Proof of Finalization: A Self-Fulfilling Function of Blockchain

Aixian Deng, Qian Ren, Yingjun Wu, Hong Lei, and Bangdao Chen

***Abstract*— Blockchain has been widely used in various industries for providing trustworthy data. On-chain data can be regarded as trusted after it is finalized by blockchain consensus, namely after the data is believed to be immutable. Unfortunately, nodes with poor/isolated network conditions are still susceptible to data spoofing attacks of blockchain view, spawning kinds of severe attacks. For example, a light node newly joining a blockchain network may request the blockchain view from a malicious full node and accept a spoof view, leading to a double spending attack. Besides, a Trusted Execution Environment (TEE), the network stack of which is fully controlled by its host, may be fed spoofed blockchain data as input, undermining the trustworthiness of TEE-based computation by cheating inputs. To resist data spoofing, existing methods rely on a trusted authority to identify trusted data, or timely provide sufficient confirmation blocks for a block *b* to prove the finalization of *b* (since the adversary holding less hash power than the honest blockchain node cannot generate the confirmation blocks timely). These methods either suffer the risks caused by centralized trust base or are only PoW-oriented and high-latency. As promising blockchains including Ethereum migrate to energy-saving consensus, *e.g.*, PoS, designing consensus-agnostic approaches against data spoofing becomes an urgent need of the industries. In this paper, we introduce a Proof of Finalization (PoF) problem for proving the finalization of blockchain to prevent data spoofing attacks of blockchain. We also contrive a novel PoF scheme, which leverages the chain quality property of blockchain to establish a trustworthy committee for proof generation. The scheme is chain-agnostic, non-interactive, non-authority-involved, and with negligible latency. Once blockchain data is finalized, the latency of proof generation in our scheme is only 106 milliseconds. Therefore, our scheme paves the way for any system, *e.g.*, light nodes, cross-chain bridges, and layer-2 systems, to read blockchains with various consensus securely.**

***Index Terms*— Blockchain, proof of finalization, chain quality.**

Manuscript received 30 March 2024; revised 2 July 2024 and 20 August 2024; accepted 21 August 2024. Date of publication 28 August 2024; date of current version 11 September 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB2700600, in part by Hainan Province Science and Technology Special Fund under Grant ZDKJ2020009, in part by the National Natural Science Foundation of China under Grant 62163011, and in part by the Research Startup Fund of Hainan University under Grant KYQD(ZR)-21071. The associate editor coordinating the review of this article and approving it for publication was Prof. Stefano Calzavara. *(Aixian Deng and Qian Ren are co-first authors.) (Corresponding author: Hong Lei.)*

Aixian Deng, Yingjun Wu, and Bangdao Chen are with Oxford-Hainan Blockchain Research Institute, SSC Holding Company Ltd., Laocheng, Chengmai, Hainan 571924, China (e-mail: aixian@oxhainan.org; yingjun@oxhainan.org; bangdao@oxhainan.org).

Qian Ren is with The Blockhouse Technology Ltd., OX2 6XJ Oxford, U.K. (e-mail: qianren1024@gmail).

Hong Lei is with the School of Cyberspace Security (School of Cryptology), Hainan University, Haikou, Hainan 570228, China, and also with Oxford-Hainan Blockchain Research Institute, SSC Holding Company Ltd., Laocheng, Chengmai, Hainan 571924, China (e-mail: leihong@ssc-hn.com).

Digital Object Identifier 10.1109/TIFS.2024.3451355

I. Introduction

BLOCKCHAIN is a distributed system where consensus nodes collectively maintain a tamper-proof ledger that provides trusted data for individuals, enterprises, and so on. In many systems, participants indisputably reach an agreement through blockchain. For example, blockchain is widely employed in cryptocurrency systems (*e.g.*, Bitcoin, Ethereum) to form an undisputed linear sequence of all confirmed transactions, thus solving transaction conflicts among participants. The support for smart contracts further expands the application of blockchain. When a smart contract is executed, its contract code, parameters, and state are immutably recorded on a blockchain, leading to a trusted world state. In conclusion, blockchain serves as a "trusted database".

However, blockchain data is not unconditionally trustworthy. For example, in Bitcoin, the current main chain may be reverted by another forked chain that is longer than it. Then transactions in reverted blocks will become invalid. In practice, data finalized by the consensus mechanism of blockchain is regarded as trusted. For example, in proof-of-work (PoW) consensus protocols, a block is believed to be trustworthy when getting *X-confirmation* on the main chain. *X-confirmation* means that after a block is generated, there are X-1 blocks following it on the main chain. The value of X depends on the difficulty of hash puzzles and the assumed upper bound of hashing power held by malicious nodes. For accuracy, in the latter paragraphs, we will use the adjective "finalized" to qualify a thing to represent its trustworthiness on a blockchain, *e.g.*, finalized data/transaction/block, and the opposite expression is "unfinalized".

Motivation example: Even if a blockchain's consensus mechanism is designed to be secure enough to clearly delineate between finalized and unfinalized data, an adversary can still make victims adopt unfinalized data as finalized to benefit from it. We call these misbehavior data spoofing attacks, which harm the security of numerous services relying on blockchain. For example, in simplified payment verification (SPV), lightweight clients only maintain the block headers of a blockchain and verify transactions by Merkle proofs. To update block headers correctly, a client should satisfy the network assumption that it is always connected to at least one honest node [9]. An adversary may isolate a client from blockchain networks and fully control its blockchain data source (which is usually called eclipse attack [23]), breaking the network assumption of the client. Then the adversary feeds the client fake block headers. As a result, a fake transaction

created by the adversary will pass the client's verification, leading to a further double-spending attack. These attacks raise an essential research problem: Can we prevent a client from data spoofing attacks (accepting unfinalized blockchain data as finalized), even if all its network I/O is controlled by an adversary?

Solutions and Limitations: The common method for preventing data spoofing attacks is leveraging a trusted authority to identify finalized data (*e.g.*, RPC service providers, the blockchain's development team). RPC service providers [1], [3], [6] identify finalized blockchain for users who are confident about their reputation and employ their blockchain access service. Similarly, the Bitcoin node software development team hard-codes the hash value of a certain block b, the checkpoint, into the Bitcoin clients, identifying the finalized blockchain which starts from the genesis block and ends with b. With the checkpoint, disoriented nodes (*e.g.*, nodes isolated from blockchain networks by an adversary) can identify and eventually follow the finalized blockchain ends with b. However, as the client needs to periodically fetch blocks/checkpoints from a trusted authority, this method suffers from single point of failures (SPOF).

Besides introducing a trusted authority, researchers explore another method [13], [15], [16] to resist blockchain data spoofing. Specifically, these works leverage the hashing power advantage of honest blockchain nodes, timely delivering sufficient confirmation blocks for a block as finalization proof. However, these methods are only PoW-oriented. What's more, to ensure that the probabilities of false negatives (*i.e.*, an adversary forges sufficient confirmation blocks successfully and timely) and false positives (*i.e.*, honest blockchain nodes fail to create sufficient confirmation blocks timely) are both less than 0.001, their latency for generating proof is around 50 block intervals (*cf.* Table I). In Bitcoin, 50 block intervals approximately equal 8 hours. Thereby, existing methods without the presence of trusted authority are PoW-oriented and high-latency, hardly fitting practical needs.

Our Work: In this paper, we introduce a Proof of Finalization (PoF) problem which is for proving that data has been published on a blockchain and finalized. We also propose a protocol to solve the problem. The protocol relies on the *chain quality* [21] property of blockchain to organize a committee by miners. The property means that among miners of k or more consecutive blocks on the main chain of a blockchain, the proportion of honest miners is no less than μ. Utilizing the *chain quality* property, the percentage of honest committee members in our protocol is at least μ. The committee is responsible for identifying the finalized blockchain and periodically generating the multi-signature of the latest finalized block as proof. The proof proves the finalization of the blockchain which starts with the genesis block and ends with the signed block. When receiving blockchain data, verifiers only accept data proved by the proof. Thereby, PoF prevents verifiers from accepting unfinalized data as finalized.

Notably, the proof generation is authority-free, chain-agnostic, and low-latency, and the proof verification is non-interactive. Specifically, committee members are randomly selected from miners and rotated, without relying on a trusted authority. The protocol is chain-agnostic since any blockchain satisfying the chain quality property supports our protocol, whilst chain quality is a basic blockchain property that can be extracted from various consensus blockchains (PoW, PoS *etc.*). Committee members can reliably access the latest blockchain views and immediately multi-sign the latest finalized blockchain. Therefore, the latency for proof generation equals an ignorable multi-signature generation time immediately after the latest block is finalized. Namely, the latency is almost minimized. Besides, verifiers independently verify the proof to identify the finalized blockchain from untrusted blockchain views without interaction with other nodes. Thereby, our PoF scheme is admirably suited for protecting any service (especially the isolated/vulnerable nodes such as light nodes of wallets, and TEE nodes of confidential smart contract frameworks [36], [37]) relying on trusted data from blockchain against data spoofing attacks.

Contributions: In summary, this work makes the following contributions:

- We introduce the PoF problem which is for proving that data has been finalized on a blockchain.
- We design a novel PoF protocol, which is chain-agnostic, low-latency, non-authority-involved, and non-interactive, thus becoming the first ready-to-use universal PoF protocol.
- We implement the PoF protocol and evaluate it on both PoW and PoS blockchains.

Organization: We introduce related work in Section II. Section III models blockchain and introduces weighted multi-signature schemes. In Section IV, we define the proof of finalization problem. Section V outlines the PoF protocol. We detail the protocol and its security analysis in Section VI. Section VII contains the evaluation of PoF on proof generation latency and proof verification time. We explore extensions (*e.g.*, feasible reward-punishment mechanism) and applications (*e.g.*, protect TEEs in TEE-blockchain systems) of PoF protocol in Section VIII, and conclude in Section IX.

II. Related Work

In this section, we explore data spoofing attacks and other related attacks, and introduce countermeasures, especially PoF solutions for data spoofing attacks.

