXs⟩ is regarded as a reference for the UTXO update. Once tx is confirmed as valid, its inputs will be spent and removed from the UTXO pool. If one or more inputs are unavailable, then tx is invalid, and the corresponding previously locked inputs need to be unlocked.

Efficiency Analysis of Transaction Processing. We compare SSHC with classic 2PC methods about the cross-shard transaction processing message complexity in Table 3. Suppose that there are n cross-shard transactions, each of which contains s inputs and 1 output on average. The number of inputs in *state* is q_s, while the number of transactions in UpdateTXs and ValidTXs is denoted as Q_{tx}. ℓ_{Input}, ℓ_{txid}, ℓ_{SIG}, ℓ_{root}, and ℓ_{hash} denote the length of an input, transaction ID, signature, root value, and hash value, respectively.

Table 3
Comparison of SSHC with classic 2PC.

System	Total BFT Calls	Total Messages for a Shard
Classic 2PC	$(2s+1)n$	$ns \cdot (\ell_{\mathsf{Input}} + \ell_{\mathsf{txid}} + \ell_{\mathsf{SIG}})$
SSHC	$\frac{sn}{q_s} + \frac{(s+1)n}{Q_{tx}}$	$(m-1) \cdot (\frac{ns}{m}(\ell_{\mathsf{Input}} + \ell_{\mathsf{txid}}) + \ell_{\mathsf{SIG}} + \ell_{\mathsf{root}} + \log_2^m \ell_{\mathsf{hash}})$

The "Total BFT Calls" denotes the number of calling BFT algorithms to process n transactions. In classic 2PC methods, e.g., Chainspace [13], the number of BFT calls is $2k + 1$ for a single transaction (k in the prepare phase, $k+1$ in the commit phase). In SSHC, the number of BFT calls in every shard is reduced by a factor of q_s and Q_{tx} in the prepare phase and commit phase, respectively. q_s and Q_{tx} are usually on the order of hundreds or even thousands. Therefore, SSHC greatly reduces the number of BFT calls.

The "Total Messages for a Shard" represents the total proof messages that a shard needs to generate, which could be calculated by multiplying the number and the length of the messages. In classic 2PC methods, each input shard should generate a proof for every input, so the number is ns. A proof contains an input address, its related transaction ID, and $2f+1$ signatures (we use 1 single multi-signature for unification) from the BFT algorithm. In SSHC, there are $m-1$ availability certificates for a shard, where each certificate $ac(S_c) = (\langle rt \rangle, \mathsf{leaf}(S_c), \mathsf{hashpath}(S_c))$ includes ns/m inputs and transaction ID in $\mathsf{leaf}(S_c)$, and the number of hash values in $\mathsf{hashpath}(S_c)$ is $\log_2^m$. Note that the first two items, i.e., Input and txid, are basically the same for SSHC and classic 2PC. However, classic 2PC requires ns SIG, while SSHC only needs $(m-1)(\mathsf{SIG}+\mathsf{root}+\log_2^m \text{ hash})$. The number of transactions n (e.g., 1000) processed at one time is usually much larger than the number of shards m (a constant, e.g., 32), so SSHC also reduces the message complexity among shards.

We give the security definition of ESC consisting of restrained executions and properties to be a secure ESC.

Definition 6 (Restrained executions for ESC)**.** *The definition is identical to Definition 2.*

Definition 7 (Secure ESC)**.** *Let T_{plbft} be a polynomially-bounded function in n, ρ, δ, τ, k and Q. Let the pair $(\mathcal{A}, \mathcal{Z})$ be $(n, \rho, \Delta, \tau, Q)$-restrained w.r.t.* ESC. *Then we say* ESC *is secure if and only if the following properties are satisfied with a probability of $1 - u(k)$ where $u(k)$ is a negligible function.*

- *Instant termination:*
  - *Any honest committee member $i \in C_e^c$ must have output chain_e^c that contains a $root_{\mathsf{utxo}_c}$ in the block $\langle B_\ell \rangle$ to $\mathcal{Z}$ by time $T_{stop} + T_{plbft}$.*
  - *Any honest node $i \in S_c$ must have output chain_e^c that contains a $root_{\mathsf{utxo}_c}$ in the block $\langle B_\ell \rangle$ to $\mathcal{Z}$ by time $T_{stop} + T_{plbft} + \delta$.*
- *Consistency:*
  - *Common prefix in shard: For any $S_c, c \in [1, m]$ and any two honest nodes $i, j \in S_c$, node i outputs chain to $\mathcal{Z}$ at time t, and node j outputs chain' to $\mathcal{Z}$ at time t', it holds that either $\mathsf{chain} \prec \mathsf{chain}'$ or $\mathsf{chain}' \prec \mathsf{chain}$.*
  - *Termination agreement in shard. For any two honest nodes i, j belonging to the same S_c, node i outputs chain_e^c that contains a $root_{\mathsf{utxo}_c}$ and node j outputs $\mathsf{chain}_e^{c\prime}$ that contains a $root_{\mathsf{utxo}_c}$ to $\mathcal{Z}$, it holds that $\mathsf{chain}_e^c = \mathsf{chain}_e^{c\prime}$.*

```
Subprotocol Epoch Sharding Consensus
Inputs: 1) start($C_e^c, v_i^{sig}$); 2) TXs; 3) stop. Outputs: chain$_e^c$ in each round;
On receiving start($C_e^c, v_i^{sig}$) from $\mathcal{Z}$, if $v_i^{sig} \in C_e^c$, then set isMem := true, else isMem := false.
Case isMem = true:
▷ plBFT Normal Operations. Node $P_i$ forks a plBFT virtual node, and sets plBFT.start($C_e^c, v_i^{sig}$).
▷ Transaction Processing.
• On receiving TXs from $\mathcal{Z}$:
  1) Set TXs$_{in}$ := $\varnothing$, TXs$_{cross}$ := $\varnothing$, $root := \perp$, $state := \perp$. For every tx $\in$ TXs, if all inputs and outputs of tx belong to $S_c$, then set
     TXs$_{in}$ := TXs$_{in} \cup$ {tx}; else, set TXs$_{cross}$ := TXs$_{cross} \cup$ {tx};
  2) a) Extract all inputs of TXs into $\{I_1, \cdots, I_q\}$. For every tx in TXs, set tx.$state[I_i] := \perp$ for all inputs;
        For every $i = 1$ to $q$, if $I_i$ belongs to $S_c$:
          If $I_i$ does not exist in utxo$_c$, set $state[($tx$.id, I_i)] := 0$;
          Else if (utxo$_c[I_i] = 1$) $\wedge$ ($state[(\_, I_i)]$ does not exist), set $state[($tx$.id, I_i)] := 1$;
          Else, set $state[($tx$.id, I_i)] := 2$.
     b) Construct a Merkel tree with $q_s$ inputs in $state$ where a leaf node is: leaf($S_{c'}$) := ($S_{c'}$, ($I_1$||tx.$id$||$b_1, \cdots, I_j$||tx.$id$||$b_j$)). For
        every $b$, $b := state[($tx$.id, I)]$. $I_1, \cdots, I_j$ are inputs related to shard $S_{c'}$;
     c) Compute the Merkel tree root $root$, and inputs $root$, $state$ to the plBFT virtual node;
     d) On receiving a committed $\langle root \rangle$ from plBFT, get the $state$ related to $\langle root \rangle$, then:
        i) For every tx $\in$ TXs$_{in}$, if (input sum = output sum) $\wedge$ (for every input $I_i \in$ tx, $state[($tx$.id, I_i)].b_i = 1$), then:
             Set ValidTXs := ValidTXs $\cup$ {tx}; For every $I_i \in$ {tx $\cap$ utxo$_c$}, remove $I_i$ from utxo$_c$;
        ii) For other inputs $I_i$ in $state$ that belongs to cross-shard transactions: // (construct availability certificate)
            A) Set tx.$state[I_i] := state[($tx$.id, I_i)].b_i$ (tx correspond to $I_i$);
            B) If $state[($tx$.id, I_i)].b_i = 1$, then set utxo$_c[I_i] := 2$; // (lock an input)
        iii) For every shard $S_{c'}$ except the current shard $S_c$:
            A) Compute hashpath($S_{c'}$) which contains hash values needed to prove leaf($S_{c'}$) is a valid leaf node;
            B) Construct an availability certificate ac($S_{c'}$) := ($\langle root \rangle$, leaf($S_{c'}$), hashpath($S_{c'}$));
            C) Send $m - 1$ ac to each corresponding shard, i.e., send ac($S_{c'}$) to shard $S_{c'}$.
• On receiving ac($S_c$) := ($\langle root \rangle$, leaf($S_c$), hashpath($S_c$) via diffusion, if ($\langle root \rangle$ is valid) $\wedge$ (leaf($S_c$) is a valid leaf node) where
  $\langle root \rangle$ is valid means the signature can pass the verification, and a valid leaf node means the root value calculated through the
  values of leaf and hashpath is equal to the signed root value, then:
  1) Set AC := AC $\cap$ {ac($S_c$)}; // (AC is a valid availability certificate pool)
  2) Parse leaf($S_c$) as ($S_c$, ($I_1$||tx.$id$||$b_1, \cdots, I_j$||tx.$id$||$b_j$), and get all related tx;
  3) For every tx in leaf($S_c$), set tx.$state[I_j] := b_j$; // (update all related transactions' state according to the received valid ac($S_c$))
  4) If any input state of tx is unknown, does nothing; else:
       If the current shard is an input shard of tx, set UpdateTXs := UpdateTXs $\cap$ {tx};
       If (for every $I_i \in$ tx, tx.$state[I_i] = 1$) $\wedge$ (at least one output belongs to $S_c$): set ValidTXs := ValidTXs $\cup$ {tx};
• If |UpdateTXs| = $Q_{\mathsf{tx}}$ or $T_{\mathsf{tx}}$ time has passed since the last committed UpdateTXs: input UpdateTXs and AC to plBFT;
  On receiving ⟨UpdateTXs⟩ from plBFT, for every tx $\in$ ⟨UpdateTXs⟩:
  1) If for every $I_i \in$ tx, tx.$state[I_i] = 1$, then for every $I_i \in$ {tx $\cap$ utxo$_c$}, remove $I_i$ from utxo$_c$; // (spend an input)
     Else, for every $I_i \in$ {tx $\cap$ utxo$_c$}, if tx.$state[I_i] = 1$, set utxo$_c[I_i] := 1$; // (unlock an input)
  2) Set UpdateTXs := $\perp$;
• If |ValidTXs| = $Q_{\mathsf{tx}}$ or $T_{\mathsf{tx}}$ time has passed since the last block $B$ = head(chain$_e^c$): // (construct a transaction block)
  1. Compute $str$ := H($B$) and $B_\ell$ := ($str$, ValidTXs) where $\ell$ denotes the level at which the current block is located;
  2. Set ValidTXs := $\perp$. Input $B_\ell$ and AC to plBFT, and wait for the committed $\langle B_\ell \rangle$.
• On receiving stop from $\mathcal{Z}$, stop other operations and set $B_\ell := (str, root_{\mathsf{utxo}_c})$. $str$ := H(head(chain$_e^c$)). $root_{\mathsf{utxo}_c}$ is the Merkel tree
  hash root of the current utxo$_c$ where a leaf node is $I_i$||utxo$_c[I_i]$. Input $B_\ell$ to plBFT.
▷ Chain Update. On receiving $\langle B_\ell \rangle$ from plBFT, if $\langle B_\ell \rangle$ contains ValidTXs, then:
  1. Get the ValidTXs from $\langle B_\ell \rangle$. For every tx $\in$ ValidTXs, create outputs in utxo$_c$ that belongs to $S_c$;
  2. Set chain$_e^c$ := chain$_e^c$||$B_\ell$, diffuse $\langle B_\ell \rangle$ among $S_c$, and output chain$_e^c$ to $\mathcal{Z}$.
  Else, i.e., $\langle B_\ell \rangle$ contains $root_{\mathsf{utxo}_c}$, set chain$_e^c$ := chain$_e^c$||$B_\ell$, output chain$_e^c$ to $\mathcal{Z}$, and diffuse $\langle B_\ell \rangle$ among $S_c$.
Case isMem = false: On receiving a valid $\langle B_\ell \rangle$, set chain$_e^c$ := chain$_e^c$||$B_\ell$. Output chain$_e^c$ to $\mathcal{Z}$.
```

Figure 7. Subprotocol epoch sharding consensus (ESC[e]).

