FAV-BFT: An Efficient File Authenticity Verification Protocol for Blockchain-Based File-Sharing System

Shuai Su[1,2(✉)], Chi Chen[1,2], and Xiaojie Zhu[3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Science, Beijing 100093, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
{sushuai,chenchi}@iie.ac.cn
[3] Abu Dhabi University, Abu Dhabi, UAE
xiaojie.zhu@adu.ac.ae

Abstract. Compared with traditional file-sharing system, the blockchain-based file-sharing system shows its superiority, such as electronic money incentive mechanism, decentralisation, information tamper resistance and so on. Benefiting from those properties, it has attracted tons of users to participate in blockchain-based file-sharing and eventually forms an indestructible electronic library. However, with such a huge amount of files, the problem of file authenticity verification is still not resolved. This paper attempts to address the challenge of file authenticity verification for blockchain-based file-sharing system, specifically, verifying that the file is really stored by the claimer and needed by the file-downloader before the file is downloaded. We propose an efficient file authenticity verification protocol, named File Authenticity Verification Byzantine Fault Tolerant (FAV-BFT). We first apply Verifiable Delay Function to bind the shared file, and then reconstruct it to a challenge-response interactive protocol for file-sharing, and finally embedded with Byzantine Fault Tolerant protocol. Due to the construction, with 2/3 of participants are honest, FAV-BFT can correctly prove how long a file has been stored and whether a file meets the requirement of the file downloader. Moreover, since all the file content is processed by hash function before transformation, FAV-BFT protects the shared-file from content disclosure during the verification process without trusted third parties.

Keywords: Blockchain · File-sharing · Authenticity verification · Efficient

1 Introduction

Peer-to-peer [1] systems allow users to share information in distributed environments because of their scalability and efficiency. Currently, the most popular P2P

H. Gao et al. (Eds.): CollaborateCom 2022, LNICST 460, pp. 455–474, 2022.
https://doi.org/10.1007/978-3-031-24383-7_25

file-sharing systems include BitTorrent, eDonkey, eMule [2], μTorrent, Napster, KaZaA [3] and so on. All these file-sharing systems requires file downloaders to search files through the central server or a *Distributed Hash Table* (DHT), and download shared-files from multiple file-sharers. The authenticity verification of the file (that is, before downloading the file, file-downloader can verify that the file is indeed the one that file-downloader needs) relies on the honesty of the central server or correctness of DHT. In practical, the central server or DHT may face various attacks, such as single point failure, information tampering, malicious nodes, and so on. Therefore, it is risky to simply rely on the trusted central server or DHT to verify the authenticity of the shared-files.

The development of Blockchain technology provides an inspiring solution to these problems. By applying blockchain technology to file-sharing system, the properties of blockchain, such as decentralization, data tamper proof and malicious peer tolerance, can solve the problems encountered by traditional file-sharing systems. Each node can use bounded resources to prove the existence of shared-files. Based on these file-existence proofs, file-sharers will obtain token rewards after sharing files. Therefore, they will form a competitive relationship in order to obtain tokens. At present, the research of file-sharing system based on blockchain is very popular. Many mature blockchain-based file-sharing mechanisms have been proposed and applied, such as BTFS [6], BlockIPFS [7], Filecoin [10], Siacoin [8], Storj [9], and so on. The goal of these systems is to use the reward mechanism to attract more users to participate in file-sharing, so as to form an indestructible electronic file-sharing library.

To maintain the quality of shared files, blockchain-based file-sharing system provides resource proof. Many consensus protocols are proposed to generate the proof, such as *Proof of Work* (PoW), *Proof of Stake* (PoS) [15], *Proof of Space* (PoSpace) [21], etc. However, these consensus protocols have their limitations. For example, PoW wastes a lot of electric power, PoS has the risk of being monopolized by oligarchs, PoSpace occupies large amount of space that is unrelated to the shared files. Moreover, these consensus protocols do not consider the file content. In such case, a user can claim a wrong file since the user only needs to show its existence instead of content. If the file content can be used as the evidence of resource proof, it can not only reduce resource consumption, but also verify the authenticity of the files. Therefore, in the blockchain based file-sharing community, the requirement of designing a practical solution that can not only prove the file existence with limited resource-consumption, but also prove the authenticity of the file.

For the proof of file existence, the existing solutions are inefficient, such as proof of sequential work [23,24], time-lock puzzle [27–29], and *Verifiable Delay Function* (VDF) [25]. Their solutions feature the time-consuming proof generation, not supported by parallel computing. In addition, for the file authenticity verification, all the previous work only considers the size of the stored file, such as PoSpace [21], *Proof of Space-Time* (PoST) [11,22]. They only support to prove the file size which is irrelevant to the file content. In order to solve above challenge, we observe that a feasible solution to this problem is to propose a mutual challenge mechanism between nodes. Fortunately, *Byzantine Fault-Tolerant* (BFT)

protocol supporting state-machine replication, uses mutual challenge mechanism to tolerate malicious nodes. If above storage proof can be properly embedded into BFT, the problem of file authenticity verification for blockchain-based file-sharing system can be addressed. Particularly, the mutual challenge mechanism is responsible for matching the file content while the storage proof takes the responsibility of proving file existence.

In short, the contribution of this paper is summarised as follows.

1 VDF is adapted for file-sharing. It is bound together with the shared file content, and reconstructed to a challenge-response interactive protocol.
2 We proposes an efficient file authenticity verification protocol for blockchain-based file-sharing system, FAV-BFT. It novelly combines adapted VDF with *Reliable Broadcast* (RBC) algorithm of BFT to verify the authenticity of files. With more than two-thirds of participants are honest, this protocol can correctly verify the file authenticity. Meanwhile, the process of file authenticity verification does not disclose the content of shared files and does not rely on any trusted third party.
3 We evaluate the performance and security of FAV-BFT in theory and practice. The theoretical analysis results show that, compared with the protocol of Filecoin, the challenge-response protocol based on VDF has lower computation complexity in the prover and verifier proof phases. In addition, we also test the FAV-BFT, and the experimental result indicates that the FAV-BFT can indeed verify the authenticity of the file and the efficiency is higher than HoneyBadgerBFT [32].

The remainder of paper is organized as follows. Section 2 presents the related work. Subsequently, the specific construction of the FAV-BFT is described in Sect. 3. Section 4 analyzes the performance and security of FAV-BFT. In Sect. 5, the experiments and test results are given. Finally, Sect. 6 concludes this paper.

