Cryptographic Design of PriCloud, a Privacy-preserving Decentralized Storage with Remuneration

Henning Kopp, David Mödinger, Franz J. Hauck, and Frank Kargl *Member, IEEE*

Abstract—Over the last years, demand for file hosting has sky-rocketed due to cost reductions and availability of services. However, centralized providers have a negative impact on the privacy of their users, since they are able to read and collect various data about their users and even link it to their identity via their payments. On the other hand, decentralized storage solutions like GNUnet suffer from a lack of participation by providers, since there is no feasible business model. We propose PriCloud, a decentralized storage system which allows users to pay their storage providers without sacrificing their privacy by employing anonymous storage smart contracts and private payments on a blockchain. We are able to provide privacy to the users and storage providers, and unlinkability between users and files. Our system offers decentralized file storage including strong privacy guarantees and built-in remuneration for storage providers.

Index Terms—Distributed Storage, Peer-to-Peer, Applied Cryptography, Blockchain, Privacy

1 Introduction

Cloud storage systems, such as Dropbox, are in use by the general population for years now. Although providing a financially viable service, privacy often plays only a minor role in these systems. The identity of users can be revealed by their e-mail address, payment information or IP addresses. Central providers are often incentivized to disregard privacy to improve efficiency, e.g., via file deduplication or personalized services. Another drawback of a centralised file storage is the lack of smaller competitors, creation of a single point of failure, as well as a lack of censorship resistance, as accounts can be suspended without appeal. Privacy improvements are important, as storage solutions are used to store privacy-sensible documents such as password lists, tax or banking information. Further, storage solutions may be used by whistle-blowers and journalists for critical documents.

Privacy-friendly decentralized storage solutions exist, e.g., GNUnet [1] and Freenet [2], in which the storage is provided by other participants of the network. They lack financial incentives for participants to contribute storage, leading to the so called free-riding problem: Users are consuming storage capacities but are unwilling to provide storage themselves. As payments provide privacy risks, adding a payment scheme is non-trivial.

The lack of a financially viable alternative for a decentralized storage system led to the creation of PriCloud, a novel peer-to-peer storage system, where users are remunerated for their contribution of resources [3]. PriCloud uses a blockchain-based token system to enable financial incentives for those participants who provide storage to other users. Unlike previous storage solutions, our system makes use of cryptographic techniques to enable private payments which guarantee unlinkability and untraceability of transactions. Storage contracts in the blockchain ensure that payments are only due for fulfilled storage promises, while retaining unlinkability of a stored file and its payment. Compared to centralized systems, PriCloud provides sender and receiver anonymity, censorship resistance and possiblity to mitigate a single point of failure.

1.1 Contribution

This paper is an extended version of a previous presentation of our scheme [3]. We provide a more detailed description of the linkable ring signature scheme and its security properties, as well as the security definitions of the proof of storage we use. Further, this exposition provides a more extensive comparison with related work and discusses requirements on other abstraction layers beside the blockchain layer. The contributions of our work can be summarized as follows:

- We propose a novel design for a privacy-preserving decentralized storage system which allows for privacy-preserving payments.
- Our privacy-preserving payment mechanism on the blockchain is based on the cryptographic constructions of ring signatures and one-time addresses to provide unlinkability and untraceability of transactions.
- To incentivize file storage and retrievability, we make use of proofs of storage and provide a formal analysis of the incentives for serving files.
- Finally, we discuss additional considerations for abstraction layers beside the blockchain, e.g., the network layer.

1.2 Paper Outline

Section 2 introduces the relevant notation used throughout this paper. Section 3 provides an overview of the mechanics of blockchain-based digital currencies. In Section 4, we introduce the PriCloud system focussing on the contract mechanics and privacy mechanisms for payments. Section 5

discusses differences and improvements of PriCloud compared to similar systems. The description of PriCloud focuses on the abstraction layer of blockchains, but for the system to remain private, additional requirements need to be met on other layers. These are discussed in Section 6. Section 7 concludes our work.

2 Notation

We write $a \leftarrow \mathcal{A}(x)$ to assign to a the output of running the randomized algorithm $\mathcal{A}$ on input x. With $a \leftarrow \mathcal{A}(x; r)$ we denote the deterministic result of running $\mathcal{A}$ on input x and the fixed randomness r. We say that an algorithm $\mathcal{A}$ is ppt if it runs in probabilistic polynomial time. With $\mathbb{Z}_p$ we denote the residue classes of the integers $\mathbb{Z}$ modulo $p \in \mathbb{N}$.

We say that a function f is negligible if for all positive polynomials p there is a natural number $N \in \mathbb{N}$ such that for all $n > N$ it holds that $|f(n)| < 1/p(n)$. Throughout the text, $\boldsymbol{f} \in \mathbb{Z}_B^\ell$ with $B \in \mathbb{N}$ denotes a file viewed as a vector with chunks $f_1, \ldots, f_\ell \in \mathbb{Z}_B$. With the symbol $\{0,1\}^\bullet$ we denote the Kleene closure of $\{0,1\}$, i.e., the set of arbitrarily long finite bit strings.

3 Background

The first truly decentralized payment system, not requiring a third party, was Bitcoin [4], where double spending, i.e., spending the same money twice, is prevented by a set of so-called miners voting with their computational power on the validity of transactions. This design was copied in numerous other blockchain-based decentralized payment systems [5] and is also used in our system. This chapter gives an overview of this design paradigm.

Bitcoin uses the notion of transactions, which are represented as a data structure of one or more inputs and one or more outputs. Each input refers to a previous output which is to be spent by this transaction. The output contains the public key of the receiver, i.e., its identity, and the amount of money to be transferred. The input contains a signature corresponding to the public key in the referenced output. The signature acts as a proof of possession of a secret key, and therefore authorization to spend the referenced output.

After creating a transaction the sender broadcasts it into the network. So called miners run an algorithm which is used to validate and bundle the valid transactions into so called blocks. A block contains a time stamp, a nonce and a reference to the most recent known block. A block is valid if the hash of the block is smaller than a difficulty parameter included in the previous block. Miners need to find this hash as a proof of work [6]. The difficulty is dynamically adjusted such that the expected block frequency is one block per 10 minutes, assuming no changes in the underlying hash rate. This data structure is called a blockchain and is a form of consensus for transactions.

The proof of work mechanism is used to prevent Sybil attacks [7], [8], the creation of identities to increase voting power, in the consensus protocol. Classical consensus protocols use static and known identities to prevent this attack. However, Bitcoin supports a dynamic set of participants.

In order to reward the miners for their participation a special transaction, called coinbase, is included into blocks. A coinbase transaction has no inputs but an output which grants the miner of this block a predetermined amount of Bitcoins. In order to encourage miners to persist transactions in blocks, these can be equipped with a transaction fee, which will be awarded to the miner including this transaction.

If two different new blocks are found at the same time, this situation is called a fork. In this case the miners continue to mine on one of the chains chosen at random, until one of them is longer, which is then considered the valid chain.

If blocks contain invalid transactions, e.g., they have a wrong signature, spend money which has already been spent before or from a block relying on an invalid block, miners reject that block. Consequently, miners are incentivized to persist only correct information in the blockchain.

If an adversary changes old transactions already persisted in the blockchain, the hash of the block changes thus breaking all references to this block. In order to convince the other miners of the validity of this chain, it needs to grow longer than the current chain. In a naive analysis, the adversary needs to control over 50% of the hashing power of the network to extend his chain fast enough. This is considered infeasible. The strongest attacker under which Bitcoin is secure is still subject to research.

Concluding, a blockchain is a distributed database without a trusted third party. The persisted data is replicated at each participant and a consensus protocol decides what data is persisted. Further, the incentive structure and the validation rules are built in such a way that only correct information is persisted. Applications of blockchains beyond decentralized currencies include secure and fair multi-party computations [9], [10], [11] and smart contracts [12], [13].