- *No conflict between shards: For any two S_c, $S_{c'}$ and any two honest nodes $i \in S_c$, $j \in S_{c'}$, node i outputs* chain *to $\mathcal{Z}$ at time t, and node j outputs* chain$'$ *to $\mathcal{Z}$ at time t'. For any two transactions* tx$_1 \in$ chain *and* tx$_2 \in$ chain$'$ *where* tx$_1 \neq$ tx$_2$*, it holds that* tx$_1$ *and* tx$_2$ *do not conflict with each other, i.e., they do not spend any same input.*

• *Liveness: For any honest node from any shard, if it receives a transaction* tx *at some time $T_{start} \leq t < T_{stop}$ from $\mathcal{Z}$, then at time $t' = t + T_{liveness}$,* tx *must be accepted or rejected. $T_{liveness}$ is the liveness parameter for a transaction to be committed.*

Theorem 2 (ESC is secure)**.** *Assume* plBFT *is secure against $(1 - Q)$-corruption with liveness parameter T_{plbft} for $Q > 2/3$ under restrained execution as per Definition 2. Let the pair $(\mathcal{A}, \mathcal{Z})$ be $(n, \rho, \Delta, \tau, Q)$-restrained w.r.t.* ESC*. Then* ESC *is secure against $(1 - Q)$-corruption as per Definition 7 with liveness parameter $T_{liveness} = T_{plbft} + \delta$.*

The proof is deferred to Section 5.

4.3 Secure Scalable Hybrid Consensus Protocol

SSHC is built on top of the subprotocol ESC and plBFT. SSHC runs instances of ESC in different epochs, allowing new nodes to join through a PoW selection process. The detailed protocol is shown in Figure 8. SSHC extends the

```
Protocol SSHC
Initialization. If node $P_i$ is initialized in the first epoch, $P_i$ sends $(\mathsf{genblock\text{-}req}, sid, P_i)$ to $\mathcal{F}_{init}$, receiving $(\mathsf{genblocks}, sid, \mathsf{chain}_0^1, \cdots, \mathsf{chain}_0^m, \mathsf{chain}_0^R)$. Set $e := 0$, $\mathsf{LOG} := \emptyset$. In epoch $e$, node $P_i$ does the following:
Mining. Node $P_i$ sends $(\mathsf{KeyGen}, sid, P_i)$ to $\mathcal{F}_{\mathrm{SIG}}$ and receives $(\mathsf{VerificationKey}, sid, P_i, v_i^{sig})$ from $\mathcal{F}_{\mathrm{SIG}}$. $P_i$ sets $\mathsf{puzzle} := \perp$.
• On receiving $(\xi_e^c, \sigma_e^c)$ via diffusion: // (use $\xi_e^c$ to confirm committee member of $C_{e+1}^c$)
    Set $m_\xi := \xi_e^{c-1}||e||c$, send $(\mathsf{Verify}, sid, m_\xi, \sigma_e^c, V^R)$ to $\mathcal{F}_{\mathrm{TSIG}}$, and receive $g$. If $g = 1$, set $\mathsf{puzzle} := \xi_e^c$.
• Loop
    If $\mathsf{puzzle} \neq \perp$, call $\mathsf{Mining}(\mathsf{puzzle}, v_i^{sig})$, and get $(\mathsf{nonce}, h_{pow})$.
      If $h_{pow} \neq 0$, send $(h_{pow}, nonce, v_i^{sig})$ to nodes in $C_e^R$ via diffusion.
Participating Nodes Operations.
  On receiving $\langle B_{e+1}^R \rangle$ via diffusion, parse it as $(str, \mathsf{list}_{e+1})$ and verify if it is a valid block. If it is:
  1) Get $\mathsf{list}_{e+1} = \{\langle C_{e+1}^1 \rangle, \cdots, \langle C_{e+1}^m \rangle, \langle C_{e+1}^R \rangle\}$ and set $e := e + 1$; // (enter next epoch)
  2) If an instance $\mathsf{ESC}[e-1]$ exists, set $\mathsf{ESC}[e-1].\mathsf{stop}$.
  3) On receiving $\mathsf{chain}_{e-1}^c$ from ESC, set $\mathsf{LOG}_c := \mathsf{LOG}_c||\mathsf{head}(\mathsf{chain}_{e-1}^c)$. Output $\mathsf{LOG}_c$ to $\mathcal{Z}$.
  4) If $v_i^{sig} \in C_e^R$, go to "Reference Committee Members Operations";
     Else if $v_i^{sig} \in C_e^c$ for some $c \in \{1, \cdots, m\}$, go to "Ordinary Committee Members Operations";
     Else, go to "Non-Committee Members Operations".
▷ Reference Committee Members Operations.
  1) Fork a plBFT virtual node, set $\mathsf{plBFT.start}(C_e^R, v_i^{sig})$, and set $c := 1$.
  2) For $c = 1$ to $(m + 1)$: // (confirm $m + 1$ committees for epoch $e + 1$ sequentially)
    a) On receiving $(h_{pow}, nonce, v_i^{sig})$, if $(h_{pow} = \mathsf{H}(\mathsf{puzzle}, nonce, v_i^{sig})) \wedge (h_{pow} < D)$, set $C_{e+1}^c := C_{e+1}^c \cup \{v_i^{sig}\}$.
    b) If $|C_{e+1}^c| = k$, then input $C_{e+1}^c$ to the plBFT virtual node; // (threshold-vote)
    c) On receiving $\langle C_{e+1}^c \rangle$ from the plBFT virtual node: // (randomness generation)
      i) Set $\mathsf{RandShare} := \varnothing$; $m_\xi := \xi_e^{c-1}||e||c$; // (use last randomness $\xi_e^{c-1}$ as a seed)
      ii) Send $(\mathsf{ThSign}, sid, m_\xi, P_i)$ to $\mathcal{F}_{\mathrm{TSIG}}$ and receive $(\mathsf{ThSignature}, sid, P_i, m_\xi, \sigma_i)$;
      iii) Broadcast $\sigma_i$ among $C_e^R$, and set $\mathsf{RandShare} := \mathsf{RandShare} \cup \{(\sigma_i, v_i^{tsig})\}$; // (own share)
      iv) On receiving $\sigma_j$ from $P_j$, send $(\mathsf{ThVerify}, sid, m_\xi, \sigma_j, v_j^{tsig})$ to $\mathcal{F}_{\mathrm{TSIG}}$ and receive $f$;
          If $f = 1$, then set $\mathsf{RandShare} := \mathsf{RandShare} \cup \{(\sigma_j, v_j^{tsig})\}$; else, ignore the message;
      v) If $|\mathsf{RandShare}| = 2f + 1$, then: // (including its own share)
        A) Send $(\mathsf{ThCombine}, sid, m_\xi, (v_1^{tsig}, \sigma_1), \cdots, (v_{2f+1}^{tsig}, \sigma_{2f+1}))$ to $\mathcal{F}_{\mathrm{TSIG}}$ and receive $(\mathsf{Signature}, sid, m_\xi, \sigma_e^c)$;
        B) Compute $\xi_e^c := \mathsf{H}(\sigma_e^c)$. Set $\mathsf{puzzle} := \xi_e^c$. Broadcast $(\xi_e^c, \sigma_e^c)$ to the whole network.
  3) Set $\mathsf{list}_{e+1} := \{\langle C_{e+1}^1 \rangle, \cdots, \langle C_{e+1}^m \rangle, \langle C_{e+1}^{m+1} \rangle\}$. // (committee members for epoch $e + 1$ are confirmed)
  4) Construct a reference block $B_{e+1}^R := (str, \mathsf{list}_{e+1})$, and input $B_{e+1}^R$ to the plBFT virtual node.
  5) On receiving $\langle B_{e+1}^R \rangle$ from plBFT, diffuse $\langle B_{e+1}^R \rangle$ and set $\mathsf{LOG}_R := \mathsf{LOG}_R||B_{e+1}^R$. Set $\mathsf{plBFT.stop}$. // (epoch $e$ stops)
▷ Ordinary Committee Members Operations.
  Query the committed block from members of $C_{e-1}^c$. If $\langle B_\ell \rangle$ is valid and could be parsed as $(str, root_{\mathsf{utxo}_c})$, then:
  1) Query $\mathsf{utxo}_c$ corresponding to $root_{\mathsf{utxo}_c}$ from members of $C_{e-1}^c$; // (correct state transfer from epoch $e$ to $e + 1$)
  2) Set $\mathsf{ESC}[e].\mathsf{start}(C_e^c, v_i^{sig})$ and $\mathsf{LOG}_c := \mathsf{chain}_0^c||\mathsf{chain}_1^c|| \cdots ||\mathsf{chain}_{e-1}^c$. // (call ESC, isMem = ture)
     On receiving $\mathsf{TXs}_c$ from $\mathcal{Z}$, input $\mathsf{TXs}_c$ to $\mathsf{ESC}[e]$. // (invoke ESC to process transactions)
     On receiving $\mathsf{chain}_e^c$ from ESC, set $\mathsf{LOG}_c := \mathsf{LOG}_c||\mathsf{head}(\mathsf{chain}_e^c)$. Output $\mathsf{LOG}_c$ to $\mathcal{Z}$.
▷ Non-Committee Members Operations.
  Query the committed block from members of $comm_{e-1}^c$, if $\langle B_\ell \rangle$ is valid and could be parsed as $(str, root_{\mathsf{utxo}_c})$, then:
  1) Select a $c$ randomly, set $\mathsf{ESC}[e].\mathsf{start}(C_e^c, v_i^{sig})$ and $\mathsf{LOG}_c := \mathsf{chain}_0^c||\mathsf{chain}_1^c|| \cdots ||\mathsf{chain}_{e-1}^c$.// (isMem = false)
  2) On receiving $\mathsf{chain}_e^c$ from ESC, set $\mathsf{LOG}_c := \mathsf{LOG}_c||\mathsf{head}(\mathsf{chain}_e^c)$. Output $\mathsf{LOG}_c$ to $\mathcal{Z}$.
```

Figure 8. Protocol SSHC.

Ideal Functionality $\mathcal{F}_{init}$

On receiving $(\mathsf{genblock\text{-}req}, sid, P_i)$ from some party P_i, return $(\mathsf{genblocks}, sid, \mathsf{chain}_0^1, \cdots, \mathsf{chain}_0^m, \mathsf{chain}_0^R)$ where $\mathsf{chain}_0^R := (B_0^R)$, $B_0^R := (\perp, C_0^R, \xi_0)$. $(\mathsf{chain}_0^c)_{c=1}^m := (B_0^c)$.

Figure 9. Ideal functionality $\mathcal{F}_{init}$.

fair selection protocol in [24] and uses FSS to select multiple committees sequentially in sharding blockchains.

Ideal Functionality $\mathcal{F}_{init}$. SSHC uses $\mathcal{F}_{init}$ to initiate the protocol. $\mathcal{F}_{init}$ returns the initial reference committee C_0^R where honest majority holds. This could be regarded as a trusted setup, and there are papers dedicated to studying this issue [25]. Besides, $\mathcal{F}_{init}$ returns the genesis block B_0^c, which contains arbitrary values of every shard.

Mining. The Mining algorithm (Algorithm 2) takes as input puzzle for PoW mining and a public key v_i^{sig}. Then, it starts computing hash values using puzzle, nonce and the public key as inputs. D is the current difficulty parameter. Once a required hash value is found, the mining function stops and returns the corresponding hash value and nonce.

Randomness Generation. To generate a randomness used as a PoW puzzle, ESC uses the last randomness ξ_e^{c-1}, an epoch number e, and a committee number c. Then it calls $\mathcal{F}_{\mathrm{TSIG}}$. After $2f + 1$ members sign the message, a threshold signature σ_e^c is obtained. Then ξ_e^c could be calculated by hashing σ_e^c. In epoch e, C_e^R generates $m + 1$ randomness used as puzzles to select $m + 1$ new committees. $\mathcal{A}$ could not just withhold his PoW solutions since they are not arranged in a chain formal, and every time the hash function employs an unpredictable randomness as a fresh puzzle. Detailed

Algorithm 2: Mining

```
input : puzzle, $v_i^{sig}$
output: (nonce, $h_{pow}$)
1 $h_{pow} := 0$; nonce := 1;
2 while nonce $\leq q$ do
3     $h = \mathsf{H}(\text{puzzle}, \text{nonce}, v_i^{sig})$;
4     if $h < D$ then $h_{pow} = h$, Break;
5     nonce = nonce + 1
6 return (nonce, $h_{pow}$);
```

analysis is deferred to Section 5.2.

