Retricoin: Bitcoin Based on Compact Proofs of Retrievability

Binanda Sengupta
Indian Statistical Institute
Kolkata, India
binanda_r@isical.ac.in

Samiran Bag
Kyushu University
Fukuoka, Japan
bag.samiran.012@m.kyushu-u.ac.jp

Sushmita Ruj
Indian Statistical Institute
Kolkata, India
sush@isical.ac.in

Kouichi Sakurai
Kyushu University
Fukuoka, Japan
sakurai@csce.kyushu-u.ac.jp

ABSTRACT

Bitcoin [24] is a fully decentralized electronic cash system. The generation of the proof-of-work in Bitcoin requires large amount of computing resources. However, this huge amount of energy is wasted as one cannot make something useful out of it. In this paper, we propose a scheme called *Retricoin* which replaces the heavy computational proof-of-work of Bitcoin by proofs of retrievability that have practical benefits. To guarantee the availability of an important but large file, we distribute the segments of the file among the users in the Bitcoin network. Every user who wants to mine Bitcoins must store a considerable portion of this file and prove her storage to other peers in the network using proofs of retrievability. The file can be constructed at any point of time from the users storing their respective segments untampered. Retricoin is more efficient than the existing Permacoin scheme [23] in terms of storage overhead and network bandwidth required to broadcast the proof to the Bitcoin network. The verification time in our scheme is comparable to that of Permacoin and reasonable for all practical purposes. We also design an algorithm to let the miners in a group (or pool) mine collectively.

CCS Concepts

•Information systems → Digital cash; Distributed storage; •Theory of computation → Cryptographic protocols;

Keywords

Bitcoin, Proofs of Retrievability, Bilinear Pairings, Mining Pool

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICDCN '16, January 04-07, 2016, Singapore, Singapore
© 2016 ACM. ISBN 978-1-4503-4032-8/16/01...$15.00
DOI: http://dx.doi.org/10.1145/2833312.2833317

1. INTRODUCTION

In the age of electronic payments, we need not carry conventional cash with us every time we want to make a transaction. However, these online transactions over the Internet suffer from a serious drawback. There must be some central authorities who will verify each and every transaction over the Internet. This not only increases the transaction cost but also results in a trust-based model. Like other trust-based systems, it requires a trusted third party (for example, a financial institution) which is difficult to achieve in real scenario. Once the trusted party is corrupted, the whole trust-based system collapses.

An alternative to overcome this drawback of the trust-based electronic cash system is a fully decentralized electronic cash system where users in the network can make transactions among themselves, and every other user in the network can check the validity of a transaction. Nakamoto introduces such a peer-to-peer cryptocurrency system known as Bitcoin [24] in 2008, and the first transaction of Bitcoins takes place the next year. Since then Bitcoin has been the most successful cryptocurrency ever. Being a peer-to-peer electronic cash system, Bitcoin does not rely on any trusted server. Bitcoin users make payment by digitally signing a transaction with their secret keys. The Bitcoin network maintains a block chain which is a public ledger of all executed valid transactions. The block chain gets longer as a new block is appended to it in every 10 minutes. These blocks are generated by the miners (users trying to mine Bitcoins) in the network who provide a cryptographic *proof-of-work* (POW) to show that they have indeed expended a large amount of computational power. Presently, Bitcoin uses Back's Hashcash [4] as the proof-of-work. In this scheme, a miner needs to find the value of a nonce, by a brute-force search, such that the hash value (using SHA-256 as the hash function) of the nonce along with all valid transactions that are not included in any previously mined block is less than a predefined target value. The miner, presenting a valid proof-of-work first, is rewarded with Bitcoins as an incentive, and the mined block is added to the block chain.

Bitcoin is often criticized because of the huge amount of computational power a miner in the Bitcoin network wastes for minting currency. This massive amount of resources cannot be used for performing any useful task. There have been some works to address this problem and propose alternative solutions to fix it. King proposes the Primecoin scheme where the proof-of-work is based on searching for prime numbers [18]. Biryukov and Pustogarov exploit the proof-of-work for covering the expenses of Tor relay [8]. Miller et al. propose Permacoin where they replace the proof-of-work (POW) of Bitcoin by *proofs of retrievability* (POR) [23]. In this scheme, there is a dealer who remains online to serve chunks from a large file (of the order of petabytes). A miner downloads some segments of the file and stores them. For minting Permacoins, she has to provide a proof that she indeed possesses those segments locally. She issues a *ticket* containing the proof along with some additional data needed to verify the proof. After receiving the ticket, any user in the Permacoin network can verify the validity of the proof. The original large file is retrievable from the chunks stored by the honest users.

Our Contribution In this paper, we propose Retricoin, a new cryptocurrency, that makes use of proofs of retrievability (POR) [28] to repurpose the mining work in order to ensure the retrievability of a large file F at any point of time. We design Retricoin in such a way that the rational miners store their data locally instead of outsourcing them to some data server. In terms of efficiency, Retricoin is an improvement over the existing Permacoin scheme [23]. To be more specific, we achieve

- less storage overhead than that of Permacoin,
- lower network bandwidth compared to Permacoin,
- verification time comparable to that of Permacoin; and
- we design an algorithm to let the miners mine collectively that is not addressed in Permacoin.

We defer the detailed comparison between Retricoin and Permacoin to Section 5. For a quick review, we summarize the comparison in Fig. 1. Following Bitcoin's idea of creating *mining pools*, we propose an algorithm in which several miners in the network can form a mining pool. The miners in a pool work collaboratively and combine their individual storage to generate a proof for a larger storage, thus increasing their chance to be rewarded.

The rest of the paper is organized as follows. In Section 2, we briefly discuss about some cryptocurrencies and describe the notion of proofs of retrievability (POR). Section 3 provides some notations and tools that we use in the following sections. In Section 4, we describe our scheme called Retricoin. In Section 5, we provide a detailed comparison between Retricoin and the existing Permacoin scheme. Section 6 describes a method to let a group of miners, in our scheme, mine in a collective fashion. In the concluding Section 7, we summarize the work done in this paper.

2. RELATED WORK

In this section, we provide a brief overview of cryptocurrency and proofs of retrievability, and we review some related literatures.

Cryptocurrency.

The mining scheme in Bitcoin involves solving a cryptographic mining puzzle which is not precomputable. A miner in the Bitcoin network finds a solution of the puzzle in the following way: Let $T_1, T_2, \ldots, T_z$ be the valid transactions for a certain *epoch* (time period, on an average, in which a Bitcoin block is generated and appended to the Bitcoin block-chain) which are not included in any previous block. The miners try to find a nonce η such that SHA-256$(BH, T_1, T_2, \ldots, T_z, \eta) \leqslant Z$, where Z is a predefined target value and BH is the hash of the previous block. All miners in the network compete to find such a nonce that yields the desired hash value under SHA-256 hash operation. Due to the preimage-resistance property of SHA-256, the only way to compute such a nonce is to search, in a brute-force manner, over all possible values of the nonce. As of August 2015, the Bitcoin network performs 426 million Gigahashes (billion hashes) per second as per *https://blockchain.info/*. The Hashcash scheme [4] constitutes the proof-of-work of Bitcoin. Multiple miners in the Bitcoin network may collude to form a mining pool and combine their computational power in order to solve the puzzle sooner than other miners in the network in order to earn reward [27, 20, 14].