4 The PriCloud System

4.1 Overview

The goal of PriCloud is to provide a distributed storage system with financial rewards for its storage providers, and strong privacy protection for participating users. We implement our file storage with anonymous payments based on a blockchain enhanced with anonymised money transfers and storage contracts, so that no information is leaked by the payment method.

PriCloud supports three roles: miners, storage providers, and users. Each participant in a PriCloud network can hold one or more of these roles.

The miners function the same as described in Section 3. The blockchain's loose synchronization of time between participants is used so that storage contracts can have an expiry date.

Users set up storage smart contracts with storage providers, locking money of the user for the duration of the contract. This money can be spent by the storage provider once the contract is fulfilled or by the user, should the contract be broken. To fulfil the contract, the storage provider needs to produce a valid proof of storage of the file at expiration time. The proof of storage is required in the blockchain, as the state of contracts needs to be publicly verifiable for miners. Storage providers are not trusted by the user, but are assumed to be mostly rational due to financial incentives.

Users wanting to participate can buy PriCloud tokens (money) on an exchange, or perform any of the other roles to earn tokens. Miners or storage providers do need little setup and can be run directly, but a storage provider should own some money for security deposits in retrieval operations to prevent cheating, further explained in Section 4.5.3.

Remark 1 (Possible Extensions)**.** *PriCloud only provides the abstraction of storing and retrieving chunks of data. More advanced functionalities, e.g., file encryption or increased storage guarantees by erasure coding can be added on top of PriCloud using standard methods, some of these are discussed in Section 6.*

4.2 Threat Model

PriCloud consists of several different components, whose security analysis is available in their respective sections. In general, the actors in PriCloud, i.e., users, storage providers and miners, are assumed to exhibit rational behaviour, i.e., improve their utilisation of the system. Hereby, utilization represents uploading (for users) or earning money by storing files (for storage providers). Different components specialize this general assumption in different ways, to yield more realistic results for their use case. Especially, privacy must not be compromised by malicious actors willing to disrupt the system.

4.3 Mining

Unlike Bitcoin (cf. Section 3), each PriCloud miner includes a new one-time public key (see Section 4.4.2) for itself in every new block in order to achieve privacy for its mining rewards. Hence, it cannot be detected if different coinbase rewards belong to the same miner.

4.4 Anonymous Payments

Since the transaction data in blockchains is public, they can be analysed by anyone to allow for inferences about the actors [14], [15]. Untraceability is even one of six properties of an ideal cash system according to Okamoto and Ohta [16]. Untraceability in their interpretation means that the privacy of the user should be protected. Especially, the relationship between the user and his purchases must be unlinkable by anyone.

In CryptoNote [17], the original requirement of untraceability is split into anonymity of the sender and of the receiver. Untraceability means that for each incoming transaction all possible senders have the same probability of being the real sender. Unlinkability means that for any two transactions it is impossible to determine if they were sent to the same recipient. Similarly, we provide untraceability by linkable ring signatures, whereas unlinkability is achieved by one-time payment addresses. We combine standard mechanisms to achieve unlinkability and untraceability of the payments, which are explained in the following.

4.4.1 Linkable Ring Signatures

Ring signatures are a special kind of digital signatures due to Rivest, Shamir and Tauman [18]. A ring signature proves that a document was signed by a member of a group of signers. Since the identity of the member is not revealed, this provides a level of privacy. To create a ring signature the signer needs its own private-public key pair as well as the list of public keys of the other members in the group. In particular, no group set-up procedure is necessary in contrast to group signatures. For verification, the signature and the list of public keys of the members in the group are needed. In a nutshell, ring signatures are digital signatures where the signer is k-anonymous, i.e., indistinguishable from $k-1$ other possible signers.

We will now define linkable ring signatures and then go on to explain their usage in our system together with a concrete instantiation. We base the definition on Liu et al. [19], however they define linkability only with respect to a fixed group of public keys, whereas in our definitions and constructions linkability needs to be achieved even across different sets of members in the ring, since otherwise double spending would become possible. To prevent privacy leaks, transaction outputs are sent to one-time keys. An identity will only be revealed, if a user decides to attempt a double spend. As this is an illegal action, it is worth to identify the user.

Definition 1 (Linkable Ring Signature [19])**.** *A linkable ring signature scheme is a quadruple of algorithms,* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link})$ *such that:*

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *is a ppt algorithm. It takes a security parameter* λ *and outputs a public and secret key pair* (pk, sk).
- $\sigma \leftarrow \mathsf{Sign}(m, sk, pk_{1,\dots,n})$ *is a ppt algorithm run by the signer to create a ring signature* σ*. Its input is a message* m*, a secret key* sk*, as well as a list of public keys* $pk_{1,\dots,n}$*. In a usual invocation of the signing algorithm,* sk *corresponds to one of the public keys* $pk_{1,\dots,n}$*.*
- $b := \mathsf{Verify}(\sigma, m, pk_{1,\dots,n})$ *is a deterministic algorithm run by the verifier which returns a single bit b, where '1' indicates acceptance and '0' indicates rejection. Its input is a message* m*, a signature* σ*, and a list of public keys* $pk_{1,\dots,n}$*.*
- $\ell := \mathsf{Link}(m, m', \sigma, \sigma')$ *is a deterministic algorithm that takes two messages* m *and* m'*, together with their signatures* σ *and* σ' *and returns one bit* ℓ*, where* $\ell = 0$ *indicates that the signatures are linked, i.e. were signed by the same secret key* sk*, and* $\ell = 1$ *indicates that they are independent.*

For a linkable ring signature we require the following properties:

Correctness *guarantees that a correctly signed message will be accepted by the verification. For all security parameters* $\lambda \in \mathbb{N}$*, all* $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$ *where* $i \in \{1, \dots, n\}$ *for some* $n \in \mathbb{N}$*, all messages* m*, all* $\sigma_i \leftarrow \mathsf{Sign}(m, sk_i, pk_{1,\dots,n})$ *it holds that* $\mathsf{Verify}(\sigma_i, m, pk_{1,\dots,n}) = 1$.

Linkability *intuitively means that two messages that are signed by the same secret key will be spotted by the algorithm* Link*.* Link *will claim only with negligible probability that two signatures from different secret keys are linked. Note that this definition assumes linkability even if the signatures have been created using different sets of public keys. For all security parameters* $\lambda \in \mathbb{N}$*, all* $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$*, where* $i \in \{1, \dots, n\}$*, all* $(pk'_j, sk'_j) \leftarrow \mathsf{Gen}(1^\lambda)$*, where* $j \in \{1, \dots, \ell\}$*, and any messages* m, m'*, the term*

$$P\big[\mathsf{Link}(m, m', \mathsf{Sign}(m, sk_1, pk_{1,\dots,n}), \mathsf{Sign}(m', sk'_1, pk'_{1,\dots,\ell})) = 0\big]$$

is negligible in λ*. Intuitively this means that the algorithm* Link *has a negligible probability of marking two independent*

signatures as linked.
Additionally for all security parameters $\lambda \in \mathbb{N}$, all $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$, where $i \in \{1, \ldots, n\}$, all $(pk'_j, sk'_j) \leftarrow \mathsf{Gen}(1^\lambda)$, where $j \in \{1, \ldots, \ell\}$, and for one j it holds that $(pk'_j, sk'_j) = (pk_i, sk_i)$ for some i, and any messages m, m', the term

$$P\big[\mathsf{Link}(m, m', \mathsf{Sign}(m, sk_i, pk_{1,\ldots,n}), \mathsf{Sign}(m', sk_i, pk'_{1,\ldots,\ell})) = 1\big]$$

is negligible in λ. This means that the algorithm Link *has a negligible probability of marking two linked signatures as independent even if some ring members change.*

