PPoS : Practical Proof of Storage for Blockchain Full Nodes

1st Jun Wook Heo
Trusted Networks Lab
School of Computer Science
Queensland University of Technology
Brisbane, Australia
junwook.heo@hdr.qut.edu.au

2nd Gowri Ramachandran
Trusted Networks Lab
School of Computer Science
Queensland University of Technology
Brisbane, Australia
g.ramachandran@qut.edu.au

3rd Raja Jurdak
Trusted Networks Lab
School of Computer Science
Queensland University of Technology
Brisbane, Australia
r.jurdak@qut.edu.au

***Abstract*—Blockchain is a distributed and immutable ledger managed by all participants. The full nodes which store the entire ledger play an essential role in managing it in a transparent and decentralised manner. However, it is difficult to verify that full nodes store the entire ledger in their dedicated storage due to Sybil, outsourcing, or generation attacks. Existing work on proving storage for cloud computing and remote data storage applications has high latency for decryption, and its impact on decentralisation is unclear, rendering it impractical for use in blockchain. In this paper, we propose a decentralised *Practical Proof of Storage (PPoS) solution for blockchain full nodes* with asymmetric latencies for encryption and decryption, which introduces a chained encryption and decryption architecture. In order to generate a unique replica of a block, each full node performs encryption with its own address and a previously encrypted block, storing the unique block in its dedicated storage. In PPoS, encryption is expensive and time consuming, enabling it to detect outsourcing and generation attacks and to deter Sybil attacks. At the same time, decryption is about 25 times faster than encryption, resulting in minimal performance overhead. The proof process is also decentralised by randomly selecting provers, verifiers, and encrypted blocks. We have conducted experiments using up to 720 real BitCoin blocks on a custom simulator and a testbed to evaluate the performance and quantify the decentralisation of PPoS. Our results show that PPoS's asymmetric design reduces decryption time 25-fold over existing approaches, while maintaining a high degree of decentralisation, confirming its suitability for blockchain full nodes.**

***Index Terms*—blockchain, proof of storage (PoS), distributed storage, blockchain scalability, blockchain decentralisation**

I. Introduction

Blockchain is a distributed and immutable ledger managed in a decentralised manner. The ledger is made up of blocks linked to previous blocks using cryptographic hashes and is shared among all participants so the ledger is secure and immutable. As blockchain is a publicly distributed database managed by nodes that do not fully trust each other, various consensus protocols are utilised to validate and update the ledger to maintain transparency. Thus, transparency of the distributed ledger underpins the blockchain to be reliable. Full nodes, which locally store the entire ledger, play a key role in transaction and block verification to maintain data transparency among the nodes. Bitcoin is one of the most well-known cryptocurrencies in which blockchain technology plays a crucial role in security and decentralisation. There are tens of thousands of applications using blockchain technology, but in terms of distributed and decentralised storage, their architectures are similar to Bitcoin's design, where full nodes store the entire ledger in their own storage.

There are two types of clients in Bitcoin: full nodes and lightweight nodes. Storing the entire history of transactions and blocks, full nodes can independently validate all transactions and blocks. On the other hand, lightweight nodes, known as simplified-payment-verification (SPV) clients, access full nodes to fetch transactions for validation. As the size of the ledger grows, a full node requires enormous resources such as computing power, network bandwidth, or storage space. Nevertheless, full nodes are not rewarded for their contribution to the Bitcoin network, while mining nodes are rewarded for processing transactions and generating new blocks [1]. The lack of incentives for full nodes creates a risk of full nodes withdrawing from the network or not maintaining their own full copies of stored data.

As blockchain networks rely on a healthy population of full nodes, reducing the number of full nodes can compromise security and decentralisation [2]. Full nodes on a peer-to-peer network have a risk of not only being unable to connect to the network due to network problems or various attacks but also being unable to provide data due to storage problems. Therefore, the number of full nodes can significantly affect data availability. In addition, the reduction in the number of full nodes can lead to a concentration of traffic, which reduces the degree of decentralisation and increases the probability of a single-point failure.

To keep blockchain networks healthy, it is important that each full node stores its own copy of the ledger in dedicated physical storage. However, there are three types of attacks on storage nodes: Sybil attacks, outsourcing attacks, and generation attacks [3]:

- Sybil Attacks : A malicious node could create multiple fake identities and pretend to store multiple copies of the entire ledger for each fake identity. However, it stores only a single copy of the ledger.

- Outsourcing Attacks : A malicious node could pretend to store data on its local storage by fetching data from other storage nodes and serving it.
- Generation Attacks : A malicious node could generate and serve data that the node does not store in its own storage.

These attacks have fueled research into decentralised algorithms that can verify the integrity and availability of ledger copies on full nodes. In general, Proof-of-storage (PoS), which verifies the integrity of remotely stored data, represents a combination of Provable Data Possession (PDP) and Proof of Retrievability (POR) [3], [9]. PDP allows users to verify the integrity of data outsourced to cloud storage without retrieving it [4], whereas POR allows users to retrieve outsourced data to cloud storage, so the users can recover the data [6]. However, PDP and POR cannot address Sybil, outsourcing, and generation attacks mentioned above. The work in [3] introduces Proof-of-Replication (PoRep), which ensures that a remote storage provider stores data in a uniquely dedicated physical storage, and Proof-of-Spacetime (PoSt), which lets that a remote storage provider spends its resources, time, and storage space, to store data in a uniquely dedicated physical storage.