There are some alternative cryptocurrencies based on different types of mining puzzles. The proof-of-work in the Primecoin scheme searches for prime numbers [18]. Another scheme is due to Biryukov and Pustogarov [8]. Here, the idea is to use the mining power of Tor [12] relay users to generate the proof-of-work on behalf of the Tor relay network which is used by Tor to earn mining reward. This reward is used to meet the expenses of Tor. The puzzle in Filecoin [1] consists of a proof-of-work similar to that of Bitcoin and also a proof that the miner possesses all the challenged pieces of a file at the time of mining. Permacoin [23] replaces the proof-of-work by proofs of retrievability.

Proofs of Retrievability.

With the advent of cloud computing, clients having large data may want to outsource the storage to the cloud server. As the cloud storage service provider (SSP) might discard old data to save some space, the clients need a guarantee that the outsourced data are not deleted or modified by the cloud server. One naive approach to ensure data integrity is that a client downloads the whole data from the server and verifies them individually segment by segment, but this process incurs a high communication bandwidth.

To resolve the issue mentioned above, researchers have come up with *proofs of storage*. Ateniese et al. [3] introduce the concept of provable data possession (PDP) where the client computes an authenticator (for example, message authentication codes) for each segment of her data (or file), and uploads the file along with the authenticators. During an audit protocol, the client samples a predefined number of random segment-indices (challenge) and sends them to the server. The server does some computations (depending upon the challenge) over the stored data, and sends a proof (response) to the client who verifies the integrity of her data based on this proof. This scheme also introduces the notion of public verifiability where anyone can perform an audit.

The first paper introducing proofs of retrievability (POR) for static data is by Juels and Kaliski [17] (a similar idea is given for sublinear authenticators by Naor and Rothblum [25]). The underlying idea is to encode the original file with an erasure code (discussed in Section 3.3), authenticate the segments of the encoded file, and then upload them on the data server. With this technique, the server has to

Parameters	Permacoin	Retricoin
Storage overhead per segment (in bits)	$256 \cdot \lceil \lg n \rceil$	256
Network bandwidth for transmitting a ticket (in bits)	$256 + \lambda + k \cdot (\zeta + \gamma + 256 \cdot \lceil \lg n \rceil)$	$256 + 4\lambda + \zeta + \lceil \lg l \rceil + k\gamma$

Figure 1: Comparison between Permacoin and Retricoin. The number of segments in the file F is denoted as n. We take ζ to be the size (in bits) of a segment, γ to be the size (in bits) of a signature in the floating preimage signature scheme (see Section 4.3), λ to be the security parameter, k to be the cardinality of a challenge set Q (see Section 4.2) and l to be the number of segments a miner is storing. We denote the logarithm of a positive real number to the base 2 by lg. We take $k = \lambda = 128$ and SHA-256 to be the hash function used. As a concrete example, for a file F of size 1 terabyte, segments of size 2016 bytes and a miner storing 4 GB data: the storage overhead per segment is 960 bytes in Permacoin and 32 bytes in Retricoin; and the size of a ticket is 411696 bytes in Permacoin and 32835 bytes in Retricoin.

delete or modify a considerable number of segments to actually delete or modify a data segment. This ensures that all segments of the file are retrievable from the responses of the server which passes an audit with some probability non-negligible in the security parameter λ. Following the work by Juels and Kaliski, several POR schemes have been proposed [28, 13, 10, 29]. Some of these schemes are designed for static data, and the rest allow the client to modify data after the initial outsourcing.

3. PRELIMINARIES

3.1 Notation

We take λ to be the security parameter. An algorithm $\mathcal{A}(1^\lambda)$ is a probabilistic polynomial-time algorithm when its running time is polynomial in λ and its output y is a random variable which depends on the internal coin tosses of $\mathcal{A}$. An element a chosen uniformly at random from a set S is denoted as $a \xleftarrow{R} S$. A function $f : \mathbb{N} \to \mathbb{R}$ is called negligible in λ if for all positive integers c and for all sufficiently large λ, we have $f(\lambda) < \frac{1}{\lambda^c}$. We denote the logarithm of a positive real number to the base 2 by lg.

3.2 Bilinear Map

Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p. Let g_1 and g_2 be generators of the groups G_1 and G_2, respectively. A bilinear map is a function $e : G_1 \times G_2 \to G_T$ such that: (1) for all $u \in G_1, v \in G_2, a \in \mathbb{Z}_p, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$ (bilinear property), and (2) e is non-degenerate, that is, $e(g_1, g_2) \neq 1$. Furthermore, properties (1) and (2) imply that $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$ for all $u_1, u_2 \in G_1, v \in G_2$.

If $G_1 = G_2 = G$, the bilinear map is symmetric; otherwise, asymmetric. Unless otherwise mentioned, we consider only the bilinear maps which are efficiently computable and symmetric. Let BLSetup(1^λ) be an algorithm which outputs (p, g, G, G_T, e) as the parameters of a bilinear map, where g is a generator of G.

3.3 Erasure Code

An $(n, f, d')_\Sigma$ erasure code is an error-correcting code [21] that consists of an encoding algorithm Enc: $\Sigma^f \to \Sigma^n$ (encodes a message consisting of f symbols into a longer codeword consisting of n symbols) and a decoding algorithm Dec: $\Sigma^n \to \Sigma^f$ (decodes a codeword to a message), where Σ is a finite alphabet and d' is the minimum distance (Hamming distance between any two codewords is at least d') of the code. The quantity $\frac{f}{n}$ is called the rate of the code. An $(n, f, d')_\Sigma$ erasure code can tolerate up to $d' - 1$ erasures. If $d' = n - f + 1$, we call the code a maximum distance separable (MDS) code. For an MDS code, the original message can be reconstructed from any f out of n symbols of the codeword. Reed-Solomon codes [26] and their extensions are examples of non-trivial MDS codes.

4. RETRICOIN: OUR SCHEME

The proof-of-work in Bitcoin is expensive and wasteful. The proof-of-storage is a useful alternative that ensures data-integrity of a large file. In this section, we propose a scheme, which we call Retricoin, for cryptocurrency that utilizes the compact proofs of retrievability [28] as the underlying mining puzzle where a miner proves that she has enough storage in order to mine blocks. A miner having more storage will benefit in our scheme. We improve our scheme to incorporate the notion of non-outsourceability by penalizing miners outsourcing their storage to some data-servers; otherwise if outsourcing of data is possible, it might happen that a few servers store the whole file, and failure of any of these servers might make the file irretrievable. Retricoin involves less storage overhead and communication bandwidth than the existing Permacoin scheme [23]. We provide a detailed comparison between Retricoin and Permacoin in Section 5.

