BOSSA: A Decentralized System for Proofs of Data Retrievability and Replication

Dian Chen, Haobo Yuan, Shengshan Hu,
Qian Wang, *Senior Member, IEEE*, and Cong Wang, *Senior Member, IEEE*

Abstract—Proofs of retrievability and proofs of replication are two cryptographic tools that enable a remote server to prove that the users' data has been correctly stored. Nevertheless, the literature either requires the users themselves to perform expensive verification jobs, or relies on a "fully trustworthy" third party auditor (TPA) to execute the public verification. In addition, none of existing solutions consider the underlying incentive issues behind a rational server who is motivated to collect users' data but tries to evade the replication checking in order to save storage resources. In this work, we propose the first decentralized system for proofs of data retrievability and replication—BOSSA, which is incentive-compatible for each party and realizes automated auditing atop off-the-shelf blockchain platforms. We deal with issues such as proof enforcements to catch malicious behaviors, new metrics to measure the contributions, and reward distributions to create a fair reciprocal environment. BOSSA also incorporates privacy-enhancing techniques to prevent decentralized peers (including blockchain nodes) from inferring private information about the outsourced data. Security analysis is presented in the context of integrity, privacy, and reliability. We implement a prototype based on BOSSA leveraging the smart contracts of Ethereum blockchain. Our extensive experimental evaluations demonstrate the practicality of our proposal.

Index Terms—Proofs of Retrievability; Proofs of Replication; Decentralized system; Blockchain.

1 Introduction

As a new computing paradigm, cloud computing provides a convenient approach for ordinary users to enjoy powerful computing and storage resources. Users can outsource their data to a cloud service provider to get rid of complex local data management. Despite promising prospects, outsourcing data to a remote server $\mathcal{S}$ also raises severe security concerns since it deprives the physical control of the data owner. One of the challenging problems is ensuring the correctness of data. A malicious server may discard the data that is rarely accessed for the purpose of saving resources, or cover up data loss accident for maintaining reputation. In addition, $\mathcal{S}$ usually claims that the data will be stored together with several replicas for ensuring high reliability [1], it takes limited liability in their Service Level Agreements (SLAs) [2], especially for the data loss. And recent accidents happened in Tencent Cloud [3], other clouds [4] have already implied the fact that $\mathcal{S}$ cannot be fully trusted.

Proofs of retrievability [5], [6], [7], [8], [9], [10], [11], [12] and proofs of replication [13], [14], [15], [16], [17], [18], [19] are two typical cryptographic methods allowing the server to prove that the original files as well as all the replicas are correctly stored. In order to avoid keeping users on-line and free them from expensive auditing tasks, the state-of-the-art solutions for proofs of retrievability and replication heavily rely on a semi-trusted third party auditor (TPA) to execute public verifications. However, such methods suffer from the following two main drawbacks:

- **Collusion Attack.** The literature usually assumes that no collusion happens between TPA and the server $\mathcal{S}$. This strong assumption may be easily corrupted in practice driven by certain interests, and $\mathcal{S}$ can easily create fake proofs that pass the verifications once it colludes with TPA. Furthermore, from the users' perspective, it is hard to tell whether or not collusion happened.
- **Corruption Attack.** Dependence on an "always-online" TPA is vulnerable to single point of failures caused by unpredictable accidents, *e.g.*, regional power outages, or malicious hacking attacks, like DDoS.

We owe these vulnerabilities to the great control power of TPA which plays a centralized role. Therefore, constructing a decentralized framework for auditing can fundamentally solve the above problems. The blockchain has been widely adopted for replacing the third party in the fields like e-voting [20], auction [21] and fair exchange [22] due to its non-repudiation and non-tampering properties. Replacing the TPA with the blockchain makes the whole auditing process trackable by users. As a result, the users are able to verify each step of proofs retrievability and replication. Moreover, the decentralized nature of the blockchain improves the tolerance of single point failures. In summary, the blockchain is a promising solution to defend against *collusion attack* and *corruption attack*.

In addition, for the data replica service that aims to guarantee the data reliability, the cloud may intentionally delete replicas for saving storage resources without concerns of being caught. Inspired by the idea of decentralized storage network (DSN) that

- *D. Chen and Q. Wang are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, the School of Cyber Science and Engineering, Wuhan University, Wuhan, Hubei 430072, China, and also with the State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China. (Corresponding author: Qian Wang.)*
  E-mail: {dianchen, qianwang}@whu.edu.cn
- *H. Yuan is with the School of Computer Science, Wuhan University, Wuhan, Hubei 430072, China.*
  E-mail: yuanhaobo@whu.edu.cn
- *S. Hu is with the School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China.*
  E-mail: hushengshan@hust.edu.cn
- *C. Wang is with the Department of Computer Science, City University of Hong Kong 999077, Hong Kong.*
  E-mail: congwang@cityu.edu.hk

tries to build a peer-to-peer network allowing peers buying and selling idle disk space, decentralizing data replication makes it much more difficult for the attackers to misbehave, as each step for the data transmission, storing and deleting is recorded and trackable.

1.1 Design Challenges and Our Solutions

In this work, we propose the Blockchain based OutSourcing Storage and Auditing (BOSSA) scheme, a brand-new general framework for proofs of retrievability and replication. BOSSA separately stores the original data on the server and the corresponding replicas on peers (also called farmers in BOSSA) of a decentralized network atop blockchain, and integrities of them will be checked respectively by carefully-designed smart contracts. BOSSA can naturally defend against the *collusion attack* and the *corruption attack*, while provides visibility-enabled data replication. BOSSA doesn't heavily rely on a specific consensus model like [23], which gives the opportunity that BOSSA can be built on off-the-shelf blockchain systems supporting Turing-complete smart contract [24], [25]. To design BOSSA, there are several challenges we have to cope with.

First of all, simply combining existing proof of retrievability schemes with the blockchain fails to achieve our desired goals since the blockchain only supports simple functionality. The most critical problem is that the blockchain cannot actively issue challenges and reacts only based on received transactions. This property allows the adversary to escape from auditing. To prevent such lazy behaviors, we make use of the feature that blocks are appended to the chain in an approximately-fixed rate, and propose a time-restricted proof forcing both the cloud and farmers to prove data availability.

Second, our scheme organizes peers, *i.e.*, farmers, to store the replicas of users' data. However, farmers are not fully trusted and may leave the network arbitrarily, which causes replicas loss, damaging the reliability of replicas. To address this problem, we design an incentive mechanism to motivate peers to store and provide replicas when needed. Specifically, we define a reward mechanism where farmers are periodically rewarded if they can provide valid proofs of replicas on a regular basis. Meanwhile, part of the reward is temporarily frozen until farmers' promised storage services expired, which discourages farmers from leaving the network. To further motivate farmers to share replicas, we define a metric called *contribution rate*, and connect farmers' frozen reward with the contribution rate, such that only farmers with 100% contribution rate can retrieve all the frozen rewards.

Third, most of blockchain platforms like Ethereum store transactions in plaintext, any participant is able to retrieve the information of transactions, including the inputs of smart contracts, the intermediate results, etc. For an auditing system naively built atop the blockchain, the open nature of the blockchain enables the adversary to record proofs and recover the original data. To prevent such privacy leakage, we incorporate privacy-enhancing techniques for the generation of proofs while ensuring their correctness.

1.2 Contribution

In summary, we make the following contributions:

- We propose a novel decentralized framework BOSSA for proofs of data retrievability and replication. Our design addresses security risks and caters to a more real-world scenario, which has not been considered so far in existing works.
- To guarantee the reliability of BOSSA, we propose a time-restricted proof forcing $\mathcal{S}$ and farmers to prove data availability, and a reward mechanism based on the new metric *contribution rate* to create a fair reciprocal environment. Privacy-enhancing techniques are also incorporated to protect users' private data.
- We provide the security analysis for BOSSA with respect to integrity, privacy, and reliability. A prototype of BOSSA is implemented on Ethereum, and evaluated. The experimental evaluations show that our proposal is feasible and incurs tolerable overheads for each party.

1.3 Related Work

Proofs of Retrievability/Data Possession. Both Proof of Data Possession (PDP) and Proofs of Retrievability (PoR) aim to ensure that the outsourced data is stored on $\mathcal{S}$ correctly. Juels and Kaliski proposed the first PoR scheme [9] and Ateniese *et al.* [5] constructed the first PDP scheme. However, their schemes do not provide public verification, which requires users keeping on-line during the auditing process. Guan *et al.* [26] leveraged the indistinguishability obfuscation to realize public verification, but their scheme has poor performance due to the usage of the indistinguishability obfuscation. Shacham and Waters [10] then constructed a public verification scheme based on BLS signature [27], but their scheme may cause privacy-leakage during the auditing [12]. Armknecht [8], Xu [7] and Yang [6] proposed low-cost public PoR schemes by modifying the private verification scheme of [10], however, the TPA in their scheme holds the secret key for auditing, which is not suitable to public blockchain platforms.

Proofs of Replication. Different from PDP/PoR, Proofs of Replication focuses on ensuring that the replicas of original data are correctly stored on $\mathcal{S}$ as it claimed. Most proofs of replication schemes are based on indistinguishable encryption/encoding [14], [16], [28] or noticeable replica generation time [13], [18], [19], [29]. MR-PDP proposed in [14] requires users to encrypt replicas and upload them to $\mathcal{S}$, and lets users audit the corresponding replicas. Hao *et al.* [28] leveraged similar methods, and their scheme supports public verification. Damgard *et al.* [16] generalized the indistinguishable encryption/encoding based model which utilizes a trapdoor function without which $\mathcal{S}$ cannot generate correct replicas on-the-fly. However, these schemes bring heavy bandwidth and computation costs to users, as the user encrypts/encodes and uploads replicas. Other solutions let $\mathcal{S}$ generate replicas, but the time cost of the replicas generation is noticeable, like RSA-based puzzle combining linear feedback shift registers [13], chained sequential encryption [17], and butterfly construction [15]. But these schemes rely on an assumption that the replicas generation is time-costly and cannot be parallelized. In BOSSA, we propose a new paradigm in which a decentralized storage network atop blockchain is used to store replicas. With the openness of the blockchain, $\mathcal{S}$'s action (*i.e.*, data replication) is publicly recorded, which is visible to the users. BOSSA neither brings heavy costs to the user, nor relies on the time-assumption. In addition, this paradigm also benefits $\mathcal{S}$ as its storage is reduced (most of clouds' basic storage services promise to store several replicas in the same region).

Decentralized storage. The main purpose of decentralized storage schemes is to reuse idle storage resources of personal

computers (PC) by motivating PC users to share their unused disk space. Existing decentralized storage network schemes, like Storj [30], Sia [31], utilize Merkle tree-based data auditing scheme whose proof size is related to the size of data, which causes high communication cost for auditing large data. IPFS [32] plans to offer an open global p2p storage network, however, it lacks an incentive mechanism such that the storage resources may be provided by few nodes. In addition, it also fails to provide an auditing mechanism to check the integrity of data stored by nodes. Filecoin [23] is built atop IPFS and provides the aforementioned missing features. Except for incentive mechanism, Filecoin leverages a computationally expensive proofs of replica scheme [17] to ensure the correct data processing and builds a consensus mechanism. Similar idea is also presented in [33], [34], [35]. Compared to Filecoin, BOSSA is a more general framework which is compatible with other blockchain platforms like Ethereum.