Theorem 3 (Secure SSHC)**.** *Suppose $(\mathcal{A}, \mathcal{Z})$ to be (n, ρ, Δ, τ)-restrained w.r.t.* SSHC *for some $n \in \mathbb{N}, \Delta > 0$. Let $\rho = 1/3 - \epsilon$ denote the computational power of $\mathcal{A}$, where ϵ denotes the restriction on $\mathcal{A}$. Let $\tau > (2m+2)k/g_0 + (2m+4)(T_{plbft} + \delta)$ be the corruption parameter of $\mathcal{A}$. Then, for any $0 < \epsilon < 1/3$, $k_T \geq 0$ and $k \geq (2+k_T)/\epsilon + 3$,* SSHC *satisfies consistency and liveness as defined in Definition 4 with liveness parameter $T_{liveness}$ where*

$$T_{liveness} = T_{plbft} + \delta = O(k\delta)$$

The above $T_{liveness}$ is for the worst case. SSHC realizes a transaction confirmation time of $O(\delta)$ in the optimistic case. The detailed proof is deferred to Section 5.

5 Security Analysis

In this section, we give the security analysis of plBFT, ESC, and SSHC.

5.1 plBFT Proofs

Note that in the analysis of SSHC in Section 5.2, we prove that $1 - Q < \frac{1}{3}$ holds for plBFT. The condition could be transferred to $k \geq 3f + 1$. We only analyze the case $k = 3f + 1$ since the case $k > 3f + 1$ could be deduced easily.

Definition 8 (A valid CERT)**.** *For a* CERT $= (\mathsf{H}(m_{\mathsf{vote}}), \sigma)$, *we say it is valid if we send* $(\mathsf{Verify}, sid, m_{\mathsf{prop}}, V)$ *to* $\mathcal{F}_{\mathrm{TSIG}}$ *and* $\mathcal{F}_{\mathrm{TSIG}}$ *returns g where $g = 1$, i.e., σ is a valid threshold signature on the message* m_{vote}.

Lemma 1 (One valid CERT in one round)**.** *For the same round r, if there are two valid certificates* CERT_r *and* CERT'_r, *then we have* $\mathsf{CERT}_r = \mathsf{CERT}'_r$.

Proof of Lemma 1. We prove by contradiction. Assume that $\mathsf{CERT}_r \neq \mathsf{CERT}'_r$, since both CERT_r and CERT'_r are valid, there must be at least $2f + 1$ members that vote for both CERT_r and CERT'_r. However, there are $k = 3f+1$ members in a committee. The number of nodes who vote twice is $f+1$, which means that these $f+1$ nodes violate the protocol. This contradicts the fact that there are only f malicious nodes in a committee, and other honest nodes adhere to the protocol's principles. So we have $\mathsf{CERT}_r = \mathsf{CERT}'_r$. □

Next, we give proof of Theorem 1.

Proof of Theorem 1. Agreement. From the condition that node i outputs a committed value $\langle p_{v,r} \rangle$, we know that i must receive a valid CERT_{r+1} in round $r + 2$. For node j who outputs a committed value $\langle p'_{v,r} \rangle$, it could be inferred that j receives a valid CERT'_{r+1} in round $r+2$. By Lemma 1, we get that for the same round $r+2$, two valid certificates are identical, i.e., $\mathsf{CERT}_{r+1} = \mathsf{CERT}'_{r+1}$. Parse the two certificates as $\mathsf{CERT}_{r+1} = (\mathsf{H}(m_{\mathsf{vote}}), \sigma)$ where $m_{\mathsf{vote}} = (\text{“vote”}, v, r + 2, \mathsf{H}(p_{v,r+1}), \mathsf{H}(\mathsf{CERT}_r))$ and $\mathsf{CERT}'_{r+1} = (\mathsf{H}(m'_{\mathsf{vote}}), \sigma')$ where $m'_{\mathsf{vote}} = (\text{“vote”}, v, r + 2, \mathsf{H}(p'_{v,r+1}), \mathsf{H}(\mathsf{CERT}'_r))$. We get that both CERT_r and CERT'_r are valid and $\mathsf{CERT}_r = \mathsf{CERT}'_r$. Similarly, we could parse CERT_r and CERT'_r as $(\mathsf{H}(\text{“vote”}, v, r+1, \mathsf{H}(p_{v,r}), \mathsf{H}(\mathsf{CERT}_{r-1})), \sigma)$ and $(\mathsf{H}(\text{“vote”}, v, r+1, \mathsf{H}(p'_{v,r}), \mathsf{H}(\mathsf{CERT}'_{r-1})), \sigma')$. So we obtain that $p_{v,r} = p'_{v,r}$. So agreement holds for plBFT.

Liveness. We first prove that liveness holds when the leader is honest. Since $k = 3f + 1$ satisfies, there are at least $2f + 1$ honest nodes voting for a valid proposal at the end of round $r - 1$. After δ time, an honest leader collects $2f + 1$ valid votes and constructs CERT_{r-1} using the threshold signature. Then he forms $m_{\mathsf{prop}} := (\text{“propose”}, v, r, \mathsf{CERT}_{r-1}, p_{v,r})$ where $p_{v,r} = \mathsf{val}$ and broadcasts it to all nodes. At this time, the round number is set to $r + 1$ for L_v, and $p_{v,r+1}$ is set to $\perp$. δ time later, nodes that receive m_{prop} verify CERT_{r-1}. Since CERT_{r-1} consists of a valid threshold signature, it will pass the verification. Then honest nodes consider a state change and vote to m_{prop} if $\mathsf{IsPValid}(p_{v,r})$ returns 1. IsPValid is a prediction related to transaction processing, block generation, or committee confirmation, which is analyzed in ESC and SSHC.

After another δ time, L_v receives $2f + 1$ valid votes for m_{prop}. From the time L_v broadcasts m_{prop} to the time he receives $2f + 1$ valid votes, if the leader gets a val' from $\mathcal{Z}$, he sets $p_{v,r+1} := \mathsf{val}'$; else $p_{v,r+1}$ remains to be $\perp$. L_v constructs $m'_{\mathsf{prop}} := (\text{“propose”}, v, r+1, \mathsf{CERT}_r, p_{v,r+1})$ and broadcasts it. Similarly, he sets the round number to $r + 2$ and $p_{v,r+2} := \perp$. Whether there is an input from $\mathcal{Z}$ or not, plBFT still makes progress. On the contrary, if L_v does not propose $\perp$ with CERT_r when $\mathcal{Z}$ does not input any value to plBFT, the previous proposals $p_{v,r-1}$ and $p_{v,r-2}$ cannot be processed in time, so the liveness of plBFT will be affected to a certain extent.

δ time later, honest nodes receiving m'_{prop} consider a state change where $p_{v,r}$ is set to $p_{v,r}.state :=$ prepared and vote for the proposal. Another δ time later, the honest leader receives $2f+1$ valid votes for m'_{prop}. Then he constructs a new proposal $m''_{\mathsf{prop}} := (\text{“propose”}, v, r + 2, \mathsf{CERT}_{r+1}, p_{v,r+2})$ and broadcasts it. δ time later, honest nodes receiving the message trigger a state change where $p_{v,r}$ is set to $p_{v,r}.state :=$ committed and output the committed value $\langle p_{v,r} \rangle$ to $\mathcal{Z}$. Therefore, val gets committed after at most 6δ time when the leader is honest. The optimistic liveness parameter is $T_{plbft} = O(\delta)$.

When the leader is malicious or offline, plBFT calls view-change to achieve liveness. Two conditions could trigger a view-change, timeout or view-change-blame.

View-change-timeout: Suppose that in some round $r - 1$, an honest node votes after receiving a valid proposal, then he steps into round r, sets a timer t_v, and starts to countdown. When the countdown ends while there is still no message from L_v, an honest node composes $m_{\mathsf{vc}} := \text{“(view-change-timeout”}, v+1, p_{v,r'}, \mathsf{CERT}_{r'})$ and sends it to the new leader L_{v+1} of $v + 1$.

View-change-blame: In normal operations of plBFT, if a leader proposes an invalid CERT (invalid hash pointer or threshold signature), or an invalid proposal (determined by IsPValid), an honest member sets $vcb_{v,r}$ to be ture, sets $\mathsf{blame}_{v,r}$ to be $(m_{\mathsf{prop}}, \sigma_L)$ as a malicious behavior proof of the leader, constructs $m_{\mathsf{vc}} := \text{“(view-change-blame”}, v + 1, \mathsf{blame}_{v,r}, p_{v,r'}, \mathsf{CERT}_{r'})$, and broadcasts it. Note that an honest member that receives a m_{vc} message verifies the validity of m_{vc}. If it is valid, the honest member constructs a view-change message and broadcasts it. This is to prevent the malicious leader from sending invalid information to only some honest nodes and to ensure that a new leader could receive enough view-change messages.

A leader is determined by a round-robin method. Honest nodes send view-change messages to L_{v+1}, so L_{v+1} can collect $2f+1$ valid m_{vc} and construct a new-view certificate $\mathsf{VC}_{v+1} = \{(m_{\mathsf{vc}}, \sigma_{\mathsf{vc}})\}_i$. This is used to prove that a new leader has the right to replace the previous one. Then L_{v+1} generates a new-view message $m_{\mathsf{nv}} = (\text{“new-view”}, v+1, 0, p_{v,r^*}, \mathsf{CERT}_{r*}, \mathsf{VC}_{v+1}, p_{v+1,0})$ where r^* is the highest prepared round number in VC_{v+1}. Honest nodes that receive m_{nv} update local states according to $p_{v,r^*}, \mathsf{CERT}_{r*}$. Since CERT_{r*} is a certificate consisting of a valid threshold signature, honest nodes set $p_{v,r^*}.state := \mathsf{prepared}$ and $p_{v,r^*-1}.state := \mathsf{committed}$. Then every honest node processes as in the normal case, treats the new-view message as a proposal, and votes to L_{v+1}. □

5.2 Scalable Secure Hybrid Consensus Proofs

We first prove that a randomness generated in every epoch for every committee is secure. Then we prove that the mining process and the member lists confirmation are secure under certain security parameters. At last, the correct environments for ESC and pIBFT are proved.

5.2.1 Analysis of the Randomness Generation Process

We first prove that the randomness is publicly verifiable, unpredictable, unbiasable, and robust.

Lemma 2 (Randomness is publicly verifiable, unpredictable, unbiasable and robust)**.** *Suppose $(\mathcal{A}, \mathcal{Z})$ to be (n, ρ, δ, τ)-restrained w.r.t.* SSHC. *Let $\rho = \frac{1}{3} - \epsilon$ denote the computational power of $\mathcal{A}$, where ϵ denotes the restriction on $\mathcal{A}$. Then for any $n \in \mathbb{N}, \delta > 0, 0 < \epsilon < \frac{1}{3}$, there exists some negligible function $\mu(k)$ such that the following properties hold with probability $1 - \mu(k)$ for $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A}, \mathcal{Z}, k)$:*

Let ξ_e^c denote the randomness generated in epoch e for committee c, for any $e, c \in \mathbb{N}$ and $1 \le c \le m+1$, ξ_e^c is:

- *Publicly verifiable. Any node could verify the correctness of a randomness using a function* $\mathsf{Verify}(\xi, \pi)$ *where π is the proof.* $\mathsf{Verify}(\xi, \pi)$ *returns 1 if ξ and π are valid.*
- *Unpredictable. Let t denote the time that (ξ_e^c, π_e^c) is first output by an honest member of C_e^R where* $\mathsf{Verify}(\xi_e^c, \pi_e^c) = 1$*, and t' denote the time that $\mathcal{A}$ outputs $(\xi_e^{c\prime}, \pi_e^{c\prime})$ where $t' < t - \delta$, then we have: $Pr[\xi_e^{c\prime} = \xi_e^c] < \mu(k)$.*
- *Unbiasable. Assume there are two cases: In the first one, $\mathcal{A}$ is removed from the protocol, and honest members of C_e^R output ξ_e^c; In the second one, $\mathcal{A}$ joins the protocol during the epoch randomness generation, and honest members of C_e^R output $\xi_e^{c\prime}$. Then: $Pr[\xi_e^{c\prime} \ne \xi_e^c] < \mu(k)$.*
- *Robust. Suppose an honest node i receives a committed $\langle C_{e+1}^c \rangle$ at some time t, then by time $t + \delta$, all honest nodes must have received the randomness ξ_e^c, i.e., the adversary could not prevent the generation of randomness.*

Proof of Lemma 2. We prove it under two cases: $e = 0$, i.e., the initial epoch; $e \ge 1$, i.e., a general epoch. For brevity, when we say a randomness is secure, we mean that it is publicly verifiable, unpredictable, unbiasable, and robust.