2 Related Work

At present, many researches on file authenticity verification for blockchain-based file-sharing system have emerged. Related work mainly includes two aspects, one is to directly combine distributed file-sharing system with existing mature blockchain mechanisms, and the other is to develop specific protocols oriented toward file-sharing scenarios.

For the first aspect, researches mainly focus on how to use the existing consensus protocol of blockchain to complete file storage and authenticity verification. For example, Sari, L. et al. and Khatal, S. et al. proposed FileTribe [4] and FileShare [5], both of them employs IPFS and Ethereum smart contract to govern, manage shared-files. *BitTorrent File System* (BTFS) [6] develops a file-sharing protocol utilizing the Tron Blockchain and the BitTorrent ecosystem. The consensus protocol of Tron is *Delegated-Proof-of-Stake* (DPoS) which is not associated with file content. Sia [8] combine cloud storage with the existing mature blockchain technology to develop blockchain-based cloud storage platform. Sia

uses file contracts to restrict storage providers to store files for customers, and uses *Proof of Storage* (PoS) to verify whether files are stored continuously. These researches mainly study how to use the existing consensus protocol of blockchain to complete the storage proof of files. The consensus protocols are not associated with the file content. Users need to provide the proof of bounded resources while saving the shared files, which may cause a waste of resources-consumption.

For the second aspect, researches mainly focus on proposing specific protocols which can prove that files have been stored for a period of time. For example, Storj [9] uses *Proof of Retrievability* (PoR) [20] to promise that a verifier sends a file to a prover and later to request a proof that the prover really stored the file. However, it is difficult to ensure that files are stored continuously in Sia and Storj. Based on *InterPlanetary File System* (IPFS) [12], "Filecoin" [10] developed *Proof-of-SpaceTime* (PoST) and *Proof-of-Replication* (PoRep) [13] to prove that the prover stores the file for an elapsed time. However, the implementation of PoRep and PoSt rely on cryptographic machinery (such as zk-SNARKs [14,16,17] and seal operation) requiring heavy computing resources. S. Dziembowski et al. proposed *Proof-of-Space* (PoSpace) [21] to replace *Proof of Work* (PoW), which prefer storage instead of computing resource to complete the qualification of mining. This consensus protocol is more energy-saving for the reason that storage consumes less cost than computing. However, PoSpace is not associated with file content, it cannot verify the authenticity of file. Benet et al. proposed *Proof-of-Replication* (PoRep) [13] which allows prover to prove that he/she is storing multiple redundant copies of a file, but it is hard to ensure that files are stored continuously. T. Moran et al. proposed *Proofs of Space-Time* (PoST) [11,22], which allow prover to convince to others that amount of "space-time" resource (storing data-space over a period of time) have been spend. PoST is also not associated with file content and cannot verify the authenticity of file. M. Mahmoody et al. and B. Cohen et al. proposed proofs of sequential work [23,24] that can prove to others that sequential computational work have been spent to solve a puzzle. D. Boneh el al proposed *Verifiable Delay Functions* (VDF) [25], which can prove that a period of time has been consumed. VDF algorithm requires a specified number of sequential steps to produce a unique output that can be efficiently and publicly verified. VDF algorithm can hardly be accelerated by parallel computing. K. Pietrzak et al. proposed a simple VDF [26] which make time-lock puzzle [27–29] to be publicly verifiable. B. Wesolowski el al proposed an efficient VDF [31], which has higher execution efficiency than previous VDF. The above VDF and proofs of sequential work can limit resource-consumption but not associated with file content, they cannot verify the authenticity of file. Shuai Su et al. proposed a efficient File-Sharing interactive verification protocol VoFSQ [35], it used storage-resources to prove that a particular file has indeed been stored for a period of time, but it is a protocol verified by both parties rather than multiple parties. The decentralization cannot be achieved.

In order to realize file authenticity verification, it is necessary to introduce a multi-party mutual verification mechanism.*Practical Byzantine Fault Tolerance* (PBFT) [18,19] uses mutual verification mechanism to tolerate malicious node.

PBFT has higher requirements for communication reliability, so it is weak practicability. HoneyBadgerBFT [32] is the first practical asynchronous BFT protocol, which guarantees liveness without making any timing assumptions. Dumbo [33] improves the consensus efficiency by implementing the *Asynchronous Binary Agreement* (ABA) algorithm through the election committee. Dumbo-MVBA [34] further improves the consensus efficiency by reducing the number of committees to a constant. The above BFT protocols are not associated with file content and cannot verify the authenticity of file. Table 1 shows details of these incentive mechanisms.

Table 1. Overview of the file authenticity verification protocols for Blockchain-based File-Sharing system

System, protocol	Description	Problems
FileTribe [4] FileShare Sia [5,8]	Combine Ethereum with file-sharing system	May cause a waste of resources
DPoS (in BTFS [6])	Prove that users own certain stake	Super nodes may appear
PoS, PoR (in Storj [9])	Prove that the file is stored	Hard to ensure that files are stored continuously
PoRep, PoST (in Filecoin [10])	Prove that files are continuously stored	Low efficiency and relies trusted third-party
PoSpace [21]	Prove that certain amount of space is occupied	Cannot verify the authenticity of file
PoRep [13]	Prove that the replication of file is stored	Hard to ensure that files are stored continuously
PoST [11,22]	Prove "space-time" have been spent	Cannot verify the authenticity of file
Proofs of sequential work [23,24]	Prove time-consuming sequential work has been done	Cannot verify the authenticity of file
VDF [25,31]	Prove time-consuming function is executed	Cannot verify the authenticity of file
VoFSQ [35]	Prove file has been stored for a period of time	Not decentralized
PBFT [18,19]	Can tolerate malicious node	Cannot verify the authenticity of file and weak practicability
HoneyBadgerBFT [32]	Asynchronous and high efficiency	Cannot verify the authenticity of file
Dumbo [33], Dumbo-MVBA [34]	Synchronous and high efficiency	Cannot verify the authenticity of file

In summary, there lack such protocol which can not only prove that the file is indeed stored for a period of time, but also prove that the file is really needed by the file-downloader on the premise of saving resource consumption. Therefore, this paper aim to design a protocol to solve these problems.