1.4 Organization

The rest of the paper is organized as follows. Section 2 presents some backgrounds. In Section 3 we define the model of our scheme, threat model and security goals, before we introduce key techniques of our scheme in Section 4. And concrete construction of our scheme is described in Section 5. We analyze our scheme in Section 6, evaluate the performance of our scheme in Section 7. We conclude the paper in Section 8.

2 Background

2.1 Blockchain and Smart Contract

Blockchain, a cryptographic primitive derived from Bitcoin proposed by Nakamoto Satoshi [36] in 2008, has been extensively studied and adopted in both academies [37], [38], [39] and industries [40]. Blockchain can be viewed as a distributed block-wise ledger allowing any participants to access and modify (appending only). Generally speaking, it can be generalized as the following formulation:

$$\begin{aligned}\text{Block}_n =& \text{Header}_n \parallel \text{Transactions},\\ \text{Header}_n =& Hash(\text{Block}_{n-1}) \parallel \text{Nonce} \parallel\\ & \text{Timestamp} \parallel \text{MerkleRoot}.\end{aligned}$$

A block consists of transactions' data and a header which generally contains information for consensus and security: a *Timestamp*, a hash of predecessor's block, a digest of transactions (*i.e. MerkleRoot*), and a *Nonce* used for solving consensus puzzles. Blocks are organized by the hash pointing to the predecessor block, forming a hierarchically expanding chain. Cryptographic mechanisms, like hashing, signature, guarantee that the blockchain is non-repudiation and non-tampering, and consensus models, *e.g.*, proof of work, proof of stake, motivate majority participants to maintain only one correct chain.

Bitcoin, the initial blockchain implementation, provides restricted customizable scripts, which limits it to a "cryptocurrency", instead of a general computational platform. The idea is later brought by the following implementations such as Ethereum [24], [41], HyperLedger [25], [42]. Ethereum, the first cryptocurrency supporting general computing, provides a virtual machine (EVM) and Turing-complete opcodes (EVM Bytes). The computation results of smart contracts inherit the nature of trust from the blockchain, hence Ethereum is widely applied in fields like e-voting [20], auction [21], fair exchange [22], etc.

2.2 Proofs of Retrievability

A Proofs of Retrievability (PoR) scheme typically contains three roles including a user, a remote server (also plays as a prover) and a verifier (*i.e.*, the TPA in public verification or the user itself in the private verification scenario). To prove that data is retrievable, a challenge-response protocol is executed between the prover and the verifier, and the data retrievability can be guaranteed with a high probability if the prover presents a valid response (proof). A PoR scheme [10] can be defined as follows:

Definition 1. *A PoR scheme consists of the following four algorithms:*

- *Setup*$(1^\lambda) \rightarrow (sk, pk)$*: This probabilistic algorithm is run by the user. It takes the security parameter as input and outputs key pair* (pk, sk) *for the scheme setup.*
- *Store*$(D, sk) \rightarrow (D', tag)$*: The user runs this algorithm to encode data and generate metadata for auditing. This algorithm takes as inputs key* sk *and user's data* D*, and outputs tags* tag *and encoded data* D'*.*
- *Prove*$(D', tag, pk, chal) \rightarrow \phi$*: This algorithm is run by the prover to prove data integrity. It takes as input the verifier's challenge* $chal$*, outsourced data* D'*, the corresponding tags* tag *and public key* pk*, and returns a proof* ϕ*.*
- *Verify*$(\phi, chal, pk[, sk]) \rightarrow (\top, \perp)$*: This algorithm is run by the verifier. It takes as input the proof* ϕ*, public key* pk *(*sk *is needed in private verification), and determines whether the prover stores data honestly.*

3 Problem Formulation

3.1 System Model

The system model of BOSSA is depicted in Fig. 1. Our system involves the following entities: the cloud server $\mathcal{S}$, the cloud user $\mathcal{U}$, and a decentralized network including nodes with two kinds of roles: miners $\mathcal{M}$ and farmers $\mathcal{F}$. $\mathcal{M}$ represents the nodes who maintain the blockchain and execute smart contracts. $\mathcal{F}$ represents the nodes joining in the decentralized storage network for renting out their idle storage resources.

In BOSSA, $\mathcal{S}$ is willing to collect the original data from $\mathcal{U}$ for accomplishing data analytics tasks such as prediction or recommendation [43], [44], [45]. Similar to existing works [7], [12], the data outsourced to $\mathcal{S}$ is kept in plaintext in order to maximize their potential value (*e.g.*, data mining). To save resources and take full advantage of idle storage resources on $\mathcal{F}$, $\mathcal{S}$ further outsources replicas to $\mathcal{F}$. At the same time, $\mathcal{U}$ delegates $\mathcal{M}$ to audit the original data on $\mathcal{S}$ and replicas on $\mathcal{F}$, respectively.

The core functionalities of BOSSA are defined below.

Definition 2. *The algorithms for proofs of retrievability and proofs of replication in BOSSA are defined as follows:*

- *Setup*$(1^\lambda) \rightarrow (sk_\mathcal{U}, pk_\mathcal{U}, sk_\mathcal{S}, pk_\mathcal{S})$*: Given the security parameter* λ*, this probabilistic algorithm outputs private and public key pairs for* $\mathcal{U}$ *and* $\mathcal{S}$*.*
- $\mathcal{U}$*.Store*$(D, sk_\mathcal{U}) \rightarrow (D^*, aux)$*: This probabilistic algorithm takes the original data* D *and* $\mathcal{U}$*'s private key* $sk_\mathcal{U}$ *as input, and outputs* D^* *and the auxiliary information* aux*.*
- $\mathcal{S}$*.Seal*$(D^*, ek) \rightarrow R$*: This algorithm takes data* D^* *received from the user and an encryption key* ek *as input, and outputs a replica* R*.*
- $\mathcal{S}$*.Distribute*$(R, sk_\mathcal{S}) \rightarrow \{R_i, aux'_i\}_{i\in[0,K)}$*: This algorithm, taking as input the replica* R *and* $\mathcal{S}$*'s secret key* $sk_\mathcal{S}$*, outputs* K *replica blocks* R_i *with auxiliary information* aux'_i*.*

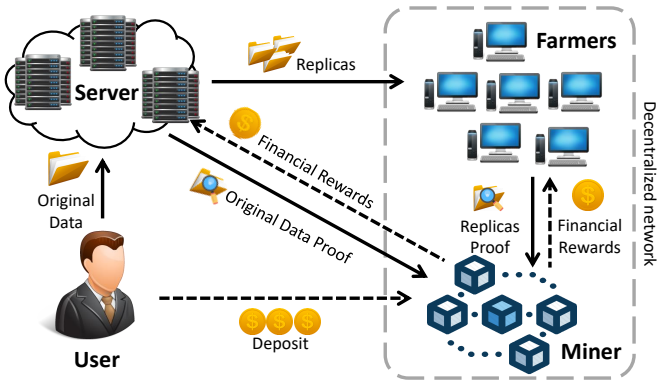


Fig. 1. System Model.

- $\mathcal{S}.\mathsf{Prove}(D^*, aux) \rightarrow \phi$: *This algorithm takes as input the data D^* stored by $\mathcal{S}$, the auxiliary information aux, then outputs the proof ϕ in a privacy-preserving manner.*
- $\mathcal{F}.\mathsf{Prove}(R_i, aux_i') \rightarrow \phi_i'$: *$\mathcal{F}$ calculates and outputs the proof ϕ_i' based on replica R_i and aux_i'.*
- $\mathsf{Verify}_X(pk_X, \phi) \rightarrow \{\top, \bot\}$: *This algorithm verifies $\mathcal{S}$'s or $\mathcal{F}$'s proof ϕ and outputs either $\top$ (TRUE) or $\bot$ (FALSE).*

3.2 Threat Model

Following previous works [13], [14] we consider a rational $\mathcal{S}$, who behaves correctly in most of the time unless misbehaving brings more benefits, *e.g.*, deleting data blocks seldom accessed for resource-saving. The blockchain network can be regarded as a trusted and public entity. Concretely, most of the computational power of the blockchain is held by honest miners who will execute smart contracts faithfully [46]. Any node in the blockchain network can learn internal states and messages sent to the blockchain.

The farmers from the decentralized network, rent out their idle disk space to store $\mathcal{U}$'s data replicas. We assume that there are massive farmers, the number of which is enough to store all the users' replicas. The farmers may leave the network irresponsibly if they lose the interest of storing replicas. Therefore, it is highly necessary to guarantee the availability of replicas, *i.e.*, motivating the farmers to act honestly.

3.3 Security Goals

Storage Correctness. This property guarantees both outsourced original data and replica are retrievable. We formalize this property by adopting extraction algorithm in [10].

Definition 3. *We say that the scheme guarantees storage correctness if the prover cannot forge a proof make the verifier accept, and there is an extraction algorithm such that for an ε-admissible prover ($\mathcal{S}$ or $\mathcal{F}$) who can provide ε fraction of a valid proof, an efficient extraction algorithm will recover data with overwhelming probability.*

Privacy Preservation. This property guarantees the nodes from the decentralized network (including $\mathcal{M}$ and $\mathcal{F}$) cannot infer any private information about the data.

Definition 4. *We say the scheme achieves privacy preservation if: 1) for a probabilistic polynomial time (PPT) adversary who acts as the verifier, namely $\mathcal{M}$, it is computationally hard to extract private information from the prover's proof; 2) the replicas stored in the decentralized network do not leak any private information about the original data.*

Replica Reliability. This property guarantees that the original data can be recovered from replica blocks stored by $\mathcal{F}$.

4 Key Techniques

4.1 Building Blocks and Notations

4.1.1 Building Blocks

Definition 5. *For a q-ary alphabet Σ_q, a (n, k, d) erasure code scheme consists of two algorithms $\mathsf{Encode}: \Sigma^k \rightarrow \Sigma^n$ and $\mathsf{Decode}: \Sigma^{n-d+1} \rightarrow \Sigma^k$. Typically, we say a code is a maximum distance separable (MDS) code, if $d = n - k + 1$.*

Definition 6. *Let $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are three multiplicative cyclic groups of prime order p. It has following properties:*

- *Given $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $x, y \in \mathbb{Z}_p$, we have $e(u^x, v^y) = e(u, v)^{xy}$.*
- *e is non-degenerate, namely, $e(g_1, g_2) \neq 1$ where g_1, g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.*
- *There is an efficient algorithm to compute $e(u, v)$ for any $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$.*

In addition, we define the following collision-resist hash functions:

- $h_1: \{0,1\}^* \rightarrow \mathbb{G}_1$, it maps arbitrary string to an element of the group $\mathbb{G}_1$.
- $h_2: \mathbb{G}_T \rightarrow \mathbb{Z}_p$, it maps elements of $\mathbb{G}_T$ to finite set $\mathbb{Z}_p$.
- $h_3: \{0,1\}^* \rightarrow \{0,1\}^\lambda$, it maps arbitrary string to constant-size bits, where λ is the security parameter.

Finally, we define $\{\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}\}$ a symmetric encryption algorithm with chose-plaintext-attack (CPA) security [47].

4.1.2 Other Notations

For ease of expression, we utilize L to represent the distributed ledger in the blockchain system. L is stored and maintained by $\mathcal{M}$. We index an entity in the blockchain by $X.Addr$, *e.g.*, $\mathcal{U}.Addr$. In the following discussion, B_{now} represents the latest block number of the blockchain.

