KopperCoin – A Distributed File Storage with Financial Incentives

Henning Kopp(✉), Christoph Bösch, and Frank Kargl

Institute of Distributed Systems, Ulm University, Ulm, Germany
{henning.kopp,christoph.boesch,frank.kargl}@uni-ulm.de

Abstract. One of the current problems of peer-to-peer-based file storage systems like Freenet is missing participation, especially of storage providers. Users are expected to contribute storage resources but may have little incentive to do so. In this paper we propose KopperCoin, a token system inspired by Bitcoin's blockchain which can be integrated into a peer-to-peer file storage system. In contrast to Bitcoin, KopperCoin does not rely on a proof of work (PoW) but instead on a proof of retrievability (PoR). Thus it is not computationally expensive and instead requires participants to contribute file storage to maintain the network. Participants can earn digital tokens by providing storage to other users, and by allowing other participants in the network to download files. These tokens serve as a payment mechanism. Thus we provide direct reward to participants contributing storage resources.

Keywords: Blockchain · Cloud storage · Cryptocurrency · Peer-to-peer · Proof of retrievability

1 Introduction

In recent years, cryptocurrencies have rapidly gained adoption. One of the pioneers and most successful e-cash system is Bitcoin [17], in which clients, called *miners*, invest computational power to create units of a virtual currency. This process of generating Bitcoins is called *mining*. Bitcoin's mining process consists in finding a pre-image of a hash function such that the resulting hash is small. This is done via brute-forcing which may be seen as a waste of computing power and ultimately energy.

Various cryptocurrencies try to replace the *proof of work (PoW)* performed in the mining process with something more useful. Primecoin [10], for example, utilizes the PoW to find Cunningham and bi-twin chains, i.e., special sequences of prime numbers which are considered useful in cryptographic systems [24].

Recently, approaches to power storage systems as a by-product of maintaining a cryptocurrency emerged. Permacoin [16] replaces the PoW with a *proof of retrievability (PoR)*, i.e., a proof of possession of a file. Clients mine Permacoins by providing a PoR over parts of a global static file which cannot be modified in any way. Thus, the system cannot be used as a flexible decentralized data

F. Bao et al. (Eds.): ISPEC 2016, LNCS 10060, pp. 79–93, 2016.
DOI: 10.1007/978-3-319-49151-6_6

storage and the storage effort is essentially wasted. Filecoin [9] on the other hand introduced a PoR in a way that allows flexible file upload and retrieval. However, it still requires the energy consuming PoW mining process of Bitcoin in its design. In addition, the system is not fair for small miners, due to the design of Filecoin's mining process, as will be explained in Sect. 4.

Current distributed storage systems like Freenet [5] or GNUnet [4] do not offer incentives for users to contribute storage resources to other users. The only incentive is reciprocity as one hopes that others likewise will contribute storage. However, free-riding and churn are common problems in those systems, substantially reducing their reliability [12].

In this paper, we propose KopperCoin, a distributed storage system where peers can store and retrieve files and which includes a token system to reward those contributing storage resources. It is based on the Bitcoin blockchain idea but replaces the PoW mining process completely with a PoR. To encourage users to participate in the network, clients who store files of other users are able to mine on these files and consequently have the chance to generate tokens, called koppercoins. In addition, participants can gain koppercoins by allowing users to retrieve files. These tokens in turn can be spent to store more files. This mechanism creates a big advantage over traditional distributed file storage systems since in our system, users have valid incentives to contribute storage to other KopperCoin users. Even commercial entities can base their business model on mining, as in Bitcoin mining farms, but with the added benefit of contributing to the decentralized file storage.

In the next section we provide an overview of Bitcoin and proofs of retrievability, as basis to understand our KopperCoin scheme which is explained in Sect. 3. We will continue to present related work in this area in Sect. 4 followed by a discussion of our scheme in Sect. 5. Section 6 concludes our work.

2 Building Blocks

Since our system architecture is heavily based on technologies used in Bitcoin and proofs of retrievability (PoRs), we will first provide a short overview of these techniques.

2.1 Bitcoin

In 2009 Satoshi Nakamoto presented Bitcoin [17], the first truly decentralized cryptocurrency. A common challenge in digital payment systems is to prevent *double-spending* of coins. Since in previous e-cash designs coins were represented as digital data they might simply be copied and spent multiple times. Bitcoin tackles this problem by not storing the valid coins, but instead by storing all valid transactions, i.e., changes of possession of Bitcoins in a publicly verifiable ledger. All valid transactions are included in a global public sequence of blocks in the peer-to-peer network called the *blockchain* which is stored by each miner. This way, each participant in the network can check if the transaction is valid by verifying the history of ownership up to the point where the coins were generated in the network.