3 The Construction of FAV-BFT

This section mainly introduces the details of a consensus protocol, which is called *File-Authenticity Verification Byzantine Fault Tolerant* (FAV-BFT). FAV-BFT contains the following steps: 1. associating identity and file contents with VDF. 2. building a challenge-response protocol based on VDF. 3. verifying the file-consistence.

3.1 Associating Identity and File Contents with VDF

In order to verify the authenticity of the file, it is necessary to take file owner's identity and file content as the input of VDF [25]. There are three phases in this protocol: *Setup*, *Eval* and $Verify$. In *Setup* phase, initialization variables are generated. In *Eval* phase, the identity and file content are associated with the VDF process of generating proof. In $Verify$ phase, the proof is verified. The improved VDF is a triple of algorithms as follows:

Protocol VDF associated with identity and file content

- $Setup(\lambda, t) \rightarrow \mathbf{pp} = (ek, pk)$. λ is a security parameter, t is difficulty, **pp** is a public parameter which contains an evaluation key ek and a verification key vk.
- $Eval(ek, x, id) \rightarrow (\pi)$. x stands for the file content or file hash value. id stands for the identity of file owner. $Eval$ takes x as input and produces proof π. Algorithm $Eval$ is time-consuming and difficult to be accelerated by parallel computing. Suppose that the fastest time of this step is t when using the fastest computing resources.
- $Verify(vk, x, id, \pi) \rightarrow \{Accept, Reject\}$. Algorithm $Verify$ takes x, id, π as input and outputs *Accept* or *Reject*. $Verify$ is much faster than $Eval$.

3.2 Building a Challenge-Response Protocol Based on VDF

The above mentioned VDF which can associate with identity and file contents is difficult to respond in real time when receiving the dynamic challenge. In order to achieve the purpose of real-time verification between nodes, it is necessary to build a challenge-response interactive protocol based on VDF. File-sharers verify each other through this protocol to check whether the same file has been preserved for a period time. The interactive protocol consists of three phases: initialization phase, proof phase, and verification phase. The two parties of the interactive protocol are prover P and verifier V. P can be viewed as a file-sharer who want to provide proof of file storage and authenticity, V acts as file-sharer who verify P. The process of VDF-based challenge-response protocol is shown in Fig. 1.

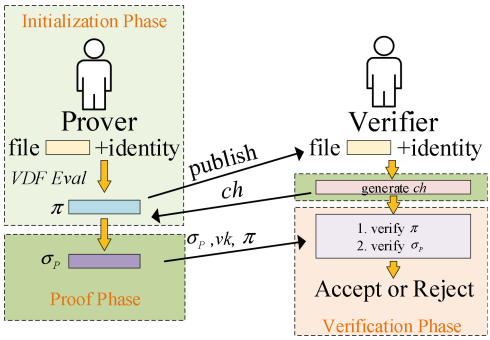

Fig. 1. A challenge-response protocol based on VDF.

Initialization Phase: P calculates the hash value of the verification key vk which generated in the VDF algorithm to generate identity ($id_P = hash(vk)$). Then P generates the VDF proof π according to the file-sharer's identity ($id_P \in \{0,1\}^*$) and file content f. The value of t can be adjusted dynamically which determines the difficulty of VDF. At the conclusion of this phase, P output proof ($\pi \in \{0,1\}^*$, respectively).

$$(\pi) \leftarrow \langle Eval_{init}\,(ek, id_P, t, f)\rangle \tag{1}$$

π will be used to generate the response within a certain time at the challenge-response phase, therefore, P should preserve π in the local storage. P broadcasts id_P and states that initialization phase has been completed.

In the initialization phase, P runs the VDF to generate the evidence-data. The VDF algorithm requires running given sequential steps and can hardly be accelerated by parallel computing. This progress is time consuming and has real-world cost.

Proof Phase: When V receives id_P from P, V sends a challenge $ch \in \{0,1\}^*$ to P. P should generate a response efficiently. P concatenates the value of id_P, f, π, ch and calculates its hash value. The calculation performed by P is as follows:

$$(\sigma_P) \leftarrow \langle hash\,(id_P||f||\pi||ch)\rangle \tag{2}$$

P sends (σ_P, vk, π) to V. Since P cannot obtain the challenge value ch in advance, she/he must run it after receiving the challenge value. If P does preserve the evidence-data, this process is efficient. Otherwise, P must rerun the initialization phase to generate the evidence-data. P will prefer continuous evidence-data preservation rather than dynamic evidence-data generation. As long as the evidence-data is preserved continuously, P can make the right response to V with a small computational cost. Preserving the evidence-data continuously should be cheaper than the cost of re-generating using computing resources.

Verification Phase: V owns the same file as the P. When V receive (σ_P, vk, π), V will recalculate the hash value according to the values of vk, f, π, ch and

compare it with the value of σ_P received. Then V will verify the correctness of π. The V will output either accept or reject ($out_v \in \{0,1\}$, where 0 is interpreted as "reject" and 1 is "accept") based on the results of the two verifications.

$$(out_v) \leftarrow \left\langle \begin{array}{c} \sigma_P \stackrel{?}{=} (\sigma'_P) \leftarrow hash(hash(vk)||f||\pi||ch) \\ Verify\,(vk, f, hash(vk), \pi) \end{array} \right\rangle \tag{3}$$

This paper lists a challenge-response protocol based on Wesolowski's VDF [31]. All file-sharers who save the same file can verify with each other in order to compete for file-sharing qualification and win token reward. The description of this protocol is shown below:

Protocol A challenge-response protocol based on Wesolowski's VDF

Public Parameters: λ: security parameter, $N = p \cdot q$ with p and q two primes and N a λ-bits number, difficulty degree $\tau, t \in \mathbb{N}$.
Initialization Phase:(Performed by P)
Input: file content $m \in \{0,1\}^*$, identity $id \in \{0,1\}^*$.

1 $x \leftarrow H(m||id)$. (H stands for hash algorithm)
2 $y \leftarrow x$.
3 **for** (1 to τ)
  (a) $y \leftarrow y^2 \ mod\ N$.
4 $l \leftarrow H_{prime}(x + y)$.($H_{prime}(x)$ stands for the closest prime number larger or equal to $H(x)$)
5 $\pi = x^{\lfloor 2^\tau / l \rfloor} \ mod\ N$.
6 Publish the metadata of shared file, id, initialization outcome.