To prevent the three attacks listed above, PoRep utilises iterative encryption and decryption schemes where it intentionally takes a long time to construct distinct replicas (physically independent copies). A Sybil attacker creating n-fake identities, for instance, has to perform n-times encryption and also store n-copies of data, requiring huge resources. Due to the time-consuming process of generating unique replicas for each node, it is difficult to mount outsourcing and generation attacks as the attackers cannot respond to users' requests for data within a short amount of time. Building on these advancements, Filecoin is a decentralised storage service that implements PoRep and PoSt to prove that storage providers are storing data on their own physical storage. However, it takes around $5 \sim 10$ minutes for a 1 MByte file for Filecoin to store in a physically unique storage [10]. In addition, PoRep has long latency for decryption because an iterative block cipher is also used to decrypt data. The long decryption time can have a negative effect on system performance when storage providers respond to data requests from users, affecting throughput and latency (or response time). Particularly, full nodes in blockchain frequently communicate with other nodes to provide blockchain ledger data. As a result, it is difficult to adapt existing PoS schemes to blockchain full nodes to ensure that they store the entire ledger in their own storage. The *symmetric* nature of PoRep in terms of the prolonged encryption *and* decryption time is therefore prohibitive for use in blockchain full nodes.

In this paper, we introduce a Practical PoS scheme with *Asymmetric Performance* and *Chained Architecture* of encryption and decryption for blockchain full nodes, addressing Sybil, outsourcing, and generation attacks. Specifically, our approach prolongs encryption time for resilience against these attacks, while ensuring decryption remains fast for high responsiveness. To create a unique replica for each full node, we use $F(x) = x^2$ over Galois fields [11] $GF(2^n)$ for $x \in GF(2^n)$ and its inverse operation to encrypt and decrypt blocks where the inverse operation takes much longer than calculating $F(x) = x^2$. As blockchain keeps generating new blocks periodically, a previously encrypted block is used to encrypt the next block as an encryption key, which links encrypted blocks to each other. This link of encrypted blocks makes it difficult for malicious nodes to generate their unique replica. The major contributions of this paper are:

- We propose Practical Proof of Storage (PPoS), as the first PoS solution for blockchain ledger data to prove that a full node stores a unique replica of the ledger in its own unique physical storage. Therefore, the number of full nodes can mean that the same number of replicas of the ledger is available on the network.
- We analyse PPoS and show that its asymmetric design of encryption and decryption significantly increases responsiveness to data access requests while maintaining resilience against key distributed storage attacks. The fast decryption can improve data retrieval performance when full nodes process users' requests.
- We evaluate PPoS in simulation and testbed experiments to confirm its performance benefits of up to 25-fold reduction in decryption time while maintaining a high degree of decentralisation. In terms of security, by using previously encrypted blocks as keys to encrypt the next blocks, it is nearly impossible to properly encrypt a block without its previous encrypted blocks. Therefore, PPoS resilience increases for longer chains, as the encryption time increases if the node does not store previously encrypted blocks.

The rest of the paper is organised as follows. Section II discusses related work on PoS. Section III outlines the details of PPoS including the architecture, proof processes and security. Section IV presents evaluation results in the aspects of performance, decentralisation, and security. Section V concludes the paper.

II. Related Work

As many applications are relying on outsourced storage services with the widespread use of cloud computing, PoS is attracting attention as a method of measuring the quality of outsourced storage services. The quality of outsourced storage services includes not only data availability and integrity but also the trustworthiness of the storage providers [12]. PDP and POR are introduced to provide data availability and integrity but cannot address the three aforementioned attacks. Therefore, Filecoin introduces PoRep to mainly address these attacks.

PDP is proposed to prove that a storage server possesses the outsourced data without retrieving it [4], [5]. The data owner locally stores pre-processed metadata to verify a proof that is generated by a storage provider to convince the data owner that the provider stores the original data. The owner repeatedly sends a challenge to the storage provider. To prove the data

TABLE I
A SUMMARY OF PROOF OF STORAGE SOLUTIONS

Solution	Countermeasure			Blockchain-Compatible	
	Sybil Attack	Outsourcing Attack	Generation Attack	Decentralisation	Low Decryption Latency
[4]	✗	✗	✗	✗	–
[5]	✗	✗	✗	✗	–
[6]	✗	✗	✗	✗	–
[7]	✗	✗	✗	✗	–
[3]	✓	✓	✓	–	✗
[8]	✓	✓	✓	✓	✗
PPoS	✓	✓	✓	✓	✓

possession, the storage provider then responds to the request with a proof which is generated with the original data and the challenge. However, PDP does not provide data recovery for data corruption, so it is difficult to recover damaged data, limiting data availability.

In addition to data integrity for remote data, POR provides data recovery using error correcting codes [6], [7]. POR embeds sentinels, which are sets of check blocks with random values, into the encrypted data. A verifier asks a prover for the sentinel values associated with the sentinels specified by the verifier as a challenge. Using the error-collecting codes, POR can recover the original data against a small portion of corruption. However, a prover cannot properly respond to a verifier's requests if significant portions of the original data have been modified or deleted.