Data Spoofing Attacks and Solutions: In data spoofing attacks, the adversary makes victims mistake data unfinalized by a blockchain consensus for finalized, *e.g.*, *eclipse attack* and *race attack* [24], while the blockchain's consensus is still secure. In an eclipse attack, recall that in Section I we exemplify that a lightweight client isolated by an adversary can be cheated by unfinalized data. In a race attack, the adversary usually targets the entity that accepts a transaction as finalized prematurely. For example, assume an exchange requires a transaction to get just $X' < X$ confirmations to accept, where the transaction actually requires *X-confirmation* to be finalized according to the blockchain's consensus mechanism. An adversary may send two conflict transactions A and B on a blockchain at the same time, *e.g.*, for the same coins, A transfers to the exchange, while B to an address held by the adversary. To launch a race attack, the adversary first mines a fork to make A satisfy X'*-confirmation*, spoofing the exchange

TABLE I

THE COMPARISON OF PoF SCHEMES. CHAIN-AGNOSTIC REFERS TO THE ABILITY TO DECOUPLE THE SCHEME FROM A SPECIFIC CONSENSUS. LATENCY INDICATES THE TIME IT TOOK TO PRODUCE THE PROOF OF FINALIZATION. ● IMPLIES THAT THE SCHEME FULLY SATISFIES THE PROPERTY OR RESISTS THE ATTACK, AND ❍ IS THE OPPOSITE OF ●

Proposal	Data spoofing	Chain-agnostic	SPOF	Latency
authority [1], [3], [6]	●	●	❍	authority-decided
Tesseract [13]	●	❍ (PoW)	●	$\approx$ 50 blocks[1]
Ekiden [15]	●	❍ (PoW)	●	$\approx$ 50 blocks[1]
reverse link [38]	●	❍	●	$\approx$ n blocks[2]
PoF	●	●	●	signature time

[1] The latency is calculated from the attacker's hash power (at the time of writing, a maximum of 25% of total power) and multiplicative slowness factor($\delta = 1.5$), ensuring that the probabilities of false negatives and false positives are both less than 0.001 (*cf.* Table 1 in [13] for details). [2] the n refers to the threshold of the required miners, which is usually 30-80 for security in practice.

with the fork. Then, it immediately shifts to the main chain containing B after receiving coins from the exchange. When B becomes finalized, which invalidates A, the adversary gets the coins it wants without paying the exchange. Data spoofing attacks are the problem that this paper aims to resolve.

To guard against data spoofing attacks, there are schemes seeking to protect blockchain users reading finalized data by providing a corresponding proof of finalization. We identify the problem as PoF, and classify existing PoF schemes into two categories: with trusted authority and without.

The first category of PoF schemes relies on a trusted authority, such as existing RPC service providers [1], [3], [6] and the blockchain's development team, to identify finalized data, where users regard the authority's endorsement as finalization proof. However, this way suffers from SPOF.

Without introducing authority, the other category of PoF schemes [13], [15], [16] timely provides X confirmation blocks of a block b as b's finalization proof. However, these works are only PoW-oriented and high-latency. Differently, [38] proposed a new method called *reverse link*. The high-level idea of [38] is to let previous miners independently sign their succeeding blocks to endorse which succeeding fork is the main chain. Specifically, each miner, mined a block b_i, is required to send a transaction to publish its signature of b_{i+m}, a block finalized after b_i where m is fixed. The signature is verified by the unique public key the miner recorded in b_i's block header. This transaction is called a hook transaction and should be included before b_{i+n}, where $n > m$, forming a reverse link $b_i \rightarrow b_{i+m}$. The reverse link indicates an endorsement of the b_i's miner to its unique succeeding fork until b_{i+m}. With reverse links, honest miners can mark out the main chain by recursively showing their endorsement of the main chain. The finalization proof of a block consists of a specific number of endorsements for the fork that contains the block. However, [38] requires consensus modification on block header to include the public key, thus being unsuitable to legacy blockchains, and it requires high latency for proof generation, as shown in Table I.

Inspired by *reverse link*, this paper proposes a novel PoF protocol. We inherited the idea in [38] of having previous miners endorse their only successor. Differently, our PoF protocol is built from the chain quality property without consensus modifications, making it chain-agnostic. We reduce the latency of proof generation to a negligible multi-signature generation time, which is around 100 milliseconds according to our evaluation results (refer to Section VII-A for details). Compared with other PoF schemes which either suffer from the SPOF of the relied trusted authority or are chain-restricted and high-latency, our PoF presents strong universality and high availability. Please refer to Table I for details on comparing the existing PoF schemes and our scheme.

Consensus Attacks and Solutions: In consensus attacks of a blockchain, the adversary aims to undermine blockchain consensus, failing the consensus in finalizing data or invalidating the data previously finalized. To subvert consensus, on the one hand, the adversary may break the security assumption of the blockchain consensus, *e.g.*, controlling $\geq 51\%$ hash power of miners in PoW, corrupting $\geq 1/3$ miners of PBFT. Relevant attacks include the *51% attack*, *Bribery attack* [20], *Sybil attack* [35] against consensus, *etc.* Specifically, the Bribery attack bribes profit-oriented miners to manipulate consensus. The Sybil attack against consensus means that an adversary creates numerous fake identities (or Sybil identities) to make a disproportionately large influence on blockchain consensus. In practice, these attacks are typically avoided by prohibitive attacking costs with meager/negative profits. For example, in Bitcoin, controlling enough hashing power for a *51% attack* costs over $96 billion [2], which far outweighs block rewards. Similarly, Sybil identities can be reduced by the deposit costs required for consensus participation.

On the other hand, the adversary may exploit the security flaws of the blockchain consensus to undermine consensus. For example, in PoS blockchains, mining a block does not cost much like in PoW. Hence, the adversary can launch the long-range attack [40] that forks a PoS blockchain at a quite early position (*e.g.*, the genesis block) by compromising early miners. When the fork is longer than the original main chain, the attack becomes successful. To mitigate the problem, *key evolving signature* (KES) [11] emerged. Concretely, each miner holds a pair of keys (pk, sk_i) to sign blocks it mined, where the public key pk is constant, while the private key sk_i is periodically rotated. sk_i in the i-th period is derived from sk_{i-1} in the previous period. Each miner erases sk_{i-1} when completing the derivation of sk_i. Note that the probability of successfully recovering the erased private keys from sk_i is negligible. Consequently, employing KES prevents the long-range attack, since the private keys used to sign early blocks cannot be reverted [18].

Note that resisting consensus attacks is considered by consensus designers to achieve consensus security, typically realizing the chain growth, chain quality, and common prefix properties, outside the aims and ability of this paper. However, the protocol proposed in this paper can help mitigate some of these attacks. For example, in the Sybil attack against consensus where Sybil miners exist but have not fully compromised the consensus security, our protocol can help to identify the

TABLE II
INTRODUCTION OF THE TRANSACTION FIELDS

transaction tx	
$sender$:	sender address of the transaction
$receiver$:	receiver address of the transaction
$transfer$:	amount of coins to transfer by the transaction
$data$:	optional transaction input data

data spoofing misbehavior of Sybil miners, *e.g.*, misleading other miners to follow their fork. Finally, these Sybil miners can be better identified and punished.

Application attacks and Solutions: Some application attacks, which happen in the application/contract layer of a blockchain, tend to be confused with the data spoofing attacks this paper solved. For example, in Sybil attacks against application layers, the adversary may create multiple Sybil accounts [31] to take away huge airdrops or cause the DDoS attacks [42]. Similar attacks in this form include the *spam attack* [10], [28], [43], in which the adversary sends numerous spam transactions to cause a denial of service. As the transactions led by these application-layer attacks are still legal transactions regarded by a secure consensus, these attacks are orthogonal topics to this paper. However, the orthogonal nature makes it easy to adopt corresponding countermeasures [25], [35] in parallel with the protocol proposed in this paper.

Efficient Query of Blockchain Data: Some works provide efficient query services of blockchain data, *e.g.*, constructing relational blockchain database [32], [39], [49], and encapsulating *authenticated data structure* in blocks [44], [47]. While these works seek to effectively handle users' specific searching requests on blockchain data, our PoF protocol helps users identify the finalized data of a blockchain, therefore, we think that PoF and these works are complementary.

III. PRELIMINARIES

In this section, we model blockchain and describe weighted multi-signature schemes, foreshadowing the following introduction of protocol design.

A. Blockchain

Referring to [17] and [26], we model a blockchain system BC as a chain of blocks finalized by a set of distributed miners following a consensus algorithm cons. Specifically, a BC is identified by a chain ID BC.id_c and it works as follows.

Each user of BC has a pair of keys (pk, sk) and its own address. A user uses its private key sk to sign its transactions, the integrity and authentication of which are verified and identified by its address derived from the public key pk. Table II lists the general information a transaction includes. A transaction is valid if its signature and format are correct.

A BC may have different blockchain forks, each of which is a chain of blocks $(b_0, b_1, \ldots, b_i)$. Each block packs a transaction sequence $\mathbf{tx} := (tx_1, tx_2, \ldots, tx_j)$. Each block also has a height: Let the height of the genesis block b_0 in the blockchain fork be 0, and the heights of the subsequent blocks $(b_1, \ldots, b_i)$ are incremented from 1 onwards. A block b is defined as the tuple $b := (Header, Body)$, and b is identified by the block hash computed from $Header$. $Body$ contains the transaction sequence **tx**, and **tx** encompasses a special coinbase transaction to give a reward to the creator (so-called miner) address of b. $Header$ consists of the hash of b's predecessor block in the blockchain fork, the Merkle tree root of **tx**, and some other elements. For clarity, the height, hash, and miner address of b are denoted by $b.height$, $b.hash$, and $b.miner$ respectively. If every block b_i in a blockchain fork has correct $(Header, Body)$ format, and the $Header$ correctly indicates b_i's predecessor in the blockchain fork, we say the blockchain fork is valid.

Blockchain Consensus: There is a set of nodes, miners, following a consensus algorithm cons to maintain and extend BC. The cons ensures that honest miners refer BC to the same finalized blockchain fork, while each miner may have different blockchain forks. Concretely, the consensus cons introduces a function mine for miners to conduct a mining process. Conforming to [21], cons extends BC in rounds. In each round r, each miner indexed at i respectively collects transactions from other users and packs them into a new block to extend the blockchain fork BC_i it holds. Let $\mathbf{tx}_i$ be a sequence of transactions collected by the miner. The function mine takes BC_i held by the miner and $\mathbf{tx}_i$ as inputs, and outputs BC'_i which is the extended version of BC_i, where $\mathrm{BC}'_i = (\mathrm{BC}_i := (b_0, b_1, \ldots, b_j)) \parallel b_{j+1}$, and the transaction sequence of b_{j+1} is $\mathbf{tx}_i$. After the round of cons, all miners are able to complete the computation of BC.mine respectively and synchronization of their blockchain forks, the set of which is denoted as $\mathbf{BC}_i$. Only one blockchain fork becomes finalized by cons, and the fork will be referred to as BC without ambiguity. Formally, in each round, cons takes $\mathbf{BC}_i$ and BC as inputs and outputs BC′, the updated version of BC. Let $\mathrm{BC}_m \preceq \mathrm{BC}_n$ represents that BC_m is a prefix of BC_n, and it is satisfied that $\mathrm{BC} \preceq \mathrm{BC}' \preceq \mathrm{BC}_i$ and $\mathrm{BC}_i \in \mathbf{BC}_i$.