Proof Phase:(Performed by P)

Upon receiving a challenge ch from the V:

1 calculate $\sigma_P = H(id||m||\pi||ch)$
2 Send σ_P,τ,id,π,l,N to V.

Verification Phase:(Performed by V)
Generate a random challenge ch and send it to P. Wait to receive σ_P,τ,id,π,l,N.

1 $\hat{x} \leftarrow H(m||id)$.
2 $r \leftarrow 2^\tau \ mod\ l$.
3 $\hat{y} \leftarrow \pi^l \cdot \hat{x}^r \ mod\ N$.
4 $\hat{l} \leftarrow H_{prime}(\hat{x} + \hat{y})$
5 $\hat{\sigma_P} = H(id||m||\pi||ch)$
6 if (l==$\hat{l}$&&$\hat{\sigma_P} == \sigma_P$)
  (a) return accept
7 else
  (a) return reject

3.3 Verifying the File-Consistence

This section formally introduces *Verification of File-Consistence* (VoFC) which can verify file authenticity on the premise that more than 2/3 nodes are honest.

In blockchain-based file-sharing scenario, multiple nodes will participate in file authenticity verification, which will form a network of N designated nodes (with distinct well-known identities (P_0 through P_{N-1})). The nodes utilize preserved files to generate evidence-data. By using the evidence-data as input, their goal is to reach common agreement that these files have been preserved for a period of time and these files is really needed by the file-downloader on the premise that more than 2/3 nodes are honest. All the verification processes will not depend on the central server and do not disclose any privacy information about file content. This protocol particularly matches the deployment scenario of a "permissioned blockchain" where file-consistence verification can be submitted by arbitrary nodes, but the nodes responsible for carrying out the protocol are fixed.

VoFC combines VDF with *Reliable Broadcast* (RBC) algorithm of HoneyBadgerBFT [32], so that multiple nodes should to challenge with each other to complete file-consistence verification using storage resources. At the same time, VoFC also has the characteristics of high throughput, practical asynchronous and so on. There are multiple nodes in VoFC. The node initiating file authenticity verification can be viewed as a file-downloader, other nodes can be regarded as file-sharers participating in the verification. There are five phases in the VoFC, namely challenge, echo-challenge, val, echo, and ready. Each node needs to perform these phases.

1) Challenge phase

Before file authenticity verification, all nodes need to utilize VDF algorithm to generate evidence-data. This step consumes computing resources and a period of time. Nodes prefer to keep these evidence-data and wait for challenges. File-downloader generates random string ch and broadcasts m^{chal} message to the whole network.

$$(m^{chal}) \leftarrow \langle \langle \text{“}challenge\text{”}, id_d, ch \rangle, sign \rangle \tag{4}$$

where “*challenge*” is the request mark, id_d is the unique identity of file-downloader, $sign$ is the digital signature of this request. When the node receives the ch value, it will execute the val and echo-challenge phase at the same time. Echo-challenge phase is to enable all nodes to challenge each other. The process of directly executing Val phase is shown in Fig. 2. The process of executing Val phase after echo-challenge phase is shown in Fig. 3.

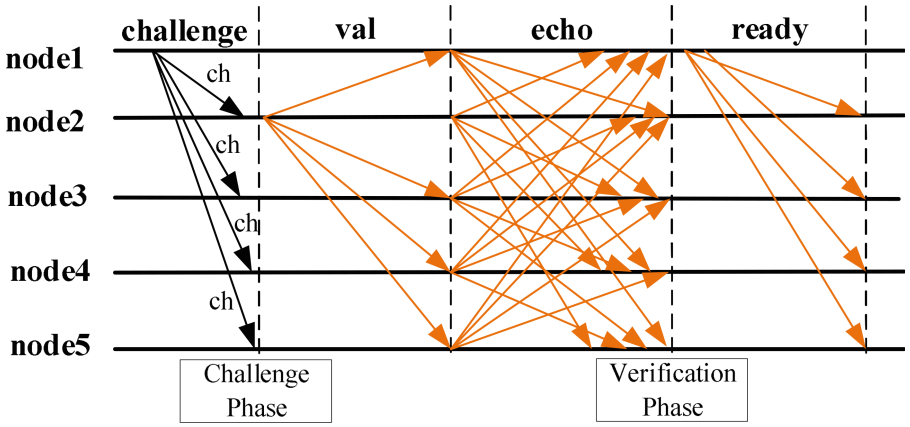

Fig. 2. The VoFC process of directly executing val phase.

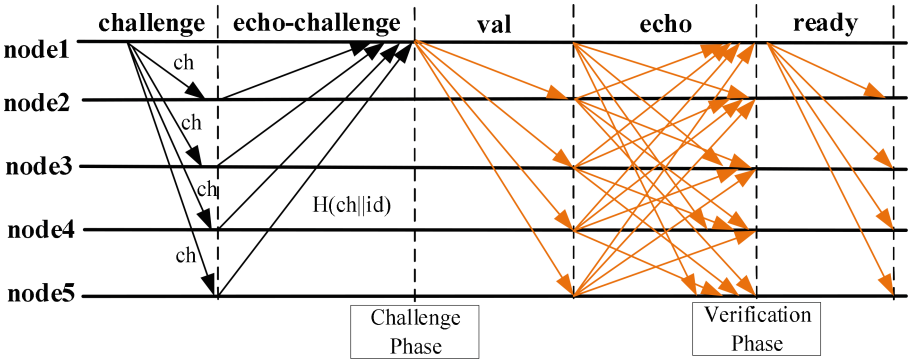

Fig. 3. The VoFC process of executing val phase after echo-challenge phase.

2) Echo-challenge phase

After receiving the message m^{chal} from file-downloader, the node verifies the digital signature. If the verification is passed, the node runs challenge phase of VDF to generate response m^{val} according to the challenge value ch, the implementation is described in val phase below. Then the node generates echo-challenge message $m^{echo-ch}$.

$$ech = hash\left(ch||id_d||id_f\right) \tag{5}$$

$$\left(m^{echo-ch}\right) \leftarrow \langle\langle \text{“}echo - chal\text{”}, id_s, ech\rangle, sign\rangle \tag{6}$$

where “$echo - chal$” is the mark of this phase, id_s is the unique identity of file-sharer. The node sends $m^{echo-ch}$ to other nodes.