Signer Ambiguity *guarantees the anonymity of the real signer of the message among the other members of the ring. An algorithm $\mathcal{E}$ without any access to secret keys that tries to guess the real signer has only a negligible advantage over randomly picking one of the members in the ring. We have to take into account that some secret keys may be leaked and thus the guessing algorithm $\mathcal{E}$ could have a proportionally higher chance of success, since it can verify if one of the leaked keys has been used in the ring. If that is the case, then $\mathcal{E}$ knows the true signer. Formally, for any ppt algorithm $\mathcal{E}$, on input of any message m, any lists of n public keys $pk_{1,\ldots,n}$, any set of t corresponding secret keys $D_t = \{\hat{sk}_1, \ldots, \hat{sk}_t\}$, and any valid signature $\sigma \leftarrow \mathsf{Sign}(m, sk_j, pk_{1,\ldots,n})$ generated by the user with public key pk_j it holds that*

$$P\left[\mathcal{E}(m, pk_{1,\ldots,n}, D_t, \sigma) = pk_j\right] = \begin{cases} \in & \left(\frac{1}{n-t} - \frac{1}{Q(\lambda)}, \frac{1}{n-t} + \frac{1}{Q(\lambda)}\right), \\ & \text{if } sk_j \notin D_t \text{ and } t < n-1 \\ > & 1 - \frac{1}{Q(\lambda)}, \text{ otherwise} \end{cases}$$

for any polynomial $Q(\lambda)$.

Regarding unforgeability of the ring signature we are able to use the attacker model of Liu et al. [19]

Definition 2 (Existential unforgeability against adaptive chosen-plain-text, adaptive chosen-public-key attackers [19])**.** *Let $\mathcal{SO}(m', pk'_{1,\ldots,n'})$ be a signing oracle that accepts as inputs a message m' and a list of n public keys $pk'_{1,\ldots,n'}$, and produces a signature σ' such that $\mathsf{Verify}(\sigma', m', pk'_{1,\ldots,n'}) = 1$. A linkable ring signature scheme is called existentially unforgeable (against adaptive chosen-plaintext and adaptive chosen public-key attackers) if, for any ppt algorithm $\mathcal{A}$ with access to a signing oracle $\mathcal{SO}$ such that $(pk_{1,\ldots,n}, m, \sigma) \leftarrow \mathcal{A}^{\mathcal{SO}}(pk_{1,\ldots,n})$ for a list $pk_{1,\ldots,n} = (pk_1, \ldots, pk_n)$ of n public keys chosen by $\mathcal{A}$, its output satisfies $\mathsf{Verify}(\sigma', m', pk_{1,\ldots,n}) = 1$ only with negligible probability. Note that $(pk_{1,\ldots,n}, m, \sigma)$ should not correspond to any query-response pair previously given to the signing oracle.*

Ring signatures can be used to provide sender anonymity for transactions. Each transaction input references multiple transaction outputs of earlier transactions where one output is the real output and the others are used to hide the sender and thus increase privacy. The public keys used in the ring signature are those of the referenced outputs. The message which is signed are the transaction outputs of the new transaction. The signature is included in the input, as in other blockchain systems. To an observer the real spent output is indistinguishable from the other fake outputs.

To detect a double spends, two real references to the same output, linkable ring signatures allow to link signatures if the same key has been used to sign, rendering a second transaction invalid.

The amounts of the referenced transactions need to be equal, otherwise the real amount cannot be determined from the set of referenced transaction outputs. Therefore, our system uses standardized denominations. This is similar to banknotes and coins where the set of possible values is fixed, and these values need to be combined to pay different amounts. The granularity of the amounts impacts scalability and privacy in opposite manners, as either many outputs are required or smaller anonymity sets are created. A preliminary choice of the denominations are powers of two. A justified choice of this parameter needs further evaluation and is still subject to research. The initial anonymity sets are given by the coinbase transactions.

CryptoNote [17] and many of its derivatives like Monero or Bytecoin use a variation of the FS linkable ring signatures [20]. In contrast, our system uses a linkable variation of LWW signatures [19] which has the advantage of being significantly shorter than FS signatures. We sketch the linkable variation of the LWW scheme scheme in the following, based on the work of Shen Noether [21], [22]. The signature algorithm can be found in Figure 1.

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, in non-elliptic curve contexts denoted as pk_i. 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 $\mathcal{H}_p : \mathbb{G} \mapsto \mathbb{G}$ is a hash function. The key image is included in the signature to enable linking of the signatures, as two signatures are linked if they have the same key image. Knowledge of the key image does not reveal the signer since x is private.

Let m be the data to sign and $\mathcal{H}$ 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. To generate the signature the following values are computed:

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

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

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

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

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

The signature consists of $\sigma = (I, c_1, s_1, \cdots, s_n)$. To verify a signature, recompute the sequence $c_1, \ldots, c_{n+1}$ and checks if $c_{n+1} = c_1$. If that is the case the signature is valid.

In the signature scheme used by CryptoNote [17], the c_i are chosen randomly and appended to the signature. Our construction uses the hash function as a PRNG, so the

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ (using an elliptic curve $\mathbb{G}$ with generator G)
  1) Choose a secret key $sk = x \in \{1, \ldots, \ell\}$ uniformly at random, where ℓ is the group order of $\mathbb{G}$.
  2) Choose a point $P \in \mathbb{G}$ uniformly at random and compute the public key pk as the point $P = xG$.
  3) Return the pair $(pk, sk) = (P, x)$.
- $\sigma \leftarrow \mathsf{Sign}(m, sk, pk_{1,\ldots,n})$
  1) Parse the points $(P_1, \ldots, P_n) = pk_{1,\ldots,n}$.
  2) W.l.o.g. assume that the secret key $x = sk$ corresponds to the j-th public key $P_j = pk_j$, i.e., $pk_j = xG$.
  3) Compute the key image as $I = x\mathcal{H}_p(P_j)$, where $\mathcal{H}_p$ is a hash function returning a point on the elliptic curve.
  4) Choose α, and s_i for $i = 1 \ldots, n$, $i \neq j$ uniformly at random from the base field of the elliptic curve $\mathbb{G}$.
  5) Compute the following values.
  $$L_j = \alpha G, \qquad R_j = \alpha \mathcal{H}_p(P_j), \qquad c_{j+1} = \mathcal{H}(m, L_j, R_j)$$
  6) For all $i \in \mathbb{Z}/n\mathbb{Z}$, $i \neq j$ compute the following.
  $$L_{i+1} = s_{i+1}G + c_{i+1}P_{i+1} \qquad R_{i+1} = s_{i+1}\mathcal{H}_p(P_{i+1}) + c_{i+1}I \qquad c_{i+2} = \mathcal{H}(m, L_{i+1}, R_{i+1})$$
  7) Define $s_j = \alpha - c_j x_j \mod \ell$, where ℓ is the group order of the elliptic curve and close the ring by computing
  $$L_j = \alpha G = s_j G + c_j x_j G = s_j G + c_j P_j \qquad R_j = \alpha \mathcal{H}_p(P_j) = s_j \mathcal{H}_p(P_j) + c_j I \qquad c_{j+1} = \mathcal{H}(m, L_j, R_j)$$
  8) Return the signature $\sigma = (I, c_1, s_1, \cdots, s_n)$.
- $b := \mathsf{Verify}(\sigma, m, pk_{i \in I})$
  1) Parse σ as $(I, c_1, s_1, \cdots, s_n)$.
  2) Recompute L_i, R_i for $i \in \{1, \ldots, n\}$, as well as $c_1, \ldots, c_{n+1}$.
  3) Return 1 if $c_{n+1} = c_1$.
- $\ell := \mathsf{Link}(m, m', \sigma, \sigma')$
  1) Parse σ as $(I, c_1, s_1, \cdots, s_n)$ and σ' as $(I', c'_1, s'_1, \cdots, s'_n)$.
  2) Return $\ell = 0$ if $I = I'$. Otherwise return $\ell = 1$.