If user Alice wants to send Bitcoins to Bob she creates a new transaction. In the *input* of this transaction she references a previous transaction which included her public key in the *output*. In the output of her newly created transaction she includes the public key of Bob. To prove that Alice is authorized to spend the funds of the previous transaction output she signs her transaction. This works, because she is the only one possessing the private key corresponding to the public key in the referenced transaction output. Finally she broadcasts her transaction into the network, where it will be included in a block.

To save storage space, the transactions are aggregated in a Merkle tree [14,15]. The root of the Merkle tree, which is comparable to a fingerprint of all the transactions, is included in the block headers which are necessary for verification of the blockchain. This is an important implementation detail which provides scalability. A whole block consists of the block header and all the transactions which are aggregated in it.

New blocks are generated through a process called *mining* and are appended to the blockchain. In an abstract way, the mining process is a distributed consensus protocol without pre-known identities. The miners receive blocks which are challenges for proofs of work and vote with their computational power on the validity of transactions. Thus they agree on the global state of the accounts. Further, the problem of Sybil attacks is solved by binding the digital identities to computational resources [2].

A simplified mining process works as follows: Let $\mathcal{B}_1, \dots, \mathcal{B}_n$ denote the block headers in the blockchain. Each block-header contains:

- The Merkle root of the transactions aggregated in the block.
- A reference to the previous block realized as a hash value.
- Other fields like a timestamp and the version number.
- Data relevant to the consensus: a *nonce* and a *difficulty* parameter.

Let $\mathcal{H}$ denote a cryptographically secure hash function. To generate a new block with header $\mathcal{B}_{n+1}$, participants try to find a *nonce* for $\mathcal{B}_{n+1}$, such that $\mathcal{H}(\mathcal{B}_{n+1})$ is below a certain threshold *difficulty*. The miner includes the transactions he received previously via broadcast in his new block.

The new block is then broadcast to the other miners. Each receiving miner checks the validity of the new block, i.e., whether

$$\mathcal{H}(\mathcal{B}_{n+1}) < \textit{difficulty}$$

and if the included transactions are correctly signed and valid. If the received block is valid the miners append it to their local copy of the blockchain and continue to mine the next block $\mathcal{B}_{n+2}$. Otherwise they reject the block and instead go on to mine a valid $\mathcal{B}_{n+1}$.

The parameter *difficulty* is agreed upon dynamically by the miners. It is also included in the respective blocks and adjusted every 2016 blocks to account for fluctuations in the overall hash rate of the network. Thereby, the block generation rate of the network remains around one block every ten minutes, independent of the total hash rate. A stable block mining rate is necessary for the functionality of the system due to the non-zero propagation delay in the network. If the difficulty would not get adjusted, a rise of the overall computing power could lead to the situation that block generation time is lower than the propagation delay. Then some nodes would not be able to see the current block and thus would have no chance to mine the next one.

To incentivize users to participate in the system, a mining reward in form of Bitcoins is given to the miner who generates a new block. The first transaction in each block is a special transaction called *coinbase* which grants a fixed amount of Bitcoins to the miner of that particular block. These Bitcoins do not have a previous owner, so they are freshly introduced into the Bitcoin system. This compensates for the computational effort spent. In addition, the coinbase transactions serve as an initial wealth distribution mechanism.

When multiple miners find a new block simultaneously both of them broadcast it. The other miners then have two possible valid blocks on which they can continue to mine. This situation is called a *fork*. The miners mine on one of the blocks until eventually one of the chains is longer. Bitcoin assumes that the honest miners control the majority of the computational resources of the network and thus try to extend the longest chain. With this assumption the network converges to one blockchain and therefore one global state of the accounts. If the assumption of an honest majority is not given, double-spending becomes possible. An attacker with more than 50 % of the computational resources of the network can buy goods with a transaction on the main chain, fork the chain at some point in the past, and extend his fork beyond the main chain. When the chain not containing his transaction is the longest one he effectively reverses his transaction. An attacker with less computational resources cannot execute this attack, since the main chain will always grow faster than his chain. At the moment 50 %-attacks are ignored because they are considered to be expensive and thus it is unlikely that a single attacker holds 50 % of the computational power in the network. However there are effects leading to centralization, like mining pools, such that 50 % attacks are not as infeasible as assumed. There is research indicating that even an honest majority does not suffice to guarantee stability of the Bitcoin system [7]. The strongest attacker model where Bitcoin works is yet unknown and subject to research.

For a more detailed description of Bitcoin we refer to the original whitepaper [17], the survey by Tschorsch and Scheuermann [21], and the book by Antonopoulos [1].