3) Val phase

When receives m^{chal} or $m^{echo-ch}$, the node runs challenge phase of VDF to generate response m^{val}.

$$\sigma_P = hash\left(id_s||f||\pi||(ch\ or\ ech)\right) \tag{7}$$

$$(val) \leftarrow (\sigma_P||vk||\pi||id) \tag{8}$$

The node divides val into $N-f$ blocks (f is the maximum number of malicious nodes, less than $N/3$), and then uses erasure coding scheme to expend them to N blocks(that is, as long as $N-f$ blocks are received, the whole message can be recovered), and utilizes these nodes as leaf to build a Merkle tree. Denote h to represent the root node of this Merkle tree. The node sends m_j^{val} to node N_j, where b_j is the jth Merkle tree branch. Construct m_j^{val} and sent it to each node N_j. "ch" or "ech" indicates that this message is a response to the ch or ech.

$$(m_j^{val}) \leftarrow \langle\langle \text{“}val\text{”}, id_s, h, b_j, s_j, \langle \text{“}ch\text{”}\ or\ \text{“}ech\text{”}, ch\ or\ ech\rangle\rangle, sign\rangle \tag{9}$$

4) Echo phase

When receives m_j^{val}, the node needs to verifies the digital signature and check that b_j is a valid Merkle branch from leaf s_j to root h, otherwise discard. When all verification passes, the node constructs an m_j^{echo} message and sends it to other nodes.

$$(m_j^{echo}) \leftarrow \langle\langle \text{“}echo\text{”}, id_s, h, b_j, s_j, \langle \text{“}ch\text{”}\ or\ \text{“}ech\text{”}, ch\ or\ ech\rangle\rangle, sign\rangle \tag{10}$$

5) Ready phase

When receives m_j^{echo} message, the node verifies the corresponding Merkel path and signature value. If the verification passed, the node constructs m_j^{ready} message and sends it to other nodes.

$$(m_j^{ready}) \leftarrow \langle\langle \text{“}ready\text{”}, h, \langle \text{“}ch\text{”}\ or\ \text{“}ech\text{”}, ch\ or\ ech\rangle\rangle, sign\rangle \tag{11}$$

When node receives $N-f$ matching $ready(h)$ messages, he/she can use $N-f$ corresponding m^{val} messages to recover the original val message. Then, the node can perform the verification phase to verify the correctness of σ_P and π which contains val message. If the verification is passed, the corresponding Asynchronous Binary Byzantine Agreement (ABA) [32] (ABA is the next process to be executed after RBC algorithm in HoneyBadgerBFT) binary bit position will be set 1, otherwise it will be set 0.

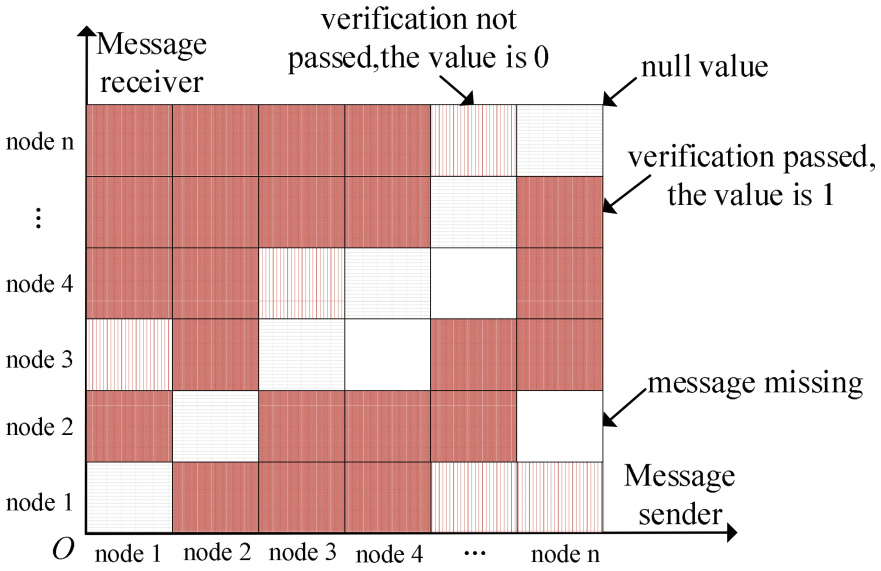

Fig. 4. ABA matrix.

After the above steps are completed, every node will receive the feedback to the corresponding challenge from other nodes, and generate a 0–1 matrix, as shown in Fig. 4. The x-axis is the message sender and the y-axis stands for the message receiver. When the verification is passed, the value is 1, otherwise the value is 0. Next, each node will execute ABA protocol, check whether the number of elements with 1 in each node matrix is greater than $2 * N/3$, if it is, set all element to 1. This step is the same as that in HoneyBadgerBFT.

Finally, every node will get a consistent matrix, showing that these nodes have preserved the same file and have afforded compute-time cost. If more than 2/3 nodes are honest, it can be sure that the shared file is exactly which file-downloader wants.

4 Evaluation of FAV-BFT

This section lists the theoretical performance analysis of the FAV-BFT, which include computing, time-consuming, storage and communication consumption. Then, we compare its differences with multiple BFT protocols, and finally analyzes its security.

4.1 Performance Evaluation

The performance evaluation includes memory M, time T, storage S, communication consumption.

Challenge-Response Protocol Based on VDF

Initialization Phase. In this phase, P have to run a computing and time consuming algorithm to generate the evidence-data. For Wesolowski's VDF, this algorithm is dominated by solving a puzzle, that is x^{2^τ}, which is conjectured to require τ sequential squarings. If $\phi(N) = (p-1)(q-1)$ is known, this algorithm can be accelerated, for the reason that:

$$y := x^{2^\tau} \ mod \ N = x^{2^\tau \ mod \ \phi(N)} \ mod \ N \tag{12}$$

It is very difficult to get $\phi(N)$ from N, so this algorithm needs to be calculated as follows:

$$x \to x^2 \to x^{2^2} \to x^{2^3} \to ... \to x^{2^\tau} \ mod \ N \tag{13}$$

There are many ways to accelerate modular multi-exponentiation algorithm [36,37]. On the premise of multithreading and s cores, the algorithm can be accelerated to at least $\frac{2\tau}{s \cdot log(\tau)}$. Computing the proof $\pi = x^{\lfloor 2^\tau / l \rfloor}$ requires a total of $O(\tau/log(\tau))$ group multiplications [30,31]. Let $t_{mul}(x^2)$ and $r_{mul}(x^2)$ denote running time and occupied memory of $\{x^2 : x \in [0, N]\}$. Using $t_{mod}(y \ mod \ N)$ and $r_{mod}(y \ mod \ N)$ to denote running time and occupied memory of $\{y \ mod \ N : y \in [0, (N-1)^2]\}$. Using $t_{hash}(z)$ and $r_{hash}(z)$ to denote running time and occupied memory of Hash algorithm $\{H(z) : z = lengthof(m||id) \in \{0,1\}^*\}$. So the results are as follows:

$$M_{init} := r_{mul}(x^2) + r_{mod}(y \ mod \ N) + r_{hash}(z) \tag{14}$$

$$T_{init} := (\frac{2\tau}{s \cdot log(\tau)} + O(\tau/log(\tau))) \cdot (t_{mul}(x^2) + t_{mod}(y \ mod \ N)) + t_{hash}(z) \tag{15}$$

$$S_{init} := sizeof(id||m||\pi||l) \tag{16}$$

It can be seen that the difficulty of generating the evidence-data is related to the value of τ. By dynamically adjusting τ, it can adjust the consumption of compute-time resource for generating the evidence-data. The consumption will cause real-world cost.

The setup phase of *Proof-of-Replication* (PoRep) in Filecoin uses file blocks as leaf nodes to generate a Merkel tree, the size of evidence-data is positively correlated with the file size. Wesolowski's VDF reduces the evidence-data to constant, therefore, it reduces storage resource consumption.

Proof Phase. P will generate a response σ_P when receiving a challenge ch. The calculation methods of computing and storage are as follows ($w = lengthof(id||m||\pi||ch)$):

$$M_{chal} := r_{hash}(w) \tag{17}$$

$$T_{chal} = t_{hash}(w) \tag{18}$$

$$S_{chal} := sizeof(\sigma_P||\tau||id||\pi||l||N) \tag{19}$$

Compared to the prove phase of *Proof-of-Replication* (PoRep) in Filecoin, this challenge phase does not use zk-SNARKs to hide the privacy information of file, but implement a less complex hash operation. Therefore, the efficiency of P proof phase is higher than PoRep.

Verification Phase. When receiving σ_P, τ, id, π, l, N, V begins to verify them. The consumption of computing and storage is as follows:

$$\begin{aligned} M_{ver} := {} & r_{hash}(m||id) + r_{mul}(2^{\tau}) + \\ & r_{mul}(\pi^{l} \cdot \hat{x}^{r}) + 2 \cdot r_{mod}(y \bmod l) \\ & + r_{hash}(\hat{x} + \hat{y}) + r_{hash}(id||m||\pi||ch) \end{aligned} \tag{20}$$

$$\begin{aligned} T_{ver} := {} & t_{hash}(m||id) + t_{mul}(2^{\tau}) + \\ & t_{mul}(\pi^{l} \cdot \hat{x}^{r}) + 2 \cdot t_{mod}(y \bmod l) \\ & + t_{hash}(id||m||\pi||ch) \end{aligned} \tag{21}$$

$$S_{ver} := S_{chal} \tag{22}$$

In this phase, the operations include multiplication, exponential operation, modular operation, etc. Comparing with the proof phase of *Proof-of-Replication* (PoRep) in Filecoin, verification phase does not use heavy cryptography mechanism zk-SNARKs. Therefore, the verification efficiency will be higher than that of PoRep.

Verification of File-Consistency. VoFC combines challenge and verification phases of VDF with the RBC algorithm of HoneyBadgerBFT. The total communication complexity of HoneyBadgerBFT is $O(N^2|v| + \lambda N^3 logN)$, where $|v|$ is the largest size of any node's input. VoFC increases challenge and echo-challenge phases compared with HoneyBadgerBFT, therefore, the communication complexity of FAV-BFT is $O(N^2|v| + \lambda N^3 logN + N^2 + N)$.

4.2 Comparison with Multiple BFT Protocol

At present, there are many popular BFT protocols, such as HoneyBadgerBFT [32], Dumbo [33], Dumbo-MVBA [34], etc. These popular BFT protocols all use RBC algorithm. FAV-BFT proposed in this paper only combines VDF with RBC protocol, while others remain unchanged. Compared with these BFT protocols, since the verification data of each stage of VoFC does not contain file privacy data, threshold encryption and randomly chosen sample can be removed. In theory, the efficiency of FAV-BFT proposed in this paper will be higher than these popular BFT protocols.

4.3 Security Analysis

The following contents introduce the security of challenge-response protocol based on VDF and VoFC respectively.

Challenge-Response Protocol Based on VDF. Challenge-response protocol based on VDF should satisfy the properties of completeness and soundness.

For completeness, it is required that for any honest P, the probability of passing the validation is close to 1.

Honest P uses the stored file to generate the evidence-data, and returns the response according to the V's challenge. When V receives the proof phase response corresponding to challenge ch, he/she will verify the correctness of the response. V holds the same file as P, so V can also calculate the value of $\hat{x}$ and r. Furthermore, V can verify the correctness of l. The completeness of this protocol is below:

Define k is the remainder of $2^\tau/l$:

$$k = 2^\tau \ mod \ l \tag{23}$$

Therefore:

$$\lfloor 2^\tau/l \rfloor = (2^\tau - k)/l \tag{24}$$

$$\begin{aligned} \pi^l \cdot x^r \ mod \ N &= x^{\lfloor 2^\tau/l \rfloor \cdot l} \cdot x^r \ mod \ N \\ &= x^{\lfloor 2^\tau/l \rfloor \cdot l} \cdot x^{2^\tau \ mod \ l} \ mod \ N \\ &= x^{(2^\tau - 2^\tau \ mod \ l)/l \cdot l} \cdot x^{2^\tau \ mod \ l} \ mod \ N \\ &= 2^{2^\tau} \ mod \ N \end{aligned} \tag{25}$$

Therefore, for honest P, the possibility of passing the verification is 1.

For soundness, malicious file-sharers attempt to pass the verification without saving the file or affording the compute-time resources. If P does not save the file or pay the cost of compute-time, it will be difficult to generate the evidence-data, and thus cannot respond to the challenge of V. The probability of a malicious P passing the verification is close to 0.

