Design of a Privacy-Preserving Decentralized File Storage with Financial Incentives

***Abstract*—Surveys indicate that users are often afraid to entrust data to cloud storage providers, because these do not offer sufficient privacy. On the other hand, peer-2-peer-based privacy-preserving storage systems like Freenet suffer from a lack of contribution and storage capacity, since there is basically no incentive to contribute own storage capacity to other participants in the network.**

We address these contradicting requirements by a design which combines a distributed storage with a privacy-preserving blockchain-based payment system to create incentives for participation while maintaining user privacy. By following a Privacy by Design strategy integrating privacy throughout the whole system life cycle, we show that it is possible to achieve levels of privacy comparable to state-of-the-art distributed storage technologies, despite integrating a payment mechanism.

Our results show that it is possible to combine storage contracts and payments in a privacy-preserving way. Further, our system design may serve as an inspiration for future similar architectures.

I. Introduction

There is a big demand for outsourcing storage to external providers, e.g. for remote backup, network-wide file access, or for sharing files among users. One major concern when using these systems is privacy, as system operators may learn file content or at least meta-data about your online activities.

Apart from commercial cloud storage providers there are decentralized peer-to-peer storage networks with a focus on privacy like Freenet [3] or GNUnet [2]. There, users allocate parts of their hard drives to store files for other network participants.

These peer-to-peer storage networks suffer from a freerider problem and missing storage capacity due to non-existent incentives to contribute storage. There are simply no advantages in storing files for other participants. One could add a financial incentive system to Freenet or GNUnet, but naïvely implemented this will negatively affect privacy.

We aim to address these two contradicting requirements by combining a privacy-enhanced payment scheme on the basis of blockchain and smart contracts technology with a distributed storage system. Users can enter smart contracts with storage providers to exchange amounts of a digital currency against storage of a file. These contracts along with money transfers are stored in a blockchain. This provides a way to reward participants who contribute their storage resources to the network. By designing a privacy-enhanced smart contracts system, those contracts cannot be abused to retrieve meta-data on user activities. For file data, we rely on existing mechanisms for file encryption in cloud storage.

Related Work: Previous approaches to design a decentralized storage system with integrated payment, like Permacoin [15] or Retricoin [24] focus on substituting the proof of work mechanism of Bitcoin-like blockchains with more ecological alternatives. These schemes support the storage of a static file, which is chosen at the setup time of the system by the creator of the blockchain. Therefore these schemes offer no distributed file storage in the proper sense.

KopperCoin [11] is an alternative concept which also replaces the proof of work, but was designed to provide storage of arbitrary user files and thus serve as distribute file storage like Freenet or GNUnet but with integrated payment mechanism.

However, KopperCoin misses to take privacy considerations into account. To store a file in the KopperCoin system, users issue a store transaction to the blockchain, thereby revealing their public key. Thus, the scheme links file identifiers to public keys of the users which can be seen as a user pseudonym. On the level of transactions KopperCoin provides the same amount of privacy as Bitcoin, which is understood to be insufficient [14], [23].

Our Contribution: The contributions in this paper can be summarized as follows:

- We propose a novel design of a privacy-preserving decentralized storage system including a privacy-preserving payment mechanism based on ring signatures and one-time addresses.
- We implemented significant parts of our design to test the practical feasibility of our system.
- Finally, we provide an extensive discussion of the privacy and security guarantees our system offers.

In Section II we introduce the foundations necessary for understanding the remainder of this article. Section III contains a detailed description of our system. The security and privacy properties of our system are discussed in Section IV. Finally, we conclude this paper with Section V.

II. Building Blocks

A. Bitcoin and the Blockchain

Bitcoin [16] is the first widely successful decentralized electronic payment system. In the Bitcoin system, participants called *miners* vote on the validity of transactions and include them in so-called *blocks*. Since there are no fixed participants in Bitcoin, the votes cannot be bound to identities. Instead, miners effectively "vote" with their computational power, since a valid block needs to have a small hash value. If a valid block is found, it is broadcast into the network.

Each block includes a reference to the previous block, thus the blocks form a chain, which is called *blockchain*. The blockchain serves as an immutable global ledger of transactions. If two or more miners find a block approximately at the same time a *fork* occurs. This means that there are multiple blocks referencing the same previous block. In this case the miners continue to mine at one of the ends until one of the chains is longer. From this point on by consensus the longer chain is extended and the other is deemed invalid.

Miners are rewarded for supporting the system with their computational power through a special transaction, called *coinbase*. This transaction grants a miner a fixed amount of a virtual currency, called Bitcoins, for each block the miner found. These Bitcoins are freshly introduced into the system and have no previous owner.

Transactions in Bitcoin consist of inputs and outputs. Each transaction input has a reference to a previous transaction output which it spends. There are mostly three types of outputs: one output for the recipient $\mathcal{R}$ of the payment, one output for change, and one output for transaction fees. The transaction output for change provides divisibility of the money. This is necessary if the sender $\mathcal{S}$ of the transaction is not in possession of a transaction output with the proper amount. The transaction fee in Bitcoin is handled implicitly as the difference between the amounts of the inputs and the amount of the outputs. The fee is given to the miner who includes the transaction in a block, thereby serving as an incentive to include this transaction. The payment channels are shown in Figure 1.