The dictionary $\mathsf{L}.Roster$ stores $(\mathsf{D}, B_{start}, B_{end}, services)$ and is indexed by farmer's address $\mathcal{F}.Addr$. The attribute D is defined as the deposit paid by $\mathcal{F}$. The list $services$ records replicas stored by $\mathcal{F}$. B_{start} represents the time when $\mathcal{F}$ starts its storage services, and B_{end} represents the time when the services expire.

The dictionary $\mathsf{L}.Audit$ is indexed by $name$, the unique identifier of $\mathcal{U}$'s file (see Section 5.2). The attributes of $\mathsf{L}.Audit$ consist of the tuple $(B_{last}, \mathsf{D}, pk_{\mathcal{U}}, \mathsf{ltr}, \mathsf{p}, \mathsf{r})$. For a file $name$, the attribute D in $\mathsf{L}.Audit(name)$ denotes the deposit paid by $\mathcal{U}$ who outsources the file $name$. $pk_{\mathcal{U}}$ is the public key of $\mathcal{U}$. The rest parameters will be explained later in our construction.

$\mathsf{L}.Replica$ is a dictionary taking $name_{rep}$ as the key. It consists of the tuple $(\mathsf{D}, pk_{\mathcal{S}}, \mathsf{ltr}, \mathsf{r}, \mathsf{p}, \mathcal{L})$, where D represents the deposit provided by $\mathcal{U}$, and $pk_{\mathcal{S}}$ is the public key of $\mathcal{S}$. The list $\mathcal{L}$ consists of $(totalAsk, totalReply, \mathsf{D}, sig_i, B_{last})$ and is indexed by $\mathcal{F}.Addr$. The attribute D in $\mathcal{L}$ represents the deposit paid by the farmer when it stores replicas. The other parameters will be explained in our construction.

TABLE 1
Notations

Notation	Description
L	Distributed ledger of the blockchain
$X.Addr$	Address of party X in blockchain
$name$	An unique identifier for outsourced original file
$name_{rep}$	An unique identifier for outsourced replica
$services$	Services $\mathcal{F}$ has accepted
B_{start}	The block number when $\mathcal{F}$ starts its storage services
B_{end}	The block number when $\mathcal{F}$'s services are ended
B_{sel}	The number of block chosen by $\mathcal{S}$ and $\mathcal{F}$ to derive challenges
B_{now}	The number of latest block
B_{last}	The block of the previous block that chosen as challenge by $\mathcal{S}$ and $\mathcal{F}$
D	Deposit secured on the blockchain
p	The price that $\mathcal{U}$ is going to give to $\mathcal{F}$ or $\mathcal{S}$
r	A radius used for relaxing strictness of the proof
ltr	A time interval before both $\mathcal{S}$ and $\mathcal{F}$ present their next proof

4.2 Design Innovations

To prove data retrievability and possession of replicas, BOSSA incorporates several auditing techniques—compact proofs of retrievability of Shacham *et al.* [10] and privacy-preserving data auditing of Wang *et al.* [12]. Meanwhile, we have also made some changes that make our design more suitable for the blockchain environment. First, different from these schemes where the challenges are generated by the auditor, we require that the provers (*i.e.*, $\mathcal{S}$ and $\mathcal{F}$) generate challenges by themselves since there are no active auditors in our scheme. To ensure an unbiased challenge, we leverage the latest block of the blockchain as a publicly traceable and uncontrollable randomness source. In order to prevent lazy behaviors of provers, we design a time-restricted proof mechanism to force them to prove. Second, directly offloading replicas to the decentralized network may cause privacy leakage. Moreover, if some nodes storing a portion of replicas leave the network, it could result in the uselessness of the whole replicas. Hence, we propose an algorithm $\mathcal{S}$.Seal to encode replica to be outsourced and then encrypt the private data with the aid of the symmetric encryption algorithm. Last but not least, we extend the structure of the authenticator inside each replica block. Specifically, we add an extra identifier of the replica and the indices of the replica block into authenticators, to prevent $\mathcal{F}$ from pretending to be storing all the data which is not at all. More details can be found in Section 5.2.

4.3 Time-Restricted Proof

We define the time-restricted proof as follows.

Definition 7. *We use B_{last} to denote the number of the block from which the prover generates challenges for calculating the last valid proof, and B_{sel} denotes the block number of the block picked for the new proof. Let ltr be a time interval represented by the block numbers and a radius r represent the maximum gap between B_{sel} and B_{now}. Thus, during the verification procedure, we say the prover provides proof on time only if the following equation holds:*

$$B_{last} + \mathsf{ltr} = B_{sel} \leq B_{now} \leq B_{last} + \mathsf{ltr} + \mathsf{r}.$$

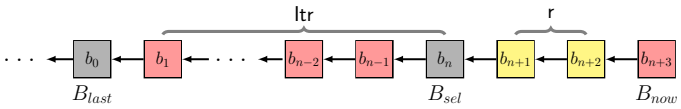


Fig. 2. Illustration of time-restricted proof. The prover must present the proof after $\mathsf{ltr} = n$ blocks. As long as the prover's proof is accepted in yellow blocks (*i.e.*, b_{n+1}, b_{n+2} as $\mathsf{r} = 2$), it can be regarded as providing a proof on time.

We illustrate the time-restricted proof mechanism in Fig. 2. As $\mathcal{M}$ cannot actively audit the prover ($\mathcal{S}$ and $\mathcal{F}$), we demand that the prover always generates challenges from the latest block (and set $B_{sel} = B_{now}$) and provides the proof to $\mathcal{M}$ periodically (after ltr blocks). The radius r is set for tolerating the processing delay of $\mathcal{M}$.

4.4 Contribution Rate

Considering the case that some farmers may be not willing to contribute replica blocks when the data is needed, we define a new metric, *contribution rate*, to measure $\mathcal{F}$'s contribution, which also directly affects $\mathcal{F}$'s final reward.

Definition 8. *Let $totalAsk$ be the amount of replica requests during $\mathcal{F}$'s services, and let $totalReply$ be the amount of $\mathcal{F}$'s responses. Then, the contribution rate cr is calculated as follows:*

$$cr = \begin{cases} \frac{totalReply}{totalAsk} & totalAsk > 0 \\ 1 & totalAsk = 0 \end{cases}, cr \in [0, 1].$$

We give an example for a better explanation. When $\mathcal{S}$ gives replica blocks to $\mathcal{F}_i$, both $totalAsk$ and $totalReply$ are set to 0. In the case that $\mathcal{S}$ needs replica blocks stored by $\mathcal{F}_i$, the counter $totalAsk$ of $\mathcal{F}_i$ is incremented by 1, and only when $\mathcal{F}_i$ provides the stored data to $\mathcal{S}$, $\mathcal{F}_i$'s counter $totalReply$ is incremented by 1. Later, when $\mathcal{F}_i$ terminates the storage service, the monetary reward it gets is its deposit multiplied by the contribution rate, and the rest of the deposit is refunded to $\mathcal{U}$.

4.5 Reward Mechanism

$\mathcal{S}$ will be rewarded for proving storage services. Suppose $\mathcal{U}$ and $\mathcal{S}$ have come to an agreement about a price p, each time, $\mathcal{S}$ honestly proves the integrity of data, $\mathcal{U}$ will pay p to $\mathcal{S}$.

Likewise, we adopt the same reward mechanism for $\mathcal{F}$ with an extra modification. Considering the following scenario, after providing several proofs, $\mathcal{F}$'s reward has far exceeded its total deposit, then it can quit the network arbitrarily without worrying about the financial loss. Therefore, we set a factor $\xi \in (0, 1)$ to prevent this behavior. After a valid proof of $\mathcal{F}$, $\mathcal{M}$ withholds $(1 - \xi) \cdot \mathsf{p}$ of monetary reward instead of giving all of them to $\mathcal{F}$. The frozen part of monetary reward is refunded only when $\mathcal{F}$'s service expires. A proper selection of ξ (*e.g.*, $\xi = 0.5$) can make $\mathcal{F}$ more sticky to the network without damaging its long-term benefits.

Algorithm 1 $\mathcal{S}$.Seal

Input: $D := \{m_0, m_1, \ldots, m_{|D|-1}\}$, encryption key ek and (n_1, k_1), (n_2, k_2) for encoding

Output: sealed replica R

1: reshape D into a $(\lceil \frac{|D|}{k_1} \rceil, k_1)$ matrix and pad the last row with 0 if the width of the last row is less than k_1
2: **for** $i = 1, \ldots, \lceil \frac{|D|}{k_1} \rceil$ **do**
3: $\{m'_{i,j}\}_{j=0,\ldots,n_1-1} = \mathsf{Enc}_{ek}\left(\mathsf{Encode}_{(n_1,k_1)}(\vec{m}_i)\right)$, and $\vec{m}_i = \{m_{i,j}\}_{j=0,\ldots,k_1-1}$
4: **end for**
5: **for** $j = 0, \ldots, k_1 - 1$ **do**
6: denote the jth column of the encoded data as $D'_j := \{m'_{0,j}, \ldots, m'_{|D'_j|-1,j}\}$ where $|D'_j| = \lceil \frac{|D|}{k_1} \rceil$
7: reshape D'_j into a $(\lceil \frac{|D'_j|}{k_2} \rceil, k_2)$ matrix and pad with zero
8: **for** $i = 0, \ldots, \lceil \frac{|D'_j|}{k_2} \rceil - 1$ **do**
9: $\vec{m^r}_{i,j} = \mathsf{Encode}_{(n_2,k_2)}(\{m'_{i\times k_2+l,j}\}_{l=0,\ldots,k_2-1})$ and $\vec{m^r}_{i,j} := (m^r_{i\times n_2,j}, m^r_{i\times n_2+1,j}, \ldots, m^r_{(i+1)\times n_2,j})$
10: **end for**
11: **end for**
12: $R = \{m^r_{i,j}\}_{i=0,\ldots,\lceil \frac{|D|\times n_2}{k_1\times k_2} \rceil - 1; j=0,\ldots,n_1-1}$

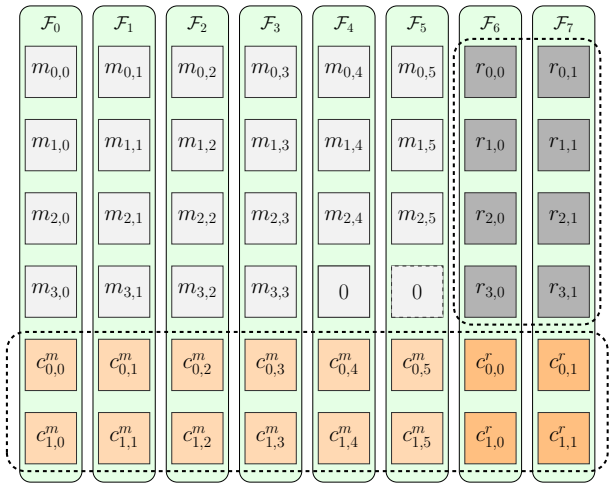


Fig. 3. A toy example of $\mathcal{S}$.Seal. We give an example about sealing 22-block data with $(n_1 = 8, k_1 = 6)$ and $(n_2 = 6, k_2 = 4)$. The light grey blocks represent original data with two zero-padding blocks. The dotted box in the upper right corner circles out blocks newly generated in the first phase (line 3) of the Algorithm 1. Blocks generated in the second phase (line 9) are circled in the dotted box below (it may have a little discrepancies with Algorithm 1 since they are put together). For simplification, we omit the representation of encryption. Each column of data later is outsourced to different farmers.