2.2 Proofs of Retrievability

A PoR is a challenge-response protocol which allows a storage provider to prove possession of a certain file. It is related to proofs of knowledge where a prover convinces a verifier that he has some knowledge. Our construction requires a PoR that is publicly verifiable and of constant size. In addition, the PoR needs to support an unlimited number of proofs over the same file. A scheme that satisfies our requirements is the one by Shacham and Waters [19] which we briefly sketch in the following.

Let $\mathcal{P}$ be the prover and $\mathcal{V}$ a verifier. We denote the user who has uploaded the file with $\mathcal{U}$. Let $m_1, \ldots, m_n$ be chunks of a file over which retrievability has to be proven. The chunks are chosen in such a way that $m_i \in \mathbb{Z}/p\mathbb{Z}$ for all $i \in \{1, \ldots, n\}$. Intuitively we use homomorphic authenticators σ_i for each chunk m_i in such a way that verifiers can be convinced that a linear combination of blocks $\mu = \sum_{(i,\nu_i)\in Q} \nu_i \cdot m_i$ was correctly computed, where Q is a challenge set chosen at random.

Let G be a group with support $\mathbb{Z}/p\mathbb{Z}$ and generator $g \in G$. Let $e : G \times G \rightarrow G_T$ be a computable bilinear pairing. The private key of the user $\mathcal{U}$ who has uploaded the chunks of the file is an element $x \in \mathbb{Z}/p\mathbb{Z}$ chosen uniformly at random. His corresponding public key is $pk_{\mathcal{U}} = (v, u)$, where $v = g^x \in G$ and $u \in G$ is another generator of G. The uploading party $\mathcal{U}$ creates and uploads authenticators $\sigma_i = \big(\mathcal{H}(i)u^{m_i}\big)^x$ over each chunk m_i, where $\mathcal{H}$ is a hash function. A verifier $\mathcal{V}$ chooses a challenge set $I \subset \{1, \ldots, n\}$ and some random coefficients $\nu_i \in \mathbb{Z}/p\mathbb{Z}$ for $i \in I$. The challenge consists of the set $Q = \{(i, \nu_i),\, i \in I\}$.

$\mathcal{P}$ sends back the proof (σ, μ), where $\sigma = \prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i}$ and $\mu = \sum_{(i,\nu_i)\in Q}$ $\nu_i \cdot m_i$. Verification is done by checking if

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

If the equation holds, then $\mathcal{P}$ stores the chunks $m_1, \ldots, m_n$ with high probability. In particular it is computationally hard for $\mathcal{P}$ to convince a verifier that he stores a file by providing a correct proof (σ, μ) without actually storing the file in question. Note that for verification one does not need any form of secret information. Thus the scheme is publicly verifiable. For details and further discussion of the security properties, we refer the reader to the original paper [19].

In our scheme we prove retrievability of chunks and not of files, so the m_i in KopperCoin are in fact subchunks of chunks of files.

Proofs of Space: In the literature there exists a similar notion of proofs of space [3,6]. To compute a proof of space the prover needs to employ a specific amount of memory. This is in contrast to a PoR where the storage provider proves possession of a specific file and not that he is in charge of a specific amount of memory.

3 KopperCoin Scheme

In this section we sketch our proposed construction of the KopperCoin scheme. We will first provide an overview and then dive into the details from Sect. 3.2 onwards.

3.1 Overview

The KopperCoin scheme identifies each entity by its public key as in Bitcoin. KopperCoin has its own blockchain as a global public transaction log. In contrast to Bitcoin, KopperCoin does not reward the miners proportionally to their computational resources, but instead proportionally to how much data of other participants in the network they store.

A file f is represented as a series of chunks $f = (c_1, \ldots, c_\ell)$ of same length, possibly padded. We always denote the pieces of a file by the term "chunk", whereas "block" always refers to blocks in the blockchain, to prevent ambiguity. The chunks cannot be linked to files, since they have identical length. A client application is needed for the splitting into chunks and reassembly on retrieval, together with optional erasure encoding for recovery of files.

Mining a new block uses a publicly verifiable proof of retrievability (PoR) over a data chunk which is close to a challenge value determined by the previous block header in the blockchain. The distance acts like a quality parameter of the block. It is computed in the address space of the chunks as will be explained later. Blocks are considered valid if this distance is less than a difficulty parameter. Invalid blocks are simply dropped as in Bitcoin. We compute the PoR over chunks and not over files, i.e., each chunk c_j is split into subchunks $(m_1, \ldots, m_n)$ in order to be able to create the PoR. Since all chunks c_j have the same size, the number of subchunks n is independent of the chunk.