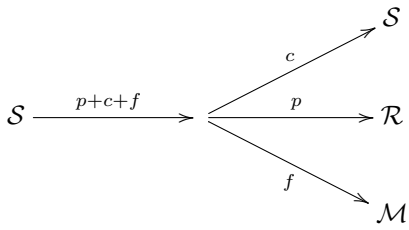

Figure 1. A payment p from a sender $\mathcal{S}$ to a recipient $\mathcal{R}$, including change c returning to $\mathcal{S}$ and a mining fee f kept by the miner $\mathcal{M}$, who mined the block including the transaction.

If there are multiple transactions claiming to spend the same transaction output this situation is detected by the miners, which only accept the first of these transactions. Such a situation is termed a *double spend*.

Transaction outputs in Bitcoin are realized with a domain specific language which specifies the terms under which the transaction can be spent. For a simple payment to another party, it consists mainly of the (hash of their) public key of the receiver which serves as a pseudonym. This implies that the transaction history of each token in the Bitcoin system is traceable, yielding very limited privacy since any observer can see the transaction recipients.

Using multiple public keys does not mitigate the problem, since they can often be linked by distinguishing between change and intended receiver in the transaction outputs (see [14], [23]).

B. One-time Payment Addresses

The blockchain-based cryptocurrency Cryptonote [27], focussing their development on privacy, introduced so called *one-time payment addresses*. This is a mechanism which takes the idea of using multiple public keys to the extreme.

Instead of simply referencing the recipient by its public key, the sender derives a new temporary public key per transaction output using a random nonce and the recipient's long-term public key. The derived one-time public key, called *destination key* and the original public key of the recipient are unlinkable without knowledge of the private key. The recipient can recover the private key corresponding to the destination key by using his private key and a *transaction public key* which is included in the transaction by the sender.

By signing with the recovered private key, the recipient can prove that she was in fact the intended recipient, and thus spend the funds, without revealing her identity or that the transaction belongs to her (long-term) private key.

A more extensive treatment can be found in the original whitepaper of Cryptonote [27] or in the work of Noether et al. [19].

One-time payment addresses provide *unlinkable transactions* ([21], [27]), i. e. , an observer cannot prove that any two transactions were sent to the same user.

C. Linkable Ring signatures

Ring signatures were first introduced by Rivest, Shamir and Tauman [22] in 2001. A ring signature is a digital signature which proves that the signer belongs to a group of signers. The signer needs its own private key, as well as the set of public keys of the other members in the group to create a ring signature. Especially, no group setup procedure is necessary. For verification, the signature, as well as the set of public keys of the members in the group is needed. Ring signatures can be used to prove membership in a group without revealing the identity.

In cryptocurrency systems ring signatures can be used to sign transactions. Using this mechanism a transaction input references multiple outputs of which one is the real output and the others are there to increase privacy. The signature is created over all outputs which are referenced in the transaction input. Of course, the amounts of the referenced transactions need to be equal, since otherwise the real amount cannot be determined from the set of referenced transaction outputs.

To prevent double spending in cryptocurrencies one needs to be able to link two signatures of the same signer. Thus, a simple ring signature does not suffice, but instead one needs a so-called traceable ring signature or linkable spontaneous ad-hoc group signature (LSAG) (cf. [6], [13], [27]). These allow an observer to link multiple signatures if they have the same signer without revealing its identity.

If an attacker tries to spend the same funds twice, it needs to create two transactions with one signature each, for the same transaction output. These signatures are linked and thus the second transaction is dropped as invalid by the miners.

Cryptonote [27] and many of its derivatives like Monero or Bytecoin currently use a variation of the FS-signatures [6]. In our system we use a linkable variation of LWW-signatures [13] since these are shorter. We sketch the scheme in the following, based on the work of Shen Noether [18].

Let $\mathbb{G}$ be an elliptic curve with generator G. Let $P_i \in \mathbb{G}$, $i = 1, \ldots, n$ be the public keys of the members in the ring. Assume that for the j-th public key we know the corresponding private key x with $P_j = xG$. Let $I = x\mathcal{H}_p(P_j)$ be the *key image*, where $\mathcal{H}_p$ is a hash function returning a point on the elliptic curve. The key image is used to enable linking of the signatures. Note that knowledge of the key image does not reveal the signer since x is private.

Let m be a message we want to sign, $\mathcal{H}$ a hash function. Let α, and s_i for $i = 1 \ldots, n$, $i \neq j$ be random values in the base field of the elliptic curve. Compute the following values.

$$\begin{aligned} L_j &= \alpha G \\ R_j &= \alpha \mathcal{H}_p(P_j) \\ c_{j+1} &= \mathcal{H}(m, L_j, R_j) \end{aligned}$$

For all $i \in \mathbb{Z}/n\mathbb{Z}$, $i \neq j$ define L_i, R_i, and c_i successively as follows.

$$\begin{aligned} L_{i+1} &= s_{i+1}G + c_{i+1}P_{i+1} \\ R_{i+1} &= s_{i+1}\mathcal{H}_p(P_{i+1}) + c_{i+1}I \\ c_{i+2} &= \mathcal{H}(m, L_{i+1}, R_{i+1}) \end{aligned}$$

The last step is closing the ring by "stitching" the two ends together. Let $s_j = \alpha - c_j x_j \mod \ell$, where ℓ is the order of the elliptic curve. Then

$$\begin{aligned} L_j &= \alpha G = s_j G + c_j x_j G = s_j G + c_j P_j \\ R_j &= \alpha \mathcal{H}_p(P_j) = s_j \mathcal{H}_p(P_j) + c_j I \\ c_{j+1} &= \mathcal{H}(m, L_j, R_j) \end{aligned}$$

The signature consists of $(I, c_1, s_1, \cdots, s_n)$.