4.1 Assumptions

Following the assumption of Permacoin [23], we assume that there is a trusted *dealer* in the Bitcoin network who always remains online. Suppose, a user owns an important but large file (may be of the order of petabytes) which she wants to be distributed across the Bitcoin network. The goal is to achieve fault-tolerance and to reduce the storage overhead at the owner's end. The Bitcoin network can store this large file in lieu of monetary benefits. We assume that the dealer has a copy of this large file and distributes the segments of this file among the Bitcoin users who request for the same. The users who want to mine Bitcoins must prove their storage to other peers in the network.

Let F_0 be the original file to be distributed in the Bitcoin network. The dealer encodes the original file F_0 with a maximum distance separable (MDS) erasure code to form another file F and distributes its segments among the Bitcoin users in the network. Now, F_0 can be retrieved from a constant fraction (depends on the rate of the error-correcting code used) of segments of F in the presence of erasures. We mention that both Retricoin and Permacoin offer a guarantee of provable data possession (PDP). However, the size of a proof in Retricoin is much smaller than that in Permacoin or other PDP schemes (where the number of audits can be unbounded, for example, [3, 2]).

In each epoch the miners have to solve a mining puzzle which is identified by a unique (corresponding to an epoch) puzzle-identifier puz. This puzzle-identifier is non-precomputable in the sense that it cannot be predicted beforehand. It is publicly known once the particular epoch starts.

4.2 Basic Scheme

In the basic scheme, a user in the Bitcoin network downloads and stores data segments of a large file F owned by the dealer. The dealer attaches with every segment a tag generated using her secret key. A user stores the segments along with the tags. However, there is no upper bound on the number of segments a particular user can request for. We incentivize miners who have more storage by letting them store as much data as they can. The basic scheme using POR consists of three phases as follows.

- **Setup** Let $\mathrm{BLSetup}(1^\lambda)$ be an algorithm that outputs (p, g, G, G_T, e) as the parameters of a bilinear map, where $e : G \times G \to G_T$ and $\mathbb{Z}_p$ is the support of the group G. The dealer chooses $x \overset{R}{\longleftarrow} \mathbb{Z}_p$ as her secret key. The public key of the dealer is $v = g^x$. Let $\alpha \overset{R}{\longleftarrow} G$ be another generator of G. Let there be f segments in the original file F_0. The dealer encodes F_0 with an $(n, f, n-f+1)$-MDS erasure code to form a file F with n segments. Therefore, F_0 can be retrieved from any f out of n segments, where $r = \frac{f}{n}$ is the rate of the erasure code. Let each segment of the file F be an element of $\mathbb{Z}_p$, that is, $F[i] \in \mathbb{Z}_p$ for all $i \in \mathbb{Z}_n$. Let $H_0 : \{0,1\}^* \to \{0,1\}^{\lceil \lg n \rceil}, H_1 : \{0,1\}^* \to \{0,1\}^{\lceil \lg p \rceil}$ be two hash functions, and $H : \{0,1\}^* \to G$ be the BLS hash [9]. These hash functions are publicly known. The dealer computes $\sigma_i = (H(i) \cdot \alpha^{F[i]})^x$ for all $i \in \mathbb{Z}_n$.
  A Bitcoin user with a public key-secret key pair (pk, sk) and storage capacity of l data segments selects a set $I \subseteq \mathbb{Z}_n$ of indices as given below.
  $$\forall i \in \mathbb{Z}_l : \qquad u[i] = H_0(pk||i) \pmod n, \qquad (1)$$
  and $I = \{u[i]\}_{i \in \mathbb{Z}_l}$. The user then downloads and stores $\{(F[j], \sigma_j)\}$ for all $j \in I$.
- **Proof generation** To generate the proof, the miner with public key pk chooses two random strings s_0 and s_1 each of size λ bits. Let puz be the publicly known and non-precomputable puzzle-identifier for the current epoch. The miner generates a random challenge set $Q = \{(r_i, \nu_i)\}$ of cardinality k as follows.
  $$\begin{aligned} \forall i \in \mathbb{Z}_k : \quad r_i &= u[H_1(puz||pk||i||s_0) \pmod l], \\ \nu_i &= H_1(puz||pk||i||s_1) \pmod p. \end{aligned} \qquad (2)$$
  The miner calculates $\sigma = \prod_{(r_i,\nu_i)\in Q} {\sigma_{r_i}}^{\nu_i}$ and $\mu = \sum_{(r_i,\nu_i)\in Q} \nu_i F[r_i] \pmod p$. Finally, she broadcasts the ticket $T = (pk, s_0, s_1, l, \sigma, \mu)$.
- **Verification** The verifier knows the public key v of the dealer and the puzzle-identifier puz for the current epoch. The verifier, upon receiving a ticket, parses it as $T = (pk, s_0, s_1, l, \sigma, \mu)$. Then, she reconstructs the set Q using Eqn. 2 and checks if
  $$e(\sigma, g) \overset{?}{=} e\left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}, v \right). \qquad (3)$$
  T is a winning ticket if it passes the verification.

In our scheme described above, the dealer produces the tags based on her secret key and any user in the network can verify a proof based on the public key of the dealer. The original file F_0 can be retrieved from any f segments of F stored untampered by the honest miners.

We assume that a miner stores l segments ($l \gg \lambda$) where l varies depending upon the storage capacity of the miner. The idea is to check whether the miner is storing data accurately at some random k locations, where k is a public parameter known to every user in the network. Corresponding to each epoch, there is a non-precomputable system-generated puzzle-identifier puz which is known to all users in the Bitcoin network. The random challenge set Q is generated non-interactively depending on the puzzle-identifier puz. Since puz is non-precomputable, so is the challenge set Q. Hence, the miner does not have any prior knowledge of the segments that are needed to compute a proof, making it impossible for her to selectively store some segments only while deleting the others. The value of k depends on the value of λ. Typically, k is taken to be λ to ensure that the miner is storing her data intact except with some probability negligible in λ [28]. The miner includes l, along with the proof, in her ticket T.

After receiving a proof from a miner, the verifier extracts, from the ticket, the relevant data for computing Q and constructs the challenge set Q non-interactively in the same way as done by the miner (prover). Finally, the verifier checks the validity of Eqn. 3 and declares the ticket as the winning ticket for the current epoch if the equality holds.

It is obvious that there would be multiple winners. The Bitcoin network will accept only the winning ticket with the highest value of l. Thus, we try to incentivize a miner who has more storage than others. Therefore, Retricoin assumes implicitly, like the classical POW scheme, that the honest miners have more resources than the dishonest miners with the exception of computational power being replaced by storage capacity.

4.3 Improved Scheme to Prevent Storage Outsourcing

A miner in the Bitcoin network mints Bitcoins by solving a mining puzzle. In our scheme, the miner stores a large number of segments of a giant file F locally, and generates a proof that she is storing the file indeed. Then she broadcasts this proof among all the users in the network who verify the proof. We incentivize the miner if the proof passes the verification. However, a malicious miner can outsource her data to a server who stores her data on her behalf. To make it worse, a particular data server can store the data of many such users in the network. In that case, the purpose of storing data in a distributed manner is defeated. If the data server fails, reconstruction of the original file F may not be possible at all. Therefore, our scheme should penalize the miners outsourcing their data to some data service providers.