Since the challenge for the PoR is not known in advance, a miner who stores more chunks has a higher probability of possessing a chunk close enough to the challenge to mine a new block. To encourage users to participate in the system, a mining reward in form of koppercoins is given to the creator of a new valid block as in Bitcoin. Thus the more chunks a miner stores the higher the probability of earning koppercoins.

KopperCoin supports all transaction types that are supported by Bitcoin, which makes it possible to transfer koppercoins to other parties in the network. Furthermore, KopperCoin introduces a new transaction `store` with inputs c, σ, $pk_{\mathcal{U}}$, and *store_amount*. With this transaction chunks can be uploaded into the network. It includes the chunk $c = (m_1, \ldots, m_n)$ consisting of n subchunks to be uploaded, its authenticators $\sigma = (\sigma_1, \ldots, \sigma_n)$ for each subchunk and the public key $pk_{\mathcal{U}}$ of the uploading user $\mathcal{U}$ needed for verification of the PoR. The *store_amount* is an amount of koppercoins which determines how long the chunk should be stored. The koppercoins used in the `store`-transaction are removed from the network and become unspendable as will be explained in Sect. 3.3. Rewards for storing are gained through mining and providing files to others.

The PoR ensures integrity of the blockchain by making it prohibitively expensive to change previous blocks, since this would require redoing many PoRs over arbitrary files. In contrast to Bitcoin the block headers alone do not suffice to check integrity of the blocks since the public key of the uploader, which is included in the `store`-transactions, is required.

The exact time of expiration of a chunk depends on the amount of koppercoins used in the initial `store`-transaction. In case a miner includes a PoR over an expired chunk into a new block, this block is considered invalid by the other miners and discarded. Thus, miners have no incentive to store expired chunks and rational miners will delete them from their local storage. Thus, the expiration mechanism allows the network to regain storage space.

3.2 The Blockchain and Mining Process

The file storage in the KopperCoin network is designed as a key-value storage. There is a global set of keys K and a corresponding set of chunks, c_j, $j \in J \subset K$. Only a subset of the keys reference chunks, such that for many keys there exists no according chunk.

A valid block header in the KopperCoin-network includes the following fields:

- The Merkle root of the transactions aggregated into the block, which we denote by *merkle_root*.
- A hash of the previous block header.
- Data which is relevant for the consensus protocol: a *timestamp*, the *difficulty*, as well as a PoR (σ, μ) over a chunk c_j, as well as a reference to the `store`-transaction where c_j was uploaded.

Algorithm 1 describes the mining process. This algorithm is executed by each miner every time the timestamp advances or a new block is received. Newly computed blocks are broadcast into the network. If a new block is received it is checked for validity of the included transactions and correctness of the PoR. Let *address* be the public key of the miner. This is not included in the block header but can be retrieved from the coinbase transaction contained in the block. Then valid blocks additionally need to fulfill the following difficulty property:

$$\mathcal{H}(\mathit{address} \| \mathit{timestamp} \| \mathit{merkle\_root}) \cdot 2^{|j \oplus H|} \leq \mathit{difficulty},$$

where *timestamp* is the timestamp when the block was mined, H is the hash of the previous block, *merkle_root* is the root of the Merkle tree containing the transactions, and $j \in J$ is the index of the chunk whose retrievability was proven. The symbol $\oplus$ denotes bitwise XOR-operation. The block is then accepted or rejected accordingly.

We will now explain Algorithm 1 in detail. Let $\mathcal{H}_{ret}$ be a cryptographically secure hash function assuming values in the set of keys K. The miner computes the challenge H from block $\mathcal{B}_n$ by $\mathcal{H}_{ret}(\mathcal{B}_n)$ in Line 1. In Line 2 he computes the index j of the chunk over which he proves retrievability. This is the index of the locally stored chunk which is nearest to the challenge H in the XOR-distance. In Line 3 he retrieves the locally stored authenticators corresponding to the chunk determined in the previous step. In Line 4 the miner tests if the index of his chunk is near enough to H and thus if he can mine the next block. If this is the case the PoR is created in Line 5 and 6, and the new block is broadcast into the network. Otherwise the next block is currently not mineable for this miner and he has to wait until the timestamp advances or until a valid block of another participant is received.