When checking the signature the verifier computes the sequence $c_1, \ldots, c_n$ and checks if $c_{n+1} = c_1$.

Two signatures are linked, i.e., from the same signer, if they have the same key image I.

In the signature scheme used by Cryptonote [27] based on FS-signatures [6] the c_i are chosen randomly and appended to the signature. Thus a signature in Cryptonote consists of $(I, c_1, \cdots, c_n, s_1, \cdots, s_n)$ and is therefore larger than the scheme described.

Ring signatures enforce *untraceability* ([21], [27]) of transactions as all identities included in the ring have the same probability of being the real sender of the transaction.

D. Proofs of Retrievability

A *proof of retrievability* or *proof of storage* is a proof of knowledge which allows a storage provider to cryptographically prove the possession of a stored file.

The first such mechanism is due to Juels and Kaliski [9]. There, the client inserts so-called sentinel blocks into the file before uploading it to the storage provider. A proof of retrievability consists in querying some of the sentinel blocks. The client can verify if it received the correct sentinels with a cryptographic key, due to their special construction. The storage provider cannot delete the file and only store the sentinels, since without the key the sentinel blocks are indistinguishable from the file. This scheme is privately verifiable, since a secret key is needed to verify the proof.

In our construction we need a *publicly verifiable* proof of retrievability of *small size*. Publicly verifiable proofs of retrievability use an asymmetric key pair, where only the public key is needed to verify the proof. This allows verification by any external auditor.

With small size, we mean in particular that the size of the proof is independent of the size of the file and only depends on the choice of the security parameter.

Candidates we could use are, e.g., the scheme of Shacham and Waters [25], [26]. Ateniese et al. [1] showed that a publicly verifiable proof of retrievability with constant size can be generated from any homomorphic identification protocol.

In the following we give the intuition behind the definition of a publicly verifiable proof of retrievability by Ateniese et al. [1].

A publicly verifiable proof of retrievability is a tuple of four algorithms $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:

1) $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ is a probabilistic algorithm that is run by the client $\mathcal{U}$ to set up the scheme. Its input is a security parameter k, and the output is a public and private key pair (pk, sk).
2) $(f', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f})$ is a probabilistic algorithm that is run by the client in order to encode the file. It takes as input the secret key sk, and a file $\boldsymbol{f} \in \mathbb{Z}_B^\ell$ viewed as a vector of chunks with fixed size B. It outputs an encoded file f' and state information st. The encoding can be thought of as splitting the file and signing each chunk with a homomorphic signature. In particular the encoding does not provide any form of confidentiality. To achieve this, a client has to encrypt the file appropriately before encoding it.
3) $\pi \coloneqq \mathsf{Prove}(pk, f', c)$ is a deterministic algorithm run by the storage provider that takes as input the public key pk, an encoded file f', and a challenge $c \in \{0,1\}^\bullet$. The challenge is expanded by a hash function to a set of indices of chunks and corresponding coefficients. It outputs a proof π by combining the chunks and the homomorphic signatures according to the indices and coefficients from the challenge.
4) $b \coloneqq \mathsf{Verify}(pk, st, c, \pi)$ is a deterministic algorithm that takes as input the public key pk, the state st, a challenge $c \in \{0,1\}^\bullet$, and a proof π. It outputs a bit, where '1' indicates acceptance and '0' indicates rejection by checking if the aggregated signatures in the proof are a correct signature for the aggregated chunks. The state st is used to check if the aggregation was done over the correct chunks.

For correctness, we require that for all $k \in \mathbb{N}$, all (pk, sk) output by $\mathsf{Gen}(1^k)$, all files $\boldsymbol{f} \in \mathbb{Z}_B^\ell$, all (f', st) output by

$\mathsf{Encode}_{sk}(\boldsymbol{f})$, and all $c \in \{0,1\}^{\bullet}$, it holds that

$$\mathsf{Verify}\,(pk, st, c, \mathsf{Prove}(pk, f', c)) = 1.$$

In summary a proof of retrievability can be used to check if a storage provider is really storing a files of a user or just pretending to do so.

III. System Design

The previous chapter supplied us with the fundamental building blocks for our system. In this section we provide a short system overview and then go on to explain our system design more deeply and shed light on our design decisions.

A. Overview

The goal of our system is to provide a distributed storage system with financial rewards for the participants and strong privacy properties. Conventional methods of payment are not privacy sensitive, banks and merchants can access names and other personal data, so an abstraction from these methods is needed to ensure private payments. This abstraction is implemented by the blockchain which contains anonymised money transfers and storage contracts.

For the following analysis a small number of roles representing different usage of the system have been identified. Every participant in the system needs to support some core functionality which consists in being able to process cryptocurrency transactions and transfer money. Beyond the core functionality the system is designed with the following three advanced roles in mind.

- *User*: A user is a node which joins the network to store and retrieve its files in the system. Users create storage contracts, a pledge to pay a storage provider money if the storage provider keeps a file for a certain amount of time. These storage contracts are broadcast and eventually included in the blockchain.
- *Storage provider*: A node joining the network with the intention to earn money by providing storage space to users is called a storage provider. Storage providers are responsible for providing storage space, the main functionality of the system. Additionally, they publish proofs of retrievability of files they stored to prove their compliance with storage contracts.
- *Miner*: A miner earns rewards for maintaining the system. Miners validate transactions and storage contracts for their correctness and aggregates them in blocks in the blockchain, comparable to miners in Bitcoin.