PoRep [3] is introduced as a new type of Proof-of-Storage, which can prove that replicated data is stored on the uniquely dedicated physical storage. This is resistant to Sybil, outsourcing, and generation attacks. To achieve replication, PDP and POR schemes need to generate a set of encrypted replicas and manage the mapping of secret keys and those replicas, which is expensive and inefficient. To solve this problem, PoRep proposes a time-bounded replication construction called sealing. The original data is repeatedly encrypted with AES-256 to intentionally increase the sealing time. This expensive sealing time plays a key role in preventing malicious attackers from generating the replica. A malicious prover who does not store a replica needs to seal the data to respond to a verifier's request, but the computation is expensive, so the response time is highly likely to exceed the required time limit. However, unsealing to reconstruct the original data is also expensive due to the repeated decryption of AES-256.

PoSt is a scheme for proving that a prover has spent time and storage resources to store a replica. PoSt entails unpredictable and frequent challenges to convince a verifier that a prover has correctly stored the data for the challenge period. Otherwise, a prover can cheat by retrieving the data from other outsourced storage in advance of the challenges due to infrequent or predictable challenges. However, frequent interactive validation can compromise network and computational efficiency. To solve this issue, PoSt proposes a non-interactive validation scheme where challenges and proofs are periodically generated and stored in chronological order. As the sequence of challenges and proofs called proof-chain is a verifiable record, a verifier can verify them at once without interactive communication each time [6]. However, PoSt like PoRep, is also based on iterative encryption and decryption and requires an auditable storage such as a blockchain to record proofs that provers generate periodically.

Filecoin [8] is a decentralised storage service implementing PoRep and PoSt with Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs). A unique physical copy of data is generated by sealing which encrypts data with a unique key. Storage miners repeatedly generate proofs of the replications to ensure they are storing the data. For the non-interactive proof, the zk-SNARK scheme consists of the following steps [13].

- $KeyGen$: In this step, a proving key and a verification key for a prover and verifier are generated with a security parameter and a function. The function is converted into an arithmetic circuit that consists of AND, OR, and NOT gates. The proving key is used as an input by a prover to generate a proof while the verification key is used by a verifier to verify a proof. In this step, Filecoin also generates a replica of the original data with a sealing key.
- $Prove$: A prover generates a proof by using the proving key, witness, and input x in this step. Since the input x is public, a prover and verifier share it, whereas the witness is secret, so the prover must prove that they know it. For Filecoin, the public input x corresponds to a random challenge and the witness corresponds to the replica.
- $Verify$: In this step, a verifier can validate the proof which is generated by a prover in the proof stage by using the public input x and the verification key.

PoRep of Filecoin is mainly focused on addressing three PoS attacks by using iterative encryption to lengthen the generation time of a unique copy for each prover. A malicious node cannot immediately prove the data storage by outsourcing attacks unless the node stores its own copy. Therefore, the long encryption time plays a key role in preventing outsourcing attacks, but the repeated encryption in Filecoin leads to increased decryption time. The long decryption time is prohibitive for blockchain full nodes, as we describe next.

Full nodes maintain an entire copy of the blockchain ledger. Without any external reference, they can verify transactions and blocks with the entire data, which is stored in their own dedicated storage, maintaining the system's decentralisation.

Based on the period of time designed to generate new blocks in Bitcoin, they receive and store a new block in their dedicated storage every 10 minutes. When a full node receives requests to verify transactions or blocks, it has to retrieve its own ledger. If a full node uses PoRep, the encrypted data involved in verification must also be decrypted. Therefore, slow decryption can lead to degrading the performance of the entire blockchain network, indicating that PoS for blockchain full nodes requires fast decryption but slow encryption.

Table I shows the summary of PoS solutions. PDP and POR solutions cannot handle Sybil, outsourcing, and generation attacks and these solutions are not suited for blockchain full nodes. PoRep and PoSt of Filecoin can prevent these three attacks but they are not suitable solutions to be adopted for blockchain full nodes due to their poor decryption performance. We propose PPoS which not only handles these attacks, but also has a fast decryption time for blockchain full nodes.

III. Practical Proof of Storage

In this section, we discuss details of encryption and decryption, and the method of proof generation and verification employed by PPoS. Fig. 1 illustrates the process of PPoS. Each full node generates an encrypted block when it receives a new block from the network (Step 1). Each full node encrypts the new block with its own address and its previously encrypted block as encryption keys then the full node stores the unique encrypted block in its own storage (Step 2). A verifier who is randomly selected among full nodes requests a prover to generate a proof in order to verify that the prover is storing its own data in its own storage (Step 3). We refer to T_{req} as the time the request is sent. The prover generates a proof (Step 4) and replies to the request at time T_{res} with the proof (Step 5) that consists of hashes of n-consecutive encrypted blocks, their merkle root and randomly selected encrypted blocks among them. The selected encrypted blocks from the n consecutive blocks are determined based on the value of the merkle root. Because the merkle root is unforeseeable, it can be used as a random seed to select the encrypted block [14], [15]. T_{proof} is the maximum allowable time from T_{req} to T_{res} to prevent outsourcing attacks.