Algorithm 1. The mining algorithm for computing new blocks in KopperCoin

```
Input: $timestamp$, newest block header $\mathcal{B}_n$, $difficulty$, root of the Merkle tree containing the transactions $merkle\_root$
Output: next block $\mathcal{B}_{n+1}$ if possible to compute
1: $H \leftarrow \mathcal{H}_{ret}(\mathcal{B}_n)$    ▷ hash of current block header
2: $j \leftarrow \text{argmin}_k \{H \oplus k \mid c_k \text{ is stored locally}\}$    ▷ index nearest to $H$ where the corresponding chunk is stored locally
3: $\Sigma \leftarrow$ authenticators $(\sigma_1, \ldots, \sigma_n)$ of $c_j = (m_1, \ldots, m_n)$
4: if $\mathcal{H}(address \| timestamp \| merkle\_root) \cdot 2^{|j \oplus H|} \leq difficulty$ then
5:     $Q \leftarrow \text{PRF}(\mathcal{B}_n)$    ▷ challenge set derived from a PRF applied to $\mathcal{B}_n$
6:     $(\sigma, \mu) \leftarrow$ PoR of the chunk $c$ with challenge $Q$ and authenticators $\Sigma$
7:     return new block with aggregated transactions and $(\sigma, \mu)$
8: end if
9: return next block is not mineable
```

A PoR internally uses a challenge Q different from H as explained in Sect. 2.2. This challenge Q contains some subchunks m_i and corresponding coefficients ν_i. Originally, PoRs are interactive, but can be transformed to non-interactive PoRs by the Fiat-Shamir transformation [8]. This means that the challenge H is generated by applying a pseudorandom function PRF, mapping from the space of blocks to the space of challenges, to the block header $\mathcal{B}_n$ in Line 5. The PoR, namely μ and σ is published in the header of the mined block $\mathcal{B}_{n+1}$.

Note that the challenges H and Q derived from blocks are all pairwise different, since otherwise there would exist two blocks $\mathcal{B}_n \neq \mathcal{B}'_n$ with the same challenge, i.e., $\mathcal{H}_{ret}(\mathcal{B}_n) = \mathcal{H}_{ret}(\mathcal{B}'_n)$. This is a collision of a cryptographically secure hash function and thus will only occur with negligible probability.

It is impossible to change the transactions contained in a block after that block is mined. If one changes a transaction in the Merkle tree the root *merkle_root* changes unpredictably. Since this is included in a cryptographically secure hash function each bit of $\mathcal{H}(address \| timestamp \| merkle\_root) \cdot 2^{|j \oplus H|}$ changes with probability 1/2. So it is infeasible to modify transactions which are included in the blockchain and thus integrity of the transactions is guaranteed.

Like in Bitcoin occasionally it can happen that two blocks are mined simultaneously by different miners thus creating a fork in the chain. The miners then try to extend the chain at the block where the value which is compared against the difficulty parameter, i.e., $\mathcal{H}(address \| timestamp \| merkle\_root) \cdot 2^{|j \oplus H|}$, is smaller. When the two chains differ in length they are mining on the longest chain by KopperCoin protocol rules. Thus this chain grows faster, since it is backed by more resources and eventually the miners abandon the shorter chain.

In Bitcoin, if some malicious miner controls the majority of computational resources, he can extend both chains at the same speed, thereby preventing consensus. In the KopperCoin system this situation can also happen, but the attacker needs more than half of the storage resources of the network, instead of computational resources. We assume that this is infeasible if our network is big enough. Additionally, an entity controlling a majority of storage resources will perhaps prefer to comply with protocol rules, since otherwise trust in the system will disappear and therefore the koppercoins, which he would be able to mine, become worthless.

3.3 The Store Transaction

The `store` transaction allows participants to store chunks. `store` takes as input a chunk c, its authenticators $\sigma_1, \ldots, \sigma_n$ computed by the client, the public key $pk_{\mathcal{U}}$ of the client, as well as an amount of koppercoins.

The koppercoins included in the `store` transaction vanish from the network and cannot be spent anymore. This payment is necessary to avoid denial-of-service attacks, since an attacker could otherwise upload an arbitrary number of chunks for free and thereby exceed the available storage in the network. The payment is, in addition, a form of inflation protection. As the amount of available koppercoins decreases the value of the remaining koppercoins increases, since only a limited amount of koppercoins are in existence at any time.

Miners can choose to store the chunk together with its authenticators to be able to create a PoR over this chunk and thus to generate a new block. In addition, the miners need to store the public key $pk_{\mathcal{U}}$ of the transaction issuer for verification purposes.

The miners do not need to store all files and are possibly not even able to do so. The incentive to store files is of economical nature, since by storing one can possibly mine a new block in the blockchain and collect mining rewards. The storage guarantees can of course be increased arbitrarily by applying an appropriate erasure code on the file to be uploaded. Beyond these financial incentives there are no further mechanisms to increase storage guarantees.

The storage period of the chunk is linearly dependent on the amount of koppercoins spent when issuing the `store`-transaction.