Figure 1. A variation of the LWW ring signature scheme

c_i are chosen pseudorandomly and can be reconstructed given c_1. Thus a signature in CryptoNote consists of $(I, c_1, \ldots, c_n, s_1, \ldots, s_n)$ and is therefore larger than signatures in the scheme described.

The proofs of security, linkability and signer ambiguity of the described scheme are simple modifications of the proofs found in the work of Liu et al. [19] and can be found in the works of Noether [21], [22].

When using linkable ring signatures in a blockchain system for the sender of transactions one remaining question is how to choose the other transactions in the anonymity set. If they are chosen in a bad way the privacy will be likely to suffer. Choosing the outputs in the anonymity set uniformly at random from all outputs published in the blockchain poses a risk, as older transactions are used in more ring signatures. Thus, younger transactions in the anonymity set more likely the output. Monero labs [23] deals with this issue in more depth, but also fails to provide an adequate solution. The impact on the current Monero system is analysed by Möser et al. [24]. It remains an open question which distributions are appropriate.

In summary, linkable ring signatures enforce sender anonymity, also called *untraceability* [17] of transactions as all identities included in the ring have the same probability of being the real sender of the transaction. We use a variation of LWW signatures, since these are comparatively short and allow for linking signatures even if the other public keys in the ring are different.

4.4.2 One-time Payment Addresses

The blockchain-based cryptocurrency CryptoNote [17] introduced so called *one-time payment addresses* [25] which increase privacy of the receiver by using different unlinkable recipient keys.

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 recipients 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 associated with the original public 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) public key.

More exactly, let $\mathbb{G}$ be an elliptic curve with generator G. Let $d \in \{1, \ldots, \ell\}$ be the private key of the receiver, where ℓ denotes the group order of the elliptic curve. $Q = dG$ is the long-term public key of the recipient.

To send a transaction, the sender generates a one-time recipient address as follows. First, the sender computes $P = eG$, where $e \in \{1, \ldots, \ell\}$ is chosen uniformly at random. The

sender defines the shared DH secret $c := \mathcal{H}(eQ)$ and sends his transaction to the one-time public key $Q' = Q + cG$. The transaction needs to include the additional information P which is needed for the recipient to recover the private key corresponding to Q'. Since

$$\begin{aligned} Q' =& Q + cG = dG + \mathcal{H}(eQ)G = dG + \mathcal{H}(edG)G \\ =& (d + \mathcal{H}(dP))G \end{aligned}$$

the recipient can recover the one-time private key $d + \mathcal{H}(dP)$ corresponding to Q'.

The unlinkablity of two one-time keys derives from the fact that cG is essentially a random offset which is added to the long-term public key. The security of the scheme in the sense that only the recipient can recover the private key follows from the DL-assumption.

This scheme does not allow the delegation of the checking of incoming transactions. As delegation would require providing the private key to a third party. However this third party would then be able to spend the funds. Delegation of transaction processing is desired in an audit by a third party, like a tax fraud investigation, or for low power devices.

In order to allow for the delegation of transaction checking, a scan key pair and a spend key pair can be created. The scan key pair is given to the third party which can then check (but also link) the incoming payments. For spending the transactions the spend key is still required.

Let $d \in \{1, \ldots, \ell\}$, $Q = dG$ be the private and public scan key of the recipient, and $f \in \mathbb{Z}_\ell$, $R = fG$ the private and public spend key of the recipient. To send a transaction the sender chooses $e \in \{1, \ldots, \ell\}$ at random as above and computes $P = eG$. We define the shared DH secret $c := \mathcal{H}(eQ) = \mathcal{H}(dP)$ as above. The sender now addresses his transaction to $R' = R + cG$. Again, P needs to be included as an additional information in the transaction.

In order to decide if the transaction is addressed to the recipient only the private scan key d is needed to check if $R' = R + \mathcal{H}(dP)G$. In order to spend the transaction the private spend key f is needed, as $R' = (f + c)G$ the private key corresponding to R' is $(f + c)$. Thus the checking of incoming transactions can be delegated at the cost of privacy.

To summarize, our one-time payment addresses provide *unlinkable transactions* [17] and transaction processing can be delegated, without delegating the authorization to spend.

4.5 File Storage

Our system provides storage contracts to provide storage of data. These contracts are realized as special transaction types, where additional requirements need to be fulfilled to spend the money. In order to set up a storage smart contract, a user searches for a storage provider by broadcasting a storage request. This storage request contains metadata like the file size and storage duration. The storage providers answer with their respective prices to fulfil the contract. The user then sends the file to the storage provider of choice (e.g. the cheapest). In parallel, the user creates and signs the storage contract. This contract contains the file identifier and the agreed storage duration. It is then included in the blockchain like any other transaction. The data stored by the storage provider is not published in any way.

The storage smart contract is a payment from the user to the storage provider which can only be spent by the storage provider, if it is able to prove storage of the data at the beginning and at expiry of the contract.

This is realized by a cryptographic mechanism known as proof of storage, explained in Section 4.5.2. The proofs of storage are persisted in the blockchain, where they can be verified by the miners, so that the user can spend the funds after expiry if the proofs are not provided. These payment rules are enforced by the miners due to the consensus protocol of the blockchain. That means, if miners generate blocks containing transactions where the storage provider receives its payment despite not proving storage of the file these transactions are rejected by the blockchain as long as there is an honest majority of (hashing power of) miners.

Our description up to now only covers the storage of files. There needs to be an additional mechanisms to allow users to retrieve their files, which is explained in Section 4.5.3.

4.5.1 Lifecycle of Storage Smart Contracts

To store a file, the user searches a storage provider accepting a file of the requested size for the given storage period c. Both create a contract for the agreed price that can be spend by a one-time key of the storage provider.

Next, the user generates a fresh public key pair pk and sk and encodes the file according to Section 4.5.2. The secret key sk is discarded afterwards, as knowledge of the secret key would allow creating valid proofs of storage without storing the file.

Afterwards the user publishes the final storage contract consisting of a public key pk for the proof of storage, the file identifier st, a refund address ref of the user, the storage duration c and the payment transaction tx. This storage contract will eventually be persisted in the blockchain by the miners, after verifying the contract.

In a next step, the storage provider can see the contract being persisted in the blockchain and accepts the file transfer. The algorithm to store a file is given by Algorithm 1.

Algorithm 1 Storing a File

Input: User secret key $sk_{\mathcal{U}}$, anonymity set size n, file $\boldsymbol{f}$, storage duration c, one-time public key ref of the user
Output: a storage contract between the user $\mathcal{U}$ and an anonymous storage provider, as well as a file transfer

1: $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ ▷ Key generation for the PoS.
2: $(f', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f})$ ▷ Encode for the PoS.
3: $(p, pk_{\mathcal{P}}) \leftarrow \text{find-SP}(\text{size}(f'), c)$
▷ Find a storage provider and receive a price p and its public key $pk_{\mathcal{P}}$, given the storage period and the size of the file
4: $tx \leftarrow \text{generate-transaction}(sk_{\mathcal{U}}, pk_{\mathcal{P}}, p, n)$ ▷ Cf. Sec. 4.4
5: **publish**(pk, st, ref, c, tx)
6: **transfer**(f') ▷ To the storage provider

The refund address ref of the user is necessary to refund the payment invested by the user in case the contract is broken. To spend the refund, the user proves the breach of the contract by referencing the broken storage contract. To reduce validation costs, the transaction contains a hint which proof of storage is missing: The first or the last one.

In our storage smart contracts, a storage provider needs to prove storage of the data once at the beginning of the contract and once at contract expiry as shown in Figure 2. The expiry proof of storage implies that the storage provider had access to the data for the whole duration of the contract. Precomputed proofs of storage are precluded, as the challenge is derived from a block in a timeframe dependent on the expiry date. The timeframe is necessary to deal with latency issues.