A node can take on a combination of these advanced roles at the same time, e.g., it can provide storage, as well as mine new blocks.

B. Functionality

The roles outlined in the previous section correspond to the core functionalities of the system: Mining, transferring money, as well as storing and retrieving files. In the following we will explain these functionalities more deeply.

1) Mining: Mining is the process of appending valid blocks to the blockchain, containing new, valid transactions. We did not see any convincing reason to deviate from the design of the mining system employed in Bitcoin (cf. [16]). The process of mining fulfills two important functions: Maintenance of a valid distributed ledger by validating transactions and the initial distribution of money in our system.

To achieve this, a miner creates a block using a list of transactions that are not yet included in a block and a new one-time address for herself. A nonce is chosen at random and the block is compared to the difficulty of the previous block (see Algorithm 1). The process of changing the nonce and retrying the evaluation is repeated for a pre-specified time, until it succeeds, or until it is known that someone else has found a new block.

The difficulty is a value which is included in the blocks. If the hash of a block is smaller than the difficulty included in the previous block, the block is considered valid. The difficulty is used to scale block creation rates, as finding blocks with hashes smaller than a difficulty d is more computationally demanding as d decreases. Assuming the hashes are distributed uniformly random the probability of getting the difficulty right in one try is roughly $\frac{d}{2^b}$, where b is the bitlength of the hash.

The difficulty is adapted so that, on average, every 10 minutes a new block is generated. This parameter is taken from Bitcoin [16] and we did not find a compelling reason to change this. The pseudo-code of our mining process is given by Algorithm 1.

Algorithm 1 Mining new Blocks

Input: newest block $\mathcal{B}_n$, *difficulty*, list of *transactions*, maximum number of *tries*, One-time public key pk of the miner
Output: next block $\mathcal{B}_{n+1}$ or *fail*

```
B_{n+1} ← Block(transactions,pk,H(B_n))
▷ Create new block
Choose B_{n+1}.nonce uniformly at random.
for 1 ... tries do
    if H(B_{n+1}) ≤ difficulty then
        return B_{n+1}
    end if
    B_{n+1}.nonce ← nonce + 1
end for
return fail
```

New blocks from mining contain a coinbase transaction. This is a transaction which is included by a miner in its mined block. The coinbase transaction grants the miner a financial reward for mining that block. In contrast to Bitcoin, we use a one-time address to hide the identity of the miner.

If a dishonest miner includes invalid transactions in a new block it is rejected by the other miners. Thus, the reward of the dishonest miner in the form of the coinbase transaction is not included in the ledger.

2) Money Transfer: To transfer money a user needs to create and publish a transaction, which consists of a set

$\{in_1, \ldots, in_m\}$ of inputs and a set $\{out_1, \ldots, out_n\}$ of outputs.

An *output* of a transaction consists of an amount and a one-time payment address of the recipient. The private key corresponding to the address is used later to prove ownership of the output.

Each *input* in_i consists of a set of references to previous outputs $\{out_1^{(i)}, \ldots, out_{\ell(i)}^{(i)}\}$ together with a linkable ring signature over the outputs $\{out_1, \ldots, out_n\}$ of the transaction containing this input. To create this signature the users' secret key and the public keys contained in the referenced outputs are used. This proves that the participant owns at least one of the referenced outputs $\{out_1^{(i)}, \ldots, out_{\ell(i)}^{(i)}\}$. Further, the outputs of the transaction $\{out_1, \ldots, out_n\}$ cannot be modified without invalidating the signature.

All outputs $\{out_1^{(i)}, \ldots, out_{\ell(i)}^{(i)}\}$ referenced in an input in_i need to have the same value, since due to the ring signature the real transaction output cannot be distinguished from the other outputs in the anonymity set. If the amounts do not have the same value the transaction cannot be validated since the amount could be any of the amounts in the referenced outputs.

To prevent the problem of having no suitable outputs with the same amount that can be used as an anonymity set, there is a limited set of valid output values. This is comparable to banknotes, where there is only a limited set of values, and combinations of them are used to pay differing sums. The granularity of the output values has a clear impact on scalability and privacy. A small set of possible output values, e.g. only one in an extreme case, provides a large anonymity set, but increases transaction size, as one needs to use many output values for sums not equivalent to the provided units. This can be compared to paying in coins of only one unit. Many possible output values lead to small, more scalable transactions, but provide small anonymity sets. A justified choice of this parameter needs further evaluation and is not yet fixed.

The steps for sending a payment are given by Algorithm 2. In Line 2 and 3, one-time addresses are created to send money to the recipient and returning the difference to oneself. Lines 4 and 5 create outputs with their respective values for the payment and change addresses. In Line 9 and 10 an anonymity set of size n is retrieved from previous transactions in the blockchain and the public keys are extracted from those anonymity outputs. Finally, Line 11 signs the created outputs with a ring signature, using the keys from the anonymity set and the users' secret key.

The previously introduced coinbase transactions act as an initial anonymity set in the system.

3) File Storage: A user who wants to store a file generates a fresh public key pair and uses it to encode the file according to the proof of retrievability. The public key of this pair and the file identifier, called st, is included in a storage contract, which is a special kind of transaction, in the blockchain. This storage contract is publicly verifiable to enable verification by the miners.

Next, the user searches a storage provider accepting a file of the requested size for the given storage period c. Both create a contract for an agreed price that can be spend by a one-time key of the storage provider. Afterwards the user publishes the contract. The storage provider can see the contract in the blockchain and accept the file transfer. The operation to store a file is formalized by Algorithm 3.