Then, the verifier verifies the proof using the prover's address and non-encrypted blocks (Step 6). The verifier first decrypts its encrypted blocks corresponding to the encrypted blocks in the proof. The verifier can then compute encryption or decryption using it in the same way that the prover has done using the encrypted blocks, non-encrypted blocks, and the prover's address. The n consecutive hashes and the merkle root are used to verify the results of the computation. Since encryption is much more expensive than generating a proof, a quick response from a prover is an essential prerequisite to convince a verifier of the prover's possession of data.

The rest of this section is organised as follows. Section III-A outlines how encryption and decryption of PPoS achieve asymmetric performance within a chained architecture. Then, Section III-B and III-C illustrates how a prover generates a proof and a verifier verifies it in a decentralised manner, respectively. Lastly, Section III-D and III-E discuss security and decentralisation.

A. Encryption and Decryption

In this paper, we use two strategies to improve decryption performance while guaranteeing long encryption time to address Sybil, outsourcing and generation attacks: (i) asymmetric encryption and decryption times; and (ii) using a previously encrypted block as a key for encrypting the next block. Asymmetric encryption and decryption use the square of elements and its inverse in a finite field for asymmetric performance of encryption and decryption. Use of encrypted blocks to encrypt the next block as an encryption key ensures that a block cannot be properly encrypted without a previously encrypted block. As shown in Fig. 2, a function $F(x) = x^2$ for $x \in GF(2^n)$ is used for decryption while the inverse of $F(x)$ is used for encryption, where B_k is the $k-th$ block, CB_k is the $k-th$ encrypted block and $P.Address$ is a unique node address for each node. For encryption, $P.Address$ makes encrypted blocks unique for each node, whereas CB_{k-1}, the previously encrypted block, makes it difficult for a malicious attacker to generate a properly encrypted block. Therefore, PPoS XORs B_k, the current block, $P.Address$ and the encrypted previous block CB_{k-1} before applying F^{-1}. Based on Fermat's little theorem [16], the inverse of $F(x)$, F^{-1}, is given by:

$$F^{-1}(x) = \{x \in GF(2^n) : x^{p/2}\} \tag{1}$$

Where p is 2^n over $GF(2^n)$ [17]. Decryption is much faster than encryption because computing x^2 is much faster than computing its inverse in a similar way to encryption and decryption for MiMC [18]. Before computing the inverse of the square for encryption, division is performed between the inputs and feedback of the previous output of the sequence of the encrypted block if the feedback is non-zero, so the encryption can be slow and non-parallelisable. Non-parallelisation can improve the security of PPoS because malicious attackers cannot speed up the encryption process through parallelisation. On the other hand, there is no feedback for decryption, so decryption can be sped up by parallelisation. In addition, it also performs an XOR between each block and the address of each full node to create unique encrypted blocks that the full node stores in its own storage. Next, we detail PPoS's proof generation process, followed by its proof verification process.

B. Proof Generation

There are two types of participants in the proof process. Provers must prove that they store their unique copy in their own storage by generating a proof while verifiers must verify the proof generated by a prover. All full nodes can be a prover and verifier. Since full nodes know the non-encrypted blocks, they can use encryption or decryption to verify the integrity of an encrypted block when they receive it.

A prover starts to generate a proof when it receives the request from a verifier as outlined in Algorithm 1. First, the prover lists up the hashes of the n-consecutive encrypted

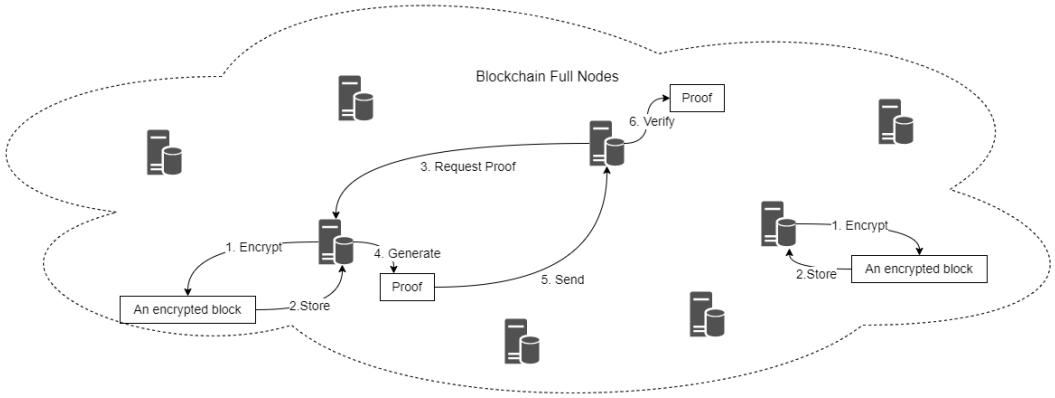
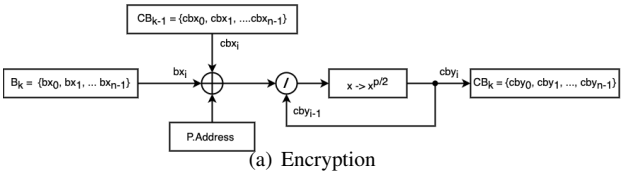
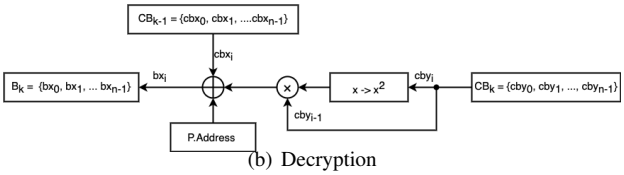