To prevent a malicious storage provider from locking funds of users for a long time, an additional proof is required in the beginning. While the storage provider can still discard the data, the storage provider had to at least store the data once. Failing to store the data at all will refund the currency of the user.

Theoretically, our design supports more complicated clauses for storage contracts than two proofs of storage. It is conceivable to design storage smart contracts where a storage provider needs to prove storage, e.g., at two out of five fixed points of time. As our design focusses on privacy, we decided against these more flexible contracts. The more unique the conditions in the smart contracts, the smaller the size of the anonymity set. An overview of the contract life cycle is given in Figure 2.

The proofs of storage transactions contain a reference to the storage contract and a transaction fee for miners, as they are published to the blockchain. To preserve storage space occupied, the proofs themselves are not included in the computation of the hash of the blocks. Thus the proofs can be removed without changing the hashes of the blocks. This means that the integrity of the proofs of storage is not guaranteed, as it is only important to guarantee their validity.

This is not a problem, as it is still impossible to switch already persisted valid proofs of storage by invalid ones, as blocks containing invalid transactions would have been discarded. Secondly, the integrity check coming from the inclusion in the blockchain does not provide any guarantees against malicious modification of the proof in transit.

4.5.2 Proofs of Storage

A proof of storage is a cryptographic proof generated by a storage provider whereby it is able to prove possession of a file. In our system, their size needs to be independent of the file size of which storage is proven, ideally it should be constant. To be used in the blockchain, proofs of storage need to be publicly verifiable. To fulfil these requirements, we use our own scheme [26]. We will now provide the formal definition and more details about publicly verifiable proofs of storage and then go on to explain its usage in our system.

Definition 3 (Proof of Storage ([27, Definition 5])). *A publicly verifiable proof of storage is a tuple of four algorithms* $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Prove}, \mathsf{Verify})$ *with the following properties:*

- $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ *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)*.*
- $(f', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f})$ *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.*
- $\pi \coloneqq \mathsf{Prove}(pk, f', c)$ *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.*
- $b \coloneqq \mathsf{Verify}(pk, st, c, \pi)$ *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 a first step the user generates a public and secret key pair. Then, the file is encoded by the user using its secret key and the file. Intuitively this step corresponds to splitting the file into chunks of size B and computing homomorphic linear authenticators over each chunk which are then appended to the file. Further, a state st is generated at the encoding step which serves as a file identifier. The encoded file f' and the file identifier st are uploaded to the storage provider. The storage provider creates a proof by running the algorithm Prove. For this, it uses the encoded file, a challenge value c, and the public key pk of the user. The challenge c is expanded by using it as a seed of a pseudorandom number generator, and the chunks of the file are homomorphically combined according to the expanded challenge. Since the encoded file consist of *homomorphic* linear authenticators, a linear combination of the linear authenticators is a valid authenticator for a linear combination of the chunks of the file. The proof of storage π consists of a linear combination of chunks and a linear combination of the authenticators. The coefficients used in the linear combination are the expanded challenge.

The algorithm Verify is used to validate such a proof π. the file identifier st, the challenge c, as well as the public key pk of the user are required. There is no secret knowledge involved and thus verification can be done by anybody.

To define the security property of proofs of storage [28], [29], we use the security definition by Ateniese et al. [27]. There, soundness is formalized using a knowledge extractor [30], [31] as in [28], [29]. In particular our definition uses so-called "witness extended emulation" [32].

Definition 4 (Security of a publicly verifiable proof of storage ([27, Definition 6])). *Let* $\Pi = (\mathsf{Gen}, \mathsf{Encode}, \mathsf{Prove}, \mathsf{Verify})$ *be a publicly verifiable proof of storage. We say that* Π *is secure if there is an expected polynomial time knowledge extractor* $\mathcal{K}$ *such that for any ppt adversary* $\mathcal{A}$ *we have that:*

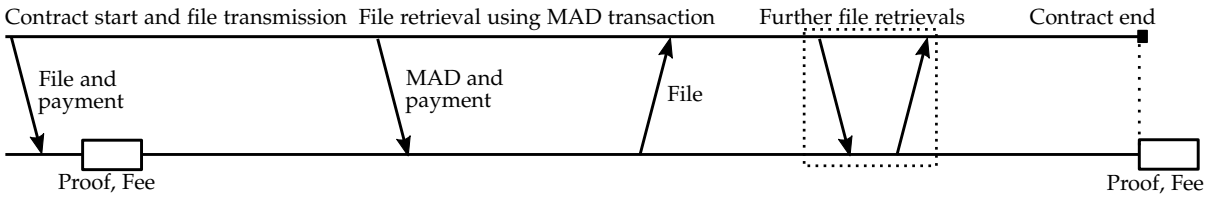

Figure 2. Visualization of the contract life cycle and payments over time. Left to right: A user commits to a payment to the storage provider and transmits its data. The storage provider provides a proof of storage which costs a small fee for the inclusion in the blockchain. This is repeated at contract expiry. To retrieve the data an additional payment is made from the user to the storage provider as will be explained in Section 4.5.3.

1) The distributions

$$\left\{ \begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k);\ (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (f', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f});\ \boldsymbol{c} \leftarrow \mathbb{Z}_p^n : \\ (\boldsymbol{c}, \mathcal{A}(st_{\mathcal{A}}, f', st, \boldsymbol{c})) \end{array} \right\}$$

and

$$\left\{ \begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k);\ (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (f', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f}) : \\ \mathcal{K}_1^{\mathcal{A}(st_{\mathcal{A}}, f', st, \cdot)}(pk, st) \end{array} \right\}$$

are identical. $\mathcal{K}_1$ denotes the first output of $\mathcal{K}$.

2) The following is negligible.

$$P\left[\begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k); \\ (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (f', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f}); \\ ((\boldsymbol{c}, \pi), \boldsymbol{f}^*) \leftarrow \mathcal{K}^{\mathcal{A}(st_{\mathcal{A}}, f', st, \cdot)}(pk, st) : \\ \mathsf{Verify}(pk, st, \boldsymbol{c}, \pi) = 1 \wedge \boldsymbol{f}^* \neq \boldsymbol{f} \end{array} \right]$$

Informally, witness-extended emulation means that given an adversary that produces a valid proof with some probability, there exist an emulator that produces a similar proof with the same probability and at the same time provides a witness, i.e., the file f'.

The first property is a restriction on the output of the knowledge extractor and guarantees that the knowledge extractor produces the same proofs as the adversary with the same probability. The second property guarantees that the knowledge extractor can provide the witness when the proof is correct, i.e., if the proof is accepted by the verifier then the prover has access to the original file.

Remark 2 (Proofs of Storage, Proofs of Retrievability, and Provable Data Possession)**.** *In the literature there are the related notions of proof of retrievability, proof of storage and provable data possession, which are used to describe similar but different concepts.*

A proof of retrievability [33], [34], [35], [36], [37], [38], [39], [40], [41] is a challenge-response protocol that proofs that a file is retrievable, i.e., recoverable without any loss or corruption. Proofs of data possession [42], [43], [44], [45], [46] are related proofs that still verify successfully if there is only a small amount of data corruption. In a proof of data possession only the existence of a knowledge extractor is required which can extract knowledge of the file, whereas a proof of storage [27], [29], [47] additionally requires the knowledge extractor to be computable in expected polynomial runtime (cf. Definition 4). Our system could apply a proof of retrievability or provable data possession, resulting in a weaker security model.