Algorithm 2 Money Transfer

Input: Sender secret key $sk_{\mathcal{S}}$, recipient public key $pk_{\mathcal{R}}$, $value$ to transmit, anonymity set size n

Output: a transaction transferring $value$ amount from the owner $\mathcal{S}$ to the recipient $\mathcal{R}$

1: $(outputs_{own}, change) \leftarrow$ find-own-outputs$(value, sk_{\mathcal{S}})$
2: $ota_{\mathcal{R}} \leftarrow$ create-ota$(pk_{\mathcal{R}})$
3: $ota_{\mathcal{S}} \leftarrow$ create-ota$(pk_{\mathcal{S}})$
4: $targetoutput \leftarrow$ create-output$(ota_{\mathcal{R}}, value)$
5: $changeoutput \leftarrow$ create-output$(ota_{\mathcal{S}}, change)$
6: $tospend \leftarrow \{targetoutput, changeoutput\}$
7: $in \leftarrow \emptyset$
8: **for** $output$ **in** $outputs_{own}$ **do**
9: $\quad anonymityset \leftarrow$ find-outputs$(output, n)$
10: $\quad anonymityset_{pks} \leftarrow$ retrieve-keys$(anonymityset)$
11: $\quad signature \leftarrow \text{sign}(tospend, \{sk_{\mathcal{S}}, anonymityset_{pks}\})$
12: $\quad in \leftarrow in \cup (\{output, anonymityset\}, signature)$
13: **end for**
14: **return** $(in, tospend)$

To prevent false accepting of contracts without accepting the file afterwards we require the miner to create a proof of retrievability directly after the storage process. Otherwise the storage provider could accept the contract while dropping the file instantly, locking the money of the user without cost for themselves. To provide the proof of retrievability, the storage provider has to accept the file, with the one-time cost of bandwidth and storage space.

To check that the storage provider keeps the file stored until the contract expires an additional proof of retrievability after the expiration of the contract is necessary. If the proof is not published the storage provider is not able to retrieve the payment from the storage contract. The challenge for this proof of retrievability is generated through the hash of the current block. Since the hash of the current block is not predictable it is not possible to precompute the proof.

The storage provider publishes both proofs of retrievability as transactions. Such a transaction includes a reference to the storage contract in the blockchain and the publicly verifiable proof. To incentivize miners to include these transactions in blocks the transactions are equipped with a small fee that can be spent by the miner of the block.

To reclaim the storage space occupied by proofs of retrievability the proofs themselves are not included in the computation of the hash of the transaction. Thus the proofs can be removed without changing the hashes of the blocks. It is not necessary to include the proofs since if they are incorrect the transaction would not have been added to the blockchain by miners. Further, the integrity check coming from the inclusion in the blockchain, does not provide any guarantees against

malicious modification of the proof in transit.

Algorithm 3 Storing a File

Input: User secret key $sk_{\mathcal{U}}$, anonymity set size n, file $\boldsymbol{f}$, contract duration c

Output: a storage contract between the user $\mathcal{U}$ and an anonymous storage provider, as well as a file transfer

1: $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$
▷ Generate a key pair for the proof of retrievability
2: $(f', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f})$
▷ Encode the file for the proof of retrievability
3: $(p, pk_{\mathcal{S}}) \leftarrow \text{find-SP}(\text{size}(f'), c)$
▷ Find a storage provider and receive a price p and its public key $pk_{\mathcal{S}}$, given the storage period and the size of the file
4: $tx \leftarrow \text{money-transfer}(sk_{\mathcal{U}}, pk_{\mathcal{S}}, p, n)$
▷ See Algorithm 2
5: **publish**(pk, st, ref, c, tx)
6: **transfer**(f')
▷ Transfer the encoded file to the storage provider

The one-time address ref is needed to refund the payment invested by the user if the contract is broken by the storage provider. To spend the refund, the user proves the breaking of the contract by referencing the broken storage contract and specifying, if the proof of retrievability is missing at the beginning a or the end c of the contract. The specification by the client is done to reduce validation cost, by halfing the blocks that need to be checked. We assume a period of goodwill, wherin a storage provider needs to provide a proof of retrievability, of length Δ. The contract is stored in block $\mathcal{B}_i$ and contains the sotrage duration c. We denote the start of the period, where the first proof of retrievability should have happened with a, this should be shortly after the contact, but not instantly, as the file transfer might take some time. This results in the period of blocks $\mathcal{B}_{i+a}, \ldots, \mathcal{B}_{i+(a+\Delta)}$ for the first proof of retrievability. The start of the period, where the second and last proof of retrievability should have happened, is denoted by c, as it is defined by the contract length, resulting in the blocks $\mathcal{B}_{i+c}, \ldots, \mathcal{B}_{i+(c+\Delta)}$. The verifier checks, depending on the point in time specified by the user, a for example, the blocks $\mathcal{B}_{i+a}$ through $\mathcal{B}_{i+(a+\Delta)}$ for a proof of retrievability referencing the contract. If the blocks do not contain a proof, the contract has been broken and the user addressed by ref is allowed to use the funds locked in the contract in a new transaction.

To increase safety of the storage, the file can be split by the client in multiple smaller chunks which are stored at multiple storage provider. Additionally an erasure code may be applied. This allows the retrieval of a subset of files to reconstruct the original file.