Fig. 1. Overview of Practical Proof of Storage



(a) Encryption



(b) Decryption

Fig. 2. Encryption and Decryption

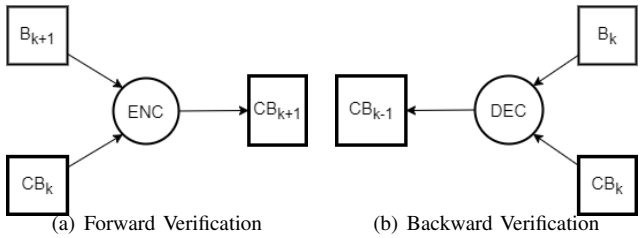
blocks in its own storage and computes their merkle root. Using the merkle root as a pseudo-random seed, the prover selects k encrypted blocks among the n-consecutive blocks. Finally, the prover sends the verifier the proof which consists of the merkle root and hash list of the n-consecutive blocks, and k encrypted blocks.

Algorithm 1 Proof Generation

Input: n and the hash of the start block
Output: PROOF
/ List up the n blocks */*
$\mathrm{HASHs} = (\mathrm{hash}(\mathrm{EB}_0), \mathrm{hash}(\mathrm{EB}_1), ...\mathrm{hash}(\mathrm{EB}_{n-1}))$
/ Compute a merkle root */*
$\mathrm{ROOT} = \mathrm{MerkleRoot}(\mathrm{HASHs})$
/ Select k encrypted blocks using the merkle root */*
$\mathrm{EBs} = (\mathrm{EB}_0, \mathrm{EB}_1, ...\mathrm{EB}_{k-1})$
return $\mathbf{PROOF}[\mathbf{HASHs}, \mathbf{ROOT}, \mathbf{EBs}]$

C. Proof Verification

PPoS supports two modes of proof verification: forward and backward verification, where a verifier can move forward or backward through the chain of blocks to verify the prover's encrypted blocks. Fig. 3 shows forward and backward verification where CB_k is the $k-th$ encrypted block that is received from a prover. A verifier can know B_k and B_{k+1} which are $k-th$ and $k+1-th$ non-encrypted blocks. For forward verification, a verifier can generate the prover's CB_{k+1} by encrypting it, and for backward verification, the verifier can also generate the key data by decrypting CB_k which the prover has used to generate CB_k. Backward verification is much faster than forward verification because decryption is faster than encryption.



(a) Forward Verification (b) Backward Verification

Fig. 3. Verification

Since new blocks are generated continuously, it is extremely difficult to verify all blocks every time. In addition, the proof block cannot be selected among all existing blocks because the size of the hash list keeps increasing. Therefore, PPoS limits the range of verifiable blocks to n-consecutive blocks. The large size of consecutive blocks makes attacks difficult while increasing the size of proof and computing time to generate a proof. A verifier randomly selects n-consecutive blocks and requests that a prover generates a proof for these blocks.

When the verifier receives the proof, the verifier can verify it using forward or backward verification as shown in Algorithm 2. T_{proof} must be less than the time that a malicious attacker can generate encrypted blocks. Thus, provers can convince verifiers that they store their own encrypted blocks in their own storage. For verification, the verifier first needs to decrypt the block corresponding to the encrypted block. Then the verifier decrypts the encrypted block received from a prover for backward verification or encrypts it for forward verification.

Finally, the verifier calculates the hash of the encrypted block and compares it with the received hash. As forward verification performs encryption and decryption once each while backward verification performs decryption twice, forward verification is much longer than backward verification. These forward and backward verifications are performed for each encrypted block in the proof.

Algorithm 2 Proof Verification

Input: PROOF = [HASHs, ROOT, EBs]
Output: TRUE or FALSE

```
/* Check timestamp */
if T_req − T_res > T_proof then
    return FALSE
end if
/* Compute a merkle root */
ROOT.v = MerkleRoot(HASHs)
if ROOT.v != ROOT then
    return FALSE
end if
/* Verify each EB in EBs */
for EB_k in EBs do
    /* Backward Verification */
    B_k = decrypt(CB_k, CB_{k−1})
    EB_{k−1}.v = decrypt(EB_k, B_k)
    hash.v = hash(EB_{k−1}.v)
    if HASHs[k] != hash.v then
        return FALSE
    end if
    /* Forward Verification */
    B_{k+1} = decrypt(CB_{k+1}, CB_k)
    EB_{k+1}.v = encrypt(EB_k, B_{k+1})
    hash.v = hash(EB_{k+1}.v)
    if HASHs[k] != hash.v then
        return FALSE
    end if
end for
return TRUE
```

D. Security Analysis

The long encryption time to generate a unique replica is key for PPoS to address Sybil, outsourcing, and generation attacks. We discuss PPoS's resilience to these attacks below.