Remark 3 (What a Proof of Storage is not)**.** *A proof of storage does not guarantee* confidentiality *of the file, but this can be achieved by standard cryptographic mechanisms as explained in Section 6.*

A proof of storage does not prevent modifications or loss *of the file, but the prover can not generate further valid proofs without access to the full original file. To prevent modification or loss of data, erasure codes can be applied. The original file cannot be recovered by a proof of storage.*

A proof of storage does not guarantee file retrievability*. The storage provider can deny other participants to retrieve the file from him. There needs to be an additional incentive structure which is described in Section 4.5.3.*

For use in our system, the user generates a random key pair used to encode its file. The corresponding secret key is deleted. The public key is published in the storage contract to allow for public verification of the proofs of storage.

Up to now we described interactive proofs of storage, but actually we need a non-interactive proof of storage, since we cannot assume the existence of a trusted challenger. Using the Fiat-Shamir heuristic [48] we can substitute the challenger by a source of randomness and consequently get rid of the interaction. Concretely, the challenge c for the proof of storage in our system is chosen as the hash of a block in the blockchain at the time the proof is needed. Note that this hash is not known in advance and thus the proof of storage cannot be precomputed.

The generated proof of storage has to be persisted in the blockchain in a predefined window of blocks after the block which is used for the challenge. This is necessary to deal with forks in the blockchain and since immediate persistence of proofs in the blockchain cannot be guaranteed.

Miners do not need to persist the whole proof of storage. When a miner receives a block containing a proof of storage, it can verify the proof of storage using the public key pk which was used for encoding the file and the file identifier st, since this is included in the storage smart contract. When the proof is invalid, the block is rejected. However, if the proof is valid, the miner only needs to store that there was a valid proof of storage of that file. Since the proofs themselves are not included in the calculation of the hash of the block, they can be removed without changing the hash of the block, similar to the pruning of spent transactions in Bitcoin [4].

4.5.3 File Retrieval

Additionally to compensating storage providers for the storage of files there needs to be an incentive to allow retrieval of files.

Current protocols for exchange distinguish between strong and weak fairness [49]. In weak fairness, an honest participant can prove to an external arbiter that the other participant has received the item it expected. This could be used for a reputation system. As a reputation system is, due to the lack of identities, infeasible, PriCloud requires strong fairness. In strong fairness, at the time of protocol termination either both participants get what they want or neither of them does. It has been shown, that strong fairness of an exchange is impossible without a trusted third party [50].

Currently known protocols for simultaneous and fair exchange [51], [52], [53], [54] suffer from two main criticisms: A participant can input garbage and still receive the item [55] or the protocol can be aborted [56]. A straightforward protocol does not satisfy strong fairness, as aborting in-between exchange of payment and file, will leave one participant with the expected item, payment or file, and the other without anything.

We propose a simpler approach than Bitcoin, which leaks a preimage of a hash [57], [58], [59], borrowing the assumption of rational actors from game theory, which is non-standard in the cryptographic literature. Despite their different mechanisms, these fields have overlap in collaborative interactions between distrusting parties [60], [61], [62]. Thus, we think that our assumptions of rational actors are justified.

To deter both parties from cheating we make use of security deposits, as proposed in [63]. Our system utilizes a special transaction for each file retrieval, called *mutually assured destruction* (MAD) transaction, comparable to NASHX[1] transactions.

To perform a MAD transaction, the user $\mathcal{U}$ and a storage provider $\mathcal{P}$ provide respective security deposits $D_{\mathcal{U}}$ and $D_{\mathcal{P}}$. The user also provides a payment p for the data and cost incurred by the storage provider during the file retrieval, e.g., bandwidth. These funds are combined as inputs in a transaction which both parties, $\mathcal{U}$ and $\mathcal{P}$ need to sign to spent. In the next step the storage provider transmits the file to the user. After receiving the data, the users check the integrity of the file and if they received the correct one. To release the funds, the user and storage provider sign the payout transaction. The user and storage provider receive their deposits $D_{\mathcal{U}}$ and $D_{\mathcal{P}}$, while the storage provider also receives the payment p. A graphical depiction of this process is provided by Figure 3.

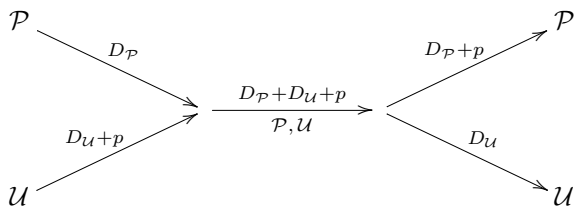

Figure 3. A MAD transaction between a user $\mathcal{U}$ and a storage provider $\mathcal{P}$, including a payment p and deposits $D_{\mathcal{U}}$ and $D_{\mathcal{P}}$.

The storage provider first creates the payout transaction and signs it. This transaction is sent to the user who broadcasts it into the network after signing herself.

1. http://nashx.com/HowItWorks

If the user or the storage provider cheat 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. The same is true if the storage provider transmits an incorrect file, as the user will not unlock the funds. This leads to the payoff matrix shown in Figure 4. In this figure, f denotes the value of the file. The two strategy pairs (cheat, cheat) where both participants cheat and (comply, comply) where both participants comply are the Nash equilibria. However (comply, comply) weakly dominates the other Nash equilibrium and thus will be chosen by rational actors.

		Storage Provider $\mathcal{P}$: comply	Storage Provider $\mathcal{P}$: cheat
User $\mathcal{U}$	comply	$f-p, p$	$-D_{\mathcal{U}}-p, -D_{\mathcal{P}}$
	cheat	$f-D_{\mathcal{U}}-p, -D_{\mathcal{P}}$	$-D_{\mathcal{U}}-p, -D_{\mathcal{P}}$

Figure 4. The payoff matrix for MAD transactions.

In theory it is possible that the user can hold the deposit of the storage provider hostage: After the second transaction and after receiving the file, the User refuses to sign the multi signature transaction and instead creates a different transaction granting a small non-negative amount ε to the storage provider. The remaining amount $D_{\mathcal{P}}+D_{\mathcal{U}}+p-\varepsilon$ is given to the user. If the storage provider signs it, only $D_{\mathcal{P}}-\varepsilon$ is lost instead of the full deposit. A rational storage provider would agree to this form of blackmailing.

In our approach, we prevent this blackmailing, since the storage provider and not the user initiates the freeing of funds. There are no further messages the software could transmit or show a user. Thus, the user can either agree on freeing the funds or lose its entire security deposit, but is unable to play a third blackmailing strategy, as there are no communication channels to communicate this blackmailing.

5 Related Work

5.1 KopperCoin

KopperCoin [63] is a previous proposal of a peer-to-peer storage with payment by us. Compared to the proposal in this article, KopperCoin has no distinction between storage providers and miners, since storing files is done by the miners. In KopperCoin the mining process of a blockchain is substituted by proofs of storage over data which has previously been uploaded by users of the system. The file over which a proof of storage needs to be computed is determined by considering the hash of the previous block in the blockchain as a file identifier. One major drawback of KopperCoin is the lack of privacy guarantees, since miners are linked to the files they store by their mining reward.

Storage in PriCloud is realized by contracts which do not interfere with the mining process. This allows implementing privacy enhancing mechanisms for storage contracts.

5.2 Permacoin and Retricoin

Permacoin [64] was the first approach to combine the ideas of decentralized file storage and blockchain. Miners prove retrievability of parts of a large digital archive where single

miners are unlikely to have the resources to store the whole archive. However, the digital archive is fixed at the time of creation of the blockchain. Users cannot upload or download data in contrast to our design. Retricoin [65] offers some efficiency improvements over Permacoin but inherits their main design decisions.

5.3 Filecoin

Filecoin is another blockchain-based decentralized storage system where users can upload and download files and pay for storage. They provide two suggestions as white papers [66], [67].

In their first white paper, the proof of work process is augmented by a proof of storage, such that there are two proofs needed to mine a block, which urges miners to be also storage providers. Files stored in Filecoin can only be used for a reward up to an expiry date. Storage providers are not incentivized to let a user access its data.