5 Our Construction

5.1 Overview

Before diving into the details of BOSSA, we give an overview of our idea. Recall the model specified in Section 3, BOSSA provides a new paradigm for data integrity auditing and data replication, which uses the blockchain network to periodically audit the server and store data replicas. Specifically, for ensuring the server to store data correctly, BOSSA adopts a conventional PoR scheme but replaces the TPA with the blockchain. As the blockchain only receives transactions passively and cannot issue challenges to the server, the *time-restricted proof* is used to force the server to provide proofs actively. Besides, another important component of BOSSA is storing replicas among the peers in the blockchain network, which are called farmers. Naively sending replicas to farmers raises concerns about data privacy, reliability, and retrievability. In BOSSA, the server encodes and encrypts replicas for guaranteeing privacy and reliability, and the farmers are spurred to prove they maintain the intactness of replicas. Besides, the farmers are rewarded for providing valid proofs (*i.e.* storing replicas honestly), and their rewards are also determined by whether they could provide replicas when needed, which is measured by *contribution rate*. Farmers, who have 100% contribution rate, can fully retrieve their reward.

In Section 5.2, algorithms of PoR are presented. In Section 5.3, we will walk through the details of BOSSA including data outsourcing, data replication, proving and verification, replicas retrieving, and rewarding.

5.2 Algorithm Specification

We give the specification of auditing algorithms. For the data auditing, we leverage some cryptographic tools in [10], [12], with modifications stated in Section 4.2.

Setup. In Setup, $\mathcal{U}$ firstly generates a key pair $(spk_{\mathcal{U}}, ssk_{\mathcal{U}})$ for signing. Then, it randomly picks the following elements: $x_{\mathcal{U}} \xleftarrow{R} \mathbb{Z}_p$, $\bar{g}_{\mathcal{U}} \xleftarrow{R} \mathbb{G}_1$, $g_* \xleftarrow{R} \mathbb{G}_1$, and calculates $v_{\mathcal{U}} = {g_2}^{x_{\mathcal{U}}}$ based on a generator g_2 of $\mathbb{G}_2$. For $\mathcal{U}$, the public key is $pk_{\mathcal{U}} = (spk_{\mathcal{U}}, v_{\mathcal{U}}, g_2, \bar{g}_{\mathcal{U}}, e(\bar{g}_{\mathcal{U}}, v_{\mathcal{U}}), g_*)$, and the secret key is $sk_{\mathcal{U}} = (x_{\mathcal{U}}, ssk_{\mathcal{U}})$.

Besides, $\mathcal{U}$ also generates key pair which will be sent to $\mathcal{S}$ during replica outsourcing. $\mathcal{U}$ picks a pair of signing keys $(ssk_{\mathcal{S}}, spk_{\mathcal{S}})$, a random element $x_{\mathcal{S}} \xleftarrow{R} \mathbb{Z}_p$, a random element $\bar{g}_{\mathcal{S}} \xleftarrow{R} \mathbb{G}_1$ and computes $v_{\mathcal{S}} = {g_2}^{x_{\mathcal{S}}}$, where g_2 is a generator of $\mathbb{G}_2$. The public key of $\mathcal{S}$ is $pk_{\mathcal{S}} = (spk_{\mathcal{S}}, v_{\mathcal{S}}, g_2, \bar{g}_{\mathcal{S}})$ and the secret key is $sk_{\mathcal{S}} = (x_{\mathcal{S}}, ssk_{\mathcal{S}})$.

Outsourcing original data. In $\mathcal{U}$.Store algorithm, $\mathcal{U}$ preprocesses its data $D \in \{0,1\}^*$ before outsourcing it to $\mathcal{S}$. Firstly, $\mathcal{U}$ encodes D with the erasure codes [48] and gets D^*. In this algorithm, $\mathcal{U}$'s data D^* is interpreted as n blocks, *i.e.*, $D^* = (m_0, m_1, \ldots, m_{n-1})$. To identify the data, $\mathcal{U}$ randomly chooses an element $name$ from $\mathbb{Z}_p$ as the identifier. For each data block m_i, $\mathcal{U}$ computes a unique authenticator $\sigma_i = (h_1(W_i) \cdot \bar{g}_{\mathcal{U}}^{m_i})^{x_{\mathcal{U}}} \in \mathbb{G}_1$ where $W_i = name \parallel i$. Furthermore, $\mathcal{U}$ generates a signature of the identifier $name$: $\omega = \omega_0 \parallel SSig_{ssk_{\mathcal{U}}}(\omega_0)$ where $\omega_0 = name \parallel n$. Then D^* and $aux = (\{\sigma_i\}_{0\le i<n}, \omega)$ are sent to $\mathcal{S}$.

Sealing replicas. To ensure reliability and protect privacy of the replica, we propose the algorithm $\mathcal{S}$.Seal. A detailed algorithm is presented in Algorithm 1, and a toy example is depicted in Fig. 3. In this algorithm, data is firstly encoded using the erasure code and encrypted with the symmetric encryption algorithm (*i.e.* line 3 of the Algorithm 1). The encrypted data is divided into pieces and will be stored by different farmers. Before being sent to the farmer, each piece of the replica is encoded again (see line 9 of the Algorithm 1). The first encoding of $\mathcal{S}$.Seal guarantees data can be retrieved even small part of farmers are unaccessible. The second encoding ensures each piece of the replica is also loss tolerance, and also reduces the verification cost (see Section 6.1). We leverage Reed-Solomon [48] code as the erasure code due to its property of being a MSD code.

Replicas distribution. In this algorithm, $\mathcal{S}$ calculates authenti-

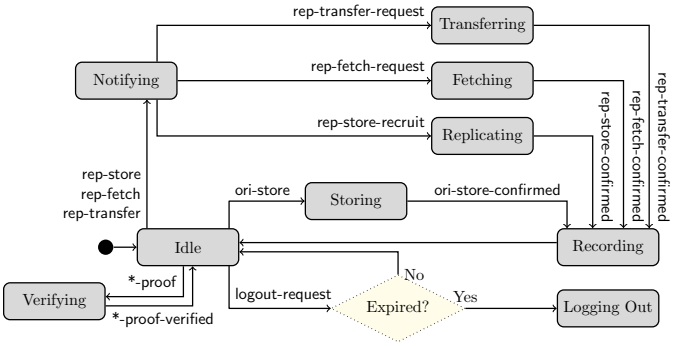


Fig. 4. State diagram of smart contract instance on $\mathcal{M}$.

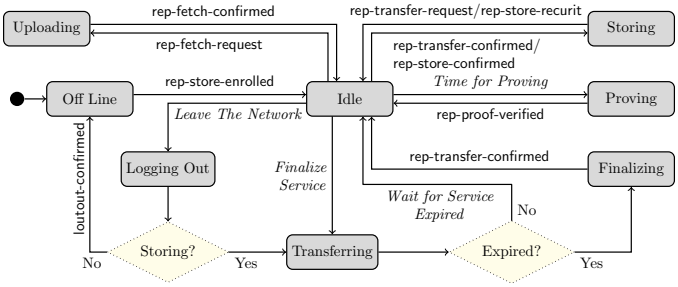


Fig. 5. State diagram of $\mathcal{F}$. Texts in *italic* font style represent active actions or events of $\mathcal{F}$, while texts in sans serif font style are messages from smart contract (*i.e.* $\mathcal{M}$).

cators for replicas. At first, the replica R is divided into K blocks, and each block contains s sectors, *i.e.*, $R = \{m^r_{i,j}\}_{0\le i<K, 0\le j<s}$. For each sector, $\mathcal{S}$ generates an authenticator $\sigma^*{}_{i,j} \leftarrow (h_1(W'_{i,j}) \cdot \bar{g}_{\mathcal{S}}^{m^r_{i,j}})^{x_{\mathcal{S}}} \in \mathbb{G}_1$. In each authenticator, we use $h_1(W'_{i,j})$, where $W'_{i,j} = name_{rep} \parallel name \parallel i \parallel j$, to represent $m^r_{i,j}$ sector. Besides, for each replica block i, an extra signature Ω_i is generated by computing $\Omega_i = name_{rep} \parallel name \parallel i \parallel s \parallel sig$, where $sig = SSig_{ssk_P}(name_{rep} \parallel name \parallel i \parallel s)$. Finally, $\mathcal{S}$ distributes K replica blocks $R_i = \{m^r_{i,j}\}_{j\in[0,s)}$ together with $aux'_i = (\{\sigma^*{}_{i,j}\}_{j\in[0,s)}, \Omega_i)$ to ith farmer, where $i \in [0, K)$.

Proofs generation for the original data. $\mathcal{S}$ runs $\mathcal{S}$.Prove to prove that the original data is stored correctly. In this algorithm, the privacy of the original data is well protected.

$\mathcal{S}$ takes the block B_{now} as the randomness source, and then calculates challenges $\{(i, \nu_i)\}_{i\in I}, \nu_i \xleftarrow{R} \mathbb{Z}_p, I \subset [0, n)$. Then $\mathcal{S}$ calculates $\mu' = \sum_{i\in I} \nu_i m_i$ and $\sigma = \prod_{i\in I} \sigma_i{}^{\nu_i} \in \mathbb{G}_1$. To blind μ' and σ, $\mathcal{S}$ firstly chooses three random elements r_σ, r_m, ρ from $\mathbb{Z}_p$ and calculates $T = e(g_*, g_2)^{r_\sigma} \cdot e(\bar{g}_{\mathcal{U}}, v_{\mathcal{U}})^{r_m} \in \mathbb{G}_T$. For hiding μ', $\mathcal{S}$ computes $\mu = r_m + \gamma\mu'$, here $\gamma = h_2(T) \in \mathbb{Z}_p$. Then, $\mathcal{S}$ calculates $\Sigma = \sigma \cdot {g_*}^{\rho}$ to hide σ, and calculates $\varsigma = r_\sigma + \gamma\rho$ to hide ρ. Finally, $\mathcal{S}$ sends $\phi = (\omega, \varsigma, \mu, \Sigma, T, B_{now})$ to $\mathcal{M}$.

Proofs generation for replicas. The farmers prove that they store replicas honestly through $\mathcal{F}$.Prove. Here, We assume that $\mathcal{F}_i$ stores the ith block of replica. Similar to $\mathcal{S}$.Prove, $\mathcal{F}_i$ generates challenges $\{(j, v_j)\}_{j\in J}$, $v_j \xrightarrow{R} \mathbb{Z}_p$, $J \subset [0, s)$, from the newest block B_{now}. Then $\mathcal{F}_i$ calculates an aggregated authenticators $\sigma^*_i = \prod_{j\in J}(\sigma^*_{i,j})^{\nu_j} \in \mathbb{G}_1$ and an aggregated sampled sectors $\mu^*_i = \sum_{j\in J} \nu_j m^r_{i,j}$. After that, $\mathcal{F}$ sends $\phi'_i = (\Omega_i, \mu^*_i, \sigma^*_i, B_{now})$ to $\mathcal{M}$.