Sybil attacks: Thanks to the long encryption time for generating unique replicas, a Sybil attacker would need enormous computational power and storage space to generate and store a unique replica for each fake identity.

Generation attacks: It is virtually impossible for a generation attacker to generate a unique copy without the original ledger, given the significant computational resources and time involved in PPoS's block encryption.

Outsourcing attacks: There are several possible outsourcing attacks for proof of storage in the blockchain. For instance, a malicious attacker who has no encrypted blocks wants to outsource the encrypted blocks to generate a proof. The attacker has no hashes of n-consecutive encrypted blocks, so the attacker should generate n-consecutive blocks first, but it takes too much time to generate n encrypted blocks. The other example of an outsourcing attack is that an attacker had generated encrypted blocks correctly then they deleted the blocks except for their hashes. In this case, the attacker can calculate the merkle root of n-consecutive hashes and also know the k encrypted blocks which are included in the proof. However, without the previously encrypted blocks, the attacker cannot generate the k correct encrypted blocks. Even if the attacker has the previous blocks, we can determine T_{proof} to be shorter than the time for the attacker to generate the k encrypted blocks, given the long encryption time.

E. Decentralisation

PPoS is decentralised by randomly selecting validators, provers, and encrypted blocks. Verifier selection is randomised by the hash of a new block which is broadcasted by miners [19]. When a full node receives a new block, it determines the verification by comparing the hash of the new block with its own address. The selected verifier randomly selects a prover which has to generate a proof to convince the verifier that it is storing its own blockchain ledger. Therefore, proof and verification are expected to be performed evenly across all full nodes. In addition, encrypted blocks that are to be verified are also randomly selected by a prover using the merkle root of the hashes of n-consecutive encrypted blocks which are selected by a verifier. Verifier selection is randomised by a hash of a new block. However, calculations can be implemented in many ways, and be customised by system designers according to different application requirements. Similarly, prover selection approaches can be customised for specific applications, and are beyond the scope of this paper.

IV. Experiment and Evaluation

This section evaluates the encryption and decryption performance compared with a *baseline* approach that uses iterative encryption and decryption with AES-256, as used by Filecoin. We implement a simulator written in Go language and run it on Raspberry Pi 4 and measure the performance with 720 Bitcoin blocks. Note that the real BitCoin blocks were downloaded and used within the simulator for the evaluation. The size of the total blocks is 736,583,544 Bytes and the average block size is 1,023,033 Bytes. Table II shows parameters for the experiment. To measure the performance of encryption and decryption, we perform encryption and decryption for 720 blocks in a sequence on Raspberry Pi 4. On the other hand, to measure the performance of proof and verification, and the level of decentralisation, we simulate PPoS with 8 nodes and 60 seconds of mining period. Since 60 seconds is 10 times faster than Bitcoin's mining time, it is expected that PPoS can be adopted for blockchains with fast throughput. The number of consecutive blocks is 4 for the experiment, but the number can be customised by system designers according to different application requirements. However, the range of blockchain nodes is wide and their performance varies. Additionally, there

are various factors that cause network latency. Since we are primarily targeting IoT blockchains, we only experimented with the Raspberry Pi in this paper.

TABLE II
EXPERIMENT PARAMETERS

Parameter	Value
GF	GF(2^{32})
T_{proof}	2 Seconds
# consecutive blocks	4
# proof blocks	1
# full nodes	8
# Bitcoin blocks	720
Mining period	60 Seconds

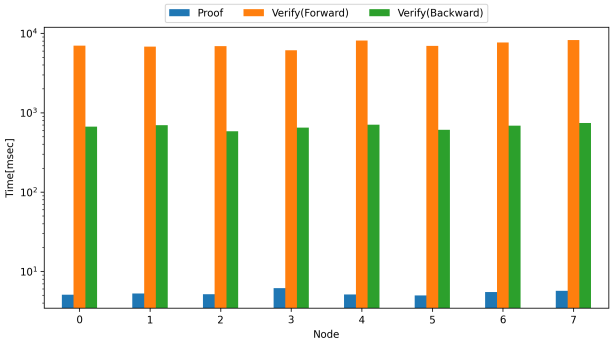
A. Performance

1) Encryption and Decryption: Table III shows the latency of encryption and decryption per megabyte of data. The baseline approach shows similar performance in encryption and decryption, whereas decryption in PPoS is about 25 times faster than encryption. For instance, it takes about 10 seconds to encrypt and decrypt 1 MByte with the baseline, but it takes about 0.42 seconds to decrypt it for PPoS. For blockchain full nodes which need to provide ledger data, a long decryption time can degrade overall system performance, so the fast decryption time can lower the barrier to PoS adoption on blockchains.