In the second white paper, the proof of storage was replaced with a proof of replication, proofing that any replica of data is stored in physically independent storage. Filecoin also includes proofs of spacetime which guarantees that data is stored *throughout a range of time*. However, only the definitions are given. Even in their dedicated technical report about proofs of replication [68] there is no construction offered. Strikingly, Chapter 4 in this report is called "Realizable Constructions (TODO)".

For storing and retrieving files, Filecoin introduces two markets. The order books are published on the blockchain, allowing the miners to automate the settlement process. However, some problems such as reordering the transactions before persisting them in the blockchain to gain a financial advantage (called frontrunning), are not addressed.

In both white papers the topic of privacy is not taken into account. In summary, the publications of the Filecoin team clearly do not constitute a working solution for the problem of decentralized storage and leave many open questions.

5.4 Anonymous Storage Systems

There are a number of anonymous storage systems like GNUnet [1], [69] or Freenet [2] which rely on voluntary contribution of storage resources and bandwidth by its participants. These systems focus on privacy. While GNUnet is geared towards anonymous and censorship-resistant file sharing, Freenet is envisioned as an anonymous replacement for the Internet, for websites only static sites are possible. Neither GNUnet nor Freenet offer remuneration for providing storage and thus their design made significantly different trade-offs. For example routing in GNUnet is based on the addresses of the files relying on a system based on a distributed hash table, similar to Kademlia [70]. In Freenet the files are replicated at each hop when they are routed and thus clusters of addresses converge. There is no single storage provider responsible for a specific file who could be remunerated. Hence, there are no storage guarantees in contrast to our system which provides financial storage guarantees. In Freenet, data which is only rarely queried slowly disappears from the system by design.

6 Additional Considerations

So far we discussed the privacy of the blockchains layer, e.g., unlinkability of transactions as they are stored in the blockchain. However, focussing on this layer is not sufficient [71]. This section discusses additional considerations on other layers for the PriCloud system to be used in practice.

6.1 Files

A truly private file storage needs to ensure a file can only be read by authorised entities. To reach this target, a strong file encryption scheme is needed. These can be provided by publicly available libraries, e.g., NaCl or dedicated software such as MiniLock. The core PriCloud system is agnostic to such systems but it is recommended to use such an addition, since otherwise the storage providers, and everyone willing to pay for the download, can read the contents.

6.2 Blockchain Transmissions

Interactions with the blockchain, such as creating transactions or disseminating new blocks, typically use a peer-to-peer network. This has consequences for the privacy: A well connected peer, or a collusion of multiple peers, can identify the originator's IP address [72]. While the originator key is protected by linkable ring signatures, IP addresses are considered personally identifiable information and can be used to track activities of users.

To prevent this identification of users, a privacy preserving broadcast mechanism is required. There are multiple solutions available for this problem, including protocols designed for blockchain broadcasts [73], [74].

6.3 File Transmissions

Contracts within the blockchain use the presented privacy mechanisms, but actual file transmission will need to traverse the network. Direct communication reveals the sender and receiver and thus violates the privacy of the user and storage provider. However, indirect communication might leak information to arbitrary participants in the network.

To preserve anonymity, a privacy preserving file transmission system is required. One possible solution is to augment contract negotiations with information for a Tor [75] rendezvous point for a hidden service, creating Tor communication tunnels for both participants. This provides sender and receiver anonymity, subject to the limitations of the Tor attacker model. A more integrated solution to the network could be modelled after decentralized storage solutions without financial incentives, e.g., GNUnet [1] or Freenet [2].

6.4 Summary

All considerations for PriCloud presented in this chapter have many solutions which are already applied in real world scenarios. Each of these solutions comes with their own advantages and drawbacks that need to be evaluated depending on the envisioned use case, especially trade-offs of privacy and speed of operations.

7 Conclusion

This article introduced PriCloud, a novel decentralized distributed storage which allows users to pay storage providers via an anonymous currency. A blockchain whose transactions have been cryptographically hardened achieves untraceability and unlinkability of payments. Storage smart contracts which determine the storage duration are included in the blockchain, thus allowing for automatic execution of payments. The storage provider can spend its payment from a contract if it produces a valid proof of storage of the file specified in the contract at some specific points in time. Hence, the storage provider only receives its payment, if it has honestly stored the file of the user. Our PriCloud system offers privacy-preserving financial rewards for participating as a storage provider and thus provides incentives for participation.

Acknowledgments

This work was funded by the Baden-Württemberg Stiftung.

References

[1] K. Bennett, T. Stef, C. Grothoff, T. Horozov, and I. Patrascu, "The gnet whitepaper," Purdue University, Technical report, 06 2002.
[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 46–66.
[3] H. Kopp, D. Mödinger, F. J. Hauck, F. Kargl, and C. Bösch, "Design of a privacy-preserving decentralized file storage with financial incentives," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. IEEE, 2017.
[4] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2009, https://bitcoin.org/bitcoin.pdf.
[5] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.
[6] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology — CRYPTO' 92*. Springer, 1993, pp. 139–147.
[7] J. R. Douceur, "The sybil attack," in *Peer-to-peer Systems*. Springer, 2002, pp. 251–260.
[8] J. Aspnes, C. Jackson, and A. Krishnamurthy, "Exposing computationally-challenged Byzantine impostors," Yale University Department of Computer Science, Tech. Rep. YALEU/DCS/TR-1332, 2005.
[9] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *IEEE Symposium on Security and Privacy 2014*. IEEE, 2014, pp. 443–458.
[10] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Advances in Cryptology–CRYPTO 2014*. Springer, 2014, pp. 421–439.
[11] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *EUROCRYPT 2016: Int. Conf. on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 705–734.
[12] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symp. on Sec. and Priv.* IEEE, 2016, pp. 839–858.
[13] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014, http://gavwood.com/paper.pdf.
[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 *Conference on Internet measurement*. ACM, 2013, pp. 127–140.
[15] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *Financial Cryptography and Data Security*. Springer, 2013, pp. 6–24.
[16] T. Okamoto and K. Ohta, "Universal electronic cash," in *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '91. Springer, 1992, pp. 324–337.
[17] N. van Saberhagen, "Cryptonote v 2.0," 2013, https://cryptonote.org/whitepaper.pdf.
[18] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology—ASIACRYPT 2001*. Springer, 2001, pp. 552–565.
[19] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," in *ACISP*. Springer, 2004, pp. 325–335.
[20] E. Fujisaki and K. Suzuki, "Traceable ring signature," in *Public Key Cryptography–PKC 2007*. Springer, 2007, pp. 181–200.
[21] S. Noether, "Ring signature confidential transactions for monero," Cryptology ePrint Archive, Report 2015/1098, 2015.
[22] S. Noether, A. Mackenzie *et al.*, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, 2016.
[23] A. Mackenzie, S. Noether, and Monero Core Team, "Improving obfuscation in the cryptonote protocol," Tech. Rep., 2015, https://lab.getmonero.org/pubs/MRL-0004.pdf.
[24] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan *et al.*, "An empirical analysis of traceability in the monero blockchain," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 143–163, 2018.
[25] S. Noether and S. Noether, "Monero is not that mysterious," Tech. Rep., 2014, https://lab.getmonero.org/pubs/MRL-0003.pdf.
[26] H. Kopp, F. Kargl, and C. Bösch, "Publicly verifiable static proofs of storage: A novel scheme and efficiency comparisons," in *Information and Communications Security*, D. Naccache, S. Xu, S. Qing, P. Samarati, G. Blanc, R. Lu, Z. Zhang, and A. Meddahi, Eds. Cham: Springer International Publishing, 2018, pp. 459–477.
[27] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Advances in Cryptology–ASIACRYPT 2009*. Springer, 2009, pp. 319–333.
[28] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS '07*. ACM, 2007, pp. 598–609.
[29] H. Shacham and B. Waters, "Compact proofs of retrievability," *Journal of cryptology*, vol. 26, no. 3, pp. 442–483, 2013.
[30] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *Advances in Cryptology–CRYPTO'92*. Springer, 1992, pp. 390–420.
[31] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *Journal of cryptology*, vol. 1, no. 2, pp. 77–94, 1988.
[32] Y. Lindell, "Parallel coin-tossing and constant-round secure two-party computation." *Journal of Cryptology*, vol. 16, no. 3, 2003.
[33] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *CCS '07*. ACM, 2007, pp. 584–597.
[34] F. Armknecht, J.-M. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter, "Outsourced proofs of retrievability," in *SIGSAC*. ACM, 2014, pp. 831–843.
[35] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *CCSW '09*. ACM, 2009, pp. 43–54.
[36] D. Cash, A. Kupcu, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," Cryptology ePrint Archive, Report 2012/550, 2012, http://eprint.iacr.org/.
[37] Y. Dodis, S. P. Vadhan, D. Wichs *et al.*, "Proofs of retrievability via hardness amplification." in *TCC*, vol. 5444. Springer, 2009, pp. 109–127.
[38] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," in *SIGSAC*. ACM, 2013, pp. 325–336.
[39] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Workshop on Security in Cloud Computing*. ACM, 2013, pp. 19–26.
[40] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *ESORICS*. Springer, 2009, pp. 355–370.
[41] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Symp. on Inf., Comp. and Comm. Sec.* ACM, 2012, pp. 79–80.
[42] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote data checking using provable data possession," *TISSEC*, vol. 14, no. 1, pp. 12:1–12:34, Jun. 2011.
[43] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *SecureComm '08*. ACM, 2008, pp. 9:1–9:10.
[44] Y. Zhang and M. Blanton, "Efficient dynamic provable possession of remote data via balanced update trees," in *ASIA CCS '13*. ACM, 2013, pp. 183–194.