Our basic scheme described in Section 4.2 does not fulfill this requirement. To prevent the outsourcing of one's data, we propose an improved scheme here. The phases of the improved scheme are described below.

- **Setup** The setup phase is the same as that of our basic scheme described in Section 4.2.

- **Proof generation** To generate the proof, the miner with a public key-secret key pair (pk, sk) chooses two random strings s_0 and s_1 each of size λ bits. Let puz be the publicly known, non-precomputable puzzle-identifier for the current epoch. The miner generates a random challenge set $Q = \{(r_i, \nu_i)\}$ of cardinality k using the pseudocode stated below.

```
$r_0 = u[H_1(puz||pk||s_0) \pmod l]$;
$\nu_0 = H_1(puz||pk||s_1||r_0) \pmod p$;
$h_0 = \mathrm{sign}_{sk}(\nu_0)$;                    (4)
$r_1 = u[H_1(puz||pk||h_0) \pmod l]$;
For $i = 1, 2, \ldots, k-2$ do
    $\nu_i = H_1(puz||pk||h_{i-1}||r_i) \pmod p$;
    $h_i = \mathrm{sign}_{sk}(\nu_i)$;                (5)
    $r_{i+1} = u[H_1(puz||pk||h_i) \pmod l]$;
$\nu_{k-1} = H_1(puz||pk||h_{k-2}||r_{k-1}) \pmod p$;
$h_{k-1} = \mathrm{sign}_{sk}(\nu_{k-1})$;            (6)
```

  The miner calculates $\sigma = \prod_{(r_i,\nu_i)\in Q} {\sigma_{r_i}}^{\nu_i}$ and $\mu = \sum_{(r_i,\nu_i)\in Q} \nu_i F[r_i] \pmod p$. Finally, she broadcasts the ticket $T = (pk, s_0, s_1, l, \sigma, \mu, \{h_i\}_{0\leqslant i<k})$.

- **Verification** The verifier knows the public key v of the dealer and the puzzle-identifier puz for the current epoch. The verifier, upon receiving a ticket, parses it as $T = (pk, s_0, s_1, l, \sigma, \mu, \{h_i\}_{0\leqslant i<k})$. She reconstructs the set Q as above, except that the signing operations in Eqn. 4, 5 and 6 are replaced by the corresponding verification operations using the public key pk. Finally, if all of the k signatures are valid, the verifier checks if

$$e(\sigma, g) \overset{?}{=} e\left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}, v\right). \qquad (7)$$

  T is a winning ticket if it passes this verification.

The verifier in Retricoin verifies if the tuples (r_i, ν_i, h_i) are generated correctly in each iteration $0 \leqslant i < k$. If they are consistent for all the iterations, the verifier checks the validity of Eqn. 7 and declares the ticket as the winning one if the equality holds.

In our basic scheme, the (r_i, ν_i) pairs in the random challenge set Q are generated independently. In the improved scheme mentioned above, we introduce some dependence between the generation of these pairs. Each ν_i depends upon r_i and the previous ν_{i-1} for $1 \leqslant i < k$, and ν_0 depends on r_0. On the other hand, each r_i depends on the previous ν_{i-1} for $1 \leqslant i < k$. To produce h_i, the miner has to sign the value ν_i for each $0 \leqslant i < k$. If the miner outsources her segments to a server, she is penalized in either of the following ways.

Firstly, we consider the case where a server stores a miner's data on her behalf without any knowledge about the the miner's signing key sk. Then, for each iteration $0 \leqslant i < k$, the miner has to compute r_i and ν_i, and then she has to send the value $h_i = \mathrm{sign}_{sk}(\nu_i)$ to the server. This communication is necessary for each of the k iterations which results in a large network bandwidth and a higher latency in terms of communication-time that reduce the miner's chance to find a winning ticket significantly. Secondly, we tie the reward for solving the puzzle to the secret signing key sk of the miner. If the miner shares her secret key sk with the data server, the server itself can claim the reward. To summarize, rational miners store their data locally instead of outsourcing them.

The value h_i is obtained by signing ν_i for each $0 \leqslant i < k$. To sign $\{\nu_i\}_{0\leqslant i<k}$ values, we require an efficient signature scheme. We use the multi-hash-based signature scheme described in the work of Miller et al. [23] which they call a floating preimage signature (FPS) scheme. In this signature scheme, each signer maintains a Merkle hash tree [22] whose leaves are random strings, and these leaves constitute the secret key of the signer. The public key is the root-digest of the Merkle hash tree. Initially, all the leaves are unrevealed. A hash function modeled as a random oracle [6] is used to generate a subset (of a predefined size) of the unrevealed leaves. The leaves in this subset along with their Merkle proofs comprise the signature. This scheme ensures non-outsourceability in the sense that if the server can sign k messages in the proof-generation phase, then, with high probability, it can produce a valid $(k+1)$-th signature which is tied to the claim of the reward.

4.4 Correctness and Security

The proofs for correctness and security of our scheme described in Section 4.3 are derived from the proofs provided by Shacham and Waters [28]. Here, we discuss them briefly. *Correctness*: For the honest miners storing their respective data segments correctly, we have

$$\begin{aligned}
\sigma &= \prod_{(r_i,\nu_i)\in Q} \sigma_{r_i}^{\nu_i} \\
&= \prod_{(r_i,\nu_i)\in Q} \left(H(r_i) \cdot \alpha^{F[r_i]}\right)^{\nu_i x} \\
&= \left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \prod_{(r_i,\nu_i)\in Q} \alpha^{\nu_i F[r_i]}\right)^x \\
&= \left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\sum_{(r_i,\nu_i)\in Q} \nu_i F[r_i]}\right)^x \\
&= \left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}\right)^x .
\end{aligned}$$

Substituting the value of σ in $e(\sigma, g)$, we get

$$\begin{aligned}
e(\sigma, g) &= e\left(\left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}\right)^x, g\right) \\
&= e\left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}, g^x\right) \\
&= e\left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}, v\right).
\end{aligned}$$

Therefore, the proof provided by an honest miner always passes the verification given by Eqn. 7.

Security: We take the security parameter λ to be 128. We consider the hash function H as a random oracle. The following theorem shows that our scheme is secure in the random oracle model. The proof of the theorem directly follows from the security proof of the publicly verifiable scheme proposed in [28] which we omit here due to space-constraint.