Proofs verification. After receiving proof from $\mathcal{S}$, $\mathcal{M}$ executes $\mathsf{Verify}_{\mathcal{S}}$ algorithm. Firstly, $\mathcal{M}$ verifies the validity of ω_0 in ω based on the signature $SSig_{ssk_{\mathcal{U}}}(\omega_0)$. If the verification fails, $\mathcal{M}$ returns $\perp$, otherwise, recovers the total block number n and $name$ from ω_0. Once the validation passes, $\mathcal{M}$ computes $\gamma = h_2(T)$ and utilizes the block number of blockchain provided by $\mathcal{S}$ to compute the same challenges $\{(i, \nu_i)\}_{i\in I}$. Then $\mathcal{M}$ checks the verification equation:

$$T \cdot e(\Sigma^\gamma, g_2) \stackrel{?}{=} e\left(\left(\prod_{i\in I} h_1(W_i)^{\nu_i}\right)^\gamma \cdot \bar{g}^{\mu}_{\mathcal{U}}, v_{\mathcal{U}}\right) \cdot e(g_*, g_2)^{\varsigma}. \tag{1}$$

If the above equation holds, then $\mathcal{M}$ outputs $\top$, otherwise returns $\perp$.

Verifying $\mathcal{F}_i$'s proof is simpler than that of $\mathcal{S}$. Firstly, $\mathcal{M}$ parses Ω_i as $name_{rep} \parallel name \parallel i \parallel s \parallel sig$. If sig is an invalid signature from $\mathcal{S}$ for $(name_{rep} \parallel name \parallel i \parallel s)$, then $\mathcal{M}$ aborts and returns $\perp$, otherwise, $\mathcal{M}$ calculates the same challenge $\{(i, \nu_j)\}_{j\in J}$ like $\mathcal{F}_i$, and verifies the following equation:

$$e(\sigma^*_i, g_2) \stackrel{?}{=} e\left(\prod_{j\in J} h_1(W'_{i,j})^{\nu_j} \cdot \bar{g}_{\mathcal{S}}^{\mu^*_j}, v_{\mathcal{S}}\right). \tag{2}$$

If the above equation holds, then the verification result is $\top$, otherwise $\perp$.

5.3 Decentralized Network Construction

Based on the auditing algorithms in Section 5.2, in this part, we give detailed construction of BOSSA. In BOSSA, each party can be viewed as a state machine, and we illustrate the state diagram of $\mathcal{F}$ and $\mathcal{M}$ in Fig. 4 and Fig. 5, respectively.

Farmer enrollment. The farmer who wants to join the network should register before providing storage services. Specifically, the farmer $\mathcal{F}$ submits a capability B_{end} representing when its services expire. To prevent $\mathcal{F}$ from leaving the network arbitrarily, it must put a deposit into the network, which is recorded by D in $\mathsf{L}.Roster(\mathcal{F}.Addr)$. The list $services$ is set to be empty during $\mathcal{F}$'s enrollment. At the same time, $\mathcal{M}$ sets B_{start} to be B_{now}.

Outsourcing original data. We assume that both $\mathcal{U}$ and $\mathcal{S}$ have reached an agreement with the following parameters: interval Itr_s, price p_s, radius r_s. $\mathcal{U}$ initializes a smart contract instance on $\mathcal{M}$ with these parameters when it uploads the data to $\mathcal{S}$. The details can be found in Fig. 6.

Outsourcing replicas. The process for outsourcing replicas is detailed in Fig. 6. Note that for ease of presentation, we assume that $\mathcal{S}$ is required to generate one replica of the original data.

A malicious $\mathcal{S}$ may control several dummy farmers and pretend to store replicas honestly, and generate proof on-the-fly during auditing. To prevent such a sybil attack from $\mathcal{S}$, $\mathcal{M}$ selects farmers randomly. Concretely, $\mathcal{M}$ calculates the hash of farmers' addresses concatenated with the latest block, and checks whether the hashed value is less than a threshold th. This guarantees that the selection of farmers is randomized. Farmers can previously calculate the hash to check whether it can be selected before it sends the message to $\mathcal{M}$. Besides, when $\mathcal{F}$ wants to store replicas, $\mathcal{M}$ checks B_{start} of $\mathcal{F}$, namely $B_{start} + \Delta < B_{now}$. This prevents a malicious $\mathcal{S}$ from generating dummy farmers on-the-fly. It is clear that only when a malicious $\mathcal{S}$ pre-generates numerous dummy farmers, can $\mathcal{S}$ performs sybil attacks, and it requires a large amount of deposit.

Outsourcing Original Data:

$\mathcal{U}$:
1: $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, sk_{\mathcal{S}}, pk_{\mathcal{S}}) \leftarrow \mathcal{U}.\mathsf{Setup}(1^{\lambda})$
2: $(D^*, aux) \leftarrow \mathcal{U}.\mathsf{Store}(D, sk_{\mathcal{U}})$
3: send D^*, aux to $\mathcal{S}$
4: put deposit of D to $\mathcal{M}$ and send ori-store, $name$, $pk_{\mathcal{U}}$, p_s, ltr_s and r_s to $\mathcal{M}$

$\mathcal{M}$:
1: upon receiving ori-store from $\mathcal{U}$
2: initialize an empty item of $\mathsf{L}.Audit$ indexed by $name$, set B_{last} as B_{now}, and store $(B_{last}, \mathsf{D}, pk_{\mathcal{U}}, \mathsf{ltr}_s, \mathsf{p}_s, \mathsf{r}_s)$ in $\mathsf{L}.Audit(name)$
3: broadcast ori-store-confirmed

Outsourcing Replicas:

$\mathcal{U}$:
1: send $sk_{\mathcal{S}}, pk_{\mathcal{S}}$, $name$ and $name_{rep}$ to $\mathcal{S}$
2: put deposit of $\mathsf{D}_{\mathcal{U}}$ in $\mathcal{M}$, and send rep-store, $name_{rep}$, $pk_{\mathcal{S}}$, ltr_f, r_f and p_f to $\mathcal{M}$

$\mathcal{M}$:
1: upon receiving rep-store from $\mathcal{M}$
2: initialize an item of $\mathsf{L}.Replica$ indexed by $name_{rep}$, initialize an empty dictionary $\mathcal{L}$ and store the tuple ($\mathsf{D}_{\mathcal{U}}$, $pk_{\mathcal{S}}$, ltr_f, p_f, r_f, $\mathcal{L}$) in $\mathsf{L}.Replica(name_{rep})$
3: broadcast rep-store-recruit

$\mathcal{F}$:
1: send deposit of $\mathsf{D}_{\mathcal{F}}$ together with rep-store-enroll, $name_{rep}$ to $\mathcal{M}$

$\mathcal{M}$:
1: upon receiving rep-store-enroll from $\mathcal{F}$
2: take the newest block as a PRF seed sd
3: **assert** $h_3(\mathcal{F}.Addr \parallel sd) < th$
4: **assert** $\mathsf{L}.Roster(\mathcal{F}.Addr).B_{start} + \Delta < B_{now}$
5: initialize a tuple (0, 0, $\mathsf{D}_{\mathcal{F}}$, $\perp$, $\perp$)
6: store the tuple in $\mathsf{L}.Replica(name_{rep}).\mathcal{L}$ with the index $\mathcal{F}.Addr$
7: broadcast rep-store-enrolled

$\mathcal{S}$:
1: generate an encryption key ek by KGen
2: $R \leftarrow \mathcal{S}.\mathsf{Seal}(D^*, ek)$
3: $\{R_i, aux'_i\}_{1 \le i \le K} \leftarrow \mathcal{S}.\mathsf{Distribute}(R, sk_{\mathcal{S}})$
4: pick K farmers and for each farmer:
5: $sig_i \leftarrow SSig_{ssk_{\mathcal{S}}}(h_3(R_i))$
6: send (R_i, aux'_i, sig_i) to the farmer

$\mathcal{F}$:
1: store replica blocks locally
2: send rep-store-finalize, sig_i to $\mathcal{M}$

$\mathcal{M}$:
1: upon receiving rep-store-finalize from $\mathcal{F}$
2: update $\mathsf{L}.Replica(name_{rep}).\mathcal{L}(\mathcal{F}.Addr)$ to $(0, 0, \mathsf{D}_{\mathcal{F}}, sig_i, B_{now})$
3: add $name_{rep}$ into $\mathsf{L}.Roster(\mathcal{F}.Addr).services$
4: broadcast rep-store-confirmed

Fig. 6. Data outsourcing.

Proving for Reward ($\mathcal{S}$):

$\mathcal{S}$:
1: set $B_{sel} = B_{now}$, $\phi \leftarrow \mathcal{S}.\mathsf{Prove}(D^*, aux)$
2: send ori-proof, ϕ and B_{sel} to $\mathcal{M}$

$\mathcal{M}$:
1: on receiving ori-proof, extract $name$ from ϕ
2: load B_{last}, ltr, r from $\mathsf{L}.Audit(name)$
3: **assert** $B_{last} + \mathsf{ltr} = B_{sel} \le B_{now} \le B_{last} + \mathsf{ltr} + \mathsf{r}$
4: **if** $\mathsf{Verify}_{\mathcal{S}}(\phi) = \top$ **then**
5: $\mathsf{L}.Audit(name).B_{last} = B_{sel}$
6: sent monetary reward of $\mathsf{L}.Audit(name).\mathsf{p}$ to $\mathcal{S}$ from $\mathcal{U}$'s deposit $\mathsf{L}.Audit(name).\mathsf{D}$
7: **end if**
8: broadcast ori-proof-verified

Proving for Reward ($\mathcal{F}$):

$\mathcal{F}$:
1: set $B_{sel} = B_{now}$, $\phi' \leftarrow \mathcal{F}.\mathsf{Prove}(R'_i, aux')$
2: send rep-proof, ϕ' and B_{sel} to $\mathcal{M}$

$\mathcal{M}$:
1: on receiving rep-proof, extract $name_{rep}$ from ϕ'
2: get B_{last} from $\mathsf{L}.Replica(name_{rep}).\mathcal{L}(\mathcal{F}.Addr)$ and get ltr, r from $\mathsf{L}.Replica(name_{rep})$
3: **assert** $B_{now} \le \mathsf{L}.Roster(\mathcal{F}.Addr).B_{end}$
4: **assert** $B_{last} + \mathsf{ltr} = B_{sel} \le B_{now} \le B_{last} + \mathsf{ltr} + \mathsf{r}$
5: **if** $\mathsf{Verify}_{\mathcal{F}}(\phi') = \top$ **then**
6: $\mathsf{L}.Replica(name_{rep}).\mathcal{L}(\mathcal{F}.Addr).B_{last} = B_{sel}$
7: load p from $\mathsf{L}.Replica(name_{rep}).\mathsf{p}$
8: deduct p from $\mathsf{L}.Replica(name_{rep}).\mathsf{D}$, send monetary reward of $\xi \cdot \mathsf{p}$ to $\mathcal{F}$ and store the rest into $\mathsf{L}.Replica(name_{rep}).\mathcal{L}(\mathcal{F}.Addr).\mathsf{D}$
9: **end if**
10: broadcast rep-proof-verified

Fig. 7. $\mathcal{F}$ and $\mathcal{S}$ proving for reward.

Proving for reward. Both $\mathcal{S}$ and $\mathcal{F}$ prove data integrity for earning rewards through the time-restricted proof, and they are rewarded based on the reward mechanism proposed in Section 4.5. If the proof is valid, $\mathcal{M}$ sets B_{last} to B_{sel}, then rewards the provers and updates $\mathcal{U}$'s deposit. Note that $\mathcal{M}$ should check that $\mathcal{F}$'s service has not expired to prevent $\mathcal{F}$ from cheating.