After the storage period has passed, blocks which include a PoR over that particular chunk are not considered valid any more. Assuming that the majority of miners do not accept such blocks, there is no incentive to store the chunks any longer. The blockchain already provides a loose synchronization of time and thus all miners can agree on when the requested storage period has passed.

3.4 Fetching Files

In order to fetch a file the client application needs to know the identifiers of the corresponding chunks. The file is restored by retrieving sufficiently many chunks. For successful retrieval not all chunks have to be fetched, depending on the erasure code that was applied before storing the file in the KopperCoin-network. The erasure code solves the problem of missing chunks and storage providers demanding unrealistically high prices for chunk retrieval.

Fetching chunks works with 2-2 multisignature transactions. These are transactions which can be spent if and only if two out of two parties agree to spend them. To our knowledge the mechanism was first used by NashX [23].

Let $\mathcal{U}$ be a user who wants to retrieve a chunk which is stored at the provider $\mathcal{P}$. Suppose $\mathcal{U}$ wants to pay the amount p for retrieving his file. Then $\mathcal{U}$ and $\mathcal{P}$ create a 2-2 multisignature transaction where the user $\mathcal{U}$ inputs $\beta + p$ and $\mathcal{P}$ inputs α. The amounts α and β are security deposits. In a next step $\mathcal{P}$ sends the chunk to $\mathcal{U}$. The user $\mathcal{U}$ checks if he has received the correct chunk. In that case he signs a multisignature transaction with two outputs: The provider $\mathcal{P}$ gets back his security deposit α, together with the price p for the chunk. In the other output the user $\mathcal{U}$ gets back his security deposit β. The process is illustrated in Fig. 1. Above the arrows are the amounts and below the arrows are the owners of the respective amounts.

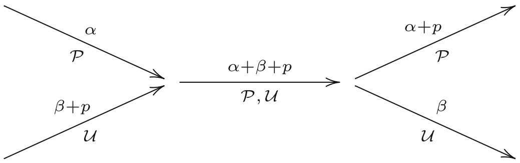

Fig. 1. File retrieval

If $\mathcal{U}$ wants to cheat he cannot set his security deposit β to zero or otherwise change the first transaction since this will be detected by the provider $\mathcal{P}$ who then refuses to sign. Nevertheless the user $\mathcal{U}$ can refuse to sign the 2-2 multisignature transaction after retrieving the chunk, thereby losing his security deposit β.

If the provider $\mathcal{P}$ cheats he can either refuse to send the chunk or refuse to sign the 2-2 multisignature transaction. In both cases he will suffer a financial damage of his security deposit α and not receive the price p for retrieval of the chunk.

4 Comparison with Related Cryptocurrencies

In this section we will present other cryptocurrencies which combine file storage with payment and compare it to our scheme where possible.

As already mentioned in the introduction, there are other cryptocurrencies which try to harness the computational effort of blockchains which is a consequence of using Proof of Work as a countermeasure against Sybil attacks and as a voting mechanism in the consensus protocol. Peercoin [11] for example exchanges proof of work (PoW) by proving possession of another scarce resource, namely the coins themselves. This approach is called Proof of Stake.

There were also some approaches before KopperCoin to include a proof of retrievability (PoR) instead of a PoW in a bitcoin-style cryptocurrency. In Permacoin [16] the miners prove retrievability of a large publicly valuable digital archive where single miners are unlikely to have the resources to store all the data. This large digital archive is globally fixed and no changes are possible. Thus, Permacoin mainly guarantees integrity of a fixed file. Compared to

Permacoin [16] we are able to store dynamic files chosen by the individual users in contrast to one large static file chosen by the creator of the blockchain. Therefore, KopperCoin provides a distinct utility advantage over Permacoin. Further, Permacoin requires a trusted dealer for initial distribution of the file, in contrast to our scheme.

Retricoin [18] offers efficiency improvements over Permacoin but suffers from the same structural problems.

Filecoin [9] is another approach to incorporate a file system into a cryptocurrency. In Filecoin it is possible to store and fetch files chosen by the users. Files stored in Filecoin have an expiry date, after which there is no reward for storing them anymore.

Filecoin extends the classical hash-based PoW of Bitcoin with an additional PoR. Thus they have two difficulty parameters to regulate the growth speed of the blockchain. One difficulty parameter is from the hash-based PoW and the second is from the PoR. In their paper it is not explained how those difficulty parameters are designed to interact. Their difficulty parameter for the PoR is realized by the amount of files of which miners need to prove retrievability. Beyond a certain difficulty parameter, small miners are never able to mine new blocks because they do not have the necessary storage. This leads to centralization pressure, since these small miners are unable to mine blocks beyond a certain difficulty.