For $e = 0$, the initial randomness ξ_0 and reference committee C_0^R are embedded in $\mathcal{F}_{init}$. There are more than 2/3 fraction of honest nodes in C_0^R. Nodes receive ξ_0 after interacting with $\mathcal{F}_{init}$ and start mining. After k PoW solutions are found and sent to C_0^R, a new member list C_1^1 is confirmed through pIBFT. Then C_0^R initiates a threshold signature on the message $\xi_0||e||c$ where $e = 0, c = 1$. After a member collects $2f+1$ valid signature shares, he could reconstruct the signature σ_0^1, then uses the hash function to get $\xi_0^1 = \mathsf{H}(\sigma_0^1)$. (ξ_0^1, σ_0^1) is broadcast, and nodes use the new randomness ξ_0^1 as a puzzle to mine. c is set to 2. Again, C_1^2 is confirmed by C_e^R, and ξ_0^2 is generated. The above process continues to loop until C_1^{m+1} is confirmed, and $(\xi_0^{m+1}, \sigma_0^{m+1})$ is generated. Since c equals $m+1$, the confirmation of B_1^R is triggered. After B_1^R is confirmed, the protocol enters into epoch 1. Honest majority of C_0^R ensures the correct execution of pIBFT and threshold signature. Next, we show that for every $1 \le c \le m+1$, ξ_0^c is secure.

- *Publicly verifiable*: ξ_0^c is broadcast with σ_0^c that serves as a proof, so $\mathsf{Verify}(\xi, \pi)$ operates as follows: on receiving (ξ_0^c, σ_0^c) via diffusion, P_i sets $m_\xi := \xi_0^{c-1}||0||c$, sends $(\mathsf{Verify}, sid, m_\xi, \sigma_0^c, V^R)$ to $\mathcal{F}_{\mathrm{TSIG}}$, and receives $(\mathsf{Verified}, sid, m_\xi, g)$. If $g = 1$ and $\mathsf{H}(\sigma_0^c) = \xi_0^c$, then ξ_0^c is valid, and $\mathsf{Verify}(\xi, \pi)$ returns 1. Else returns 0.
- *Unpredictable*: It could be reduced to the *unforgeability* property of $\mathcal{F}_{\mathrm{TSIG}}$. Let t denote the time that (ξ_0^c, σ_0^c) is first output by some honest member of the reference committee where $\mathsf{Verify}(\xi_0^c, \sigma_0^c) = 1$, while $\mathcal{A}$ outputs $(\xi_0^{c\prime}, \sigma_0^{c\prime})$ at some time $t' < t - \delta$. t is the time $2f+1$ valid signature shares are collected, and $\mathsf{ThCombine}()$ is first called to generate σ_0^c. Let t^* denote the time that $2f+1$ valid shares are first generated, so we have $t^* \ge t - \delta$. Hence, at any time $t' < t - \delta$, there are less than $2f+1$ valid shares generated. $\mathcal{A}$ cannot call $\mathsf{ThCombine}()$ to generate a valid signature, which means $\sigma_0^{c\prime}$ is not generated legally. By the unforgeability of $\mathcal{F}_{\mathrm{TSIG}}$, $\sigma_0^{c\prime}$ will not pass $\mathcal{F}_{\mathrm{TSIG}}$'s verification of an honest node.
- *Unbiasable*: It could be reduced to the *uniqueness* property of $\mathcal{F}_{\mathrm{TSIG}}$. To generate ξ_0^c, members in C_0^R need to sign on $m_\xi := \xi_0^{c-1}||0||c$. After $2f+1$ valid signature shares are generated and collected by a member, he could reconstruct the σ_0^c using $\mathsf{ThCombine}()$. Due to the uniqueness property of $\mathcal{F}_{\mathrm{TSIG}}$, any set of $2f+1$ valid signature shares leads to the same signature σ_0^c. The randomness $\xi_0^c = \mathsf{H}(\sigma_0^c)$ is unique, and $\mathcal{A}$ cannot influence the result of ξ_0^c.
- *Robust*: It follows the *robustness* property of $\mathcal{F}_{\mathrm{TSIG}}$. Since honest majority holds in C_0^R, there are at least $2f+1$ honest nodes. The threshold parameter $2f$ and robustness property of $\mathcal{F}_{\mathrm{TSIG}}$ ensures that honest nodes will generate σ_0^c. So ξ_0^c is sure to be generated. About the time cost, honest members of the reference committee broadcast signature shares, so it takes δ time for an honest node to collect $2f+1$ valid shares and generate a randomness.

For the general case $e \ge 1$, we assume ξ_{e-1}^c is secure for every $1 \le c \le m+1$, and we show that ξ_e^c is secure for every $1 \le c \le m+1$. Since ξ_0^c is secure as proved above, then all the randomness ξ_e^c is secure. By the assumption that ξ_{e-1}^c is secure for every $1 \le c \le m+1$, honest majority holds for every committee C_e^c until the end of epoch e according to Lemma 9 and Lemma 10. Honest majority of C_e^R ensures the correct generation of threshold signature and running of pIBFT. Besides, at the beginning of epoch e, nodes mine using ξ_{e-1}^{m+1} as the puzzle. The rest of the process is identical to that of epoch 0. So ξ_e^c is secure for $e \ge 1$. □

5.2.2 *Analysis of the Mining Process*

We extend the security analysis in [24]. We first prove the mining speed has a lower and upper bound, so the time to mine k PoW solutions is bounded. Then we show that the mining process is fair, i.e., honest nodes controlling $1-\rho$ fraction of the computational power occupy at least $(1-\omega)(1-\rho)$ fraction of the k nodes received by an honest reference committee member. p denotes the probability that one node finds a valid PoW solution in a single round.

Definition 9 (Lower and upper bound of speed to find PoW)**.** *Let $t_s \geq 0$ denote a starting time to find PoW, and let $t_0 > 0, t_1 > 0$ denote some time steps. Let k_{t_0} and k_{t_1} denote the PoW solution number received by honest members in the reference committee at t_s+t_0 and t_s+t_1. $g_0 > 0, g_1 > 0$ denote the speed to find PoW solutions. If there exist some negligible functions $\mu_0(k)$ and $\mu_1(k)$ such that for some $k > 0, t_0 \geq \frac{k}{g_0}, t_1 \leq \frac{k}{g_1}$, the following conditions are satisfied*

$$\Pr[\mathsf{min}(k_{t_0}) \geq k] \geq 1-\mu_0(k), \Pr[\mathsf{max}(k_{t_1}) \leq k] \geq 1-\mu_1(k)$$

where $\mathsf{min}(\cdot)$ and $\mathsf{max}(\cdot)$ return the minimum and maximum value, respectively. Then we have:

g_0 and g_1 are the lower and upper bound of speed to find PoW solutions, respectively.

k_{t_0} and k_{t_1} received by honest reference committee members might differ due to the network latency. So we use the function $\mathsf{min}(\cdot)$ and $\mathsf{max}(\cdot)$ to get the minimum of k_{t_0} and maximum of k_{t_1}, respectively.

Lemma 3 (Lower bound of speed to find PoW)**.** *Suppose $(\mathcal{A}, \mathcal{Z})$ to be (n, ρ, Δ)-restrained w.r.t. SSHC for some $n \in \mathbb{N}, \Delta > 0$. Then for any $0 \leq \omega < 1, k \geq 3/\omega$, there exists some negligible function $\mu_0(k)$ such that the following property holds with probability $1-\mu_0(k)$ for $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A}, \mathcal{Z}, k)$:*

Let $\alpha = (1-\rho)np$ denote the expected PoW solutions number mined by honest nodes in a single round, then $g_0 = (1-\omega)\alpha$.

Proof of Lemma 3. To analyze the mining process, we assume that $n\Delta p < 1$ [22], [26]. This condition should be satisfied in most blockchains systems that use PoW to select nodes to guarantee system security. The value of p is related to n and Δ and determines the difficulty parameter D.

By Lemma 2, ξ_e^c is robust and public verifiable, so every node could get a valid (ξ_e^c, σ_e^c) and set puzzle as ξ_e^c. Besides, since ξ_e^c is unpredictable and unbiasable, $\mathcal{A}$ cannot predict the value of ξ_e^c or affect the distribution of ξ_e^c. However, $\mathcal{A}$ controls the network message transmission and could delay messages between honest nodes, so he gets ξ_e Δ time earlier than honest nodes. Furthermore, $\mathcal{A}$ might delay PoW solutions for Δ time submitted by honest nodes to a reference committee. Let t denote the time to mine PoW solutions, then the available time for honest nodes to mine is $t-2\Delta$. Since $\mathcal{A}$ can only delay Δ time, the PoW solution uploaded by honest nodes before time $t-\Delta$ must have been received by the reference committee.

For the lower bound, we assume that $\mathcal{A}$ prevents the finding of PoW solutions. $\mathcal{A}$ delays honest nodes' messages for Δ time and finds no PoW solution. Honest nodes find all solutions. Let N_{pow} denote the solution number mined during time t. The number of honest nodes is $(1-\rho)n$, and honest nodes' total time to compute PoW is $t-2\Delta$. This could be seen as $(1-\rho)n(t-2\Delta)$ independent binary random variables. The value for each variable is 1 with probability p and 0 with probability $1-p$. N_{pow} equals the sum of these $(1-\rho)n(t-2\Delta)$ independent binary random variables, so the expectation of N_{pow} is

$$E[N_{pow}] = (1-\rho)n(t-2\Delta)p = (t-2\Delta)\alpha$$

By the Chernoff bound, for every constant $0 \leq \omega' < 1$

$$\Pr[N_{pow} \leq (1-\omega')(t-2\Delta)\alpha] \leq e^{-\frac{\omega'^2}{2}(t-2\Delta)\alpha} \quad (1)$$

We choose a lower bound of speed to find PoW g_0 to be

$$g_0 = \frac{1}{1+\omega}\alpha \geq (1-\omega)\alpha$$

By our definition, it holds that

$$t \geq \frac{k}{g_0} = \frac{k(1+\omega)}{\alpha} \quad (2)$$

By Equation (1) and (2), during the time period t, except for a negligible probability $e^{-\frac{\omega'^2}{2}(t-2\Delta)\alpha}$, the number of PoW solutions N_{pow} grows at least

$$\begin{aligned} &(1-\omega')(t-2\Delta)\alpha \geq \\ &(1-\omega')\Big(\frac{k(1+\omega)}{\alpha} - 2\Delta\Big)\alpha = \quad (3) \\ &(1-\omega')\Big(1+\frac{\omega}{3}\Big)k + \frac{2\omega}{3}\Big(1-\omega'\Big)k - 2\Delta\alpha(1-\omega') \end{aligned}$$

By choosing a sufficiently small ω', the first term $(1-\omega')(1+\frac{\omega}{3})k$ of the above formula is greater than k. Except for a negligible probability, N_{pow} grows at least

$$\begin{aligned} (1-\omega')\Big(1+\frac{\omega}{3}\Big)k + \frac{2\omega}{3}\Big(1-\omega'\Big)k - 2\Delta\alpha(1-\omega') &> \\ k+(1-\omega')\Big(\frac{2\omega}{3}k - 2\Delta\alpha\Big) &\geq \\ k+(1-\omega')(2-2\Delta np(1-\rho)) &> \\ k+(1-\omega')(2-2(1-\rho)) &> k \end{aligned}$$

where the second inequality is obtained by the condition that $k \geq \frac{3}{\omega}$ and $\alpha = (1-\rho)np$. The third inequality is calculated by the assumption that $n\Delta p < 1$. Thus, the number of PoW solutions N_{pow} grows at least k. So we get

$$\Pr[\mathsf{min}(k_{t_0}) - k_t \geq k] = \Pr[N_{pow} \geq k] \geq 1-\mu_0(k)$$

Therefore, the lower bound of speed to find PoW is $g_0 = (1-\omega)\alpha$ except for a negligible probability $\mu_0(k)$. ☐

Lemma 4 (Upper bound of speed to find PoW)**.** *Suppose $(\mathcal{A}, \mathcal{Z})$ to be (n, ρ, Δ)-restrained w.r.t. SSHC for some $n \in \mathbb{N}, \Delta > 0$. Then for any $\omega \geq 0, k \geq 3/\omega$, there exists some negligible function $\mu_1(k)$ such that the following property holds with probability $1-\mu_1(k)$ for $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A}, \mathcal{Z}, k)$:*

Let g_1 denote the upper bound of speed to find PoW solutions, then we get $g_1 = (1+\omega)np$.