To increase privacy, a file can be encrypted with a securely guarded encryption key prior to upload. This is not enforced by our protocol to allow the storage of publicly accessible files, though encryption will be the default option in the system for privacy reasons.

4) File Retrieval: While storage contracts address the cost of storage, bandwidth is not free either. Hence, users need to pay for each file retrieval.

It is important to note that the order of payment and file transfer opens the gate to different forms of cheating. If the user pays first the storage provider has no incentive to continue by transferring the file. However, if the storage provider transfers the file first the problem is not solved either. In this case the user loses its incentive to pay the storage provider since it already received its data.

To deter both parties from cheating we make use of security deposits, as in KopperCoin [11]. The high-level idea is to bring both parties into a situation, where they have a game-theoretic disadvantage if they do not behave honestly, i.e., they lose their security deposits.

In particular, the system utilizes a payment transaction for each file retrieval, called mutual assured destruction (MAD) transaction. A MAD-transactions consists of a multi signature transaction. This is a transaction that needs to be signed by multiple parties in order to be spent.

To perform a MAD-transaction, a user $\mathcal{U}$ and storage provider $\mathcal{P}$ provide respective security deposits $D_{\mathcal{U}}$, $D_{\mathcal{P}}$ and the user additionally provides a payment p for the data. These funds are combined as inputs in a multi signature transaction which both parties, $\mathcal{U}$ and $\mathcal{P}$ need to sign to spent them. In the next step the storage provider needs to transmit the file to the user. After receiving the data the user checks the integrity of the file. If it is correct both sign the payout transaction, so the user receives its deposit $D_{\mathcal{U}}$ and the storage provider receives its deposit and the payment $D_{\mathcal{P}} + p$.

If the user or the storage provider cheats by not paying or not transferring the file, the respective other party will refuse to sign the second transaction and both parties lose their security deposits.

A graphical depiction of a MAD-transaction is provided by Figure 2.

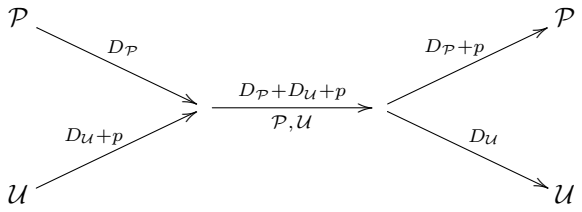

Figure 2. A MAD-transaction between a user $\mathcal{U}$ and a storage provider $\mathcal{P}$, including a payment p and deposits $D_{\mathcal{U}}$ and $D_{\mathcal{P}}$.

A full review of the lifecycle of a contract is given by Figure 3.

IV. Security and Privacy Discussion

In this section we discuss the privacy and security properties of our system. Since the specification of the network is ongoing work, we focus on privacy and security properties of the upper layers. At some parts we require certain properties of the network layer to achieve our privacy goals. We will indicate this where necessary.

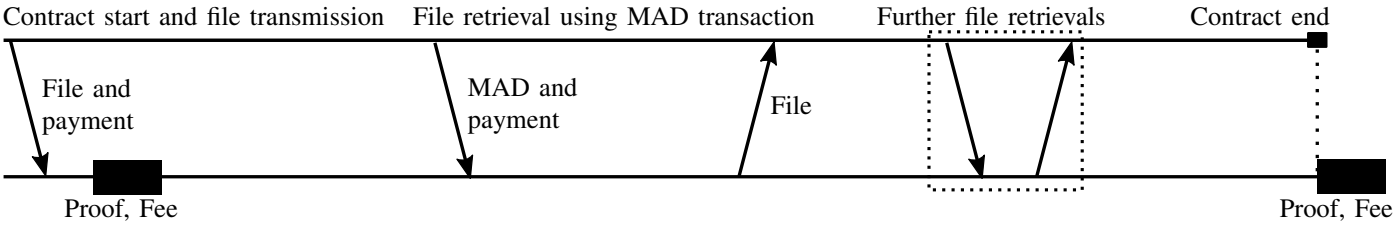

Figure 3. Visualization of the contract lifecycle and involved payments over time. Starting on the left, a user commits to a payment to the storage provider and transmits its file. The storage provider provides a proof of retrievability costing a small fee. This is repeated at the end of the contract. To retrieve the file a MAD transaction is initiated and a payment is made from the user to the storage provider.

A. Privacy Properties

In the following our attacker is a passive observer with access to all transactions in the blockchain. We organized this section according to the different entities in the system.

In a nutshell, we provide anonymity of senders and receivers in money transfers. Only the amount in the transaction is public. Regarding storage, we hide the user, the storage provider, as well as the file. Of course, the identity of a file is linkable with its corresponding proofs of retrievability, but the content of the file is never revealed to a passive observer. We assume that the client application takes care of hiding the content of the file towards a storage provider, e.g., by encrypting it.

Our system achieves that even two storage contracts with the same user, storage provider and file cannot be linked. Only the amount is known.

1) Sender Anonymity: With *sender anonymity* or *untraceability* ([21], [27]) we describe the property that the sender of a transaction is hidden.

Our scheme achieves this on the level of the blockchain, since linkable ring signatures are used when referencing previous transaction outputs. Thus, all identities included in the ring have the same probability of being the sender of the transaction [27]. In practice one may choose a ring size between two and five. Since the anonymity of a sender depends on the anonymity of other senders in the ring, care must be taken to disallow rings with only one member. Otherwise an attacker controlling a large part of the signatures could set a chain reaction in motion which can deanonymize a large part of the senders of past transactions [20]. Note that the signatures are nevertheless one-time and thus anonymity of users may not necessarily be violated.