Blockchain Interfaces: A blockchain BC provides the following BC's functions invoked by users:

- BC.read(m[, n]): The function has two input fields: a required block height m and an optional block height $n > m$. The function returns the block b_m on BC when only taking m as input, or returns the block sequence $(b_m, \ldots, b_n)$ on BC when taking the complete two inputs m and n. Assume b_i is the last block of BC with height i, if $m > i$ or $n > i$, the returned block(s) after b_i would get padding by $\emptyset$.
- BC.send(tx): If the transaction tx is valid, then tx would eventually be included into BC.
- BC.validate($\mathbf{b} := (b_x, b_{x+1}, \ldots, b_{x+y})$): The function takes a block sequence which constitutes a blockchain (fragment) as input and outputs 1 if the blockchain (fragment) is valid, otherwise 0. Note that BC.validate($\mathbf{b}$) $= 1$ only determines the validity of **b**, not the finalization that $\mathbf{b} \subseteq$ BC.

Blockchain Properties: After modeling blockchain, now we introduce three blockchain properties: *common prefix*, *chain growth*, and *chain quality* [21]. The properties have been widely employed as evaluation indicators in various consensus designs, involving PoW [34], PoS [41], PBFT [33], *etc.* The definitions of the three properties are as follows:

- *Common prefix*: The common prefix property with parameter $d \in \mathbb{N}$ states that for any two honest miners respectively

holding blockchain fork BC_1 and BC_2 at round l_1 and l_2 ($l_1 < l_2$), *i.e.*, $\mathrm{BC}_1 \in \mathbf{BC}_i$ at round r_1 and $\mathrm{BC}_2 \in \mathbf{BC}_i$ at round r_2, BC_1 after cutting the d last blocks will be the prefix of BC_2.

• *Chain growth*: The chain growth property with parameter $\tau \in \mathbb{R}$ states that for $l \in \mathbb{N}$ and any honest miner holding blockchain fork $\mathrm{BC}_i \in \mathbf{BC}_i$ at a round, BC_i will expand by at least $\tau \cdot l$ blocks after passing l consecutive rounds.

• *Chain quality*: The chain quality property with parameters $k \in \mathbb{N}$ and $\mu \in \mathbb{R}^+$ states that for any honest miner holding blockchain fork $\mathrm{BC}_i \in \mathbf{BC}_i$ at a round, in any consecutive k blocks of BC_i, the proportion of blocks created by honest miners is at least μ.

B. Weighted Multi-Signature

A weighted multi-signature is a joint signature generated by signature group members $\mathbf{M}$ which consist of t members ($\mathbf{M} := \{m_1, m_2, \ldots, m_t\}$). Each member m_i has a signature weight w_i. A joint signature is valid if it reaches a specific weight threshold. Formally, in a ts-of-n weighted multi-signature scheme, ts is the weight threshold, and n is the total weight of members $\mathbf{M}$ who hold signature weight. $\mathbf{M}$'s weight distribution $\mathbf{w_M}$ should satisfy:

$$(\mathbf{w_M} := (w_1, w_2, \ldots, w_t)) \ \wedge \ (\Sigma \mathbf{w_M} = n)$$

A valid weighted multi-signature can be created by members $\mathbf{m} \subseteq \mathbf{M}$ whose weight distribution is $\mathbf{w_m}$, and satisfies:

$$\Sigma(\mathbf{w_m} := (w_1, \ldots, w_{|\mathbf{m}|})) \geq ts$$

Weighted multi-signature schemes are classified into two types: (i) Collect the independent signatures of $\mathbf{m}$ as a joint signature. (ii) Based on (i), use an aggregation function to aggregate the collected independent signatures into a joint signature, whose size is the same as an independent signature [12], [14]. Formally, each member $m_i \in \mathbf{M}$ uses a digital signature scheme $\mathcal{DS}(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vrf})$. A key generation algorithm $\mathsf{KGen}(1^\lambda)$ takes the security parameter 1^λ as input and outputs a public/secret key pair (pk, sk). Each member m_i holds a key pair (pk_i, sk_i) generated by KGen. A signing algorithm $\mathsf{Sig}(sk; m)$ takes a secret key sk and a message m as inputs and outputs a signature s. A verification algorithm $\mathsf{Vrf}(pk; m, s)$ takes a public key pk, a message m, and a signature s as inputs and outputs 1 if s is a valid signature of m, otherwise 0. We simply model the verification algorithm for blockchain users' signatures (refer to Section III-A) as $\mathsf{Vf}(ad; m, s)$, where ad is not a public key but a user's address derived from its public key. A valid joint signature $\mathbf{s}$ in scheme (i) can be expressed as:

$$\mathbf{s} := \{s_i := \mathcal{DS}.\mathsf{Sig}(sk_i; m) \mid m_i \in \mathbf{m}\}$$

Denoted the aggregation function in scheme (ii) and a size query function as aggre and sizeof, then a valid joint signature S in scheme(ii) satisfies:

$$(S := \mathsf{aggre}(\mathbf{s})) \ \wedge \ (\mathsf{sizeof}(S) = \mathsf{sizeof}(s_i))$$

Scheme (i) has excellent compatibility since nearly all current major blockchains support it. The advantage of scheme (ii) is that the size of its multi-signature is constant and does not float with the threshold value. However, it falls short in compatibility because its relevant principles, mainly bilinear maps or point multiplication in elliptic curve cryptography (ECC) [48], are more sophisticated than scheme (i) and harder to implement on-chain. Besides, the aggregate signature based on bilinear maps typically uses the BLS signature, which is unsuitable to some mainstream blockchains (*e.g.*, Ethereum using ECDSA to sign user transactions). In our PoF protocol, some specific miners would be the signature group. Thereby, we employ scheme (i) due to its compatibility, which helps to achieve chain-agnostic PoF by only requiring miners to have the bare minimum ability to generate and verify signatures. The compatibility is also useful when introducing a PoF smart contract which can verify proofs, *i.e.*, multi-signature (*cf.* Section VIII).

IV. Proof of Finalization

In this section, we introduce the PoF protocol. We start with an intuitive sketch. Given the blockchain BC (refer to Section III-A for details), the data recorded on BC must be finalized. PoF is a protocol that aims to generate proof for the data to prove its finalization. By verifying the proof, even without trusted blockchain data sources, a node can identify finalized data, ensuring that it never mistakes unfinalized data for finalized. We define PoF as a triple of functions (setup, $\mathsf{genProof}$, $\mathsf{vrfProof}$), which are used to initialize a setup, generate a proof for finalized blockchain data, and verify the proof, respectively. We model the three functions as follows:

• setup: It takes *a security parameter* λ as input, and outputs a pair of parameters (arg_e, arg_v), where arg_e is a *proof generation parameter* and arg_v is a *proof verification parameter*.

$$(arg_e, arg_v) := \mathsf{setup}(\lambda).$$

• $\mathsf{genProof}$: It takes as inputs any data dt on BC (There exist m, n to satisfy $dt \in \mathrm{BC}.\mathsf{read}(m[, n])$) and arg_e, then returns a *finalization proof* prf of dt.

$$prf := \mathsf{genProof}(dt, arg_e).$$

• $\mathsf{vrfProof}$: It takes a proof prf, data dt, and arg_v as inputs and outputs a verification result bl. bl is a Boolean value where 1 means that prf is a valid proof of dt and 0 otherwise.

$$bl := \mathsf{vrfProof}(prf, dt, arg_v).$$

A secure PoF protocol satisfies completeness and soundness properties defined as follows:

• **Completeness**: Given a security parameter λ, there is a negligible function negl such that for any $prf := \mathsf{genProof}(dt, arg_e)$, where dt is any data on BC, it holds that:

$$\Pr\left[\mathsf{vrfProof}(prf, dt, arg_v) = 1\right] \geq 1 - \mathsf{negl}(\lambda).$$

• **Soundness**: Given a security parameter λ, there is a negligible function negl such that for any dt' not on BC (There exist no m, n to satisfy $dt' \in \mathrm{BC}.\mathsf{read}(m[, n])$), any proof prf' of dt' holds that:

$$\Pr\left[\mathsf{vrfProof}(prf', dt', arg_v) = 1\right] \leq \mathsf{negl}(\lambda)$$

V. Protocol Overview

We design a PoF protocol to resolve the PoF problem. In this section, we introduce the protocol's system model, adversary model, and system goals, and present the overview of the protocol.

A. System Model

The PoF protocol involves three entities:

*Blockchain (*BC*):* The blockchain BC conforms to the blockchain model depicted in Section III-A, which mainstream blockchains such as Bitcoin and Ethereum satisfy. Therefore, the blocks on BC are finalized, and BC satisfies the common prefix, chain quality, and chain growth properties [21] so that it can continuously handle and finalize new transactions according to its consensus mechanism.

*Committee Members (***M***):* Committee members are some specific miners of BC selected by our protocol to generate proof for the blocks on BC. Each member determines BC based on its blockchain view and BC's consensus mechanism.

*Verifiers (***V***):* Any node susceptible to being spoofed by unfinalized blocks can optionally become a verifier. When receiving a block claimed as finalized, a verifier validates the block's finalization by verifying the corresponding proof provided by our protocol. In this way, the verifier will never mistake unfinalized block fed by unreliable connected nodes for finalized.

B. Adversary Model

We assume a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ exists in the protocol execution. The adversary aims at spoofing verifiers with unfinalized blockchain data, while its ability is as follows:

Blockchain: $\mathcal{A}$ can statically corrupt miners of BC, *i.e.*, each miner node has been corrupted or not before becoming the miner of blocks in BC; each honest (*resp.* corrupted) miner remains honest (*resp.* corrupted) in protocol executions. However, as entailed in our assumption of BC, the miners corrupted by $\mathcal{A}$ are insufficient to destroy the common prefix, chain quality, and chain growth properties of BC, and all honest members can reliably access the latest blockchain view of BC.

Committee Members: We assume all selected honest BC's miners will join the committee. Besides, honest committee members can synchronize information through a committee network. These assumptions are practical, and we demonstrate existing solutions meeting the assumptions in Section VIII.

Verifiers: Each verifier itself is honest. $\mathcal{A}$ can fully control the I/O of any verifier, *e.g.*, isolating a verifier from the real blockchain network and sending unfinalized blockchain data to the verifier. However, before $\mathcal{A}$ launches isolation, we assume each verifier can be trustworthily set up with a specified finalized block. Although this setup is essentially a trusted setup, we note that there already exist many mechanisms for a verifier to obtain the block with mitigated spoofing risks. For example, an SPV node usually synchronizes block headers from reputable RPC service providers [1], [3], [6], checkpoints sync endpoints [4], the infrastructure in underlying blockchain [5], *etc.* and can cross-validate the data obtained. We stress that each verifier only needs to synchronize the finalized block once as the root of trust. Thereafter, the verifier's network I/O can be fully controlled by $\mathcal{A}$.