Proof of Lemma 4. About the upper bound, during the time period t, we consider that $\mathcal{A}$ starts mining when he receives ξ_e^c. $\mathcal{A}$ does not delay messages sent by honest nodes. In this situation, the number of nodes is n, and the whole time to mine is t. So this could be seen as tn independent binary random variables. N_{pow} equals the sum of these tn independent binary random variables, so the expectation of N_{pow} is $E[N_{pow}] = tnp$.

By the Chernoff bound, for every constant $\omega' \geq 0$,

$$\Pr[N_{pow} \geq (1+\omega')tnp] \leq e^{-\frac{\omega'^2}{2+\omega'}tnp} \quad (4)$$

We choose an upper bound of speed to find PoW $g_1 = (1+\omega)np$. By our definition, it holds that

$$t \le \frac{k}{g_1} = \frac{k}{(1+\omega)np} \tag{5}$$

By Equation (4) and (5), during time period t, except for a negligible probability $e^{-\frac{\omega'^2}{2+\omega'}tnp}$, N_{pow} grows at most

$$(1+\omega')tnp \le (1+\omega')np\frac{k}{(1+\omega)np} = \frac{1+\omega'}{1+\omega}k \tag{6}$$

For any $\omega \ge 0$, we choose a $\omega' \le \omega$ such that $\frac{1+\omega'}{1+\omega}k \le k$ holds. So we get

$$\Pr[\mathsf{max}(k_{t_1}) - k_t \le k] = \Pr[N_{pow} \le k] \ge 1 - \mu_1(k)$$

Thus, the upper bound of speed to find PoW is $g_1 = (1+\omega)up$ except for a negligible probability $\mu_1(k)$. □

Lemma 5 ($T_{mine\text{-}k}$ is bounded)**.** *Suppose $(\mathcal{A},\mathcal{Z})$ to be (n,ρ,Δ)-restrained w.r.t.* SSHC *for some $n \in \mathbb{N}, \Delta > 0$. Let $T_{mine\text{-}k}$ denote the time to find k PoW solutions. Then for any $0 \le \omega < 1, k \ge 3/\omega$, there exist some negligible function $\mu_0(k)$ and $\mu_1(k)$ such that the following conditions hold for* $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A},\mathcal{Z},k)$*:*

$$\Pr[T_{mine\text{-}k} \le \frac{k}{g_0}] \ge 1-\mu_0(k),\ \Pr[T_{mine\text{-}k} \ge \frac{k}{g_1}] \ge 1-\mu_1(k)$$

Proof of Lemma 5. It is straightforward to prove Lemma 5 by combining Lemma 3 and 4. □

Lemma 6 (Honest majority holds in new nodes through mining)**.** *Suppose $(\mathcal{A},\mathcal{Z})$ to be (n,ρ,Δ)-restrained w.r.t.* SSHC *for some $n \in \mathbb{N}, \Delta > 0$. Let $\rho = 1/3 - \epsilon$ denote the computational power of $\mathcal{A}$, where ϵ denotes the restriction on $\mathcal{A}$. Then for any $0 < \epsilon < 1/3, 0 \le \omega \le 3\epsilon/(2+3\epsilon)$ and $k \ge 3/\omega \ge 2/\epsilon + 3$, there exists some negligible function $\mu_f(k)$ such that the following property holds with probability $1-\mu_f(k)$ for* $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A},\mathcal{Z},k)$*:*

Let Q_m denote the honest fraction of C^c_{e+1} received by an honest member in C^R_e. Then for any $e \in \mathbb{N}, c \in \mathbb{N}$ and $1 \le c \le m+1$, we have $Q_m \ge \frac{2}{3}$.

Proof of Lemma 6. Let N_h denote the number of PoW solutions mined by honest nodes. To ensure the honest majority, we need to analyze the lower bound of the PoW solution number found by honest nodes during a period of time. Since as long as the lower bound (worst case) of PoW solution number exceeds $k \cdot Q_m$, the mining result is secure. Assume that the total time to find k PoW solutions is t. To calculate the lower bound of the PoW solution number found by honest nodes, we need to calculate the lower bound of mining time available for honest nodes. As analyzed in *Proof of Lemma 3*, considering $\mathcal{A}$'s network delay advantage, the available mining time of honest nodes is $t - 2\Delta$. Therefore, during the time period t, by the Chernoff bound, for every constant $0 \le \omega' < 1$, we have

$$\Pr[N_h \le (1-\omega')(t-2\Delta)\alpha] \le e^{-\frac{\omega'^2}{2}(t-2\Delta)\alpha} \tag{7}$$

The lower bound of t corresponds to the upper bound of mining speed g_1. By the time bound condition derived in Lemma 5, for any $\omega'' \ge 0$, we have

$$t \ge \frac{k}{g_1} = \frac{k}{(1+\omega'')np} \tag{8}$$

By Equation (7) and (8), during the time period t, except for a negligible probability $e^{-\frac{\omega'^2}{2}(t-2\Delta)\alpha}$, the PoW solution number found by honest nodes is at least

$$\begin{aligned}(1-\omega')(t-2\Delta)\alpha &\ge (1-\rho)(1-\omega')\Big(\frac{k}{1+\omega''} - 2\Delta np\Big)\\ &> (1-\rho)\Big(\frac{1-\omega'}{1+\omega''}k - 2(1-\omega')\Big)\\ &\ge (1-\rho)\Big(\frac{1-\omega'}{1+\omega''}k - 2\Big)\end{aligned}$$

where the second inequality is obtained by $n\Delta p < 1$. The last inequality is calculated by the condition that $0 \le \omega' < 1$.

By choosing a sufficiently small ω' and ω'', we have

$$(1-\rho)\Big(\frac{1-\omega'}{1+\omega''}k-2\Big) \ge (1-\rho)\Big(\frac{1-\omega'}{1+\omega''}\Big)(k-2.5) \ge (1-\rho)(k-3) \tag{9}$$

For any constant $0 \le \omega < 1$, we choose a k that satisfies $k \ge \frac{3}{\omega}$, then we get $(1-\omega)k \le k-3$. So it holds that

$$(1-\rho)(k-3) \ge (1-\omega)(1-\rho)k$$

Thus, the PoW solution number found by honest nodes in k solutions is at least $(1-\omega)(1-\rho)k$. So we get that for any $\rho > 0, 0 \le \omega < 1$ and $k \ge \frac{3}{\omega}$, except for a negligible probability $\mu_f(k) = e^{-\frac{\omega^2}{2}(1-\rho)k}$, the condition $Q_m \ge (1-\omega)(1-\rho)$ is satisfied. Let $Q_m \ge \frac{2}{3}$ and $\rho = \frac{1}{3} - \epsilon$, we get

$$\frac{2}{3} \le (1-\omega)(1-\rho) = (1-\omega)(\frac{2}{3}+\epsilon) \le Q_m$$

We then get the condition that ω should satisfy is $\omega \le \frac{3\epsilon}{2+3\epsilon}$. We conclude that for any $\epsilon > 0, \rho = \frac{1}{3} - \epsilon$, by choosing a sufficiently small ω where $\omega \le 3\epsilon/(2+3\epsilon)$ and a large enough k where $k \ge 2/\epsilon + 3$, $Q_m \ge \frac{2}{3}$ holds except for a negligible probability of $e^{-\frac{\omega^2}{2}(1-\rho)k}$. □

5.2.3 Analysis of the New Member Lists Confirmation

Next, we analyze the process of confirming a new member list. A list is proposed by a leader. If a leader is malicious, he could censor some honest nodes on the list. C_{e+1} and C'_{e+1} denote two member lists proposed by a leader and held by an honest node in C^R_e. A malicious leader removes a certain number of honest nodes on C_{e+1} and adds the same number of malicious nodes later than the former honest nodes to find PoW solutions [24].

In this case, if honest members in C^R_e vote without checking the validity of C_{e+1}, then the safety property will be ruined. On the contrary, if an honest member only votes when C_{e+1} equals C'_{e+1} hold by himself, then the liveness property will be broken with a high probability since the list held by different honest members may have some differences due to the network latency. So a proper threshold k_T is chosen for the differences between C_{e+1} and C'_{e+1} [24]. We add the threshold-vote rule to IsPValid.

Lemma 7 (Honest majority holds in every confirmed new member list)**.** *Suppose $(\mathcal{A},\mathcal{Z})$ to be (n,ρ,Δ)-restrained w.r.t.* SSHC *for some $n \in \mathbb{N}, \Delta > 0$. Let $\rho = 1/3 - \epsilon$ denote the computational power of $\mathcal{A}$, where ϵ denotes the restriction on $\mathcal{A}$. Then for any $0 < \epsilon < 1/3, k_T \ge 0, 0 \le \omega \le 3\epsilon/(2+3\epsilon)$ and $k \ge (3 + k_T/(1-\rho))/\omega \ge (2+k_T)/\epsilon + 3$, there exists some negligible function $\mu_f(k)$ such that the following property holds with probability $1-\mu_f(k)$ for* $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A},\mathcal{Z},k)$*:*

Let Q_f denote the honest fraction in $\langle C^c_{e+1}\rangle$ committed by plBFT *under the threshold-vote rule. Then for any $e \in \mathbb{N}, c \in \mathbb{N}$ and $1 \le c \le m+1$, we have $Q_f \ge \frac{2}{3}$.*

Proof of Lemma 7. By Equation (9), during time period t, except for a negligible probability $e^{-\frac{\omega'^2}{2}(t-2\delta)\alpha}$, the PoW solution number found by honest nodes is at least $(1-\rho)(k-3)$. Since we adopt the threshold-vote rule, for a confirmed list $\langle C^c_{e+1}\rangle$, the worst case for the number of PoW solutions that belong to honest nodes is $(1-\rho)(k-3) - k_T$. By the assumption that $k \ge \frac{1}{\omega}(3 + \frac{k_T}{1-\rho})$, we have

$$k-3-\frac{k_T}{1-\rho} \ge k-k\omega \Rightarrow (1-\rho)(k-3)-k_T \ge (1-\rho)(1-\omega)k$$

So for any $\rho > 0, 0 \le \omega < 1$ and $k \ge \frac{1}{\omega}(3 + \frac{k_T}{1-\rho})$, the following condition is satisfied except for a negligible probability $\mu_f(k) = e^{-\frac{\omega^2}{2}(1-\rho)k}$

$$Q_f \ge (1-\omega)(1-\rho) \tag{10}$$

Let $Q_f \ge \frac{2}{3}$ and $\rho = \frac{1}{3} - \epsilon$, we have

$$\frac{2}{3} \le (1-\omega)(1-\rho) = (1-\omega)(\frac{2}{3}+\epsilon) \le Q_f$$

The condition that ω should satisfy is $\omega \le \frac{3\epsilon}{2+3\epsilon}$. Then we have

$$k \ge \frac{1}{\omega}(3+\frac{k_T}{1-\rho}) \ge \frac{2+3\epsilon}{3\epsilon}(3+\frac{k_T}{\frac{2}{3}+\epsilon}) = \frac{2+k_T}{\epsilon}+3 \tag{11}$$

By Equation (10) and (11), we conclude that for any $\epsilon > 0, \rho = \frac{1}{3} - \epsilon$, by choosing a sufficiently small ω where $\omega \le \frac{3\epsilon}{2+3\epsilon}$ and a large enough $k \ge \frac{2+k_T}{\epsilon}+3$, $Q_f \ge \frac{2}{3}$ holds except for a negligible probability of $e^{-\frac{\omega^2}{2}(1-\rho)k}$.