THEOREM 4.1. *An adversary passing the verification as given by Eqn. 7 is indeed storing all the challenged segments, except with some probability negligible in λ.*

5. COMPARISON WITH PERMACOIN

In this section, we provide a comparison between the Retricoin scheme described in Section 4.3 and the Permacoin scheme proposed by Miller et al. [23]. We note that Filecoin [1] also uses proofs of retrievability along with the proof-of-work (similar to Bitcoin) to mine a Bitcoin block. However, Filecoin uses the existing Bitcoin network and adds another functionality: *distributing the data pieces of a file over the network (via 'Put' and 'Get' transactions) and proving storage with the help of POR.* Therefore, this scheme does *not* eliminate the computationally heavy proof-of-work. On the contrary, our scheme Retricoin replaces the proof-of-work completely by proofs of retrievability so that the mining work can be utilized for some useful purpose. Thus, the schemes Retricoin and Filecoin do not fall under the same scope. Moreover, Filecoin does not use an error-correcting code to encode the pieces. Therefore, it does not provide any guarantee of retrieving some pieces when some of the Bitcoin miners go offline. On the other hand, Filecoin does not offer any mechanism to prevent the outsourceability of data to a storage server. To the best of our knowledge, Permacoin is the only e-cash scheme found in the literature that is based solely on the concept of proofs of retrievability, where the dealer constructs a Merkle hash tree [22] (for authenticated storage) on the segments of a file and distributes these segments to guarantee that it is retrievable at any point of time. So, we limit the comparison between Retricoin and Permacoin only.

5.1 Bilinear Setting and Size of Parameters in Our Scheme

We consider the construction of the pairing function e and the structure of the groups involved in it. In general setting, we take the pairing function $e : G_1 \times G_2 \rightarrow G_T$ with parameters $(p, g_1, g_2, G_1, G_2, G_T)$, where $|G_1| = |G_2| = |G_T| = p = \Theta(2^{2\lambda})$ and g_1, g_2 are generators of the groups G_1 and G_2, respectively. We take $\lambda = 128$. Practical constructions of pairings are done on elliptic (or hyperelliptic) curves defined over a finite field. We write $E(\mathbb{F}_q)$ to denote the set of points on an elliptic curve E defined over the finite field $\mathbb{F}_q$. Then G_1 is taken as a subgroup of $E(\mathbb{F}_q)$, G_2 is taken as a subgroup of $E(\mathbb{F}_{q^{k'}})$ and G_T is taken as a subgroup of $\mathbb{F}^*_{q^{k'}}$, where k' is the embedding degree [19, 16, 30].

Now, let us refer to the Retricoin scheme described in Section 4.3. We assume that $F[i] \in \mathbb{Z}_p$ for all $i \in \mathbb{Z}_n$, and the random strings s_0, s_1 are of size λ bits. The ν_i values in a random challenge set Q are taken to be elements of $\mathbb{Z}_p$, and $|Q| = k = \lambda$. The BLS hash function H hashes a binary string of arbitrary length into G_1, and α is a random element of G_1. Moreover, $\sigma = \prod_{t\in\mathbb{Z}_m} \sigma_t$ also is an element of G_1, and $\mu = \sum_{(r_i,\nu_i)\in Q} \nu_i F[r_i] \pmod p$ belongs to $\mathbb{Z}_p$. For the value of the security parameter λ up to 128, Barreto-Naehrig (BN) curves [5, 15] are suitable for our scheme [28]. In this setting, the segments, tags, q, σ and μ are of size 256 bits each.

5.2 Efficient Storage

Let us consider a user in the network storing l segments. In the Permacoin scheme, the user stores a set $\{(F[j], \pi_j) : j \in I \text{ and } |I| = l\}$, where I is the set of indices of F corresponding to these l segments and π_j is the Merkle proof of the segment $F[j]$. For each $j \in I$, the Merkle proof π_j for $F[j]$ consists of total $\lceil \lg n \rceil$ hash values corresponding to the associated path of length $\lceil \lg n \rceil$. If SHA-256 is used to compute these hash values, then the size of each π_j becomes $256 \cdot \lceil \lg n \rceil$ bits. This incurs a large storage overhead of $256 \cdot \lceil \lg n \rceil$ bits for each of the segments, since n, the number of segments in the file F, is huge in general.

In Retricoin, we make this storage overhead constant and independent of n. Instead of the Merkle proof, the user stores a small tag or authenticator of size 256 bits along with each segment $F[j]$. Therefore, a user in our scheme enjoys around $256 \cdot (\lceil \lg n \rceil - 1)$ bits *less* storage overhead per segment compared to Permacoin.

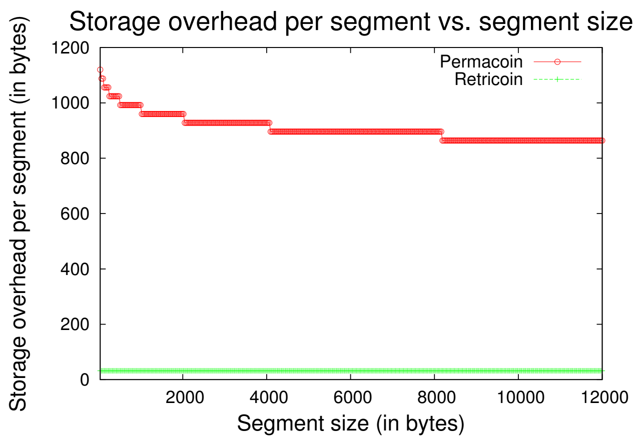
We note that our scheme as well as the Permacoin scheme can be modified such that each segment is a collection of d sectors, each sector being an element of $\mathbb{Z}_p$. Now, we can assign a tag (or a Merkle proof) with each segment in Retricoin (or Permacoin), thus making both schemes more efficient with respect to storage overhead per segment. In this setting also, the storage overhead per segment for a user in our scheme is around $256 \cdot (\lceil \lg n' \rceil - 1)$ bits *less* than that in the Permacoin scheme, where $n' = \lceil \frac{n}{d} \rceil$ is the number of segments in F now. In Fig. 2(a), we compare Retricoin with Permacoin in terms of storage overhead per segment when the size of a segment varies.

5.3 Efficient Bandwidth

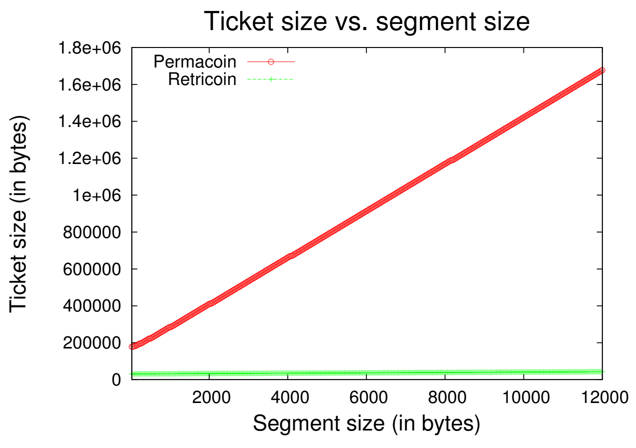
A miner attaches a ticket, as a proof of her storage, with a Bitcoin block. In the Permacoin scheme, a ticket is of the form $(pk, s, \{F[r_i], \beta_i, \pi_{r_i}\}_{0\leqslant i<k})$, where r_i is the first element of the i-th pair in the random challenge set Q, β_i is the i-th signature in the proof-generation phase and π_{r_i} is the Merkle proof of the segment $F[r_i]$. Let us assume that s is a random string of length λ bits. If SHA-256 is used as the hash function for the Merkle tree in the floating preimage signature (FPS) scheme [23], then the size of a signature is around $\gamma = \lambda + 256 \cdot \lceil \lg \lambda \rceil$ bits. As we mention in Section 5.2, the size of each π_{r_i} is $256 \cdot \lceil \lg n \rceil$ bits. If each $F[r_i]$ is an element of $\mathbb{Z}_p$, it is represented in $\lceil \lg p \rceil \approx 2\lambda$ bits. The public key pk, the root-digest of the Merkle tree used in FPS scheme, is of size 256 bits. Then the communication of a ticket requires $256 + \lambda + k \cdot (2\lambda + \gamma + 256 \cdot \lceil \lg n \rceil)$ bits of bandwidth.