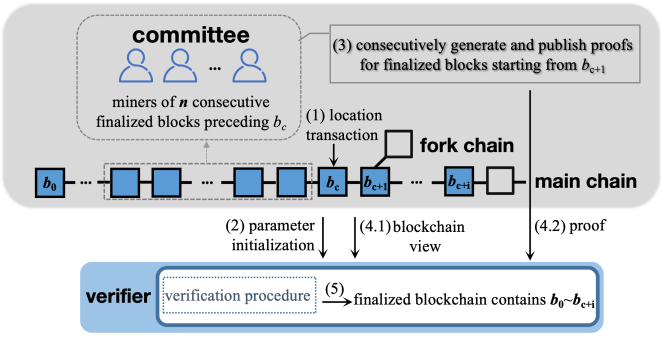

Fig. 1. Overview of the PoF protocol. The PoF committee members are miners of the n consecutive blocks preceding the block containing the location transaction. The blue blocks represent finalized blocks. After verifying proof, the verifier can identify the finalized blockchain containing the genesis block b_0 to the signed block b_{c+i}.

C. System Goals

In this section, we describe the system goals in devising PoF protocols:

Completeness: The blocks on BC and the corresponding proofs can successfully pass the validation of each verifier, satisfying the completeness in PoF definition (Section IV).

Soundness: $\mathcal{A}$ can generate a proof for an unfinalized block. Any unfinalized block and its proof generated by $\mathcal{A}$ can pass the validation of a verifier with negligible probability, satisfying the soundness in PoF definition.

D. Overview

Conforming to the definition in Section IV, our PoF protocol runs in three phases: setup, genProof, and vrfProof, corresponding to the global setup, proof generation, and proof verification respectively. Now we present the three phases as follows:

setup phase. During the global setup phase, the protocol configuration is published, the committee members are determined and initialize the proof generation parameter, and verifiers initialize the proof verification parameter.

Any blockchain user posts a *location transaction* on BC (Step (1) in Fig. 1). The transaction contains a protocol configuration, including the required committee size n and threshold ts (*i.e.*, the least weight of a set of independent signatures created by committee members to secure a proof). The transaction would eventually be packed into a finalized block (denoted as b_c), and locate the miners of the consecutive n blocks immediately preceding b_c as committee members. Each member initializes the proof generation parameter, including ts and the committee's weight distribution. Next, each verifier is initialized with the block b_c. According to our assumption, the verifier trusts that the b_c it obtains is indeed the finalized block that contains the location transaction. After validating the

determined protocol config in b_c, the verifier fetches the valid blockchain fragment consisting of the consecutive n blocks before b_c, then initializes the proof verification parameter, including ts and the committee's weight distribution, from b_c and the blockchain fragment (Step (2)).

genProof phase. The committee starts to generate proofs for finalized blocks. Specifically, each committee member independently determines the block b_{c+1} following b_c based on its blockchain view, and signs the chain ID and the block hash of b_{c+1}. Committee members will synchronize their independent signatures through the committee network. When the weight of independent signatures for b_{c+1} achieves the threshold ts, a valid weighted multi-signature can be generated as proof. Considering the compatibility of signature schemes, we adopt scheme (i) in Section III-B for proof generation. Namely, the proof is the set of independent signatures with weights of at least ts. The proof proves the finalization of the blockchain which starts from the genesis block b_0 and ends with b_{c+1}, since b_{c+1} functions like a checkpoint. Then any committee member can provide the proof when being requested. Starting from b_{c+1}, the committee would consecutively generate proof for finalized blocks subsequent to b_c to identify the latest finalized blockchain (Step (3)).

vrfProof phase. After the setup phase, each verifier can identify the finalized blockchain even if its connected nodes are untrusted. Concretely, the verifier updates its blockchain view from its connected nodes (Step (4.1)), and requires a proof simultaneously (Step (4.2)). With the verification parameter initialized in the setup phase, the verifier can execute its verification procedure. Only if the proof passes the verification procedure, the verifier accepts the finalized blockchain which starts from b_0 and ends with the block signed in the proof (Step (5)). We note that the proving phase and verification phase are independent of each other, committee members and verifiers don't need to interact with each other.

VI. Protocol Detail

In this section, we describe in detail the PoF protocol π_{pof}, introduce optimizations, and make a security analysis.

A. Protocol Details

In Fig. 2 we illustrate the protocol's workflow, which involves three phases: the setup phase, genProof phase, and vrfProof phase.

setup phase. During the setup phase, the PoF committee $\mathbf{M}$ is established, each member $m_i \in \mathbf{M}$ initializes the proof generation parameter arg_e, and each verifier v initializes the proof verification parameter arg_v.

location transaction $tx_c(n, ts)$	
$sender$:	the blockchain user's address
$data$:	PoF config including n and ts

Any user of BC can initiate a location transaction tx_c including PoF configuration by invoking BC.send$(tx_c(n, ts))$, where (n, ts) is generated from a security parameter λ via the function Gen(1^λ). The protocol configuration includes a required committee size n and threshold ts (*i.e.*, the least

$\pi_{pof}(\mathrm{BC}, v, \lambda)$

Parameters:

tx_c: the location transaction

$\mathbf{b}_s$: a block sequence consisting of blocks $(b_x, b_{x+1}, \cdots, b_{x+y})$

setup(λ) :

$(n, ts) :=$ Gen(1^λ)

BC.send$(tx_c(n, ts))$

// tx_c will be packed in a block, denoted as $\mathrm{BC}.b_c$, after certain rounds.

$tx_c \in \mathrm{BC}.b_c$

$h_c := b_c.height$

$\mathbf{b} :=$ BC.read$(h_c - n, h_c - 1)$

// let $(m_1, m_2, \cdots, m_t)$ be the miners of $\mathbf{b}$.

$\mathbf{M} := \{m_1, m_2, \cdots, m_t \mid \forall b \in \mathbf{b}, b.miner \in \mathbf{M}\}$

for each member $m_i \in \mathbf{M}$ **do**

// proof generation parameter and the first block to prove

$(arg_e, b_j) := m_i$.initialize(b_c)

$arg_v := v$.initialize(b_c) // proof verification parameter

genProof(b_j, arg_e) :

$\mathbf{prf}_j := m_i$.generateProof(b_j, arg_e)

$b_j :=$ BC.read$(b_j.height + 1)$

goto genProof(b_j, arg_e)

vrfProof$(\mathbf{prf}_{x+y}, \mathbf{b}_s, arg_v)$:

v.verifyProof$(\mathbf{prf}_{x+y}, \mathbf{b}_s, arg_v)$

Fig. 2. The PoF protocol.

weight of a set of independent signatures created by committee members to secure a proof). To ensure the uniqueness of the location transaction, miners only admit the first valid location transaction recorded on BC. After tx_c is recorded in a finalized block b_c with the height h_c (*i.e.*, $h_c := b_c.height$), the committee members of $\mathbf{M}$ are determined, which are the miners of the n consecutive blocks $\mathbf{b} :=$ BC.read$(h_c - n, h_c - 1)$. Note that the miners may be repetitive (*e.g.*, 2 blocks in $\mathbf{b}$ are mined by the same address). Thus, the committee $\mathbf{M}$ consists of $t \leq n$ members and runs a weighted multi-signature scheme. Each member m_i holds a public/private key pair (pk_i, sk_i) for the generation and validation of its signature, and its blockchain address ad_i is derived from its public key pk_i. m_i will be assigned a weight value w_i linked to its address ad_i, and w_i is determined by the number of blocks mined by m_i in $\mathbf{b}$. Formally, the committee $\mathbf{M} := (m_1, m_2, \ldots, m_t)$, where each member m_i holds a key pair (pk_i, sk_i). Let $\mathbf{w_M}$ be a key-value map to represent $\mathbf{M}$'s weight distribution where keys are member addresses and values are the corresponding weights, it satisfies $\sum_{i=1}^{t} \mathbf{w_M}[ad_i] = n$ where

$$\mathbf{w_M} := (ad_1 \to w_1, \ldots, ad_t \to w_t)$$

Each member m_i initializes the proof generation parameter $arg_e := (ts, \mathbf{w_M})$ and the block b_j, which is the first block to prove (refer to Algorithm 1). Specifically, m_i first obtains the threshold ts recorded in $tx_c \in b_c$. Second, m_i obtains $\mathbf{w_M}$ by retrieving the coinbase transactions in $\mathbf{b}$. Finally, m_i initializes b_j with BC.read$(h_c + 1)$.

Algorithm 1 Committee member (m_i)

Procedure initialize (b_c)
  assert $tx_c(n, ts) \in b_c$ is the first location transaction
  $\mathbf{b} := \text{BC.read}(h_c - n, h_c - 1)$
  for each block $b_i \in \mathbf{b}$ **do**
  // initialize an address-to-weight map $\mathbf{w_M}$
    $ad_i := b_i.miner$
    // initial value of $\mathbf{w_M}[ad_i]$ is 0
    $\mathbf{w_M}[ad_i] := \mathbf{w_M}[ad_i] + 1$
  $arg_e := (ts, \mathbf{w_M})$
  $b_j := \text{BC.read}(h_c + 1)$
  return (arg_e, b_j)
Procedure generateProof (b_j, arg_e)
  assert $b_j := \text{BC.read}(b_j.height) \neq \emptyset$
  $s_i := \mathcal{DS}.\text{Sig}(sk_i; (\text{BC}.id_c, b_j.hash))$
  // collect a set of signatures $\mathbf{s}$ of b_j from members $\mathbf{m} \subseteq \mathbf{M}$
  collect $\mathbf{s} := \{s_1, s_2, \cdots, s_{|\mathbf{m}|} \mid \forall m_i \in \mathbf{m} \subseteq \mathbf{M}, s_i \in \mathbf{s}\}$
  $w_\mathbf{s} := 0$
  for each signature $s_i \in \mathbf{s}$ **do**
    if $\mathcal{DS}.\text{Vf}(ad_i; \text{BC}.id_c, b_j.hash, s_i) = 1$ **then**:
      $w_\mathbf{s} := w_\mathbf{s} + arg_e.\mathbf{w_M}[ad_i]$
  if $w_\mathbf{s} \geq arg_e.ts$ **then**:
    $\mathbf{prf}_j := \mathbf{s}$
    post $\mathbf{prf}_j$ on the proof publication location
    return $\mathbf{prf}_j$

Algorithm 2 Verifier (v)