Therefore, in every epoch e, the fraction of honest nodes is greater or equal to $\frac{2}{3}$ in every new member lists $C^1_{e+1}, \cdots, C^{m+1}_{e+1}$ confirmed by C^R_e. □

5.2.4 Correct Environments for ESC and plBFT Executions

Lemma 8 ($T_{initial}$, T_{epoch} is bounded)**.** *Suppose $(\mathcal{A}, \mathcal{Z})$ to be (n, ρ, Δ)-restrained w.r.t.* SSHC *for some $n \in \mathbb{N}, \Delta > 0$. Let $\rho = 1/3-\epsilon$ denote the computational power of $\mathcal{A}$, where ϵ denotes the restriction on $\mathcal{A}$. Then for any $0 < \epsilon < 1/3, k_T \ge 0$ and $k \ge (2+k_T)/\epsilon + 3$, there exists some negligible function $\mu_f(k)$ such that the following property holds with probability $1 - \mu_f(k)$ for* $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A}, \mathcal{Z}, k)$*:*

Let $T_{initial}$ denote the time to initialize the protocol SSHC*, and let T_{epoch} denote the time of an epoch, then we get $T_{initial} = T_{epoch}$, and their value range is*

$$[\frac{(m+1)k}{g_1}+(m+2)(T_{plbft}+\delta), \frac{(m+1)k}{g_0}+(m+2)(T_{plbft}+\delta)] \tag{12}$$

Proof of Lemma 8. $T_{initial}$ denotes the time to initiate SSHC, i.e., epoch 0. At first, every node takes ξ_0 as a PoW puzzle to mine. Once a solution is found, the node sends it to C^R_0. After receiving k solutions that takes about $T_{mine\text{-}k}$ time, C^R_0 runs plBFT to reach an agreement on C^1_1, which takes about T_{plbft} time. Then C^R_0 generates ξ^1_0 which takes δ time. Nodes use ξ^1_0 as a puzzle to mine. Since there are $m+1$ committees to be confirmed, the above mining, plBFT, and randomness generation process repeat $m+1$ times. Finally, C^R_0 runs plBFT to reach consensus on B^R_1 that contains list_1 of epoch 1 and broadcast B^R_1. After another $T_{plbft} + \delta$ time, each node enters into epoch 1 and operates normally. So the total time taken by the initial epoch is:

$$T_{initial} = (m+1)T_{mine\text{-}k} + (m+2)(T_{plbft} + \delta)$$

By Lemma 5, $T_{initial}$ is bounded by Equation 12.

T_{epoch} denotes the time that an epoch e takes where $e \ge 1$. The only difference from $T_{initial}$ is that nodes use ξ^{m+1}_{e-1} of epoch $e-1$ as a puzzle at the beginning of epoch e. This does not influence the time spent by an epoch. C^R_e runs plBFT to reach an agreement on C^1_{e+1}, taking about T_{plbft} time. Then they generate ξ^1_e, which takes about δ time. The above process repeats $m+1$ times, then C^R_e runs plBFT to produce B^R_{e+1} containing list_{e+1} and broadcasts B^R_{e+1}, taking about $T_{plbft} + \delta$ time. So we get $T_{initial} = T_{epoch}$. □

Lemma 9 (Honest majority holds in every committee during each epoch)**.** *Suppose $(\mathcal{A}, \mathcal{Z})$ to be (n, ρ, Δ, τ)-restrained w.r.t.* SSHC *for some $n \in \mathbb{N}, \Delta > 0$. Let $\rho = 1/3 - \epsilon$ denote the computational power of $\mathcal{A}$, where ϵ denotes the restriction on $\mathcal{A}$. Let $\tau > (2m+2)k/g_0 + (2m+4)(T_{plbft}+\delta)$ be the corruption parameter of $\mathcal{A}$. Then for any $0 < \epsilon < 1/3, k_T \ge 0$ and $k \ge (2+k_T)/\epsilon + 3$, there exists some negligible function $\mu_f(k)$ such that the following property holds with probability $1 - \mu_f(k)$ for* $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A}, \mathcal{Z}, k)$*:*

Let Q denote the honest fraction of C^c_e. Then for any $e \in \mathbb{N}, c \in \mathbb{N}$ and $1 \le c \le m+1$, $Q \ge \frac{2}{3}$ holds during each epoch.

Proof of Lemma 9. We analyze it from two aspects: honest majority holds in a new confirmed list_{e+1}; from the time that committees are confirmed to the time that committees are updated, $\mathcal{A}$ could not complete its corruption attack.

By Lemma 7, we know that in epoch e, the honest fraction $Q_f \ge 2/3$ holds in every $C^1_{e+1}, \cdots, C^m_{e+1}, C^R_{e+1}$. So $Q \ge 2/3$ holds in every C^c_{e+1} at the beginning of epoch $e+1$.

Next, we analyze the corruption attack of $\mathcal{A}$. The committee members in epoch $e+1$ are determined in epoch e through the mining process. Since the submission of a PoW solution is public, $\mathcal{A}$ could launch a corruption attack against a target node once a valid PoW solution is uploaded in epoch e. So the condition that τ should satisfy is $\tau > 2T_{epoch}$. By Lemma 8, we know that T_{epoch} is bounded. To prevent honest nodes from being corrupted by $\mathcal{A}$, τ should be larger than the upper limit of T_{epoch}. So we get

$$\tau > \frac{(2m+2)k}{g_0} + (2m+4)(T_{plbft} + \delta)$$

When the above condition is satisfied, $Q \ge \frac{2}{3}$ holds in every C^c_e during epoch operation. □

Lemma 10 (The environments of ESC and plBFT are restrained)**.** *Suppose $(\mathcal{A}, \mathcal{Z})$ to be (n, ρ, Δ, τ)-restrained w.r.t.* SSHC *for some $n \in \mathbb{N}, \Delta > 0$. Let $\rho = 1/3 - \epsilon$ denote the computational power of $\mathcal{A}$, where ϵ denotes the restriction on $\mathcal{A}$. Let $\tau > (2m+2)k/g_0 + (2m+4)(T_{plbft}+\delta)$ be the corruption parameter of $\mathcal{A}$. Then for any $0 < \epsilon < 1/3, k_T \ge 0$ and $k \ge (2+k_T)/\epsilon + 3$, there exists some negligible function $\mu_f(k)$ such that the following property holds with probability $1 - \mu_f(k)$ for* $\mathsf{EXEC}_{\mathsf{SSHC}}(\mathcal{A}, \mathcal{Z}, k)$*:*

Let $(\mathcal{A}, \mathcal{Z})[\mathsf{ESC}[e]]$ and $(\mathcal{A}, \mathcal{Z})[\mathsf{plBFT}[e]]$ be the adversary and environment pairs that ESC*[e] and* plBFT *interface with. Then for any $e \in \mathbb{N}$, $(\mathcal{A}, \mathcal{Z})[\mathsf{ESC}[e]]$ is $(n, \rho, \delta, \tau, Q)$-restrained w.r.t.* ESC *as per Definition 6. $(\mathcal{A}, \mathcal{Z})[\mathsf{plBFT}[e]]$ is $(n, \rho, \delta, \tau, Q)$-restrained w.r.t.* plBFT *as per Definition 2.*

Proof of Lemma 10. The environment of ESC is restrained, meaning that it satisfies *committee correctness*, *close start and stop*, and *honest majority*.

Committee correctness. In SSHC, ESC is run by ordinary committees. A node that receives list_{e+1} fist sets $e = e + 1$ and then verifies if his public key v_i^{sig} is on the list. If $v_i^{sig} \in C_e^c$, he sets $\mathsf{ESC}[e].\mathsf{start}(C_e^c, v_i^{sig})$ where isMem is true. Else, he selects a c randomly, then sets $\mathsf{ESC}[e].\mathsf{start}(C_e^c, v_i^{sig})$ where isMem is false. For a shard, committee agreement follows from the definition of SSHC. For two different shards, after a node's public key is contained in a confirmed list, he could generate a new public key to mine. So there is no public key that belongs to two or more than two committees, i.e., no conflict among shards holds for $\mathsf{ESC}[e]$.

Close start and stop. At the end of epoch $e - 1$, after receiving a valid reference block $\langle B_e^R \rangle$, the members of C_e^c query the last block $\langle B_\ell \rangle$ from members of C_{e-1}^c. If $\langle B_\ell \rangle$ is valid with a $root_{\mathsf{utxo}_c}$, then members of C_e^c query utxo_c from C_{e-1}^c and trigger the start command, i.e., sets $\mathsf{ESC}[e].\mathsf{start}(C_e^c, -)$. Since the actual network delay is δ, the difference of time for any two nodes to get the last committed block $\langle B_\ell \rangle$ and utxo_c is δ. So close start holds for $\mathsf{ESC}[e]$. Similarly, the stop command is triggered by a reference block $\langle B_{e+1}^R \rangle$, so close stop holds for $\mathsf{ESC}[e]$.

Honest majority. By Lemma 9, honest majority holds.

For the environment of pIBFT, we consider the following two cases. Case 1: $\mathsf{pIBFT}[e]$ is run by $\mathsf{ESC}[e]$. In $\mathsf{ESC}[e]$, a node receives $\mathsf{start}(C_e^c, v_i^{sig})$ from $\mathcal{Z}[\mathsf{ESC}[e]]$. Then if $v_i^{sig} \in C_e^c$, the node sets $\mathsf{pIBFT}.\mathsf{start}(C_e^c, v_i^{sig})$. By the above proof of ESC, $(\mathcal{A}, \mathcal{Z})[\mathsf{ESC}[e]]$ satisfies honest majority, committee correctness, close start and stop. $\mathcal{Z}[\mathsf{pIBFT}[e]]$ inputs the identical C_e^c in start to $\mathsf{pIBFT}[e]$ as $\mathcal{Z}[\mathsf{ESC}[e]]$ inputs to $\mathsf{ESC}[e]$. So $(\mathcal{A}, \mathcal{Z})[\mathsf{pIBFT}[e]]$ satisfies honest majority, committee correctness, close start and stop. Case 2: $\mathsf{pIBFT}[e]$ is run directly by SSHC. In SSHC, pIBFT is called by reference committee members. A node that receives a new list_e verifies if his public key v_i^{sig} is on the list. If $v_i^{sig} \in C_e^R$, he sets $\mathsf{pIBFT}.\mathsf{start}(C_e^R, v_i^{sig})$. The proof for committee correctness and close start and stop for pIBFT is the same as that of ESC. Furthermore, by Lemma 9, the honest node fraction exceeds $2/3$ in C_e^R, i.e., honest majority holds for pIBFT. □

5.3 Epoch Sharding Consensus Proofs

To prove that ESC is secure, we describe the concrete process of an honest member. For any honest member in some C_e^c, it receives TXs and classifies it into TXs_{in} and TXs_{cross}. For all inputs of TXs, ESC parses them into $\{I_1, \cdots, I_q\}$. For every input I_i, it queries the availability state of I_i from utxo_c. It obtains $state$, constructs a Merkel tree using each shard as a leaf node, gets its root value $root$, and inputs $root, state$ to pIBFT. We prove that there will be a $\langle root \rangle$ committed by pIBFT in Lemma 11. Moreover, only one block is to be committed at any level ℓ in Lemma 12.

Lemma 11 (A $root$ are to be committed)**.** *After* TXs *is input to C_e^c, there must be a corresponding committed $\langle root \rangle$ output by all honest committee members of C_e^c.*

Proof of Lemma 11. After a TXs is input to a committee C_e^c, an honest member inputs $root, state$ to pIBFT by the above analysis. Then in pIBFT, a leader constructs a proposal message $m_{\mathsf{prop}} := (\text{“propose”}, v, r, \mathsf{CERT}_{r-1}, p_{v,r})$ where $p_{v,r} = root$. The leader broadcasts $(m_{\mathsf{prop}}, \sigma_L)$ with the corresponding $state$. As an honest member of C_e^c, after receiving $(m_{\mathsf{prop}}, \sigma_L)$, he calls $\mathsf{IsPValid}(p_{v,r})$ to verify the validity of $p_{v,r}$. The type of P is root, so an honest member gets the $state$ related to $root$ received from the leader. Then he queries the local UTXO to verify if every input state in $state$ is valid and $root$ is a valid root value. If the leader is honest, then $\mathsf{IsPValid}(p_{v,r})$ returns 1 for every honest member since their UTXO pools are consistent with each other (this is explained later). So every honest member votes for the proposal $p_{v,r} = root$. By the agreement and liveness properties of pIBFT, a committed $\langle root \rangle$ will be output by honest members. If a malicious leader proposes an invalid $root$, IsPValid will return 0, and honest members will set $\mathsf{blame}_{v,r}$ to be $(m_{\mathsf{prop}}, \sigma_L)$ and $vcb_{v,r}$ to be ture. A view-change is then launched to blame the malicious leader and change a new leader until an honest leader proposes a valid $root$. In this way, a committed $\langle root \rangle$ is sure to be output by all honest members of C_e^c.