Verification of File-Consistency. FAV-BFT combines challenge-response protocol base on VDF with RBC algorithm of BFT protocol. The communication and verification methods are consistent with RBC algorithm. Therefore, the security of FAV-BFT can be regulated to the security of these BFT protocols.

5 Experiment and Evaluation Result

This paper implements the FAV-BFT and tested its efficiency[1]. The experiment was performed on a machine with dual-core 12th Gen Intel(R) Core(TM) i7-12700F CPU @ 2.00 GHz, 8 GB RAM, Ubuntu 18.04 virtual machine, 20 GB

[1] The source code is open source on the website: https://github.com/buptis073114/FAV-BFT.

SATA hard disk. The challenge-response protocol based on VDF is implemented in C language and be compiled into so dynamic link library, in which GMP library is used for large number operation. RBC algorithm is implemented in Python language based on the open source code of HoneyBadgerBFT. The ctype library is used to call so shared libraries functions in Python3. Suppose that the shared file sizes are 1 MB, 5 MB, 50 MB, 100 MB, 500 MB, 700 MB and 1 GB respectively, the values of τ are 500, 5000, 10000, 50000, 100000, 300000, 500000 and 1000000 respectively. The size of N is 4096-bits.

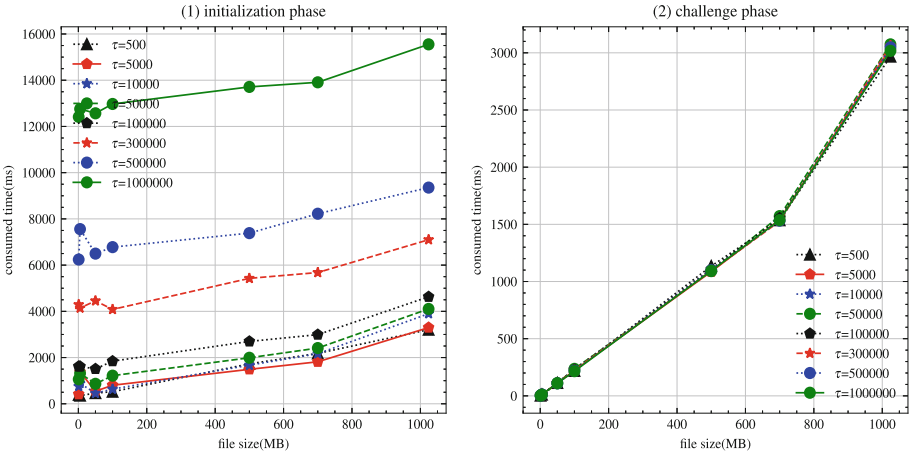

Fig. 5. Time consumed in initialization and challenge phases

Figure 5 (1) shows the consumed time of initialization phase with the different values of τ. As the shared-file size and the value of τ increase, the consumed time also increases. In this phase, the compute-time consumption can be adjusted by dynamically adjusting the value of τ. It is difficult to speed up this process by adding parallel computing. The size of evidence-data generated by this phase is 2231-bytes.

In proof phase, the experiment tested the efficiency for P to generate a response when receiving the challenge from V. The challenge is a 256-bit string randomly generated by P. The consumed time of P proof phase is shown in Fig. 5 (2). The consumed-time in this phase is positively correlated with file size and has little association with the value of τ. If the P does not hold the evidence-data, he/she must rerun the initialization phase, which will cause timeout and be rejected.

The experiment tests the efficiency of verification phase, which is shown in Fig. 6 (1). The consumed time in this phase is proportional to the file size. It can be shown that the consumed time mainly occurs in calculating the hash value of the file. We test the execution efficiency of VoFC and RBC algorithm, which is shown in Fig. 6 (2). The experimental results show that the efficiency of VoFC is higher than RBC under the premise that the input length is same.

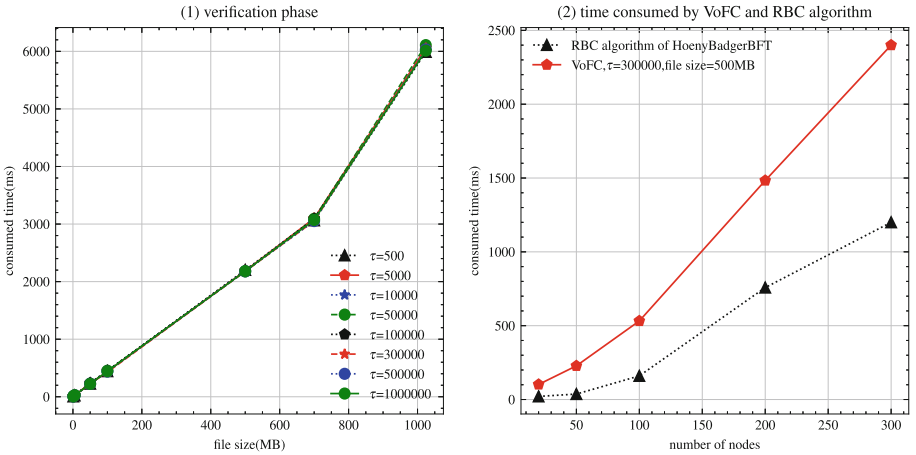

Fig. 6. Time consumed in verification phase, VoFC and RBC algorithm

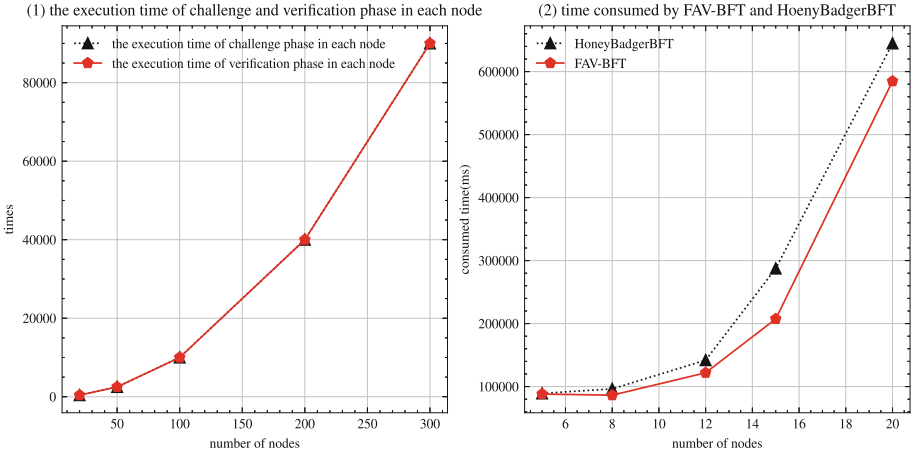

Fig. 7. Time consumed in verification phase, VoFC and RBC algorithm

In the above challenge and verification phases, verifying the hash value of the file takes most time. Therefore, the method of improving calculation efficiency is adopted by calculating the hash value of the file sampling instead of the hash value of the whole content of the file. The result shows that these two phases can be completed more efficiently.