Procedure initialize (b_c)
  if $n < k$ **or** $ts \leq \lfloor (1-\mu)n \rfloor$ **then output** *setupFail*
  else wait for $(\mathbf{b}' := \text{BC.read}(h_c - n, h_c - 1))$
  if $\text{BC.validate}((\mathbf{b}', b_c)) = 0$ **then output** *setupFail*
  else for each block $b_i \in \mathbf{b}'$ **do**
  // initialize an address-to-weight map $\mathbf{w_M}$
    $ad_i := b_i.miner$
    // initial value of $\mathbf{w_M}[ad_i]$ is 0
    $\mathbf{w_M}[ad_i] := \mathbf{w_M}[ad_i] + 1$
  $arg_v := (ts, \mathbf{w_M})$
  return arg_v
Procedure verifyProof ($\mathbf{prf}_{x+y}, \mathbf{b}_s, arg_v$)
  if $\text{BC.validate}(\mathbf{b}_s) = 0$ **then return** 0
  $w_{x+y} := 0$
  for each signature $s_i \in \mathbf{prf}_{x+y}$ **do**
    if $\mathcal{DS}.\text{Vf}(ad_i; \text{BC}.id_c, b_{x+y}.hash, s_i) = 1$ **then**:
      $w_{x+y} := w_{x+y} + arg_v.\mathbf{w_M}[ad_i]$
  if $w_{x+y} < arg_v.ts$ **then return** 0
  else return 1

Each verifier v is initialized with the block b_c in a trustworthy manner (assumed in Section V-B), then extracts and validates the PoF config in the location transaction $tx_c \in b_c$ according to the chain quality property (refer to Algorithm 2). Concretely, the chain quality property entails that in any consecutive k or more blocks of BC, the proportion of honest miners is at least μ. To leverage the property, for $tx_c(n, ts)$, v validates $n \geq k$ and $ts > \lfloor (1-\mu)n \rfloor$. We note that the protocol configuration is flexible and can be adjusted for different balances/levels of availability and security. We will present it in Section VI-B. It does not matter if tx_c is proposed by a malicious blockchain user because v will reject tx_c if its PoF config contradicts the above conditions. If the config is correct, the verifier accepts the miners of the consecutive n blocks $\mathbf{b}$ as PoF committee members.

Each verifier v independently learns about the proof verification parameter $arg_v := (ts, \mathbf{w_M})$ from b_c and $\mathbf{b}$ (refer to Algorithm 2). Specifically, v first obtains the threshold ts recorded in tx_c included in b_c. Second, v requests the consecutive n blocks $\mathbf{b}$ which precedes b_c from its connected nodes, and obtains each member's address ad_i and corresponding weight w_i to initialize and assign $\mathbf{w_M}$ by retrieving the coinbase transactions in $\mathbf{b}$. Crucially, it is virtually impossible for an assumed PPT adversary to forge a $\mathbf{b}'$ corresponding to incorrect $\mathbf{w_M}$ to spoof v, because v holds the trusted b_c as a checkpoint and can validate the received $\mathbf{b}'$ by invoking $\text{BC.validate}((\mathbf{b}', b_c))$, which checks the validity of the blockchain fragment $(\mathbf{b}', b_c)$. With b_c as the root of trust, it is ensured that any v that completes the global setup has obtained the correct arg_v.

genProof phase. After tx_c is finalized and determines the committee $\mathbf{M}$, each committee member m_i generates and broadcasts the signature for b_j, the finalized block following b_c, independently. Concretely, m_i signs the tuple of chain ID and b_j's hash $(\text{BC}.id_c, b_j.hash)$ using its private key sk_i. m_i broadcasts the signature and collects signatures from other members over the public committee network (*e.g.*, the public and dedicated mempool in EIP-4337). Let $\mathbf{s}$ be a set of signatures collected by m_i from members $\mathbf{m} \subseteq \mathbf{M}$, and $w_\mathbf{s}$ is the total weight of $\mathbf{s}$, which is equal to the accumulation of the weights of each signature $s_i \in \mathbf{s}$. If $w_s \geq arg_e.ts$, $\mathbf{s}$ will be b_j's proof $\mathbf{prf}_j$. Upon $\mathbf{prf}_j$ is created, any m_i can post $\mathbf{prf}_j$ on the proof publication location, which has many choices (a web server, a cloud database, *etc.*) and can be chosen depending on verifiers' convenience. Starting from b_j, each m_i continuously generates proof for finalized blocks after b_c to identify the latest BC.

In addition, the committee can be rotated to improve its security and liveness. With the rotation, a committee can continuously sign the newly finalized blocks until the committee rotates. To further enhance liveness, imposing a reward-punishment mechanism on the committee members is feasible. We demonstrate the committee rotation in Section VI-C and a reward-punishment mechanism in Section VIII.

vrfProof phase. After obtaining the proof verification parameter arg_v, each verifier v can identify the finalized blockchain even if its connected nodes are untrusted. Concretely, when updating its blockchain view from connected nodes, v receives a block sequence $\mathbf{b}_s := (b_x, b_{x+1}, \ldots, b_{x+y})$ and the proof $\mathbf{prf}_{x+y}$ that is claimed to be the valid proof for b_{x+y}. Then, the v proceeds validation as follows (refer to Algorithm 2):

(i) Verify the validity of $\mathbf{b}_s$ by invoking BC.validate($\mathbf{b}_s$). (ii) Verify that each independent signature s_i in $\mathbf{prf}_{x+y}$ can be successfully verified by the corresponding address ad_i in $arg_v.\mathbf{w_M}$, and (iii) the signature weight of $\mathbf{prf}_{x+y}$ achieves the threshold $arg_v.ts$. Then if $\mathbf{b}_s$ passes the verification, v accepts $\mathbf{b}_s$ as finalized.

Note that our assumptions allow an adversary to compromise the connected nodes to withhold updated blockchain views or proof sent to the verifier, making the verifier fail to obtain the finalized blockchain. We pinpoint that this problem is about denial of service instead of data spoofing, thus being out of this paper's scope. However, we stress that in any assumed cases including the above, since the adversary cannot forge a valid proof for any unfinalized block (*cf.* Section VI-D), the verifier will never mistake unfinalized data for finalized, which achieves the goal of our PoF protocol.

B. Threshold Setting

Recall we leverage the chain quality property to organize a committee whose total weight is n and threshold $ts > \lfloor(1-\mu)n\rfloor$. In this section, we elaborate that the threshold setting is deemed as a trade-off between security and availability of PoF and demonstrate how to tune ts and n to meet the real-world need in different scenarios, *e.g.*, Ethereum.

We model the process of mining n blocks as a Bernoulli process, wherein in each Bernoulli experiment, we define a Boolean random variable x_i that has $x_i := 1$ when the block b_i is created by an honest miner, otherwise $x_i := 0$. Denote the probability that a miner is honest as p, *i.e.*, $p := \Pr[x_i = 1]$. Let the random variable x be the number of blocks mined by honest miners in the sequence, *i.e.*, $x := \sum_{i=1}^{n} x_i$. Then the expected value of x is $\mathbb{E}[x] := np$, and x satisfies binomial distribution, *i.e.*, $x \sim b(n, p)$. The probability that honest blocks are no more than a certain value x' would be:

$$\Pr[x \le x'] = \sum_{i=0}^{x'} \binom{n}{i} (p)^i (1-p)^{n-i} \tag{1}$$

Assume that a PoF committee is composed of the miners of 48 consecutive blocks on a PoS blockchain, which has the security assumption that the stake held by honest nodes is over $\frac{3}{4}$ of total stakes. We refer to the validator committee in the Ethereum Beacon chain, which achieves the security that the probability of the proportion of malicious validators reaching $\frac{2}{3}$ is less than 2^{-40} [7]. According to the equation 1, we can set the threshold to no less than 36 to be comparable with the Ethereum consensus security requirement.

Although a rising threshold ts increases the security of PoF protocol, it leads to decreased protocol availability. Assume each block has a different miner to simplify the scenario. As our protocol requires at least ts signatures of committee members to be a proof of block finality, it means that at least ts members should be active for proof generation. Thereby, with an excessive ts (*e.g.*, is equal to committee size), the accidental downtime of any member can undermine the PoF availability. To balance the security and availability for real-world needs, we list some reference values in Table III. Specifically, we set the committee size n to the maximum of 120, which is close to the Ethereum committee (128 validators). We require the security probability $\Pr[n-x \ge ts] \le 2^{-40}$, and the lower ratio ts/n is associated with higher availability. The PoF protocol can adjust the committee threshold flexibly to balance security and availability, meeting practical needs better.

TABLE III
REFERENCE VALUES FOR THRESHOLD SETTING

p	n	ts	ts/n	$\Pr[n-x \ge ts]$
$\frac{3}{4}$	48	36	$\frac{3}{4}$	2^{-40}
$\frac{2}{3}$	30	29	≈ 1	2^{-40}
$\frac{2}{3}$	75	60	$\frac{4}{5}$	2^{-52}
$\frac{2}{3}$	120	80	$\frac{2}{3}$	2^{-43}
$\frac{1}{2}$	120	100	$\frac{5}{6}$	2^{-45}

C. Optimization

To avoid confusing the focus of PoF protocol, we omit some design details of the committee in previous sections. In this section, we elaborate on committee election and committee rotation to reinforce the details.

Committee Election: In this work, we leverage the location transaction to determine committee members. Naturally, the method we propose is not the only way to organize the committee. The key point is to publicly and deterministically select the n consecutive blocks. For example, the location transaction can combine with the random source (*i.e.*, block hash) provided by the blockchain. Then the location transaction's position and a random number jointly determine the consecutive n blocks. This method avoids the tendency that blocks adjacent to the location transaction are more likely to be selected, further enhancing the unpredictability of the PoF committee members. Besides, some early miners (*e.g.*, miners who mine blocks close to the genesis block) may already have exited the blockchain network or resold their accounts to someone else. This problem can be mitigated by limiting the range of block selection, *e.g.*, choosing n blocks from the recent 500 blocks. In summary, the selection method can flexibly change according to practical needs.

Committee Rotation: The committee can be rotated to reduce further the liveness requirement to its members. Specifically, we divide the blocks on BC into different epochs, each of which is assigned a committee to generate proof for the epoch's blocks. Every committee's epoch is deterministically derived by the first location transaction tx_c (*e.g.*, by a pseudo-random number in tx_c). Except for the first committee, each committee's members are selected by the last block of the previous epoch. Besides, we can also support the active update, where the current committee can optionally post a new location transaction tx'_c (with a newer PoF version number to be distinguished from previous location transactions) on BC to identify the next committee. The current committee will generate proof until proving the finalized block b'_c containing tx'_c.