In case of a double-spend both transactions become linked and the second is discarded by the miners. The identity of the sender is not revealed thereby, if the same set of keys is used. If the sender double-spends with two disjoint sets of other public keys in the rings, an observer learns the one-time key which is used in the double-spend. This is a one-time key and even though sender anonymity is broken, the long-term identity of the sender is not revealed.

To provide anonymity of senders of transactions we require that the network hides the senders of broadcast messages. Clearly, linking a sender of a transaction to the participant broadcasting the transaction in the network violates our privacy requirements.

2) Receiver Anonymity: The notion of *receiver anonymity* means that the identity of the receiver of a transaction is hidden. Our scheme provides *unlinkable transactions*, i.e., an observer cannot decide if any two transactions were sent to the same user. We accomplish this through the use of one-time addresses. If two amounts are sent to the same receiver the senders derive two different unlinkable one-time payment addresses from the long-time key of the receiver [19], [27]. Only the receiver is able to check if both transactions belong to him, and recover the private key which is needed to spend the transactions.

3) User Anonymity: The anonymity of the storage user $\mathcal{U}$ could potentially be violated through his actions on the blockchain, as well as through the file handling of the system.

When a user wants to store a file, a storage contract is issued on the blockchain. This contract includes her payment, i.e., references to sets of previous transactions outputs, a public key which is used for the proof of retrievability, as well as the address of the storage provider $\mathcal{P}$, and details under which conditions $\mathcal{P}$ receives the payment.

As explained above in Section IV-A1, the public key of the sender of the payment, in this case the user $\mathcal{U}$, is not revealed in a storage contract, because of the use of linkable ring signatures.

The public key used for the proof of retrievability is different from the other keys of the user and unique per file. It is needed to enable public verifiability of the proof of retrievability. The corresponding private key is needed only once to run the algorithm Encode of the proof of retrievability prior to the upload and can then be deleted. Since the key pair is unique per file it cannot be used to identify the user or link different files of the same user.

Unlinkability between the user and his files is a consequence of the anonymity of the user in the storage contract due to the linkable ring signatures. Thus if a user uploads multiple files, an observer cannot decide if they originated from the same user.

Though note that if a user creates multiple storage contracts with very unique conditions on the storage provider, these contracts may be linked as corresponding to the same user.

Since a MAD transaction is just a multi signature transaction and does not contain any references to the file, the privacy of the user is protected through the same mechanisms as in a normal transfer of funds (see Sections IV-A1 and IV-A2).

To provide our strong privacy guarantees on the network layer, we require a mechanism to hide the sender of messages when uploading the file. When the user downloads the file the network needs to hide the receiver of messages. If the network additionally provides unlinkability of message senders and receivers users and storage providers are unlinkable.

4) Storage Provider Anonymity: As with user anonymity, we need to check the storage contracts and the MAD transactions to check the anonymity of the storage provider. Regarding the published proofs of retrievability it is easy to see that these do not identify the storage provider, since they only contain a reference to the storage contract in the blockchain.

In a storage contract the identity of the sender is not revealed, since a one-time payment address is used. Our system even achieves unlinkability of two storage contracts with the same storage provider.

In a MAD transaction the anonymity of the storage provider is also preserved, again since one-time addresses are used and the security deposits are done with ring signatures.

Again we require the network to provide unlinkability of the communicating parties.

5) File Anonymity: With *file anonymity* we mean the identifiability of the file throughout its life-cycle.

From the storage contract an observer can learn an identifier for the file, which is later used in the proofs of retrievability. This is the only time this identifier is used in the blockchain.

In particular an attacker cannot link a storage contract concerning a specific file and a corresponding MAD transaction, since a MAD transaction contains no reference to the file. Hence, an observer cannot learn how often a specific file is requested in the network.

A passive attacker can neither learn the identity of a user, nor of a storage provider of a specific file, since the identities in the storage contract are hidden. The attacker can only learn the price for the storage contract on which the participants agreed.

The identifiability of a file in a network when a download request occurs depends heavily on the concrete network employed. If the packets sent in the network are kept confidential and sender and receiver are unlinkable, a passive attacker is not able to trace the file. For a non-global passive attacker a mixing system like TOR [4] could be used.

B. Security discussion

This section discusses the security properties of our scheme. We discuss only attackers specific to our scenario and will omit general issues concerning blockchain architectures, such as the necessary conditions for double-spending or assumptions on the attackers to obtain a secure consensus algorithm. For a security discussion of these general problems we refer the reader to previous literature on the subject (see, e.g., [5], [7], [8], [10], [12], [17]).

1) Denial of Service Attack: The most simple attack is a denial of service attack.

Concerning transactions, this means to repeatedly transfer money to other addresses of the same participant. Like in Bitcoin, the attacker needs to pay transaction fees to get its transactions included in the blockchain. This means that such an attack has a financial cost associated with it which provides an upper limit for the attacking capability of the attacker.

Concerning storage contracts the same reasoning holds.

An attacker can also try to include many proofs of retrievability in the blockchain, thereby increasing the size of the blockchain. This attack is prevented by the miners checking the proofs of retrievability and only including them in the blockchain if these are valid and required to fulfil a storage contract.

2) Malicious Storage Providers: Let us take a look at the potential actions of a malicious storage provider.

A malicious storage provider can offer to store a file of the user, but instead delete it. This way the data of the user is lost.