In contrast, the stochastic nature of Bitcoin's PoW scheme ensures that even small miners can mine blocks, albeit with proportionally less probability. In KopperCoin we also encourage small miners to provide resources to the network. KopperCoin uses the distance of a chunk of which retrievability needs to be proven to a challenge predetermined by the blockchain as difficulty. Thus small miners are always able to mine koppercoins proportionally to their storage contribution to the network.

In particularly we defined

$$\mathcal{M}(\mathcal{B}_n, address) = \mathcal{H}(address \| timestamp \| merkle\_root) \cdot 2^{|j \oplus \mathcal{H}_{ret}(\mathcal{B}_n)|}$$

as the "quality" and therefore the difficulty of the PoR of the chunk c_j. Recall that *timestamp* is a timestamp with appropriate resolution, *address* is the public key of the miner, and *merkle_root* is the root of the Merkle tree containing the transactions.

This fulfills the following properties:

(i) The more chunks one stores, the higher the probability to store a chunk whose address is close to $\mathcal{H}_{ret}(\mathcal{B}_n)$. And the nearer the key of the chunk whose retrievability is proven is to $\mathcal{H}_{ret}(\mathcal{B}_n)$, the smaller the result of our mapping $\mathcal{M}$ is. Therefore, $\mathcal{M}$ behaves like a difficulty parameter. Note that the probability of mining a block is proportional to how many files the miner stores:

$$P\left[\begin{array}{c}\forall address' \neq address : \\ \mathcal{M}(\mathcal{B}_n, address) > \mathcal{M}(\mathcal{B}_n, address')\end{array}\right] = \frac{\#\text{ files stored by } address}{\#\text{ files in the system}}$$

(ii) The mapping $\mathcal{M}$ depends on the miner. If two or more miners prove retrievability of a chunk with the same distance to the key $\mathcal{H}_{ret}(\mathcal{B}_n)$, they get different values since *address* is included in the hash function. If the mapping would not depend on the miner such a situation would create a fork of the blockchain.

(iii) It is impossible to end up with a block $\mathcal{B}_n$ where no-one can successfully append a next block $\mathcal{B}_{n+1}$, since the timestamp will change and thus also the challenge. This provides liveness of the blockchain, i.e., it is always possible to find a subsequent block after sufficient time has passed.

In particular, we have chosen the XOR-distance $d(x, y) = x \oplus y$ as a metric because it is unidirectional [13]. This means that for each distance δ and each fixed bit sequence x there is exactly one y satisfying $d(x, y) = \delta$. Thus we have a unique distance to each chunk and therefore a clearly defined priority over which chunk retrievability needs to be proven.

In Filecoin the files of which one has to prove retrievability are chosen deterministically. Thus if one of these files is not available in the network anymore it is impossible to mine a future block leading to the death of the network, since no one can append blocks to the blockchain. KopperCoin solves this problem by allowing files near a deterministically chosen index and by including a timestamp in the index choosing mechanism.

KopperCoin further distinguishes itself from Filecoin in that we do not use a Bitcoin-style PoW at all, since we consider this a waste of energy.

Another peer-to-peer cloud storage network offering incentivisation is Storj [22]. In Storj the PoRs are not integrated into the blockchain but are handled by a heartbeat protocol. The PoRs are done with Merkle trees and thus their size depends on the size of the files. In contrast to our scheme the data locations in Storj are included into the blockchain which could lead to efficient censorship.

5 Discussion

In contrast to other cryptocurrencies, we use less computational resources since in order to mine koppercoins it is not necessary to brute-force a hash function. Instead we require storage resources which are used to power the underlying distributed data storage.

Other distributed file storages like Freenet or GNUnet provide similar advantages as KopperCoin, but are not very successful, since not many storage providers participate. We believe that this can be changed if incentives, financial or otherwise, exist for providing storage for other participants in the network.

KopperCoin incorporates such incentives for joining the scheme. When contributing storage to the KopperCoin network and thus storing files of other parties, one can generate koppercoins and earn unclonable tokens. The generated value is directly proportional to the amount of storage provided. Thus we expect that commercial entities will engage in KopperCoin similar to commercial Bitcoin miners.

One disadvantage of KopperCoin is the lack of deterministic storage guarantees, since currently we cannot know if a chunk is stored by any peer at all. We can guard against losing a fraction of the chunks with erasure-coding of the files, but this does not solve the problem completely. We remark that classical peer-to-peer systems like BitTorrent or Freenet also do not enforce any storage guarantees. We assume that participants in the system are aware of this issue and thus the price of the koppercoins will adjust accordingly due to the market mechanisms of supply and demand.