Fetching replica. When $\mathcal{S}$ tries to reconstruct the original data from replicas, it sends message rep-fetch to $\mathcal{M}$ together with $name_{rep}$. Upon receiving the message from $\mathcal{S}$, $\mathcal{M}$ increases the $totalAsk$ of farmers by 1 who are in $\mathsf{L}.Replica(name_{rep}).\mathcal{L}$, and broadcasts a message rep-fetch-request for acknowledging farmers. A rational farmer will share its local replica block R_i with $\mathcal{S}$ for maximizing his profits (see Section 6.2.3). When $\mathcal{S}$ receives enough replica blocks, it can reconstruct the original data and validate replicas through the comparison with its previously-stored hashes. For those farmers who contribute their local replica blocks, $\mathcal{S}$ will notify $\mathcal{M}$ to increase $totalReply$ with message rep-fetch-finalize. And $\mathcal{M}$ broadcasts rep-fetch-confirmed for notifying farmers.

Farmer logout. When a farmer wants to leave the network and takes back all its deposit (including the frozen part of the reward), the following conditions must be met: 1) it is not storing replicas *i.e.*, $\mathsf{L}.Roster(\mathcal{F}.Addr).services = \emptyset$; 2) the storage service it promised should expire, *i.e.*, $B_{end} \le B_{now}$; 3) it proves correctly each time *i.e.*, $0 \le B_{end} - B_{last} < \mathsf{ltr}$. The first two conditions allow the farmer to take back its deposit during the enrollment, and the last one allows it to obtain the deposit for storing replica blocks (which is refunded based on the contribution rate). Besides, a new farmer must be found for restoring its replicas. Only after transferring all the replicas to the new farmer, can it gets back the frozen deposit. Details can be found in Fig. 8.

6 System Analysis

6.1 Choice of Challenge Size

The challenges against both $\mathcal{S}$ and $\mathcal{F}$ are based on the probabilistic framework [5], through which the workload of the provers can be noticeably reduced. This is because, during the auditing, the provers randomly sample a subset of stored data, as required by the verifier, instead of looking through all of the data. Meanwhile,

Replicas Transferring:

$\mathcal{F}_s$:
1: calculate $cm = h_3(r)$ based on a randomly picked seed r
2: send rep-transfer, $name_{rep}$ and cm to $\mathcal{M}$

$\mathcal{M}$:
1: upon receiving rep-transfer
2: load ltr from $\mathsf{L}.Replica(name_{rep})$, load B_{last} from $\mathsf{L}.Replica(name_{rep}).\mathcal{L}(\mathcal{F}_s.Addr)$, and load B_{end} from $\mathsf{L}.Roster(\mathcal{F}_s.Addr)$
3: **assert** $B_{end} \leq B_{now}$
4: **assert** $0 \leq B_{end} - B_{last} <$ ltr
5: store cm and publish $name_{rep}$
6: broadcast rep-transfer-request

$\mathcal{F}_d$:
1: upon receiving rep-transfer-request from $\mathcal{M}$
2: request r and replica block from $\mathcal{F}_s$
3: access $\mathsf{L}.Replica(name_{rep})$ and load sig_i from $\mathcal{L}(\mathcal{F}_s.Addr)$
4: **assert** replica block is valid based on sig_i
5: put deposit of D to $\mathcal{M}$ and send rep-transfer-finalize, r, $\mathcal{F}_s.Addr$ and $name_{rep}$ to $\mathcal{M}$

$\mathcal{M}$:
1: upon receiving rep-transfer-finalize
2: **assert** $cm = h_3(r)$
3: remove $name_{rep}$ from $\mathsf{L}.Roster(\mathcal{F}_s.Addr).services$
4: remove $\mathcal{F}_s$ from $\mathsf{L}.Replica(name_{rep}).\mathcal{L}$
5: calculate contribution rate cr of $\mathcal{F}_s$
6: refund $\mathsf{L}.Replica(name_{rep}).\mathcal{L}(\mathcal{F}_d.Addr).\mathsf{D}$ to $\mathcal{F}_s$ based on the contribution rate cr, and rest of deposit is refunded to $\mathcal{U}$
7: add a new tuple $(0, 0, \mathsf{D}, sig_i, B_{now})$ into $\mathsf{L}.Replica(name_{rep}).\mathcal{L}(\mathcal{F}_d.Addr)$
8: broadcast rep-transfer-confirmed

Farmer Logout:

$\mathcal{F}_s$:
1: give all of local replica blocks to new farmers
2: send logout-request to $\mathcal{M}$ for logout

$\mathcal{M}$:
1: upon receiving logout-request from $\mathcal{M}$
2: load B_{end} from $\mathsf{L}.Roster(\mathcal{F}_s.Addr)$
3: **assert** $B_{end} \leq B_{now}$
4: **assert** $\mathsf{L}.Roster(\mathcal{F}_s.Addr).services$ is empty
5: give $\mathcal{F}_s$'s deposit back
6: broadcast logout-confirmed

Fig. 8. $\mathcal{F}$ leaves the network.

with the aid of unbiased-random challenges (usually picked by the verifier), the cheater will be caught with an overwhelming probability, if the data loss is beyond a bound.

Supposing that the prover tampers with $n \cdot t, t \in (0, 1)$ blocks out of n data blocks in total, and he is asked to sample c blocks. We let X denote a discrete random variable representing the number of deleted blocks chosen during the proof, and we define P_X as the corresponding probability of X. Thus we have:

$$\begin{aligned} P_X &= P\{X \geq 1\} \\ &= 1 - P\{X = 0\} \\ &= 1 - \frac{n - n \cdot t}{n} \cdot \frac{n - 1 - n \cdot t}{n - 1} \cdots \frac{n - c + 1 - n \cdot t}{n - c + 1} \\ &\geq 1 - (1 - t)^c. \end{aligned}$$

Clearly, if we fix the number of t, P_X is independent of the number of data blocks. For example, when $t = 1\%$, P_X can reach 95% as long as the number of sampled blocks is not less than 300 (see Fig. 9).

The workloads of proving and verification are related to the number of challenges, namely the size of c. However, since both original data and replica are encoded with the erasure code, we can sacrifice storage space in exchange for efficient verification and proving. Recall that a (n_2, k_2) encoding is applied on each column of replica which is individually stored by farmers, and the redundancy rate is n_2/k_2. If we raise the redundant rate, we could take a less rigorous challenge with a smaller challenge size. For example, if we take a $2\times$ redundancy-rate encoding, we can aggressively reduce challenge size to 7, as the data loss of 50% can be detected with a 99% probability (as depicted in Fig. 9) while the whole replica can still be recovered.

6.2 Security Analysis

6.2.1 Storage Correctness

Definition 9. *The Computational Diffie-Hellman assumption is define as follows. Given a tuple $(g, g^x, g^y) \in \mathbb{G}$, where $\mathbb{G}$ is a multiplicative cyclic group in order of p, g is a generator of $\mathbb{G}$, and $x, y \xleftarrow{R} \mathbb{Z}_p$, then the probability that a PPT adversary $\mathcal{A}$ outputs g^{xy} is no more than a negligible probability ϵ:*

$$Pr[\mathcal{A}(g, g^x, g^y) = g^{xy}] < \epsilon.$$

Theorem 1. *In the random oracle model, if CDH assumption holds in bilinear groups, then BOSSA guarantees the storage correctness.*

Proof (sketch). Firstly, we prove that an honest $\mathcal{S}$ can always pass the verification. To show that, for the equation (1), we have:

$$\begin{aligned} LHE =& T \cdot e\left(\Sigma^{\gamma}, g_2\right) \\ =& e\left(g_*, g_2\right)^{r_\delta} \cdot e\left(\bar{g}_{\mathcal{U}}, \upsilon_{\mathcal{U}}\right)^{r_m} \cdot \\ & e\left(\prod_{i \in I} \left(h_1\left(W_i\right)^{\nu_i} \cdot \bar{g}_{\mathcal{U}}^{m_i \cdot \nu_i}\right)^{x_{\mathcal{U}} \cdot \gamma} \cdot g_*^{\rho\gamma}, g_2\right) \\ =& e\left(g_*, g_2\right)^{r_\sigma + \rho\gamma} \cdot e\left(\prod_{i \in I} h_1\left(W_i\right)^{\nu_i \cdot \gamma} \cdot \bar{g}_{\mathcal{U}}^{r_m + \gamma\mu'}, \upsilon_{\mathcal{U}}\right) \\ =& e\left(g_*, g_2\right)^{\varsigma} \cdot e\left(\left(\prod_{i \in I} h_1\left(W_i\right)^{\nu_i}\right)^{\gamma} \cdot \bar{g}_{\mathcal{U}}^{\mu}, \upsilon_{\mathcal{U}}\right) = RHE. \end{aligned}$$

Thus, this property ensures the soundness of BOSSA, and guarantees the interests of $\mathcal{S}$. Then, we prove that for an ε-*admissible* $\mathcal{S}$ who can correctly provide ε-fraction proofs, original data can be extracted from it.

We assume the verifier controls a random oracle h_2 and it answers hash queries from $\mathcal{S}$. Upon receiving the proof from $\mathcal{S}$, the verifier rewinds to the point when $\mathcal{S}$ queries $h_2(T)$, and sends a different $\gamma^* = h_2(T)$ to $\mathcal{S}$. Having two different proofs (μ, Σ, ς) and $(\mu^*, \Sigma, \varsigma^*)$, the verifier extracts μ' and σ by calculating

$$\begin{aligned} \mu' &= \frac{\mu - \mu^*}{\gamma - \gamma^*} = \frac{(r_m + \gamma\mu') - (r_m + \gamma^*\mu')}{\gamma - \gamma^*}, \\ \rho &= \frac{\varsigma - \varsigma^*}{\gamma - \gamma^*} = \frac{(r_m + \gamma\rho) - (r_m + \gamma^*\rho)}{\gamma - \gamma^*}, \\ \sigma &= \frac{\Sigma}{g_*^{\rho}}. \end{aligned}$$

The pair (μ', σ) is the proof for verification in [10], and based on Theorem 4.2 in [10], if $\mathcal{S}$ forges both μ' and σ, it cannot pass

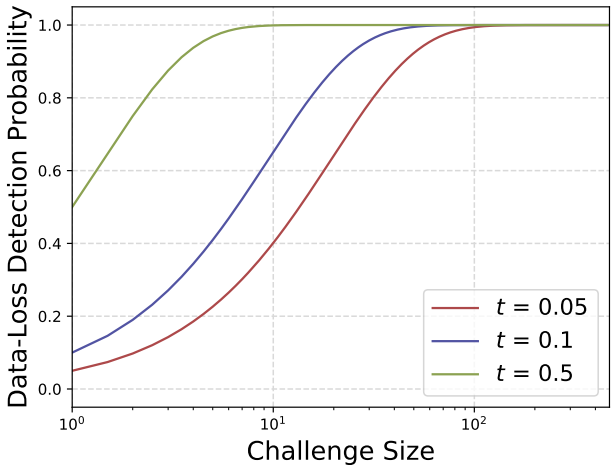


Fig. 9. Data-loss detection probability *vs.* challenge size. We give an illustration about challenge sizes for achieving various loss-detection probabilities under different degrees of t.

the verification, except with a negligible probability. Then, based on Theorem 4.3 in [10], if $\mathcal{S}$ gives at least ε valid proofs to the verifier, the data can be recovered from the proofs through an extraction algorithm.