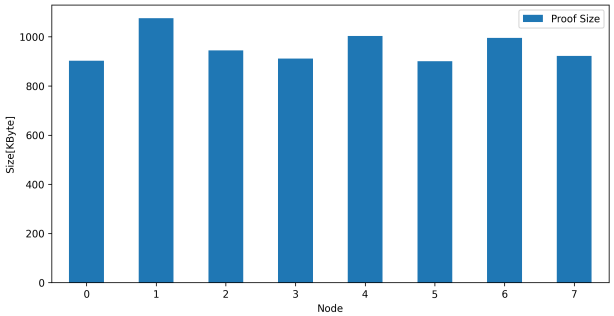
TABLE III
LATENCY OF ENCRYPTION AND DECRYPTION PER MBYTE

	Encryption[Second]	Decryption[Second]
Baseline	10.411	10.383
PPoS	10.631	**0.420**

2) Proof and Verification: Fig. 4 shows the average time of generating proof as well as forward and backward verification on a logarithmic scale. Since forward verification is based on one encryption operation and a decryption operation, while backward verification is based on two decryption operations, forward verification is approximately 12.4 times slower than backward verification. Compared to verification, proof generation is much faster because no encryption or decryption is required. In terms of proof size, the size of the proof mainly depends on the size of the block since the size of the encrypted block is much larger than the size of n-consecutive hashes in the proof. Table IV shows the performance of PPoS compared to SNARKs. The average proof size is similar to the average block size, which is significantly larger than the one of SNARKs. Compared to SNARKs, the average proof generation time is much shorter, whereas the average verification time is longer. We can determine that T_{proof} is 2 seconds which is long enough to generate a proof and short enough to detect an outsourcing attack for the experiment. When verifying an encrypted block using both forward and backward verification, a verifier can ensure that a prover generates the encrypted block with an appropriate previous key and generates the next encrypted block with the encrypted block. To achieve the same level of security using only backward verification, the verifier requires two consecutive encrypted blocks, increasing proof size but reducing the computational resources compared to forward encryption.



(a) Average Time of Proof and Verification



(b) Average Size of Proof

Fig. 4. Average Time and Size for Each node

TABLE IV
PERFORMANCE COMPARISON BETWEEN PPOS AND SNARKS

	Proof		Verification	
Solution	Size	Time	Forward	Backward
PPoS	965.40KB	5.12ms	10431.40ms	841.99ms
SNARKs [20]	288 Bytes	2.3s	10ms	

B. Decentralisation

An important requirement for a proof-of-storage algorithm for blockchain is its degree of decentralisation. To quantify decentralisation, we adopt the following metrics:

- Fairness index is proposed to measure the fairness level of resource allocation schemes [21]. Fairness index can quantify decentrality as,

$$FI(p) = \frac{(\sum_{i=1}^{i=N} p_i)^2}{\sum_{i=1}^{i=N} p_i^2} \tag{2}$$

where p_i is the total number of proof or verification by a node i and where N is the number of full nodes. Normalised fairness index is also defined as, [22].

$$NF(p) = \frac{FI(p) - \frac{1}{N}}{1 - \frac{1}{N}} \tag{3}$$

- As entropy represents the quantification of uncertainty or randomness, we use normalised entropy to measure the decentrality of the PPoS scheme. Normalised entropy is defined as,

$$NE(p) = \frac{\sum_{i=1}^{i=N} -p_i \cdot log_2(p_i)}{log_2(N)} \quad (4)$$

where p_i is the total number of proof or verification by a node i and where N is the number of full nodes [22].

In the remainder of this section, we use these metrics for quantifying PPoS's decentralisation of node selection for proof and verification.

We randomise the selection of the prover and verifier using the hash of new blocks to evenly perform proof and verification for each node. Fig. 5 shows the number of proofs and verifications for each node. We evaluate the decentrality level using normalised fairness index and normalised entropy [22] as shown in Table V. A decentralisation value close to 1 means decentralised, while a value close to 0 means centralised. All decentralisation indices are greater than 0.97, which means that the system is predominantly decentralised.

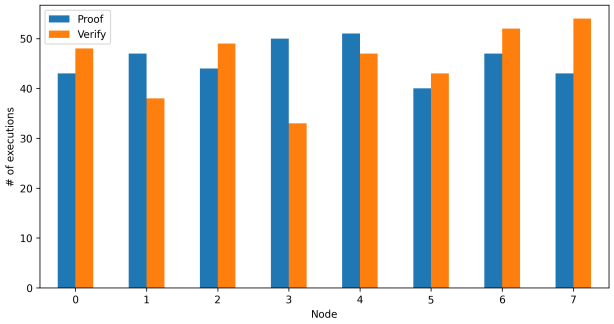


Fig. 5. The number of proof and verification for each node

TABLE V
DECENTRALISATION OF PROOF AND VERIFICATION

Decentralisation Index	Proof	Verification
Normalised Fairness Index	0.993	0.976
Normalised Entropy	0.999	0.995

C. Security

PPoS is resistant to the three attacks, forcing full nodes to generate their own replicas through a lengthy encryption process. In the case of a Sybil attack, a malicious node has to perform encryption for all the blocks, from the genesis block to the latest block, and store the encrypted blocks for each fake identity. Figure 6 shows encryption and decryption time by the height of blocks, highlighting the progressive increase in encryption time with increased block height. For instance, it takes 127.6 minutes to encrypt 720 blocks. Sybil attacks, therefore, require enormous computational power and storage space, becoming increasingly difficult over time as the chain grows longer.