As we require the last block of an epoch finalized by its committee to decide the next epoch, the absence of its committee liveness would affect the liveness of our protocol. But we emphasize that the absence of liveness can be very rare under a proper PoF config and financial control. Specifically, with the flexible threshold setting (*cf.* Section VI-B), our PoF can tolerate the accidental downtimes of an adjustable proportion of members. We can also reduce the deliberate inactivity of misbehaving members further by introducing a reward-punishment mechanism, which incentivizes honest and active members and slashes inactive members (*cf.* Section VIII).

Besides, we designed a mechanism that when the current committee is down, we ask the previous committees to endorse the new rotation. Specifically, besides the own epoch, a committee $\mathbf{M}$ should follow the blockchain until the next two epochs finish and timeout. If either of the next two committees loses liveness, $\mathbf{M}$ should take over to generate missed proofs. From verifiers' view, besides the proof signed by $\mathbf{M}_i$, the committee of the epoch i, they also accept the proof co-signed by $\mathbf{M}_{i-1}$ and $\mathbf{M}_{i-2}$, where both $\mathbf{M}_{i-1}$ and $\mathbf{M}_{i-2}$ should reach the threshold ts. Recall that the probability of compromising ts members is negligible, the endorsement by the two committees is also trusted and even stronger. Although this mechanism cannot absolutely guarantee PoF liveness since on rare occasions previous committees may also be inactive, we can make this mechanism work in conjunction with the reward-punishment mechanism and threshold setting adjustment mechanism, mitigating the inactivity tendency of committee members collectively and efficiently.

D. Security Analysis

Given the assumption of a PPT adversary $\mathcal{A}$ (*cf.* Section V-B), a blockchain BC conforming to the assumed model (*cf.* Section III-A), an EU-CMA secure signature scheme $\mathcal{DS}$, and a second preimage resistant hash function, this section demonstrates the arguments that PoF protocol π_{pof} meets both completeness and soundness introduced in Section IV by enumerating possible attacking cases.

Recall the main symbols, variables, and workflow of our protocol (Section VI): The location transaction tx_c contained in the block b_c locates the n consecutive blocks preceding b_c for determining committee members, which periodically generate proof for finalized blocks (*i.e.*, the blocks on BC). Each member signs the chain ID BC.id_c and the hash of a finalized block b using its private key sk_i as proof. As the miners of the n blocks may be repetitive, the committee runs a weighted multi-signature scheme. Formally, the committee $\mathbf{M}$ consists of $t \leq n$ members. Let $\mathbf{w_M}$ represent the address-to-weight map of $\mathbf{M}$, it satisfies $\sum_{i=1}^{t} \mathbf{w_M}[ad_i] = n$ where

$$\mathbf{w_M} := (ad_1 \rightarrow w_1, \ldots, ad_t \rightarrow w_t)$$

In the consecutive n blocks $\mathbf{b} := \text{BC.}\mathsf{read}(h_c - n, h_c - 1)$, where $h_c := b_c.height$, denote the number of blocks mined by compromised miners by n_c. Based on the chain quality property, we have $n_c \leq \lfloor (1-\mu)n \rfloor$. With the block b_c as the root of trust, a verifier v obtains the threshold ts and $\mathbf{w_M}$ to constitute the proof verification parameter arg_v in the global setup phase. As we set the threshold $ts > \lfloor (1-\mu)n \rfloor$, v accept a fed block as finalized iff it obtains a valid weighted multi-signature $\mathbf{s}$, the total weight $w_{\mathbf{s}}$ of $\mathbf{s}$ satisfies $w_{\mathbf{s}} \geq arg_v.ts$.

Claim 1: π_{pof} satisfies the completeness that given the proof verification parameter arg_v being initialized based on b_c, for any proof $\mathbf{prf} := \{\mathbf{s} \mid w_{\mathbf{s}} \geq arg_v.ts\}$ of any block b recorded on BC, each verifier v holds that

$$\Pr\left[v.\mathsf{verifyProof}(\mathbf{prf}, b, arg_v) = 1\right] \geq 1 - \mathsf{negl}(\lambda)$$

Proof: In π_{pof}, the proof verification function $\mathsf{verifyProof}$ is a signature verification algorithm, and the proof verification parameter is $arg_v := (ts, \mathbf{w_M})$. If the weighted multi-signature ($\mathbf{prf}$) of b is valid, and arg_v is correct, the verification will output 1, due to the completeness of the signature algorithm itself. To realize $v.\mathsf{verifyProof}(\mathbf{prf}, b, arg_v) \neq 1$, as the $\mathbf{prf}$ of b is assumed to be valid, the only chance for an adversary $\mathcal{A}$ is to make a verifier v initialize an incorrect arg_v to invalidate $\mathbf{prf}$. Recall that v extracts $arg_v := (ts, \mathbf{w_M})$ from the blockchain fragment $(\mathbf{b}, b_c)$, where b_c is the finalized block containing the location transaction $tx_c(n, ts)$, and the coinbase transactions in $\mathbf{b} := \text{BC.}\mathsf{read}(h_c - n, h_c - 1)$ indicate committee members' addresses $(ad_1, \ldots, ad_t)$ and corresponding weights $(w_1, \ldots, w_t)$. As v is assumed as honest and can be trusted setup with b_c, v can obtain correct ts and verify received $\mathbf{b}$ with b_c as a checkpoint. Then the last chance for $\mathcal{A}$ is to make v initialize an incorrect $\mathbf{w_M}$ under the limit of b_c. Specifically, $\mathcal{A}$ may try to provide v with spoofed blocks $\mathbf{b}'$ which satisfy $n'_c \geq ts$ and $\text{BC.}\mathsf{validate}((\mathbf{b}', b_c)) = 1$. To achieve this, $\mathcal{A}$ should at least alter $ts - n_c$ block(s) in $\mathbf{b}$, which is impossible since v holds b_c as a checkpoint and the hash function used to compute block hash satisfies second preimage resistance. Thereby, π_{pof} satisfies completeness.

Claim 2: π_{pof} satisfies the soundness that given the proof verification parameter arg_v being initialized based on b_c, for any proof $\mathbf{prf}'$ that $\mathcal{A}$ generates for b', where b' is any block not being recorded on BC, each verifier v holds that

$$\Pr\left[v.\mathsf{verifyProof}(\mathbf{prf}', b', arg_v) = 1\right] \leq \mathsf{negl}(\lambda)$$

Proof: Conceive possible cases in which $\mathbf{prf}'$ destroys soundness: (i) $\mathcal{A}$ deceives v with an incorrect arg_v to make $v.\mathsf{verifyProof}(\mathbf{prf}', b', arg'_v) = 1$. (ii) If the arg_v held by v is correct, the last chance for $\mathcal{A}$ is trying to corrupt a set of members (miners) $\mathbf{m}$ which satisfies

$$\mathbf{m} \subseteq \mathbf{M} \ \wedge \ \sum_{i=1}^{|\mathbf{m}|} \mathbf{w_m}[ad_i] \geq ts$$

making $v.\mathsf{verifyProof}(\mathbf{prf}', b', arg_v) = 1$, where $\mathbf{w_m}$ is the address-to-weight map of $\mathbf{m}$, and $\mathbf{prf}' := \{\mathbf{s} \mid \forall m_i \in \mathbf{m}, s_i \in \mathbf{s}\}$. Particularly, due to the EU-CMA secure signature scheme run by committee members, corrupting insufficient members (whose total weight is less than ts) cannot enable $\mathcal{A}$ to forge a valid weighted multi-signature whose weight achieves ts.

We have proved that the case (i) is impossible (*cf.* proof of Claim 1). Now we consider the case (ii). As $\mathcal{A}$ is a

static adversary that corrupts miner nodes before they become the miners of blocks in BC, honest members are the honest miners of the consecutive n blocks. Let $w_{\mathbf{h}}$ be the total weight of honest members, then the probability of the case (ii) is $\Pr[w_{\mathbf{h}} \leq n - ts]$. Let p be the probability of a block being mined by an honest member, then we have:

$$\Pr[w_{\mathbf{h}} \leq n - ts] = \sum_{i=0}^{n-ts} \binom{n}{i} (p)^i (1-p)^{n-i}.$$

As the blockchain satisfies the chain quality property that $w_{\mathbf{h}} \geq \lceil \mu n \rceil$, and the threshold $ts > \lfloor (1-\mu)n \rfloor$, the probability $\Pr[w_{\mathbf{h}} \leq n - ts < \lceil \mu n \rceil]$ is negligible (*cf.* Section VI-B). Thereby, π_{pof} satisfies soundness.

VII. Evaluation

Implementation and Setup: In this section, we design experiments to answer two questions about the PoF protocol:

- What is the latency of proof generation? Specifically, how long does it take from a block becomes finalized until its proof can be accessed?
- How long does it take to verify the proof?

We evaluate the latency in PoW and PoS consensus protocols, corresponding to pre- and post-Ethereum merge, respectively. The Ethereum merge is the joining of Ethereum Mainnet (PoW consensus) with Ethereum's Beacon Chain (PoS consensus). In the Mainnet, a block is finalized when getting 7 confirmations. While in the Beacon chain, a block is finalized by a recent checkpoint after it, and two checkpoints are usually 32 slots[1] apart. In other words, the Mainnet finalizes blocks one by one, while the Beacon chain finalizes in bulk. Considering the difference, we make the committee sign every finalized block in the Mainnet, while only signing checkpoints in the Beacon chain. The reason is that signing only checkpoints is sufficient to prove the finalization of block in bulk, while also mitigating the volume of proof data, which can degrade cost when proofs are posted on a blockchain.

We leverage the public API of Ethereum (getBlock function and subscribe function, which are used to subscribe information of the latest block) to simulate a member's behavior of accessing the latest blockchain view. Then the member can determine when a block becomes finalized. For example, in the Ethereum mainnet, the time of receiving the latest block with height h, is the finalized time of the block whose height is $h-6$. Besides, we have developed two programs to implement the genProof function and vrfProof function of PoF protocol. Each member's key pairs used to generate and verify proofs are generated by the public API of Ethereum (accounts.create() function).

We perform the experiment on Intel(R) Xeon(R) Platinum 8269CY CPU with 1GB of memory, and the two programs are written in JavaScript. In the experiment, the block interval time is τ, the proportion of the resources (*i.e.*, hashing power or stake) held by honest blockchain nodes is p, and h represents the height of a block. Members synchronize signatures in the dedicated committee network in which transmission speed is quite fast. Hence, we simulate members by the multiple threads in the genProof program. By default, the committee contains $n := 48$ members, and the committee threshold $ts := 36$. We will adjust n and ts for more comprehensive evaluation results. For convenience, we assume that members have equivalent signature weights. Similarly, a verifier is simulated by the vrfProof program.