The experiment also tests the efficiency of each node executing challenge and verification phases in VoFC, which is shown in Fig. 7 (1). The results show that each node will receive the challenge from all other nodes, and will verify all received responses. The execution number of challenge and verification phases is the square of node's number.

At last, by replacing the RBC algorithm of HoneyBadgerBFT with VoFC and removing the threshold encryption in HoneyBadgerBFT, the whole process of FAV-BFT is implemented. The experimental results show that FAV-BFT can indeed verify the authenticity of the file and the efficiency is higher than that of HoneyBadgerBFT which is shown in Fig. 7 (2). In addition, a certain number of bad participants are deliberately set in the experiment (less than 1/3 of the total number of nodes), but they can not affect the process of FAV-BFT, which demonstrates the characteristics of tolerating a certain number of malicious participants.

6 Conclusion

This paper proposes an efficient file authenticity verification protocol for Blockchain-based file-sharing system, namely *File Authenticity Verification Byzantine Fault Tolerant* (FAV-BFT). Compared with state-machine replication BFT protocol, FAV-BFT combines VDF with RBC algorithm to tolerate malicious file-sharers who do not save real files. FAV-BFT solves the problem of competing for file-sharing qualification in the scenario where multiple file-sharers hold the same file. It allows users to convince others that file-shares have stored the same file for a period of time and ensures that shared-file is indeed wanted by the file-downloader under the condition that more than 2/3 of all participants are honest. In addition, this paper analyzes the performance and security of FAV-BFT. Compared to the consensus protocol of Filecoin, the computation complexity of FAV-BFT is low. Finally, this paper implements FAV-BFT, and tests each phase of the FAV-BFT. The experimental results show that FAV-BFT can complete the verification of file authenticity efficiently.

References

1. Steinmetz, R., Wehrle, K.: Peer-to-Peer Systems and Applications, vol. 3485. Springer, Heidelberg (2005). https://doi.org/10.1007/11530657
2. Kulbak, Y., Bickson, D., et al.: The eMule protocol specification. eMule project (2005). http://sourceforge.net
3. Liang, J., Kumar, R., Ross, K.W.: Understanding kazaa
4. Sari, L., Sipos, M.: FileTribe: blockchain-based secure file sharing on IPFS. In: 25th European Wireless Conference, pp. 1–6 (2019)
5. Khatal, S., Rane, J., Patel, D., Patel, P., Busnel, Y.: FileShare: a blockchain and IPFS framework for secure file sharing and data provenance. In: Advances in Machine Learning and Computational Intelligence, pp. 825–833 (2021)
6. BTFS. https://www.bittorrent.com/token/bittorrent-file-system/
7. Nyaletey, E., Parizi, R.M., Zhang, Q., Choo, K.R.: BlockIPFS - blockchain-enabled interplanetary file system for forensic and trusted data traceability. In: 2019 IEEE International Conference on Blockchain (Blockchain), pp. 18–25 (2019)
8. Vorick, D., Champine, L.: Sia: simple decentralized storage. Nebulous Inc. (2014)
9. Storj Labs: Storj: a decentralized cloud storage network framework (2018)
10. Protocol Labs: Filecoin: a decentralized storage network (2017). https://filecoin.io/filecoin.pdf

11. Moran, T., Orlov, I.: Simple proofs of space-time and rational proofs of storage. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 381–409. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_14
12. Benet, J.: IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561 (2014)
13. Benet, J., Dalrymple, D., Greco, N.: Proof of replication, vol. 27, p. 20. Protocol Labs (2017)
14. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
15. King, S., Nadal, S.: PPCoin: peer-to-peer crypto-currency with proof-of-stake, 19 August 2012. Self-published paper
16. Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_18
17. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6
18. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI 1999, vol. 99, pp. 173–186 (1999)
19. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. (CSUR) **22**(4), 299–319 (1990)
20. Juels, A., Kaliski Jr, B.S.: PORs: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 584–597 (2007)
21. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 585–605. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_29
22. Moran, T., Orlov, I.: Rational proofs of space-time. Technical report. Cryptology ePrint Archive, vol. 35 (2016)
23. Mahmoody, M., Moran, T., Vadhan, S.: Publicly verifiable proofs of sequential work. In: Proceedings of the 4th Conference on Innovations in Theoretical Computer Science, pp. 373–388 (2013)
24. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 451–467. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_15
25. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 757–788. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_25
26. Pietrzak, K.: Simple verifiable delay functions. In: 10th Innovations in Theoretical Computer Science Conference (ITCS 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
27. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
28. Mahmoody, M., Moran, T., Vadhan, S.: Time-lock puzzles in the random oracle model. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 39–50. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_3

29. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 620–649. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_22
30. Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. IACR Cryptology ePrint Archive 2018/712 (2018)
31. Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 379–407. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_13
32. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 31–42 (2016)
33. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: faster asynchronous BFT protocols. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 803–818 (2020)
34. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-MVBA: optimal multi-valued validated asynchronous byzantine agreement, revisited. In: Proceedings of the 39th Symposium on Principles of Distributed Computing, pp. 129–138 (2020)
35. Su, S., Yuan, F., Yuan, Y., Zeng, L., Chen, C.: VoFSQ: an efficient file-sharing interactive verification protocol. In: 2021 IEEE Symposium on Computers and Communications (ISCC), pp. 1–7 (2021)
36. Dimitrov, V., Jullien, G., Miller, W.: Complexity and fast algorithms for multiexponentiations. IEEE Trans. Comput. **49**, 141–147 (2000)
37. Chang, C., Lou, D.: Parallel computation of the multi-exponentiation for cryptosystems. Int. J. Comput. Math. **63**, 9–26 (1997)