It could be conceivable, that the underlying P2P-network assigns files to special nodes to store them like, e.g., in Chord [20]. This mechanism could be included in the block verification procedure, such that blocks are only valid if the proof of retrievability (PoR) is created by the node responsible for the storage of the chunk. This could increase scalability and storage guarantees. However this reveals which nodes store which files what could be seen as a privacy problem.

Future work will address determining the optimal parameters like, e.g., size of the chunks, maximum blocksize, and adjusting of the difficulty parameter of the PoR.

6 Conclusion

This paper presented KopperCoin, a decentralized token system combined with a peer-to-peer file storage system which provides direct reward for participants contributing storage resources. It is based on the idea of a blockchain to manage ownership and files. The mining process to maintain the network is realized by a proof of retrievability (PoR) instead of a proof of work (PoW). Miners create cryptographic proofs that they store files, thereby mining koppercoins, which are unclonable tokens with an owner, managed decentrally by the blockchain of the KopperCoin system. Koppercoins provide incentives to offer storage resources for the peer-to-peer file storage system.

We outlined basic concepts and discussed benefits of KopperCoin in terms of tight integration of the file storage system with the token system as a reward mechanism.

For insight into the usability we need to tune the parameters by performing large-scale experiments and investigate the performance in realistic environments.

KopperCoin is a promising approach to implement a distributed peer-to-peer file storage system that provides usability and offers incentives for participation. Thus, participation could be improved beyond traditional peer-to-peer file storage systems that rely on voluntary resources.

References

1. Antonopoulos, A.M.: Mastering Bitcoin, Unlocking Digital Cryptocurrencies. O'Reilly Media, Sebastopol (2014)
2. Aspnes, J., Jackson, C., Krishnamurthy, A.: Exposing computationally-challenged Byzantine impostors. Technical report YALEU/DCS/TR-1332, Yale University Department of Computer Science (2005)
3. Ateniese, G., Bonacina, I., Faonio, A., Galesi, N.: Proofs of Space: When Space is of the Essence. Cryptology ePrint Archive, Report 2013/805 (2013)
4. Bennett, K., Stef, T., Grothoff, C., Horozov, T., Patrascu, I.: The GNeT whitepaper, June 2002
5. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: a distributed anonymous information storage and retrieval system. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 46–66. Springer, Heidelberg (2001). doi:10.1007/3-540-44702-4_4
6. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. Cryptology ePrint Archive, Report 2013/796 (2013)
7. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45472-5_28
8. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). doi:10.1007/3-540-47721-7_12
9. filecoin.io: Filecoin: a cryptocurrency operated file storage network (2014). http://filecoin.io/filecoin.pdf
10. King, S.: Primecoin: cryptocurrency with prime number proof-of-work (2013). http://primecoin.io/bin/primecoin-paper.pdf
11. King, S., Nadal, S.: PPCoin: peer-to-peer crypto-currency with proof-of-stake (2012). https://peercoin.net/whitepaper
12. Ma, R.T.B., Lee, S.C.M., Lui, J.C.S., Yau, D.K.Y.: Incentive and service differentiation in P2P networks: a game theoretic approach. IEEE/ACM Trans. Netw. **14**(5), 978–991 (2006)
13. Maymounkov, P., Mazières, D.: Kademlia: a peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002). doi:10.1007/3-540-45748-8_5
14. Merkle, R.C.: Method of providing digital signatures. US Patent 4,309,569, 5 Jan 1982. https://www.google.com/patents/US4309569
15. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). doi:10.1007/3-540-48184-2_32
16. Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: repurposing bitcoin work for data preservation. In: Security and Privacy, pp. 475–490. IEEE (2014)
17. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2009). https://bitcoin.org/bitcoin.pdf
18. Sengupta, B., Bag, S., Ruj, S., Sakurai, K.: Retricoin: Bitcoin based on compact proofs of retrievability. ICDCN 2016. ACM (2016). http://doi.acm.org/10.1145/2833312.2833317
19. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89255-7_7

20. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Comput. Commun. Rev. **31**(4), 149–160 (2001)
21. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: a technical survey on decentralized digital currencies. Cryptology ePrint Archive, Report 2015/464 (2015)
22. Wilkinson, S., Buterin, V.: Storj: peer-to-peer cloud storage network (2014). https://storj.io/storj.pdf
23. Yoo, S.Y.: How a NASHX transaction works (2013). http://nashx.com/HowItWorks
24. Young, A., Yung, M.: Auto-recoverable auto-certifiable cryptosystems. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 17–31. Springer, Heidelberg (1998). doi:10.1007/BFb0054114