[1]In the Beacon chain, blocks are separated by epoch which contains 32 slots, and either one block is generated in a slot or not.

TABLE IV

THE COMPARISON OF LATENCY. p REPRESENTS THE SECURITY ASSUMPTION. SECURITY REQUIREMENT DETERMINES THE PROBABILITY $\Pr[X_c \geq X]$ IN EXISTING POF SCHEMES, OR DETERMINES THE PROBABILITY $\Pr[n_c \geq ts]$ IN OUR POF SCHEME. THE SECURITY REQUIREMENT IS SATISFIED BY ADJUSTING OTHER PARAMETERS PROPERLY UNDER THE FIXED p VALUE. THE PINK (*resp.* YELLOW) CELLS CONTAIN THE LATENCY IN EXISTING SCHEMES (*resp.* OUR POF SCHEME)

p	security requirement	other parameters	latency
$\frac{3}{4}$	2^{-40}	$X := 70, \beta := 1.5$	819.6s
		$n := 48, ts := 36$	0.11s
$\frac{2}{3}$	2^{-43}	$X := 140, \beta := 1.5$	1730.7s
		$n := 120, ts := 80$	0.20s
$\frac{2}{3}$	2^{-52}	$X := 170, \beta := 1.5$	2121.5s
		$n := 75, ts := 60$	0.14s

A. Latency of Proof Generation

In the Ethereum mainnet, if the first block signed by the committee is b_i, then all signed blocks are $b_i, b_{i+1}, b_{i+2}, \ldots$. The committee will immediately sign these blocks once they become finalized. Hence, the latency of proof generation for any above block is equal to the negligible time required for the committee to synchronize information and deliver the weighted multi-signature, which is 105.99 milliseconds according to our evaluation results.

We also compare the latency with existing PoF schemes [13], [15], in which provers provide X confirmation blocks for a block within the time limit $X \cdot \beta \cdot \tau$ (β is a multiplicative slowness factor) as the block's finalization proof. Denote the number of confirmation blocks provided by an adversary within the time limit as X_c, and n_c represents the number of malicious members in our PoF committee. We compare the latency under different security assumptions and security requirements (*cf.* Table IV). The latency of existing schemes is dramatically higher than us. Hence, our PoF scheme is low-latency.

Additionally, we evaluate the latency with the signing interval $H > 1$ (*i.e.*, all signed blocks are $b_i, b_{i+H}, b_{i+2H}, \ldots$) for references. Specifically, a public blockchain is one of the choices for the proof publication location. If proofs are published on-chain, H closer to 1 is associated with more expensive interactions with the blockchain. Therefore, we evaluate the latency with $H > 1$. We find that the latency for a block not signed by the committee is not negligible. For example, the latency of the block b_{i+1} is the relative time from the generation of b_{i+7} until the proof of b_{i+H} is disclosed (*i.e.*, until b_{i+H+6} emerges). We conclude that for any block with height $h \neq i + jH$, the latency is equal to $(H - (h-i)\%H) \cdot \tau$;

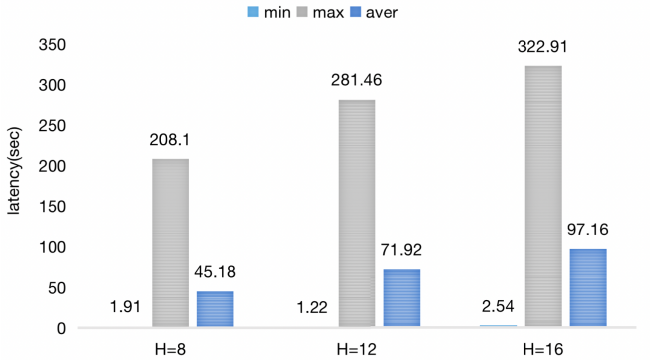

Fig. 3. The latency under different committee signature intervals H. The three-dimensional histogram shows minimum latency, maximum latency, and average latency respectively.

for any block with height $h = i+jH$, the latency is negligible. Based on the latency formula, we set the interval $H := 8$, 12, and 16 respectively and measure the corresponding latency. Fig. 3 records the evaluation results. In general, the average block interval time (*i.e.*, τ) of the Ethereum mainnet is between 12s and 14s. Under $\tau := 13s$, the maximum latency at $H := 16$ is 322.91s $\approx 25\tau$, while the average latency is 97.16s $\approx 7.5\tau$, which is still much less than the existing PoF schemes. When $H := 12$ (resp. 8), the latency can be reduced to 5.5τ (resp. 3.5τ).

For the Beacon chain, the committee periodically signs checkpoints. We assume that the first block signed by the committee is the recent checkpoint, and the committee signs it immediately after it becomes finalized. Then the committee's activities will be in sync with the finalization mechanism of the Beacon chain, *i.e.*, the committee will sign the follow-up checkpoints immediately after they become finalized. Thereby, the latency of proof generation is 105.99 milliseconds, which is the negligible multi-signature generation time.

B. Latency of Proof Verification

We expect that given a block and the corresponding proof, a verifier can complete the proof validation quickly. Proof verification mainly involves signature verification and block verification. To comprehensively evaluate the latency of proof verification, we set the signing interval $H \geq 1$ and execute the vrfProof program for a block signed by the committee or not respectively. When $H > 1$, for any signed block with height $h = i + jH$, only the block itself and corresponding proof should be verified. While for any unsigned block with height $h \neq i + jH$, for example, the block b_{i+1}, its proof is the multi-signature of b_{i+H} and the consecutive blocks $(b_{i+1}, \ldots, b_{i+H})$. On the other hand, the larger the threshold ts, the more independent signatures need to be verified. Thereby, the latency is affected by H and ts. We then measure the latency under $ts := 36$, and we find the latency is 93.84 milliseconds for $H := 1$ and 100.81 milliseconds for $H := 8$. Further, under $ts := 80$, the latency is 156.12 milliseconds for $H := 1$ and 165.16 milliseconds for $H := 16$. According to the evaluation results, we concluded that the latency of proof verification fluctuates slightly with increasing H and ts. In other words, the impact of H and ts is negligible. Therefore, our PoF protocol is low-latency.

VIII. Discussion

A Reward-Punishment Mechanism: We explore and sketch a promising reward-punishment mechanism for the committee that can incentivize members to behave actively and honestly, enhancing the security and availability of our protocol. Deploying a PoF contract to implement the mechanism is feasible. Each member seeking to obtain rewards should register on the contract with deposits. A registered member m_i would be financially punished in two cases: (i) Any blockchain user sent to the contract a transaction which contains the signature signed an unfinalized block by m_i, proving m_i's misbehavior. (ii) Any blockchain user challenged the committee which includes m_i to sign a finalized block b via sending a challenge transaction, especially when detecting the committee as inactive, ending up with neither b's proof nor m_i's signature of b provided. The contract is able to obtain the information about finalized blocks, *e.g.*, solidity contracts [8] use the predefined function blockhash(uint $blockNumber$) to obtain the block hash of a specific block, thus the contract can handle received challenges correctly. Note that an honest member can always respond to any challenge with its valid signature of b to avoid punishment. After a committee completes its three duty epochs (*cf.* the committee rotation in Section VI-C), if a member is not punished, it will be rewarded. Thereby, the PoF contract incentivizes honest members and punishes malicious members for misbehavior or inactivity.

Additionally, there are many practical and widely adopted choices for the reward source, fees paid by verifiers which rely on PoF to prevent data spoofing attacks [22], [30], a kind of ERC-20 token exclusively issued for PoF [29], [46], *etc.* For example, we can set up several distributed PoF service nodes to collect and store signature proofs from the committee. We stress that the security of PoF protocol still relies on committee members' signatures instead of the introduced service nodes. When any verifier requests proofs from the service nodes, the service nodes respond with proofs to the verifier only after observing that the verifier has paid service fees via the contract. Additionally, as long as at least one service node is honest, the verifier can obtain proofs. Then the contract manages fees from verifiers as incentives for members.

In the above reward-punishment scheme, it is verifiable that a member receives a reward or punishment only after the contract has validated its membership through Merkle proof. Exemplified by Ethereum, we can deploy the contract on-chain. Recall that each member should register on the contract with deposits. Assuming that a member is the miner of a block b_i, then its deposit transaction carries the block number of b_i, the coinbase transaction tx_{cb} of b_i, and the Merkle proof of tx_{cb}. When executing the deposit transaction, the contract obtains the block hash of b_i through the blockhash(uint $blockNumber$) function. The input $blockNumber$ is the block number carried by the deposit transaction. Then the contract can validate membership based on tx_{cb}, Merkle proof, and the corresponding block hash, and accept the corresponding miner address as member address if the proof is correct and all members complete deposit [27]. Thereafter, the contract

verifiably executes rewards and punishments related to the address.

What's more, the above scheme's costs can be decreased by offloading the verification of Merkle proofs to off-chain systems. Executing the verification in an off-chain TEE with an attestation function, or constructing a zero-knowledge proof (ZKP) for off-chain verification [45], is feasible. We outline a TEE-based scheme since there are many works on TEE-based off-chain execution [15], [19], [36]. As the contract is no longer responsible for validating Merkle proofs, the deposit transaction of a member only carries the block number of b_i to make the contract able to obtain the block hash of b_i via the blockhash function. To prove its membership, the member delivers the block hash of b_i, tx_{cb}, and the Merkle proof of tx_{cb} to a TEE. TEE refers to a trusted execution environment for confidential computation and offers verifiable computation correctness via remote attestation. Additionally, the TEE has registered on the contract with its remote attestations and the public key of a TEE-hold account to verify the TEE's result. The TEE is responsible for validating the Merkle proof and returning its signed validation results to the contract. Thereby, when validating membership, the contract is only responsible for validating that (i) the returned results are validly signed by a registered TEE, and (ii) the returned results correspond to the block hash specified by the deposit transaction, decreasing the on-chain costs.

Besides, each miner just needs to deposit once for unlimited multiple epochs when it honestly behaves. Members are allowed to claim their reward in batches and by need rather than claiming per each epoch or member. Therefore, in the optimistic epoch where committee members honestly behave and all members have deposited in previous epochs, no deposit transaction is needed, and reward transactions happen sometime later. Although reward transactions finally happen with Merkle proof, we note that the claiming can be performed in batches and members can always further reduce the cost of proof verification by the TEE-based method. Suppose committee members misbehave in the pessimistic epoch. In that case, the epoch can lead to two transactions for batch challenging and punishment respectively, where the cost will be paid from the deposits of malicious members. Thereby, the reward-punishment mechanism is low-cost.