On the other hand, a ticket T in Retricoin is of the form $(pk, s_0, s_1, l, \sigma, \mu, \{h_i\}_{0\leqslant i<k})$, where h_i is the i-th signature in the proof-generation phase. Assuming the parameters to be the same as discussed above, $256 + 6\lambda + \lceil \lg l \rceil + k\gamma$ bits of bandwidth are required to communicate a ticket which is much less than that in Permacoin.

Again, each segment can be a collection of d sectors, each sector being an element of $\mathbb{Z}_p$. Therefore, the size of a segment is $\zeta = 2d\lambda$ bits, $n' = \lceil \frac{n}{d} \rceil$ is the number of segments in



(a)



(b)

Figure 2: Comparison between Permacoin and Retricoin. (a) Estimated storage overhead per segment vs. segment size and (b) estimated ticket size vs. segment size, for a file F of size 1 terabyte and a user storing 4 GB data. We take $k = \lambda = 128$ and SHA-256 to be the hash function used. As an example, for segments of size 2016 bytes, the storage overhead per segment is 960 bytes in Permacoin and 32 bytes in Retricoin; and the size of a ticket is 411696 bytes in Permacoin and 32835 bytes in Retricoin.

F, and $l' = \lceil \frac{l}{d} \rceil$ is the number of segments a user is storing now. Consequently, the size of the aggregated segment μ is $2d\lambda$ bits in Retricoin and l' (of size $\lceil \lg l' \rceil$ bits) is included in the ticket instead of l. Since each of these two elements appears only once in a ticket and the other parameters in the ticket remain unchanged, so the relative change in the ticket size is very small. Therefore, the size of a ticket (in bits) is $256+4\lambda+\zeta+\lceil \lg l' \rceil+k\gamma$ in Retricoin. On the other hand, in Permacoin, k segments are included in a ticket. Thus, larger the size of a segment, larger is the size of a ticket. The size of a ticket (in bits) is $256 + \lambda + k \cdot (\zeta + \gamma + 256 \cdot \lceil \lg n' \rceil)$ in Permacoin. Though the value of n reduces to n', this affects the size of the ticket to a smaller extent due to the logarithmic factor. In Fig. 2(b), we compare Retricoin with Permacoin in terms of the size of a ticket when the size of a segment varies.

5.4 Cost of Verification

Retricoin enjoys much more efficiency than Permacoin [23] in terms of storage overhead and communication bandwidth required. However, this efficiency comes at a cost of the time required to verify the proof sent by the miner. The verification phase of Retricoin takes more time than Permacoin due to the use of expensive bilinear pairings. In this section, we precisely calculate the cost of the verification phase of Retricoin, and we show that this timing is comparable to that of Permacoin and reasonable for all practical purposes.

During the reconstruction of the random challenge set $Q = \{(r_i, \nu_i)\}_{0 \leqslant i < k}$ in the verification phase of Retricoin, the i-th iteration requires:

1. one invocation of H_1 for generating r_i,
2. one invocation of H_1 for generating ν_i and
3. $\lceil \lg \lambda \rceil$ invocations of the hash function to verify the Merkle proof in FPS scheme.

If all of these hash functions are realized as SHA-256, then there are $k \cdot (2 + \lceil \lg \lambda \rceil)$ invocations of SHA-256 for all k iterations. We estimate the time taken, on a 2.5 GHz Intel Core i5-3210M processor, to compute SHA-256 from *eBASH*, a benchmarking project for hash functions [7]. Here, we mention that the computation time of SHA-256 is more when the length of the input message is larger.

After Q is reconstructed consistently, we verify Eqn. 7 (see Section 4.3) that involves $|Q| = k$ hashes to G_1, one multi-scalar multiplication in G_1, one scalar multiplication in G_1, one addition in G_1, two pairings and one comparison in G_T. A multi-scalar multiplication computes k scalar multiplications followed by $k - 1$ additions in G_1 in an efficient manner. For the timing analysis of these operations, we use PandA, a recent software framework for *Pairings and Arithmetic* developed by Chuengsatiansup et al. [11]. In Table 1, we mention the cycle-counts of these operations for a 128-bit secure, Type-3 pairing framework involving a pairing-friendly BN curve. We take $k = \lambda = 128$. These counts are taken as the median of 10000 measurements on a 2.5 GHz Intel Core i5-3210M processor [11]. Summing up

API function	Cycle-counts
Hash 59-bytes message into G_1	83168
Multi-scalar multiplication in G_1 for $k = 128$	14605364
Scalar multiplication in G_1	288240
Addition in G_1	2468
Ate pairing	3832644
Comparison in G_T	8320

Table 1: Cycle-counts on a 2.5 GHz Intel Core i5-3210M processor.

over required number of these operations, we get the total cycle-counts for Eqn. 7 as around $(128 * 83168 + 14605364 + 288240 + 2468 + 2 * 3832644 + 8320) = 33215184$. These many cycles take approximately 13.29 milliseconds on this processor. On the other hand, the total time for hashing using SHA-256 is around 1.12 milliseconds in Retricoin.

Therefore, the total verification time in our scheme is around $13.29 + 1.12 = 14.41$ milliseconds. Assuming a Bitcoin block is generated in the network every 10 minutes on an average, it takes at most 15 seconds for a user to validate Bitcoin blocks of one week.

In Permacoin, the verification time is reported as around 6 milliseconds for $k = 20$ iterations using SHA-1 as the hash function on an "ordinary" computer. In Retricoin, the verification time is approximately 14.41 milliseconds for $k = 128$ iterations when measured using SHA-256 as the hash function on a 2.5 GHz Intel Core i5-3210M processor. We note that the verification time depends upon the value of k and the hash function we use. Larger the value of k, larger is the verification time. The execution time of SHA-256 is more compared to that of SHA-1 when both of them implemented on the same platform.

6. SCHEME FOR COLLECTIVE MINING

In the classical mining scheme of Bitcoin, miners create pools [27] to unite their computational resources to have a large computing powerhouse in order to do hash computation of the order of 2^{55}. In our scheme as well, some miners may want to combine their storage to have a big storage at their disposal. This concern is not addressed in [23].

We introduce an algorithm that allows multiple miners belonging to the same pool to mine collectively. For simplicity, we present a modified version of the basic scheme described in Section 4.2, although one can verify easily that this modification can be ported readily to the improved scheme described in Section 4.3. Let us assume that each miner in a pool stores the same number (say, l) of segments. However, if they store different numbers of segments, small changes are required which are discussed at the end of this section.