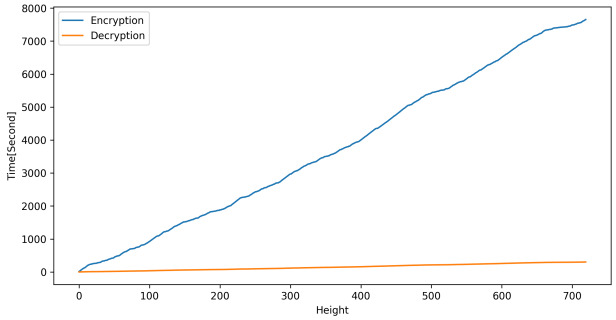


Fig. 6. Encryption and decryption time by height

In terms of outsourcing and generation attacks, an honest prover must provide its proof to the verifier within T_{proof} and T_{proof} must be less than the time for a malicious node to generate encrypted blocks. There are two considerations to determine an appropriate T_{proof}. If T_{proof} is too short, an honest prover cannot respond to a verifier's request in time, whereas if T_{proof} is too long, a verifier would not be able to detect outsourcing attacks. To safely increase T_{proof} to account for network delay, it is possible to increase the number of encrypted blocks in the proof. However, increasing the number of encrypted blocks also increases the proof size. Since the size of encrypted blocks is the majority of a proof, increasing the number of encrypted blocks n times also increases the size of a proof produced by honest provers as well as the time to generate the encrypted blocks by outsourcing or generation attacks by approximately n times. These parameters can be chosen by system designers to customise the approach to their application to balance its security requirements and responsiveness.

V. CONCLUSION

This paper proposed a decentralised PPoS for blockchain full nodes with chained architecture and asymmetric performance of encryption and decryption. Decryption is about 25 times fast than encryption and previously encrypted blocks are used to encrypt the next blocks. Therefore, it takes long enough to generate a unique replica of a blockchain ledger to prevent Sybil, outsourcing, or generation attacks while the fast decryption leads to minimisation of the performance degradation for blockchain full nodes. The proof process proposed is also decentralised. As a result, PPoS can be adopted for blockchain full nodes to improve security and decentralisation. To the best of our knowledge, PPoS is the first proof-of-storage protocol dedicated to blockchain full nodes. An interesting direction for future work is the design of a non-interactive proof protocol with the zero-knowledge proof to enhance security and to reduce the proof size.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] M. Florian, S. Henningsen, S. Beaucamp, and B. Scheuermann, "Erasing data from blockchain nodes," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019, pp. 367–376.

[3] J. Benet, D. Dalrymple, and N. Greco, "Proof of replication," *Protocol Labs, July*, vol. 27, p. 20, 2017.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 598–609.

[5] R. Mukundan, S. Madria, and M. Linderman, "Efficient integrity verification of replicated data in cloud using homomorphic encryption," *Distributed and Parallel Databases*, vol. 32, no. 4, pp. 507–534, 2014.

[6] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 584–597.

[7] H. Shacham and B. Waters, "Compact proofs of retrievability," in *International conference on the theory and application of cryptology and information security*. Springer, 2008, pp. 90–107.

[8] J. Benet and N. Greco, "Filecoin: A decentralized storage network," *Protoc. Labs*, pp. 1–36, 2018.

[9] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, 2012, pp. 1–12.

[10] "Filecoin docs," filecoin's, November 12, 2022. [Online]. Available: https://docs.filecoin.io/

[11] A. Vourdas, "Quantum systems with finite hilbert space: Galois fields in quantum mechanics," *Journal of Physics A: Mathematical and Theoretical*, vol. 40, no. 33, p. R285, 2007.

[12] G. Ateniese, L. Chen, M. Etemad, and Q. Tang, "Proof of storage-time: Efficiently checking continuous data availability," *Cryptology ePrint Archive*, 2020.

[13] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct {Non-Interactive} zero knowledge for a von neumann architecture," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 781–796.

[14] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *Ieee Access*, vol. 7, pp. 22 328–22 370, 2019.

[15] V. Buterin, "Starks, part ii: Thank goodness it's fri-day," Nov 2017. [Online]. Available: https://vitalik.ca/general/2017/11/22/starks_part_2.html

[16] U. Daepp and P. Gorkin, "Fermat's little theorem," in *Reading, Writing, and Proving*. Springer, 2011, pp. 315–323.

[17] E. Muñoz-Coreas and H. Thapliyal, "Design of quantum circuits for galois field squaring and exponentiation," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 68–73.

[18] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2016, pp. 191–219.

[19] A. Dorri and R. Jurdak, "Tree-chain: A lightweight consensus algorithm for iot-based blockchains," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2021, pp. 1–9.

[20] S. Kassaras and L. Maglaras, "Zkps: Does this make the cut? recent advances and success of zero-knowledge security protocols," *arXiv preprint arXiv:2006.09990*, 2020.

[21] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Beyond jain's fairness index: Setting the bar for the deployment of congestion control algorithms," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, 2019, pp. 17–24.

[22] S. P. Gochhayat, S. Shetty, R. Mukkamala, P. Foytik, G. A. Kamhoua, and L. Njilla, "Measuring decentrality in blockchain based systems," *IEEE Access*, vol. 8, pp. 178 372–178 390, 2020.