[45] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "Mr-pdp: Multiple-replica provable data possession," in *ICDCS'08*. IEEE, 2008, pp. 411–420.
[46] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2231–2244, 2012.
[47] J. Xu, A. Yang, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage." *IACR Cryptology ePrint Archive*, vol. 2014, p. 395, 2014.
[48] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," pp. 186–194, 1987.
[49] N. Asokan, "Fairness in electronic commerce," Ph.D. dissertation, 1998.
[50] H. Pagnia and F. C. Gärtner, "On the impossibility of fair exchange without a trusted third party," Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, Tech. Rep., 1999.
[51] M. Blum, "How to exchange (secret) keys," *ACM Trans. Comput. Syst.*, vol. 1, no. 2, pp. 175–193, May 1983.
[52] M. Luby, S. Micali, and C. Rackoff, "How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin," in *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, ser. SFCS '83. IEEE Computer Society, 1983, pp. 11–22.
[53] T. Okamoto and K. Ohta, "How to simultaneously exchange secrets by general assumptions," in *Proceedings of the 2Nd ACM Conference on Computer and Communications Security*, ser. CCS '94. New York, NY, USA: ACM, 1994, pp. 184–192.
[54] I. B. Damgård, "Practical and provably secure release of a secret and exchange of signatures," *Journal of Cryptology*, vol. 8, no. 4, pp. 201–222, 1995.
[55] E. F. Brickell, D. Chaum, I. B. Damgård, and J. van de Graaf, "Gradual and verifiable release of a secret (extended abstract)," in *Advances in Cryptology — CRYPTO '87*, ser. Lecture Notes in Computer Science, C. Pomerance, Ed. Springer, 1988, vol. 293, pp. 156–166.
[56] T. Tedrick, "Fair exchange of secrets (extended abstract)," in *Advances in Cryptology*, ser. Lecture Notes in Computer Science. Springer, 1985, vol. 196, pp. 434–438.
[57] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Towards fairness of cryptocurrency payments," *CoRR*, vol. abs/1609.07256, 2016.
[58] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *Future Generation Computer Systems*, 2017.
[59] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," *Commun. ACM*, 2017.
[60] Y. Dodis and T. Rabin, "Cryptography and game theory," *Algorithmic Game Theory*, pp. 181–207, 2007.
[61] G. Kol and M. Naor, "Cryptography and game theory: Designing protocols for exchanging information," *Theory of Cryptography*, pp. 320–339, 2008.
[62] G. Asharov, R. Canetti, and C. Hazay, "Towards a game theoretic view of secure computation." in *EUROCRYPT*, vol. 6632. Springer, 2011, pp. 426–445.
[63] H. Kopp, C. Bösch, and F. Kargl, "Koppercoin - a distributed file storage with financial incentives," in *Information Security Practice and Experience: 12th Int. Conf., ISPEC 2016, Proceedings*. Springer, 11 2016, pp. 79–93.
[64] 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*. IEEE, 2014, pp. 475–490, http://cs.umd.edu/%7Eamiller/permacoin.pdf.
[65] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai, "Retricoin: Bitcoin based on compact proofs of retrievability," in *Proceedings of the 17th Int. Conf. on Distributed Computing and Networking*, ser. ICDCN '16. ACM, 2016, pp. 14:1–14:10.
[66] filecoin.io, "Filecoin: A Cryptocurrency Operated File Storage Network," 2014, https://filecoin.io/filecoin-jul-2014.pdf.
[67] Protocol Labs, "Filecoin: A Decentralized Storage Network," 2017, http://filecoin.io/filecoin.pdf.
[68] ——, "Proof of replication," 2017, https://filecoin.io/proof-of-replication.pdf.
[69] C. Grothoff, K. Grothoff, T. Horozov, and J. T. Lindgren, "An encoding for censorship-resistant sharing," 2003. [Online]. Available: http://www.gnu.org/software/GNUnet/
[70] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.
[71] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," *arXiv preprint arXiv:1410.6079*, 2014.
[72] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proc. of the ACM SIGSAC Conf. on Comp. and Comm. Sec. (CCS)*. ACM, 2014, pp. 15–29.
[73] S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath, "Dandelion: Redesigning the bitcoin network for anonymity," *Proc. of the ACM Measurement and Analysis of Comp. Sys. (POMACS)*, vol. 1, no. 1, pp. 22:1–22:34, Jun. 2017.
[74] D. Mödinger, H. Kopp, F. Kargl, and F. J. Hauck, "Towards enhanced network privacy for blockchains," in *Proc. of the 38th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 2018.
[75] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

Henning Kopp received his master's degree in mathematics at Technical University Kaiserslautern, Germany, in 2014. He successfully defended his PhD in 2018 at the Institute of Distributed Systems at Ulm University, Germany, in the area of applied cryptography and its intersection with computer security and privacy. Currently, he is employed at the IT Security company SCHUTZWERK GmbH.

David Mödinger received his B.S. (2012) and M.S. (2015) from Ulm University, Germany. Currently he is with the Institute of Distributed Systems at Ulm University as a research and teaching assistant, pursuing his doctorate degree.

Franz J. Hauck earned his dissertation in 1994 and his habilitation in 2001 from the University of Erlangen-Nürnberg, interrupted by a one year stay at the Vrije Universiteit Amsterdam. Since 2002 he is professor at and deputy director of the Institute of Distributed Systems at Ulm University, Germany. His research interests are middleware systems for special purposes, recently for fault tolerance, scalability, weak real-time requirements, and privacy. Prof. Hauck is a member of the ACM and the German Computer Society, GI.

Frank Kargl is the director of the Institute of Distributed Systems at Ulm University. His research interests include system security and privacy engineering and he is particularly eager to demonstrate that strong privacy guarantees do not necessarily limit system functionality. Prof. Kargl is a member of the ACM, IEEE and the German Computer Association GI.