In a nutshell, we outline a verifiable and low-cost reward-punishment mechanism to enhance the security and liveness of PoF protocol. This is only for demonstrating the feasibility of introducing an incentive mechanism for PoF protocol, as a rigorous design and analysis of incentive mechanism for PoF is outside the scope of this paper. We leave the design of incentive mechanisms, such as the specification of a challenge transaction, the collateral costs for members' deposits and potential attacks, to future work.

PoF protocol for TEE-blockchain system: PoF protocol is well adapted to isolated nodes such as TEE, we would describe a PoF protocol for TEE-blockchain systems which have become a research hotspot. In the systems, a TEE typically executes confidential computation based on blockchain data and delivers computing results to the blockchain [15], [17], [37]. However, the TEE is an isolated computation environment whose network stack is fully controlled by the TEE executor, which can modify, delay, and drop messages sent to the TEE. If the TEE accepts unfinalized data fed by the executor, it would output incorrect computing results, destroying the trustworthiness of TEE computation results.

To resist data spoofing attacks, the TEE can choose to be a verifier in PoF. Concretely, recall that PoF committee members are determined by the location transaction tx_c recorded in the block b_c, and we have assumed that verifiers can obtain b_c. In PoF global setup phase, the TEE loads a verification program based on b_c and several blocks preceding b_c. Thereafter, when receiving blockchain data from the executor, the TEE only accepts the data corresponding to valid proofs passing the verification program. Note that the executor can refuse to provide the TEE with (i) the several blocks or (ii) finalized data with valid proof, and only feed TEE unfinalized data. Without (i), the TEE rejects to accept any blockchain data since it has not loaded a verification program; Without (ii), the TEE also rejects to accept any data since no data can pass the verification program. Thereby, even if the TEE interacts only with a malicious executor, it can resist data spoofing attacks through PoF.

Future Works: In this paper, we assume a static adversary that corrupts miner nodes before they become the miners of the blocks in BC. However, in practice, an adversary may concentrate its efforts on corrupting members (miners) selected by the location transaction. Thereby, in future works, we will extend the static-adversary assumption to an adaptive-adversary assumption, which allows an adversary to corrupt arbitrary miners at any point without breaking the common prefix, chain growth, and chain quality properties.

IX. Conclusion

PoF is a protocol used to guard against data spoofing attacks, and we describe a chain-agnostic, non-interactive, non-authority-involved, and effective PoF scheme in this paper. We evaluate the scheme and find that the latency of proof generation and proof verification are both negligible in milliseconds, proving the effectiveness of the scheme.

References

[1] *Alchemy—The Web3 Development Platform*. Accessed: Sep. 6, 2024. [Online]. Available: https://www.alchemy.com/

[2] *Cost of a 51% Attack*. Accessed: Sep. 6, 2024. [Online]. Available: https://gobitcoin.io/tools/cost-51-attack/

[3] *Ethereum API*. Accessed: Sep. 6, 2024. [Online]. Available: https://www.infura.io/

[4] *Ethereum Beacon Chain Checkpoint Sync Endpoints*. Accessed: Sep. 6, 2024. [Online]. Available: https://eth-clients.github.io/checkpoint-sync-endpoints/

[5] *Ethereum Glossary|Ethereum*. Accessed: Sep. 6, 2024. [Online]. Available: https://ethereum.org/en/glossary/

[6] *Quicknode | The Blockchain Development Platform*. Accessed: Sep. 6, 2024. [Online]. Available: https://www.quicknode.com/

[7] *Sharding*. Accessed: Sep. 6, 2024. [Online]. Available: https://web.archive. org/web/20190504131341/https://vitalik.ca/files/Ithaca201807_Sharding.pdf

[8] *Solidity*. Accessed: Sep. 6, 2024. [Online]. Available: https://docs.soliditylang.org/en/latest/units-and-global-variables.html

[9] B. Alangot, D. Reijsbergen, S. Venugopalan, and P. Szalachowski, "Decentralized lightweight detection of eclipse attacks on Bitcoin clients," in *Proc. IEEE Int. Conf. Blockchain*, Nov. 2020, pp. 337–342.

[10] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver, "Stressing out: Bitcoin 'stress testing,"' in *Proc. Int. Workshops*, Feb. 2016, pp. 3–18.

[11] M. Bellare and S. K. Miner, "A forward-secure digital signature scheme," in *Proc. 19th Annu. Int. Cryptol. Conf.*, Aug. 1999, pp. 431–448.

[12] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, Oct. 2006, pp. 390–399.

[13] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1521–1538.

[14] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2003, pp. 416–432.

[15] R. Cheng et al., "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. 2019 IEEE Eur. Symp. Secur. Privacy*, Jun. 2019, pp. 185–200.

[16] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an unfair world: Fair multiparty computation from public bulletin boards," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 719–728.

[17] P. Das et al., "FastKitten: Practical smart contracts on Bitcoin," in *Proc. USENIX Secur. Symp.*, 2019, pp. 801–818.

[18] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Proc. 37th Annu. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol.*, May 2018, pp. 66–98.

[19] T. Frassetto et al., "POSE: Practical off-chain smart contract execution," 2022, *arXiv:2210.07110*.

[20] S. Gao, Z. Li, Z. Peng, and B. Xiao, "Power adjusting and bribery racing: Novel mining attacks in the Bitcoin system," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 833–850.

[21] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin backbone protocol: Analysis and applications," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, Apr. 2015, pp. 281–310.

[22] J. He, G. Zhang, J. Zhang, and R. Zhang, *An Economic Model of Blockchain: The Interplay Between Transaction Fees and Security*, document SSRN 3616869, 2020.

[23] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 129–144.

[24] G. O. Karame, E. Androulaki, and S. Capkun, "Two Bitcoins at the price of one? Double-spending attacks on fast payments in Bitcoin," *Cryptol. EPrint Arch.*, vol. 1, pp. 1–24, Jun. 2012.

[25] R. M. Kouame, K. Saito, and H. Inaba, "Detection of spam transactions in blockchain by graph analysis," in *Proc. IEEE 11th Global Conf. Consum. Electron. (GCCE)*, Oct. 2022, pp. 306–310.

[26] P. Lafourcade and M. Lombard-Platet, "About blockchain interoperability," *Inf. Process. Lett.*, vol. 161, Sep. 2020, Art. no. 105976.

[27] R. Lan, G. Upadhyaya, S. Tse, and M. Zamani, "Horizon: A gas-efficient, trustless bridge for cross-chain transactions," 2021, *arXiv:2101.06000*.

[28] K. Li, Y. Wang, and Y. Tang, "DETER: Denial of Ethereum txpool sERvices," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 1645–1667.

[29] A. Lisi, A. De Salve, P. Mori, L. Ricci, and S. Fabrizi, "Rewarding reviews with tokens: An Ethereum-based approach," *Future Gener. Comput. Syst.*, vol. 120, pp. 36–54, Jul. 2021.

[30] Y. Liu, Z. Fang, M. H. Cheung, W. Cai, and J. Huang, "An incentive mechanism for sustainable blockchain storage," *IEEE/ACM Trans. Netw.*, vol. 30, no. 5, pp. 2131–2144, Oct. 2022.

[31] Z. Liu and H. Zhu, "Fighting Sybils in airdrops," *arXiv:2209.04603*, 2022.

[32] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, and P. Jayachandran, "Blockchain meets database: Design and implementation of a blockchain relational database," 2019, *arXiv:1903.01919*.

[33] J. Niu, F. Gai, M. M. Jalalzai, and C. Feng, "On the performance of pipelined HotStuff," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[34] R. Pass and E. Shi, "FruitChains: A fair blockchain," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2017, pp. 315–324.

[35] M. Platt and P. McBurney, "Sybil in the haystack: A comprehensive review of blockchain consensus mechanisms in search of strong Sybil attack resistance," *Algorithms*, vol. 16, no. 1, p. 34, Jan. 2023.

[36] Q. Ren, H. Liu, Y. Li, and H. Lei, "Demo: Cloak: A framework for development of confidential blockchain smart contracts," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 1102–1105.

[37] Q. Ren et al., "Cloak: Transitioning states on legacy blockchains using secure and publicly verifiable off-chain multi-party computation," in *Proc. 38th Annu. Comput. Secur. Appl. Conf.*, Dec. 2022, pp. 117–131.

[38] A. W. Roscoe, P. Antonino, and J. Lawrence, "Embedding reverse links in a blockchain," in *Proc. Workshop Encouraging Building Better Blockchain Secur.*, 2022, pp. 1–18.

[39] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang, "Fine-grained, secure and efficient data provenance on blockchain systems," *Proc. VLDB Endowment*, vol. 12, no. 9, pp. 975–988, May 2019.

[40] O. Sanda, M. Pavlidis, S. Seraj, and N. Polatidis, "Long-range attack detection on permissionless blockchains using deep learning," *Expert Syst. Appl.*, vol. 218, May 2023, Art. no. 119606.

[41] S. Siddiqui, V. Srivastava, R. Maheshwari, and S. Gujar, "QuickSync: A quickly synchronizing PoS-based blockchain protocol," 2020, *arXiv:2005.03564*.

[42] P. Swathi, C. Modi, and D. Patel, "Preventing Sybil attack in blockchain using distributed behavior monitoring of miners," in *Proc. 10th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, 2019, pp. 1–6.

[43] Y. Tang, K. Li, Y. Wang, and S. B. Somuncuoglu, "Scalable log auditing on private blockchains via lightweight log-fork prevention," in *Proc. 4th Workshop Scalable Resilient Infrastructures Distrib. Ledgers*, Dec. 2020, pp. 1–4.

[44] H. Wang, C. Xu, C. Zhang, J. Xu, Z. Peng, and J. Pei, "VChain+: Optimizing verifiable blockchain Boolean range queries," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, May 2022, pp. 1927–1940.

[45] M. Westerkamp and J. Eberhardt, "ZkRelay: Facilitating sidechains using zKSNARK-based chain-relays," in *Proc. IEEE Eur. Symp. Security Privacy Workshops*, Jul. 2020, pp. 378–386.

[46] S. Yoo, "How to design the token reinforcement based on token economy for blockchain model," *Int. J. Adv. Culture Technol.*, vol. 8, no. 1, pp. 157–164, 2020.

[47] C. Zhang, C. Xu, J. Xu, Y. Tang, and B. Choi, "Gem$\hat{2}$-tree: A gas-efficient structure for authenticated range queries in blockchain," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, May 2019, pp. 842–853.

[48] Y. Zhao, "Practical aggregate signature from general elliptic curves, and applications to blockchain," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Jul. 2019, pp. 529–538.

[49] Y. Zhu, Z. Zhang, C. Jin, A. Zhou, and Y. Yan, "SEBDB: Semantics empowered blockchain database," in *Proc. 35th Int. Conf. Data Eng.*, 2019, pp. 1820–1831.