As for an ε-$admissible$ $\mathcal{F}$, the storage correctness is achieved based on theorems in [10]. □

6.2.2 Privacy Preservation

Theorem 2. *In the random oracle model, if* Enc *is CPA-secure and Diffie-Hellman problem is hard,* BOSSA *achieves privacy preservation defined in Definition 4.*

Proof (sketch). The replicas of original data are encrypted through Enc before being distributed, hence participants of the decentralized network cannot learn private information as long as they don't have the encryption key, and the auditing procedure also leaks no private information. So, the point is that there should be no privacy leakage during auditing $\mathcal{S}$.

To prove privacy preservation holds for $\mathcal{S}$, we show a simulator without knowing μ' and σ can provide a valid proof to $\mathcal{M}$. We assume it controls the random oracle h_2 and answers queries from $\mathcal{M}$. To provide a valid proof to $\mathcal{M}$, the simulator randomly picks γ and $\mu' \in \mathbb{Z}_p$ and $\sigma \in \mathbb{G}_1$, and calculates μ, ς and Σ. Then, the simulator lets

$$T = e\left(\left(\prod_{i\in I} h_1\left(W_i\right)^{\nu_i}\right)^{\gamma} \cdot \bar{g}_{\mathcal{U}}^{\mu}, \upsilon_{\mathcal{U}}\right) \cdot \frac{e\left(g_*, g_2\right)^{\varsigma}}{e\left(\Sigma^{\gamma}, g_2\right)}$$

and fills the random oracle $h_2(T)$ with γ. The simulator's proof is $(T, \varsigma, \mu, \Sigma)$. It is clear that the simulator's proof is valid if the simulator answers γ when $\mathcal{M}$ queries $h_2(T)$. The simulator knows nothing about μ' and σ, it means the proof leaks no information about $\mathcal{S}$. □

6.2.3 Replica Reliability

According the analysis in Section 6.2.1, as long as $\mathcal{F}$ provides a valid proof, it can convince the verifier that the data is stored properly. According to our reward distribution mechanism, rewards from the user will be given to $\mathcal{F}$. Since partial rewards are temporarily frozen in the deposit of $\mathcal{F}$, the more $\mathcal{F}$ proves, the more deposit it will accumulate. When it wants to get the deposit back, the deposit is refunded based on the contribution rate cr, which is determined by whether $\mathcal{F}$ contributes its local replicas when there is a request for replicas from $\mathcal{S}$. Hence, a rational $\mathcal{F}$ who is eager to maximize its profits will actively contribute its local replica blocks.

Finally, we remark that as replicas are encoded by erasure codes, when $\mathcal{F}$ goes offline accidentally during the replica fetching, BOSSA can still guarantee the reliability of the replicas, as long as the geographical locations of the farmers are sufficiently dispersed.

TABLE 2
Overheads of Proof Generation and Verification for 100M Data

	$\mathcal{S}$		$\mathcal{F}$
File size (MB)	100		100
Sample blocks	300	480	7
Proving (ms)	33.2	48.8	10.8
Verification (gas)	27,063,392	42,979,515	997,151
USD	12.18	19.34	0.45

7 Implementation & Evaluation

7.1 Implementation Setup

All the parties in our prototype are instantiated on a local Ubuntu 18.04 LTS with an Intel Core i5-8400 CPU, 12GB of RAM, and a 7200 RPM Seagate 1 TB hard disk. We adopt Ethereum as the underlying blockchain system in our implementation. The smart contracts are written using Solidity and deployed to Ganache-cli (a local Ethereum simulation) and Ropsten (an official testnet). Other parties are implemented with JavaScript combined with C/C++. We utilize Keccak256 as the hash function and 128-bit-of-security Barreto-Naehrig curve as the ellipse curve. We use ate-pairing [49] for bilinear paring since it is based on the elliptic curve suitable for Ethereum. And we use Jerasure, an instance of Reed-Solomon coding, for implementing erasure codes. To evaluate economic costs, we take the exchange rate of $3 \cdot 10^{-9}$ Ether per gas (namely 3 Gwei per gas), and 150 USD per Ether (December, 2019).

7.2 Circumventing Gas Limitation of Ethereum

To prevent malicious users from launching DDoS attacks by sending infinite loops to miners, Ethereum uses *gas* to measure overheads for running smart contracts, and sets a limitation upon it. To circumvent the gas limitation, the implementation of our prototype is slightly different from our design. In BOSSA, the main gas costs are caused by the multiplications and hashes related to the size of challenges. In our prototype, we split those procedures into several sub-functions whose gas costs are below the gas bound. For example, if it is required to perform 300 times of $\mathbb{G}_1$ multiplications, we then run sub-functions 10 times, each of which runs $\mathbb{G}_1$ multiplication 30 times. This enables us to avoid exceeding the gas limitation. Note that this substitution increases total gas costs since we must store temporary variables and trigger multiple calls of the smart contracts.

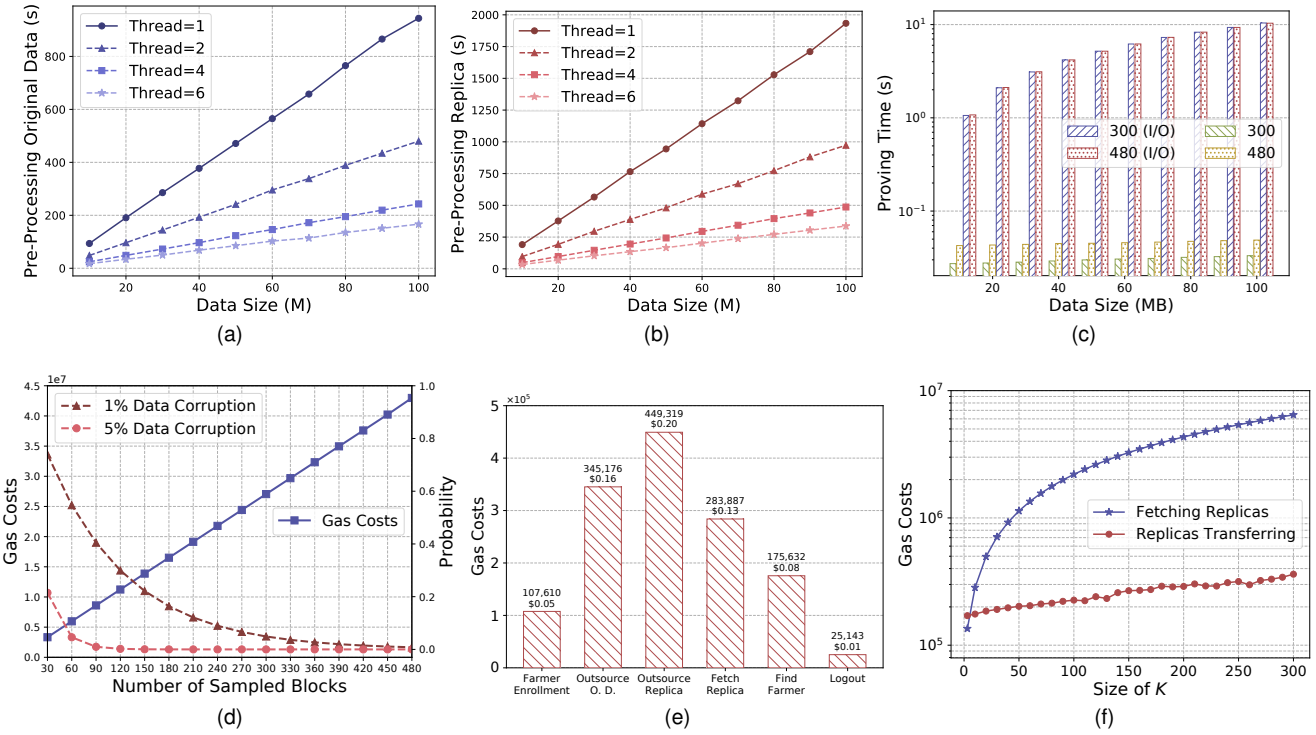


Fig. 10. Time and gas costs of BOSSA. (a) Time costs for pre-processing original data using multiple threads *vs.* data size. (b) Time costs for pre-processing one replica using multiple threads *vs.* data size. (c) The time costs of $\mathcal{S}$'s proof generation under different challenge sizes with and without I/O. (d) The probabilities of failing to detect data corruption and the proof verification gas costs *vs.* the number of sampled blocks. (e) Gas costs of $\mathcal{M}$' actions when replica is stored by $K = 10$ farmers (O.D. stands for Original Data). (f)The gas cost of replica fetching and transferring *vs.* the number of farmers storing replica.

7.3 Experimental Results

Pre-processing. We firstly test time for generating key pairs (*i.e.*, the $\mathcal{U}$.Setup). The time cost is about 0.84ms, which is negligible. In Fig. 10(a), we measure the time costs of $\mathcal{U}$ for pre-processing the original data before outsourcing it to $\mathcal{S}$ (*i.e.*, the $\mathcal{U}$.Store algorithm) and Fig. 10(b) presents the time cost of $\mathcal{S}$ to pre-process one replica before distributing it to farmers (*i.e.*, the algorithms of $\mathcal{S}$.Distribute and $\mathcal{S}$.Seal). We randomly generate testing data ranging from 10MB to 100MB. The result shows that the time cost increases linearly with the data size. The time costs are dominated by authenticators generation, while both replica encryption and replica coding incur negligible costs. The generation of authenticators of 100M size costs about 164s on $\mathcal{U}$, and 330s for a replica of 200M (as we encode original data into double size) on $\mathcal{S}$.

Proof generation and on-chain verification. The overheads of proof generation on $\mathcal{S}$ and $\mathcal{F}$ are only related to the size of the challenge. We test the overheads of sampling 7 sectors of a replica block, and sampling both 300 and 480 (for about 99% loss detection probability) blocks of original data as discussed in Section 6.1. The results, shown in Table 2, demonstrate our proving process is efficient, and it only costs tens of milliseconds. In Fig. 10(c), we evaluate the I/O effect on the proof generation on $\mathcal{S}$. We can see that the data size greatly affects the time cost. This is due to the fact that the time cost of proof generation is mainly dominated by the I/O cost (*e.g.*, reading data and authenticators from hard drivers).

For the on-chain verification, the gas costs are $27,063,392$ and $42,979,515$ when sampling 300 blocks and 480 blocks (Fig. 10(d)), respectively. On the other hand, the gas cost of verifying

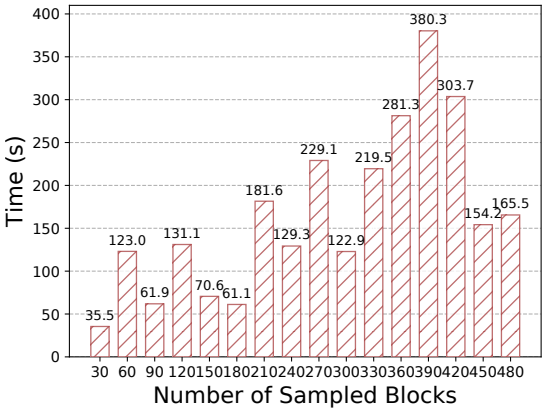


Fig. 11. Time cost of verification in Ropsten *vs.* the number of sampled blocks.

a farmer's proof is much lower, which is $997,151$ gas (*i.e.*, about \$0.45). Note that this cost is for a single farmer, and the total costs are K times larger when there are K farmers to store replica. Since the underlying blockchain platform may affect the time cost of the verification process, we further deploy the smart contracts to Ropsten, an official public Ethereum testnet. We measure the time cost from sending the proof to recording verification result on the chain. The time cost for verifying $\mathcal{S}$'s proof is presented in Fig. 11. It can be seen that although heavily affected by the throughput of the testnet and the latency of the network, the time cost still roughly increases with the number of the sampled blocks. Note that techniques like sharding [50], [51] and state channel [52] can

further improve the efficiency of verification.