We consider a pool M containing a set of miners $\Gamma \subset \mathcal{P}$ of size m, where $\mathcal{P}$ is the set of all miners in the Bitcoin network. Let $P_0, P_1, \ldots, P_{m-1}$ be the miners in the pool M. So, the pool collectively stores $L = ml$ data segments where each miner stores l segments. We assume that there is a pool operator or administrator who manages the pool in addition to mining Bitcoins. Without loss of generality, let P_0 be the administrator of the pool M. To maintain fairness, the miners in a pool can choose their administrator in a round-robin fashion for different epochs. A trivial solution for collective mining is that each miner stores her data, generates a proof and sends her proof to P_0 independently. After receiving all the proofs, P_0 broadcasts m proofs to the Bitcoin network. The verifier checks each proof separately and accepts if each of the m verifications passes. However, this requires $2m$ expensive pairing operations for verification. Our scheme involves only two pairing operations to verify the proofs from the group of miners Γ. We describe the scheme as follows.

- **Setup** Let BLSetup(1^λ) be an algorithm that outputs (p, g, G, G_T, e) as the parameters of a bilinear map, where $e : G \times G \to G_T$ and $\mathbb{Z}_p$ is the support of the group G. The dealer chooses $x \overset{R}{\longleftarrow} \mathbb{Z}_p$ as her secret key. The public key of the dealer is $v = g^x$. Let $\alpha \overset{R}{\longleftarrow} G$ be another generator of G. Let there be f segments in the original file F_0. The dealer encodes F_0 with an $(n, f, n - f + 1)$-MDS erasure code to form a file F with n segments. Therefore, F_0 can be retrieved from any f out of n segments, where $r = \frac{f}{n}$ is the rate of the erasure code. Let each segment of the file F be an element of $\mathbb{Z}_p$, that is, $F[i] \in \mathbb{Z}_p$ for all $i \in \mathbb{Z}_n$. Let $H_0 : \{0,1\}^* \to \{0,1\}^{\lceil \lg n \rceil}$, $H_1 : \{0,1\}^* \to \{0,1\}^{\lceil \lg p \rceil}$ be two hash functions, and $H : \{0,1\}^* \to G$ be the BLS hash. These hash functions are publicly known. The dealer computes $\sigma_i = (H(i) \cdot \alpha^{F[i]})^x$ for all $i \in \mathbb{Z}_n$.

  A group of miners Γ with a public key-secret key pair (pk, sk) for the group and storage capacity of L data segments forms a pool M and selects a set $I \subseteq \mathbb{Z}_n$ of indices as follows. Each miner P_t, where $t \in \mathbb{Z}_m$, computes
  $$u[i] = H_0(pk||i) \pmod n \quad \text{for } lt \leqslant i < l(t+1), \qquad (8)$$
  and the set $I_t = \{u[i]\}_{lt \leqslant i < l(t+1)}$. Let $I = \cup_{t \in \mathbb{Z}_m} I_t = \{u[i]\}_{i \in \mathbb{Z}_L}$. Each miner P_t then downloads and stores $\{(F[j], \sigma_j)\}$ for all $j \in I_t$.

- **Proof generation** To generate the proof, the group of miners Γ with public key pk chooses two random strings s_0 and s_1 each of size λ bits. Let puz be the publicly known, non-precomputable puzzle-identifier for the current epoch. Without loss of generality, we assume that P_0 is the administrator of the pool M. The administrator P_0 generates a random challenge set $Q = \{(r_i, \nu_i)\}$ of cardinality k as given below.
  $$\begin{aligned} \forall i \in \mathbb{Z}_k : \quad & d_i = H_1(puz||pk||i||s_0) \pmod L, \\ & r_i = u[d_i], \\ & \nu_i = H_1(puz||pk||i||s_1) \pmod p. \end{aligned} \qquad (9)$$
  Then P_0 distributes the set Q among the members of Γ in such a way that a miner P_t gets a set Q_t of these $\{(r_i, \nu_i)\}$ pairs if the corresponding d_i values satisfy $lt \leqslant d_i < l(t+1)$. Let $|Q_t| = k_t$ for each $t \in \mathbb{Z}_m$. Clearly, $Q = \cup_{t \in \mathbb{Z}_m} Q_t$ and $k = \sum_{t \in \mathbb{Z}_m} k_t$. Each miner P_t calculates $\sigma_t = \prod_{(r_i, \nu_i) \in Q_t} {\sigma_{r_i}}^{\nu_i}$ and $\mu_t = \sum_{(r_i, \nu_i) \in Q_t} \nu_i F[r_i] \pmod p$, and sends (σ_t, μ_t) to P_0. Upon receiving $\{(\sigma_t, \mu_t)\}_{1 \leqslant t \leqslant m-1}$, P_0 calculates $\sigma = \prod_{t \in \mathbb{Z}_m} \sigma_t$ and $\mu = \sum_{t \in \mathbb{Z}_m} \mu_t \pmod p$. Finally, P_0 broadcasts the ticket $T = (pk, s_0, s_1, L\sigma, \mu)$.

- **Verification** The verifier knows the public key v of the dealer and the puzzle-identifier puz for the current epoch. The verifier, upon receiving a ticket, parses it as $T = (pk, s_0, s_1, L, \sigma, \mu)$. Then, she reconstructs the set Q using Eqn. 9 and checks if
  $$e(\sigma, g) \stackrel{?}{=} e\left(\prod_{(r_i, \nu_i) \in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}, v\right). \qquad (10)$$
  T is a winning ticket if it passes the verification.

One drawback of our scheme described above is that the administrator P_0 has to transfer, for each $1 \leqslant t \leqslant m-1$, the set Q_t to the miner P_t which consumes large communication bandwidth. On the other hand, as the miners cannot proceed to generate their proofs until P_0 sends them their challenge sets, they have to wait. Let us consider the situation where each miner P_t, for each $0 \leqslant i < k$, computes d_i values (as given in Eqn. 9) by herself. Now depending upon whether $lt \leqslant d_i < l(t+1)$, P_t computes ν_i and includes (r_i, ν_i) in Q_t. Therefore, it is a rational alternative for the miners $P_1, P_2, \ldots, P_{m-1}$ to generate, instead of waiting for P_0, their own challenge sets. This technique eliminates the communication needed earlier to distribute the challenge sets among miners. However, the aggregation of the values $\{\sigma_t\}_{t\in\mathbb{Z}_m}$ and $\{\mu_t\}_{t\in\mathbb{Z}_m}$ should be done centrally, and the communication of these values is necessary to generate the aggregate proof (σ, μ). We show this improved scheme for collaborative mining below.