Next, we prove that $\mathsf{IsPValid}(p_{v,r})$ returns the same value for honest members. The essence is that each input state change in UTXO is committed by pIBFT. In the beginning, an input shard composes all the inputs into $state$ and $root$ and commits $root$ through pIBFT. When $\langle root \rangle$ is committed, all honest members update their UTXO according to $\langle root \rangle$ and its corresponding $state$. By the definition of ESC, valid intra-shard transactions are written to ValidTXs, and their inputs are removed from utxo_c. For cross-shard transactions, if $state[(\mathsf{tx}.id, I_i)].b_i = 1$, then $\mathsf{utxo}_c[I_i]$ is set to 2 to lock the input. After receiving ac sent from other shards, if all input states for a transaction tx are known, then tx is put into UpdateTXs, and the corresponding ac is put into the AC pool. UpdateTXs and AC are input to pIBFT, where AC is to prove the input states in UpdateTXs. In pIBFT, IsPValid judges the validity of ac in AC and verifies each input state in UpdateTXs (the input state may be $0, 1, 2$). Then an honest member votes to UpdateTXs. The committed $\langle \mathsf{UpdateTXs} \rangle$ serves as a reference for honest members to update their utxo_c. For a transaction in UpdateTXs, if for every $I_i \in \mathsf{tx}$, $\mathsf{tx}.state[I_i] = 1$, then tx is accepted as valid, and the inputs of tx is removed from utxo_c. If tx is invalid, i.e., at least one input state of tx is not 1, then tx is rejected, and the previously locked input should be unlocked. For an output shard, valid transactions are added to ValidTXs, and a transaction block B_ℓ will be committed by pIBFT. New unspent transaction outputs will be created in utxo_c according to the committed $\langle B_\ell \rangle$. □

Lemma 12 (Only one block is committed at any level ℓ)**.** *A block B_ℓ proposed by an honest leader of any C_e^c is sure to be committed. The committed $\langle B_\ell \rangle$ will be output by all honest members of C_e^c. Besides, only one block B_ℓ is committed at any level ℓ.*

Proof of Lemma 12. By Lemma 11, honest members obtain a committed $\langle root \rangle$ and for every shard, ESC constructs an availability certificate $\mathsf{ac}(S_{c'}) = (\langle root \rangle, \mathsf{leaf}(S_{c'}), \mathsf{hashpath}(S_{c'}))$. Then honest members transfer each $\mathsf{ac}(S_{c'})$ to the corresponding input or output shard's members. For a receiving shard, an honest member receives a valid $\mathsf{ac}(S_c)$ from another shard, parses it, and obtains $\mathsf{leaf}(S_c)$. For each tx in $\mathsf{leaf}(S_c)$, if the input states of tx are all known and available, tx is regarded as a valid transaction. If the current shard is an output shard of tx, ESC adds tx into ValidTXs. For TXs_{in}, ESC adds valid intra-shard transactions into ValidTXs.

When a leader of C_e^c inputs a new transaction block B_ℓ and related AC to plBFT, ValidTXs is included in B_ℓ. Every honest member could verify if every transaction in ValidTXs is valid, according to the ac in AC. If every input of every transaction is available and B_ℓ contains a valid str, honest members vote for B_ℓ in plBFT. If a malicious leader puts some invalid transactions into the block or tries to censor some transactions, an honest member could broadcast a view-change-blame message. By the agreement and liveness properties of plBFT, there will be a valid B_ℓ committed, and $\langle B_\ell \rangle$ will be output by honest members of C_e^c. Besides, the agreement property of plBFT ensures that there is only one block that is committed in any level ℓ. □

Next, we give the proof of Theorem 2.

Proof of Theorem 2. By Definition 7, ESC is secure means that instant termination, consistency, and liveness hold for $\mathsf{EXEC}_{\mathsf{ESC}}(\mathcal{A}, \mathcal{Z})$ except with negligible probability.

Instant termination. For an honest member of C_e^c in epoch e, after the committee list for epoch $e+1$ is confirmed, SSHC inputs stop to ESC. This is when T_{stop} starts. Then ESC constructs a $B_\ell := (str, root_{\mathsf{utxo}_c})$ and inputs it to plBFT. $root_{\mathsf{utxo}_c}$ serves as a "stop" signal to plBFT and ESC. After T_{plbft} time, B_ℓ containing a $root_{\mathsf{utxo}_c}$ is committed. Then honest members set $\mathsf{chain}_e^c := \mathsf{chain}_e^c || B_\ell$ and output chain_e^c to SSHC. So by time $T_{stop} + T_{plbft}$, honest members must have output chain_e^c containing a $root_{\mathsf{utxo}_c}$ in $\langle B_\ell \rangle$ to $\mathcal{Z}$.

For honest nodes of S_c that are not committee members, they monitor the network. Honest members broadcast $\langle B_\ell \rangle$ that contains a $root_{\mathsf{utxo}_c}$ among S_c in $T_{stop} + T_{plbft}$ time. After at most δ time, honest nodes in S_c receive $\langle B_\ell \rangle$. They update their local chain after verification and output chain_e^c that contains a $root_{\mathsf{utxo}_c}$ to $\mathcal{Z}$.

Consistency. We prove it from the following three aspects.

1. Common prefix in shard. For members in S_c, by Lemma 12, every honest member of C_e^c outputs the same committed $\langle B_\ell \rangle$ for any level ℓ. By the definition of ESC, honest members update their local chain to $\mathsf{chain}_e^c || B_\ell$. In this way, for any level ℓ, honest members of C_e^c add the same block to their local chain. For two time points t and t', the views of two honest nodes may be different due to network delays, yet this difference may only happen when the two local chains are at different levels. The blocks are identical at the same level. So common prefix holds for honest members. For non-committee members in S_c, they collect messages broadcast by members. After a block is committed, honest members broadcast $\langle B_\ell \rangle$. Note that $\langle B_\ell \rangle$ contains a valid threshold signature, which could be seen as a commit certificate. So non-committee members update their local chain as soon as they receive a $\langle B_\ell \rangle$ and complete the signature verification. In this way, common prefix holds for non-committee members.

2. Termination agreement in shard. For honest members, ESC inputs $B_\ell := (str, root_{\mathsf{utxo}_c})$ to plBFT as a "stop" signal. plBFT commits the B_ℓ, then outputs $\langle B_\ell \rangle$ and stops other operations. After this, every honest member updates their local chain to $\mathsf{chain}_e^c || B_\ell$ and outputs chain_e^c. By the analysis of common prefix, chain_e^c output by honest members must be identical at this time. For non-committee members, after receiving $\langle B_\ell \rangle$ that contains a $root_{\mathsf{utxo}_c}$, they could verify its validity, update their local chain, and output chain_e^c that contains a $root_{\mathsf{utxo}_c}$. At this time, chain_e^c must be the same as that of honest members since every committed block could be verified by a commit certificate. So termination agreement holds in the same shard.

3. No conflict between shards. For any two different transactions tx and tx′, they do not conflict with each other means they can not spend the same input. For a tx to be confirmed, all its inputs, which might be managed by different shards, must be available. By Lemma 11, every related shard forms a $\langle root \rangle$ and sends a specified ac to other related shards. In every related shard, after receiving all related ac, tx could be seen as valid if all its input states are 1. Different shards have an identical view on tx since every ac contains a threshold signature that serves as a proof of validity. $\mathcal{A}$ cannot forge any ac by the unforgeability property of $\mathcal{F}_{\mathrm{TSIG}}$.

If tx is rejected in some S_c, which means one or more inputs are in state 2 or 0, all relative inputs that are locked from state 1 to 2 should be unlocked in the corresponding input shards. In this case, every related shard regards tx as invalid since the ac they received contain identical input states for the same tx. The availability certificates ensure a consistent view of all shards on each cross-shard transaction.

For any input, it could only be spent once due to the lock mechanism. Suppose that an input I_i is managed by S_c. At first, its state is 1 for available, then a tx using I_i as one of its inputs is uploaded. After $root$ is committed, I_i is locked to state 2. It is impossible for any other tx′ to spend I_i since it is locked, and the state query returns 2. Besides, to defend against the replay attack [27], each input is bonded with its corresponding transaction ID. This information is included in ac. So for two transactions tx and tx′ that spend the same input I_i, $\mathcal{A}$ cannot use the ac of tx to complete the commitment of tx′

Liveness. Liveness of ESC is guaranteed by plBFT's liveness property and the transaction processing mechanism. After a TXs containing tx is submitted, its input shards verify the input availability and construct a ac. The liveness property of plBFT ensures that after T_{plbft} time, ac must be output by all input shards. Then ac is transferred to every relative shard. After δ time, every shard must have received all ac related to tx. At this time, tx could already be regarded as valid or invalid. To realize responsiveness, output shards could return the proof to the client that submits tx without waiting for tx to be written into the block. So a transaction is determined to be accepted or rejected after $T_{plbft} + \delta$. Liveness holds for ESC with a parameter $T_{liveness} = T_{plbft} + \delta$, and in an optimistic case, $T_{liveness} = 6\delta + \delta = O(\delta)$. The transaction confirmation time is only related to the actual network latency, so it satisfies responsiveness. □

5.4 Safety for SSHC

Based on Section 5.2 and 5.3, we prove Theorem 3.

Proof of Theorem 3. Consistency. By Lemma 10, the environment of plBFT is restrained. By Theorem 1, we get that plBFT is secure, i.e., satisfies agreement and liveness. By Lemma 10, it is proved that SSHC provides a restrained environment for ESC for any $e \in \mathbb{N}$. So by Theorem 2, $\mathsf{ESC}[e]$ is secure. We prove consistency for SSHC out of the instant termination, consistency property of ESC.

1. Common prefix in shard. We prove this from two periods: the execution period of an epoch and the switching period between epochs.

In epoch e, honest nodes first set $\mathsf{LOG}_c := \mathsf{chain}_0^c || \cdots || \mathsf{chain}_{e-1}^c$. Then they input a start command to $\mathsf{ESC}[e]$. On receiving TXs from $\mathcal{Z}$, SSHC inputs TXs to $\mathsf{ESC}[e]$. By the liveness property of $\mathsf{ESC}[e]$, $\mathsf{ESC}[e]$ outputs chain_e^c to SSHC, where chain_e^c includes a new committed block. Then SSHC sets $\mathsf{LOG}_c := \mathsf{LOG}_c || \mathsf{head}(\mathsf{chain}_e^c)$ and outputs LOG_c to $\mathcal{Z}$. Assume there are two honest nodes i and j in the same S_c, and they output chain_e^c and $\mathsf{chain}_e^{c\prime}$, respectively. By the common prefix property of $\mathsf{ESC}[e]$, it holds that $\mathsf{chain}_e^c \prec \mathsf{chain}_e^{c\prime}$ or $\mathsf{chain}_e^{c\prime} \prec \mathsf{chain}_e^c$. Suppose the logs output by node i and j are LOG_c and $\mathsf{LOG}_c{}'$, respectively. Then their SSHC update the local log using $\mathsf{LOG}_c := \mathsf{LOG}_c || \mathsf{head}(\mathsf{chain}_e^c)$ and $\mathsf{LOG}_c{}' := \mathsf{LOG}_c{}' || \mathsf{head}(\mathsf{chain}_e^{c\prime})$. So it holds that $\mathsf{LOG}_c \prec \mathsf{LOG}_c{}'$ or $\mathsf{LOG}_c{}' \prec \mathsf{LOG}_c$.

Next, we analyze the switching period between epochs. At the end of epoch e, C_e^R broadcasts $\langle B_{e+1}^R \rangle$. Members of C_e^c input a "stop" command to $\mathsf{ESC}[e]$. Then ESC outputs chain_e^c that contains a $root_{\mathsf{utxo}_c}$ in $\langle B \rangle$, where $\langle B \rangle$ is the last block of epoch e. Assume that there are two honest nodes i and j of the same S_c, while their $\mathsf{ESC}[e]$ output chain_e^c and $\mathsf{chain}_e^{c\prime}$ that both contain $root_{\mathsf{utxo}_c}$, respectively. Then their SSHC update logs based on $\mathsf{LOG}_c := \mathsf{LOG}_c || \mathsf{head}(\mathsf{chain}_e^c)$ and $\mathsf{LOG}_c{}' := \mathsf{LOG}_c{}' || \mathsf{head}(\mathsf{chain}_e^{c\prime})$. By the termination agreement property of $\mathsf{ESC}[e]$, we get that $\mathsf{chain}_e^c = \mathsf{chain}_e^{c\prime}$. So at the end of an epoch, $\mathsf{LOG}_c = \mathsf{LOG}_c{}'$.