Interactions with the smart contract. In BOSSA, other processes, such as initialization, data fetching, also need to interact with $\mathcal{M}$ (through the smart contracts). In Fig. 10(e), we evaluate the gas costs of those processes. We can see that those costs are much lower (about several cents) than that of proof verification. Meanwhile, among these processes, the costs of outsourcing original data and replicas are the highest. This is caused by the large call parameters (420B and 460B) for storing $\mathcal{U}$'s and $\mathcal{S}$'s public keys respectively.

Fig. 10(f) shows the impact of the number of farmers on the gas costs when fetching and transferring replicas. The gas costs increase linearly with the number of the farmers. This is due to the limitation of data structures supported in Ethereum. We add farmers' addresses into an array, such that it needs to traverse the whole array for searching a specific farmer. In addition, we have to simultaneously increase the counter $totalAsk$ for each farmer by 1 when fetching replicas, and it charges an extra fee due to the changing of the blockchain states. To avoid exceeding the gas limitation, $\mathcal{S}$ can fetch replica blocks separately.

8 Conclusion

In this paper, we proposed BOSSA, a novel decentralized framework for proofs of data replication and retrievability. Different from existing schemes, BOSSA is built atop off-the-shelf blockchain platforms, where each participant is fairly treated and incentivized to faithfully follow the auditing protocol. We proposed a proof enforcement mechanism to catch lazy behaviors, and a new metric as well as a reward distribution mechanism to create a fair reciprocal environment. Our experiments show that BOSSA incurs tolerable costs and is feasible in practice.

Acknowledgments

Qian Wang's research was supported in part by the NSFC under Grants 61822207 and U1636219, in part by the Outstanding Youth Foundation of Hubei Province under Grant 2017CFA047, and in part by the Fundamental Research Funds for the Central Universities under Grant 2042019kf0210. Cong Wang's research was supported in part by Research Grants Council of Hong Kong under Grants CityU 11212717 and CityU 11217819, and by the Innovation and Technology Commission of Hong Kong under ITF Project ITS/145/19. Shengshan Hu's research was supported by the Fundamental Research Funds for the Central Universities under Grant 2020kfyXJJS075. Qian Wang is the corresponding author.

References

[1] "Data protection in amazon s3," https://docs.aws.amazon.com/AmazonS3/latest/dev/DataDurability.html.
[2] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, and Z. Zhao, "A blockchain based witness model for trustworthy cloud service level agreement enforcement," in *Proc. of INFOCOM*. IEEE, 2019, pp. 1567–1575.
[3] "Tencent cloud user claims $1.6 million compensation for data loss," https://technode.com/2018/08/06/tencent-cloud-user-claims-1-6-million-compensation-for-data-loss.
[4] "Google loses data as lightning strikes," https://www.bbc.com/news/technology-33989384.
[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. of CCS*. ACM, 2007, pp. 598–609.
[6] A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage," *IEEE Transactions on Cloud Computing*, 2018.
[7] J. Xu, A. Yang, J. Zhou, and D. S. Wong, "Lightweight delegatable proofs of storage," in *Proc. of ESORICS*. Springer, 2016, pp. 324–343.
[8] F. Armknecht, J.-M. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter, "Outsourced proofs of retrievability," in *Proc. of CCS*. ACM, 2014, pp. 831–843.
[9] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proc. of CCS*. ACM, 2007, pp. 584–597.
[10] H. Shacham and B. Waters, "Compact proofs of retrievability," *Journal of Cryptology*, vol. 26, no. 3, pp. 442–483, 2013.
[11] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011.
[12] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.
[13] F. Armknecht, L. Barman, J.-M. Bohli, and G. O. Karame, "Mirror: Enabling proofs of data replication and retrievability in the cloud." in *Proc. of USENIX Security*, 2016, pp. 1051–1068.
[14] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "Mr-pdp: Multiple-replica provable data possession," in *Proc. of ICDCS*. IEEE, 2008, pp. 411–420.
[15] I. Leontiadis and R. Curtmola, "Secure storage with replication and transparent deduplication," in *Proc. of CODASPY*. ACM, 2018, pp. 13–23.
[16] I. Damgard, C. Ganesh, and C. Orlandi, "Proofs of replicated storage without timing assumptions," in *Proc. of CRYPTO*. Springer, 2019, pp. 355–380.
[17] J. Benet, D. Dalrymple, and N. Greco, "Proof of replication," Tech. Rep., 2017.
[18] B. Fisch, "Poreps: Proofs of space on useful data," Tech. Rep., 2018.
[19] B. Fisch, J. Bonneau, N. Greco, and J. Benet, "Scaling proof-of-replication for filecoin mining," Tech. Rep., 2018.
[20] A. B. Ayed, "A conceptual secure blockchain-based electronic voting system," *International Journal of Network Security & Its Applications*, vol. 9, no. 3, pp. 1–9, 2017.
[21] S. Wu, Y. Chen, Q. Wang, M. Li, C. Wang, and X. Luo, "Cream: a smart contract enabled collusion-resistant e-auction," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 7, pp. 1687–1701, 2018.
[22] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proc. of CCS*. ACM, 2018, pp. 967–984.
[23] "Filecoin," https://filecoin.io/filecoin.pdf.
[24] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
[25] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on distributed cryptocurrencies and consensus ledgers*, vol. 310, no. 4, 2016.
[26] C. Guan, K. Ren, F. Zhang, F. Kerschbaum, and J. Yu, "Symmetric-key based proofs of retrievability supporting public verification," in *Proc. of ESORICS*. Springer, 2015, pp. 203–223.
[27] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proc. of ASIACRYPT*. Springer, 2001, pp. 514–532.
[28] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in *Proc. of ISDPE*. IEEE, 2010, pp. 84–89.
[29] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Proc. of CRYPTO*. Springer, 2018, pp. 757–788.
[30] "Storj," https://storj.io/white-paper.
[31] "Sia: Simple decentralized storage," https://sia.tech/sia.pdf.
[32] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[33] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. of S&P*. IEEE, 2014, pp. 475–490.
[34] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai, "Retricoin: Bitcoin based on compact proofs of retrievability," in *Proc. of ICDCN*. ACM, 2016, pp. 1–10.
[35] S. Ruj, M. S. Rahman, A. Basu, and S. Kiyomoto, "Blockstore: A secure decentralized storage framework on blockchain," in *Proc. of AINA*. IEEE, 2018, pp. 1096–1103.
[36] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

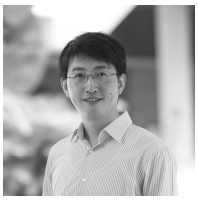
[37] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. of CRYPTO*. Springer, 2017, pp. 357–388.
[38] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *Proc. of INFOCOM*. IEEE, 2018, pp. 792–800.
[39] S. Hu, C. Cai, Q. Wang, C. Wang, Z. Wang, and D. Ye, "Augmenting encrypted search: A decentralized service realization with enforced execution," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, DOI: 10.1109/TDSC.2019.2 957 091, 2019.
[40] "Libra," https://libra.org/en-US/white-paper/.
[41] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *White paper*, 2014.
[42] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proc. of EuroSys*. ACM, 2018, pp. 1–15.
[43] S. Hu, L. Y. Zhang, Q. Wang, Z. Qin, and C. Wang, "Towards private and scalable cross-media retrieval," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, DOI: 10.1109/TDSC.2019.2 926 968, 2019.
[44] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, pp. 496–510, 2018.
[45] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, 2012.
[46] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and A. Mohaisen, "Exploring the attack surface of blockchain: A systematic overview," *arXiv preprint arXiv:1904.03487*, 2019.
[47] J. Katz and Y. Lindell, *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.
[48] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
[49] "ate-pairing," https://github.com/herumi/ate-pairing.
[50] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. of CCS*. ACM, 2016, pp. 17–30.
[51] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: scaling blockchain via full sharding," in *Proc. of CCS*. ACM, 2018, pp. 931–948.
[52] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Proc. of FC*. Springer, 2019, pp. 508–526.

Dian Chen received the B.E. degree from Wuhan University, Wuhan, China, in 2018. He is working toward Master degree in the School of Cyber Science and Engineering in Wuhan University. His research interests include information security and blockchain technology.

Haobo Yuan is an undergraduate student majoring in Computer Science and Technology at Wuhan University. His research interests include information security and blockchain technology.

Qian Wang received the Ph.D. degree from the Illinois Institute of Technology, Chicago, IL. He is currently a professor with the School of Cyber Science and Engineering, Wuhan University. His research interests include AI security, data storage, search and computation outsourcing security and privacy, wireless system security, and applied cryptography etc. He received National Science Fund for Excellent Young Scholars of China, in 2018. He is also an expert under National "1000 Young Talents Program" of China. He is a recipient of the 2018 IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researcher), and the 2016 IEEE Asia-Pacific Outstanding Young Researcher Award. He is also a co-recipient of several best paper awards from IEEE DSC'19, IEEE ICDCS'17, IEEE TrustCom'16, WAIM'14, and IEEE ICNP'11 etc. He serves as associate editors for the IEEE Transactions on Dependable and Secure Computing (TDSC), IEEE Transactions on Information Forensics and Security (TIFS), and IEEE Internet of Things Journal (IoT-J). He is a Senior Member of the IEEE and a Member of the ACM.

Shengshan Hu received the B.E. and Ph.D. degrees in computer science and technology from Wuhan University in 2014 and 2019, respectively. He is currently an Associate Professor in the School of Cyber Science and Engineering, Huazhong University of Science and Technology. His research interest focuses on privacy-enhancing technologies, AI security, and blockchain.



Cong Wang is an Associate Professor in the Department of Computer Science, City University of Hong Kong. His current research interests include data and network security, blockchain and decentralized applications, and privacy-enhancing technologies. He is one of the Founding Members of the Young Academy of Sciences of Hong Kong. He received the Outstanding Researcher Award (junior faculty) in 2019, the Outstanding Supervisor Award in 2017 and the President's Awards in 2019 and 2016, all from City University of Hong Kong. He is a co-recipient of the IEEE INFOCOM Test of Time Paper Award 2020, Best Student Paper Award of IEEE ICDCS 2017, and the Best Paper Award of IEEE ICPADS 2018 and MSN 2015. His research has been supported by multiple government research fund agencies, including National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He serves/has served as associate editor for IEEE Transactions on Dependable and Secure Computing, IEEE Internet of Things Journal and IEEE Networking Letters, and TPC co-chairs for a number of IEEE conferences/workshops. He is a senior member of the IEEE, and member of the ACM.