When modifying or deleting the data of the user before the storage contract has expired the storage provider is unable to compute a proof of retrievability since the challenge is not known in advance. This way, a malicious storage provider loses a potential reward from the storage contract, as well as from potential MAD transactions.

A solution against the loss of her data on the side of the user is to split the file in multiple chunks and apply an erasure code, before uploading it to multiple storage providers, since then the data can be recovered if only a sufficiently small fraction of storage providers is malicious. Though, currently there is no mechanism in our scheme to ensure that the chunks will be stored at different storage providers.

Concluding, a malicious storage provider misses a financial profit by cheating and thus will not cheat if she is rational.

3) Malicious Users: On the side of the user there is few potential for malicious actions, since the user always has to pay for using storage resources of the network.

One idea for a user to cheat is by blackmailing the storage provider in the MAD transaction. When requesting a file a user $\mathcal{U}$ can refuse to sign the second multi signature transaction. Instead $\mathcal{U}$ offers to sign a different multi signature transaction granting only a small non-negative amount ε to the storage provider $\mathcal{P}$ and the remaining amount $D_{\mathcal{P}} + D_{\mathcal{U}} + p - \varepsilon$ to $\mathcal{U}$. If the storage provider does not sign this transaction, she loses her deposit $D_{\mathcal{P}}$. On the other hand, if she signs it, she only loses $D_{\mathcal{P}} - \varepsilon$, which is less. Thus, a rational storage provider will agree to the blackmailing transaction.

On the other hand the risk of $\mathcal{U}$ when blackmailing the storage provider in such a way is losing her deposits $D_{\mathcal{U}}$.

In our system this problem of blackmailing cannot occur since in our system the storage provider $\mathcal{P}$, and not the user, sends the second multi signature transaction. Thus, the user can only accept or refuse to sign, in which case she loses her deposit. She does not have the opportunity to send a blackmailing transaction to the storage provider.

V. Conclusion

This paper presented a design for a privacy-preserving distributed storage system where the participants have financial incentives to contribute. It is based on a blockchain architecture

to handle funds and storage contracts. Storage providers can prove compliance with storage contracts by publishing proofs of retrievability, i.e., cryptographic proofs which show that they indeed provided the storage.

We have implemented large parts of the payment mechanics to test their feasibility, and are currently working on the network, as well as the file mechanics.

Privacy and security are often perceived as conflicting requirements, but as we have shown in our design, they are not. We did not have to achieve a trade-off between privacy and security, but rather between security and privacy on the one hand and usability and scalability on the other hand. Privacy and security can well be integrated into a coherent design without compromising on functionality.

REFERENCES

[1] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *Advances in Cryptology–ASIACRYPT 2009*, pages 319–333. Springer, 2009.

[2] K. Bennett, T. Stef, C. Grothoff, T. Horozov, and I. Patrascu. The gnet whitepaper. Technical report, Purdue University, 06/2002 2002.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[4] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[5] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[6] E. Fujisaki and K. Suzuki. Traceable ring signature. In *Public Key Cryptography–PKC 2007*, pages 181–200. Springer, 2007.

[7] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*, pages 281–310. Springer, 2015.

[8] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. Cryptology ePrint Archive, Report 2015/263, 2015.

[9] 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 '07, pages 584–597, New York, NY, USA, 2007. ACM.

[10] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun. Misbehavior in Bitcoin: A Study of Double-Spending and Accountability. *ACM Trans. Inf. Syst. Secur.*, 18(1):2:1–2:32, May 2015. http://doi.acm.org/10.1145/2732196.

[11] H. Kopp, C. Bösch, and F. Kargl. Koppercoin – a distributed file storage with financial incentives. In *Proceedings of the 12th International Conference on Information Security Practice and Experience, ISPEC 2016*, pages 79–93. Springer, 2016.

[12] J. A. Kroll, I. C. Davey, and E. W. Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013, 2013.

[13] J. K. Liu, V. K. Wei, and D. S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Information Security and privacy, ACISP 2004*, pages 325–335. Springer, 2004.

[14] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[15] 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, 2014*, pages 475–490. IEEE, 2014. http://cs.umd.edu/%7Eamiller/permacoin.pdf.

[16] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009. https://bitcoin.org/bitcoin.pdf.

[17] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. Cryptology ePrint Archive, Report 2015/796, 2015.

[18] S. Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015.

[19] S. Noether and S. Noether. Monero is not that mysterious. Technical report, 2014. https://lab.getmonero.org/pubs/MRL-0003.pdf.

[20] S. Noether, S. Noether, and A. Mackenzie. A note on chain reactions in traceability in cryptonote 2.0. Technical report, 2014. https://lab.getmonero.org/pubs/MRL-0001.pdf.

[21] T. Okamoto and K. Ohta. Universal electronic cash. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 324–337, London, UK, 1992. Springer-Verlag. http://dl.acm.org/citation.cfm?id=646756.705374.

[22] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Advances in Cryptology—ASIACRYPT 2001*, pages 552–565. Springer, 2001. http://dx.doi.org/10.1007/3-540-45682-1_32.

[23] D. Ron and A. Shamir. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[24] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai. Retricoin: Bitcoin based on compact proofs of retrievability. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 14:1–14:10, New York, USA, 2016. ACM.

[25] H. Shacham and B. Waters. Compact proofs of retrievability. In *Advances in Cryptology–ASIACRYPT 2008*, pages 90–107. Springer, 2008.

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

[27] N. van Saberhagen. Cryptonote v 2.0. 2013. https://cryptonote.org/whitepaper.pdf.