The new committee members of C_{e+1}^c query committed blocks from members of C_e^c. If $\langle B \rangle$ is valid with a $root_{\mathsf{utxo}_c}$, they query utxo_c and historical blockchain from C_e^c and set $\mathsf{ESC}[e+1]$.start. Then they set $\mathsf{LOG}_c := \mathsf{chain}_0^c || \cdots || \mathsf{chain}_e^c$. Since every committed block in any chain contains a threshold signature as a certificate, every honest member of C_{e+1}^c gets an identical LOG_c. Besides, $root_{\mathsf{utxo}_c}$ in the last committed block of epoch e could serve as a proof for the UTXO pool utxo_c. So honest members of C_{e+1}^c get the same valid utxo_c. The subsequent process and analysis are identical to epoch e. So common prefix in shard holds for SSHC.

2. No conflict between shards. For any two honest nodes i and j, we assume they belong to S_c and $S_{c'}$ where $c \neq c'$, respectively. They maintain their logs as $\mathsf{LOG}_c = \mathsf{LOG}_c || \mathsf{head}(\mathsf{chain}_e^c)$ and $\mathsf{LOG}_{c'} = \mathsf{LOG}_{c'} || \mathsf{head}(\mathsf{chain}_e^{c'})$, where chain_e^c and $\mathsf{chain}_e^{c'}$ are output from their $\mathsf{ESC}[e]$. By the consistency property of ESC, we know that any two transactions $\mathsf{tx}_1 \in \mathsf{chain}_e^c$ and $\mathsf{tx}_2 \in \mathsf{chain}_e^{c'}$ do not conflict with each other, i.e., do not spend the same input. So it is straightforward to obtain that any two transactions $\mathsf{tx}_1 \in \mathsf{LOG}_c$ and $\mathsf{tx}_2 \in \mathsf{LOG}_{c'}$ do not conflict with each other.

Liveness. In epoch $e \geq 1$, SSHC transfers TXs to $\mathsf{ESC}[e]$ and gets committed transactions in return. By the liveness property of ESC, liveness satisfies for SSHC with the same $T_{liveness}$ parameter $T_{plbft} + \delta$.

For every epoch, T_{epoch} is bounded by Lemma 8. After $(m+1)k$ new members are confirmed, B_{e+1}^R is produced, and $\mathsf{ESC}[e]$ stops, then $\mathsf{ESC}[e+1]$ starts. New members in C_{e+1}^c could download the UTXO and LOG_c and finish key generation of $\mathcal{F}_{\mathrm{TSIG}}$ before the start of epoch $e+1$ since they are confirmed earlier. So when a new epoch starts, members are ready for processing transactions. Therefore, epoch switch, i.e., committee reconfiguration, will not influence the liveness property of SSHC. □

6 Related Work

Luu et al. propose ELASTICO [6] to improve blockchain performance using sharding technology, yet it could not process cross-shard transactions. Al-Bassam et al. propose Chainspace [13], using a sharded Byzantine atomic commit protocol to process cross-shard transactions. However, to commit a cross-shard transaction, BFT needs to be called many times in relative shards. Omniledger [28] designs a client-driven cross-shard transaction processing mechanism where a client asks the input shard leader to get a proof-of-acceptance for a transaction input. However, a leader might behave maliciously by providing proof-of-rejection or refusing to respond. Besides, letting a client act as a coordinator increases its workloads. RapidChain [10] uses a synchronous BFT algorithm [29] for intra-shard consensus, splitting a cross-shard transactions into multiple ones. This enlarges the total number of transactions, resulting in an increased processing and storage burden on the entire network. Dang et al. [30] adopt random numbers generated by trusted hardware Intel SGX to assign nodes into different shards. Monoxide [31] proposes atomic transfer, which uses a relay transaction to process cross-shard transactions. Besides, chu-ko-nu mining is designed to defend against the 1% attack where an adversary focuses his computational power on a single shard. Parallel Chains [32] designs a PoS-based ledger using Ouroboros Praos [33] as a basic chain.

7 Conclusion

This paper proposes a secure and scalable hybrid consensus protocol realizing decentralization, security, and scalability. SSHC realizes optimal resistance against an arbitrary adversary with a computational power $\rho = 1/3 - \epsilon$. The committee reconfiguration mechanism is innovative and applies for sharding blockchains, and transaction confirmation is responsive.

Acknowledgments

This paper is supported by the National Key R&D Program of China through project 2017YFB0802500 and 2017YFB1400702, the National Cryptography Development Fund through project MMJJ20170106, the Natural Science Foundation of China through projects 61972019, 61932011, 61772538, 61672083, 61532021, 61472429, 91646203, 61402029, 61972017, 61972018, 61932014, 61972310, the foundation of Science and Technology on Information Assurance Laboratory through project 61421120305162112006, the China Scholarship Council through project 201906020015.

References

[1] S. Nakamoto, et al., Bitcoin: A peer-to-peer electronic cash system, `https://bitcoin.org/bitcoin.pdf` (2008).

[2] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, G. Danezis, Sok: Consensus in the age of blockchains, in: AFT 2019, 2019, pp. 183–198.

[3] D. E. Kouicem, Y. Imine, A. Bouabdallah, H. Lakhlef, A decentralized blockchain-based trust management protocol for the internet of things, IEEE Transactions on Dependable and Secure Computing (2020) 1–1.

[4] M. Conti, S. K. E, C. Lal, S. Ruj, A survey on security and privacy issues of bitcoin, IEEE Communications Surveys and Tutorials 20 (4) (2018) 3416–3452.

[5] J. C. Corbett, et al., Spanner: Google's globally distributed database, ACM Trans. Comput. Syst. 31 (3) (2013) 8:1–8:22.
[6] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, P. Saxena, A secure sharding protocol for open blockchains, in: ACM SIGSAC 2016, 2016, pp. 17–30.
[7] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, M. H. Rehmani, Applications of blockchains in the internet of things: A comprehensive survey, IEEE Communications Surveys and Tutorials 21 (2) (2019) 1676–1717.
[8] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, B. Ford, Enhancing bitcoin security and performance with strong consistency via collective signing, in: USENIX Security 2016, 2016, pp. 279–296.
[9] I. Eyal, E. G. Sirer, Majority is not enough: bitcoin mining is vulnerable, Commun. ACM 61 (7) (2018) 95–102.
[10] M. Zamani, M. Movahedi, M. Raykova, Rapidchain: Scaling blockchain via full sharding, in: CCS 2018, 2018, pp. 931–948.
[11] M. Castro, B. Liskov, Practical byzantine fault tolerance, in: OSDI 1999, 1999, pp. 173–186.
[12] J. N. Gray, Notes on data base operating systems, in: Operating Systems, Springer, 1978, pp. 393–481.
[13] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, G. Danezis, Chainspace: A sharded smart contracts platform, in: NDSS 2018, 2018, pp. 18–21.
[14] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, W. J. Knottenbelt, Sok: Communication across distributed ledgers, https://eprint.iacr.org/2019/1128.pdf (2019).
[15] R. Pass, E. Shi, Hybrid consensus: Efficient consensus in the permissionless model, in: DISC 2017, 2017, pp. 39:1–39:16.
[16] G. Avarikioti, E. Kokoris-Kogias, R. Wattenhofer, Divide and scale: Formalization of distributed ledger sharding protocols, https://arxiv.org/pdf/1910.10434.pdf (2019).
[17] R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in: FOCS 2001, 2001, pp. 136–145.
[18] R. Canetti, Y. Dodis, R. Pass, S. Walfish, Universally composable security with global setup, in: CRYPTO 2007, Springer, 2007, pp. 61–85.
[19] A. Boldyreva, Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme, in: PKC 2003, 2003, pp. 31–46.
[20] B. Libert, M. Joye, M. Yung, Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares, Theor. Comput. Sci. 645 (2016) 1–24.
[21] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, I. Abraham, Hotstuff: Bft consensus with linearity and responsiveness, in: PODC 2019, ACM, 2019, pp. 347–356.
[22] J. A. Garay, A. Kiayias, N. Leonardos, The bitcoin backbone protocol: Analysis and applications, in: EUROCRYPT 2015, 2015, pp. 281–310.
[23] Y. Liu, J. Liu, J. Yin, G. Li, H. Yu, Q. Wu, Cross-shard transaction processing in sharding blockchains, in: ICA3PP 2020, 2020, pp. 324–339.
[24] Y. Liu, J. Liu, Z. Zhang, H. Yu, A fair selection protocol for committee-based permissionless blockchains, Computers & Security (2020) 101718.
[25] J. A. Garay, A. Kiayias, N. Leonardos, G. Panagiotakos, Bootstrapping the blockchain, with applications to consensus and fast pki setup, in: PKC 2018, Springer, 2018, pp. 465–495.
[26] R. Pass, L. Seeman, A. Shelat, Analysis of the blockchain protocol in asynchronous networks, in: EUROCRYPT 2017, 2017, pp. 643–673.
[27] A. Sonnino, S. Bano, M. Al-Bassam, G. Danezis, Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers, in: EuroS&P 2020, 2020, pp. 397–406.
[28] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, B. Ford, Omniledger: A secure, scale-out, decentralized ledger via sharding, in: SP 2018, 2018, pp. 583–598.
[29] I. Abraham, S. Devadas, K. Nayak, L. Ren, Brief announcement: Practical synchronous byzantine consensus, in: DISC 2017, 2017, pp. 41:1–41:4.
[30] H. Dang, T. T. A. Dinh, D. Loghin, E. Chang, Q. Lin, B. C. Ooi, Towards scaling blockchain systems via sharding, in: SIGMOD 2019, 2019, pp. 123–140.
[31] J. Wang, H. Wang, Monoxide: Scale out blockchains with asynchronous consensus zones, in: NSDI 2019, 2019, pp. 95–112.
[32] M. Fitzi, P. Gazi, A. Kiayias, A. Russell, Proof-of-stake blockchain protocols with near-optimal throughput, https://eprint.iacr.org/2020/037 (2020).
[33] B. David, P. Gazi, A. Kiayias, A. Russell, Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain, in: EUROCRYPT 2018, 2018, pp. 66–98.

Yizhong Liu Yizhong Liu (Student Member, IEEE) received his B.S. degree in School of Electronic and Information Engineering from Beihang University, Beijing, China, in 2014. He is now pursuing a Ph.D. degree in School of Electronic and Information Engineering, Beihang University, China. His research interests include information security, cryptography, blockchain and smart contracts.

Jianwei Liu Jianwei Liu received his B.S. and M.S. degrees in Electronic and Information from Shandong University, China in 1985 and 1988, respectively. He received his Ph.D. degree in Communication and Electronic System from Xidian University, China in 1998. He is now a Professor in School of Cyber Science and Technology, Beihang University, Beijing, China. His current research interests include wireless communication network, cryptography and information security.

Qianhong Wu Qianhong Wu (Member, IEEE) received the M.S. degree in applied mathematics from Sichuan University, Sichuan, China, in 2001, and the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2004. He is an Associate Research Fellow with the University of Wollongong, Wollongong, Australia, a Professor in School of Cyber Science and Technology, Beihang University, Beijing, China, and a Senior Researcher with the Universitat Rovirai Virgili, Tarragona, Catalonia. He has been a Main Researcher or a Project Holder/Coholder for more than ten Chinese-, Australian-, and Spanish-funded projects. He has authored over 60 publications and served on the program committees of several international conferences on information security and privacy. His research interests include cryptography, information security and privacy, and ad hoc network security.

Hui Yu Hui Yu received the B.S. degree in School of Electronic and Information Engineering from Beihang University, Beijing, China, 2017. He received the M.S. degree in School of Electronic and Information Engineering, Beihang University, Beijing, China, in 2020. His major research interests include blockchain and cryptocurrency.

Yiming Hei Yiming Hei received the B.S. degree from Xidian University, Xian, China, in 2017. He received the M.S. degree in School of Cyber Science and Technology from Beihang University, Beijing, China, in 2020. Now he is pursuing a Ph.D. degree in School of Cyber Science and Technology from Beihang University, Beijing, China. His research interests include applied cryptography and smart contract.

Ziyu Zhou Ziyu zhou received her B.S. Degree in Shenyuan Honor College School from Beihang University, China, 2018. She is now pursuing a Ph.D. degree in School of Cyber Science and Technology from Beihang University, China. Her research interests include cryptography and blockchain.