- **Setup** The setup phase is the same as that described in the previous scheme for collaborative mining.
- **Proof generation** To generate the proof, the group of miners Γ with public key pk chooses two random strings s_0 and s_1 each of size λ bits. Let puz be the publicly known, non-precomputable puzzle-identifier for the current epoch. Without loss of generality, we assume that P_0 is the administrator of the pool M. Each miner P_t computes d_i values (as given in Eqn. 9) for all $i \in \mathbb{Z}_k$. If $lt \leqslant d_i < l(t+1)$, P_t includes the pair $\{(r_i, \nu_i)\}$ in her random challenge set Q_t, where $r_i = u[d_i]$ and $\nu_i = H_1(puz||pk||i||s_1) \pmod p$. Let $|Q_t| = k_t$ for each $t \in \mathbb{Z}_m$. Let $Q = \{(r_i, \nu_i)\}$ were the random challenge set of cardinality k if the administrator P_0 would have generated it by following Eqn. 9. Clearly, $Q = \cup_{t\in\mathbb{Z}_m} Q_t$ and $k = \sum_{t\in\mathbb{Z}_m} k_t$. Each user P_t calculates $\sigma_t = \prod_{(r_i,\nu_i)\in Q_t} {\sigma_{r_i}}^{\nu_i}$ and $\mu_t = \sum_{(r_i,\nu_i)\in Q_t} \nu_i F[r_i] \pmod p$, and sends (σ_t, μ_t) to P_0. Upon receiving $\{(\sigma_t, \mu_t)\}_{1\leqslant t\leqslant m-1}$, P_0 calculates $\sigma = \prod_{t\in\mathbb{Z}_m} \sigma_t$ and $\mu = \sum_{t\in\mathbb{Z}_m} \mu_t \pmod p$. Finally, P_0 broadcasts the ticket $T = (pk, s_0, s_1, L, \sigma, \mu)$.
- **Verification** The verifier knows the public key v of the dealer and the puzzle-identifier puz for the current epoch. The verifier, upon receiving a ticket, parses it as $T = (pk, s_0, s_1, L, \sigma, \mu)$. She reconstructs the set $Q = \{(r_i, \nu_i)\}$ of cardinality k as stated below.

$$\begin{aligned} \forall i \in \mathbb{Z}_k: \quad & d_i = H_1(puz||pk||i||s_0) \pmod L, \\ & r_i = u[d_i], \\ & \nu_i = H_1(puz||pk||i||s_1) \pmod p. \end{aligned} \tag{11}$$

Then, the verifier checks if

$$e(\sigma, g) \overset{?}{=} e\left(\prod_{(r_i,\nu_i)\in Q} H(r_i)^{\nu_i} \cdot \alpha^{\mu}, v\right). \tag{12}$$

T is a winning ticket if it passes the verification.

The verification phase remains unchanged, so the verifier would have no idea about whether the proof has been generated by a single miner or by a group of collaborating miners. The payment for a winning ticket is tied to the secret key sk shared among the miners of the pool. Finally, if the miners in a pool store different numbers of segments, then for each miner P_t, the range of i values to form I_t in the setup phase and the range of d_i values to form Q_t in the proof-generation phase are changed accordingly.

7. CONCLUSION

In this paper, we have proposed an alternative scheme, which we call Retricoin, of the existing Permacoin scheme. We replace the computationally heavy work required for Bitcoin-mining by proofs of retrievability that can be utilized to store an important file redundantly over the distributed network. From the comparison with Permacoin, we observe that Retricoin is more efficient than Permacoin in terms of the storage overhead and the bandwidth required to transmit a ticket over the network. On the other hand, the verification time in Retricoin is comparable to that of Permacoin. We have shown that one user can validate one week's Bitcoin blocks in around 15 seconds that is much less compared to the average time (10 minutes) for generation of a Bitcoin block. We have also designed an algorithm by which a group of miners can form a pool and mine Bitcoins collectively.

8. ACKNOWLEDGMENTS

The authors were partially supported by JSPS and DST under the Japan-India Science Cooperative Program of research project name: "Computational Aspects of Mathematical Design and Analysis of Secure Communication Systems Based on Cryptographic Primitives".

9. REFERENCES

[1] 1e96a1b27a6cb85df68d728cf3695b0c46dbd44d. Filecoin: A cryptocurrency operated file storage network, July 2014. http://filecoin.io/filecoin.pdf.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song. Remote data checking using provable data possession. *ACM Transactions on Information and System Security*, 14(1):12:1–12:34, June 2011.

[3] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*, pages 598–609, 2007.

[4] A. Back. Hashcash - a denial of service counter-measure, August 2002. http://www.hashcash.org/papers/hashcash.pdf.

[5] P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer Berlin Heidelberg, 2006.

[6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS 1993, pages 62–73, New York, NY, USA, 1993. ACM.

[7] D. J. Bernstein and T. Lange. eBASH: ECRYPT benchmarking of all submitted hashes. http://bench.cr.yp.to/ebash.html.

[8] A. Biryukov and I. Pustogarov. Proof-of-work as anonymous micropayment: Rewarding a Tor relay. Cryptology ePrint Archive, Report 2014/1011, 2014. http://eprint.iacr.org/.

[9] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[10] D. Cash, A. Küpçü, and D. Wichs. Dynamic proofs of retrievability via oblivious RAM. In T. Johansson and

P. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 279–295. Springer Berlin Heidelberg, 2013.

[11] C. Chuengsatiansup, M. Naehrig, P. Ribarski, and P. Schwabe. PandA: Pairings and arithmetic. In Z. Cao and F. Zhang, editors, *Pairing-Based Cryptography - Pairing 2013*, volume 8365 of *Lecture Notes in Computer Science*, pages 229–250. Springer International Publishing, 2014.

[12] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[13] Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009*, pages 109–127, 2009.

[14] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 436–454, 2014.

[15] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[16] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, Sept. 2008.

[17] 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*, CCS 2007, pages 584–597, New York, NY, USA, 2007. ACM.

[18] S. King. Primecoin: Cryptocurrency with prime number proof-of-work, July 2013. http://primecoin.io/bin/primecoin-paper.pdf.

[19] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In N. Smart, editor, *Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer Berlin Heidelberg, 2005.

[20] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis, 2015. http://www.cs.huji.ac.il/~yoadlew/bitcoin.pdf.

[21] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, 1977.

[22] R. C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO 1987*, pages 369–378, 1987.

[23] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing Bitcoin work for data preservation. In *IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 475–490, 2014.

[24] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. http://bitcoin.org/bitcoin.pdf.

[25] M. Naor and G. N. Rothblum. The complexity of online memory checking. *Journal of the ACM*, 56(1):2:1–2:46, February 2009.

[26] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[27] M. Rosenfeld. Analysis of Bitcoin pooled mining reward systems. *CoRR*, abs/1112.4980, 2011.

[28] H. Shacham and B. Waters. Compact proofs of retrievability. *Journal of Cryptology*, 26(3):442–483, 2013.

[29] E. Shi, E. Stefanov, and C. Papamanthou. Practical dynamic proofs of retrievability. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS 2013, pages 325–336, New York, NY, USA, 2013. ACM.

[30] N. P. Smart and F. Vercauteren. On computable isomorphisms in efficient asymmetric pairing-based systems. *Discrete Applied Mathematics*, 155(4):538–547, Feb. 2007.