university of groningen

University Medical Center Groningen

University of Groningen

Balancing privacy and accountability in digital payment methods using zk-SNARKs

Bontekoe, Tariq; Everts, Maarten; Peter, Andreas

Published in:
2022 19th Annual International Conference on Privacy, Security & Trust (PST)

DOI:
10.1109/PST55820.2022.9851987

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2022

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Bontekoe, T., Everts, M., & Peter, A. (2022). Balancing privacy and accountability in digital payment methods using zk-SNARKs. In *2022 19th Annual International Conference on Privacy, Security & Trust (PST)* IEEE Xplore. https://doi.org/10.1109/PST55820.2022.9851987

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 14-02-2023

Balancing privacy and accountability in digital payment methods using zk-SNARKs

Tariq Bontekoe
TNO, Unit ICT
Groningen, the Netherlands
tariq.bontekoe@tno.nl

Maarten Everts
University of Twente
Linksight
Enschede, the Netherlands
maarten.everts@utwente.nl

Andreas Peter
University of Oldenburg
Oldenburg, Germany
andreas.peter@uol.de

2022 19th Annual International Conference on Privacy, Security & Trust (PST) | 978-1-6654-7398-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/PST55820.2022.9851987

***Abstract*—In this paper we propose and implement a digital permissioned decentralized anonymous payment scheme that finds a balance between anonymity and auditability. This approach allows banks to ensure that their clients are not participating in illegal financial transactions, whilst clients stay in control over their sensitive, personal information. Existing anonymous payment schemes often provide good privacy, but only little or mostly no auditability. We provide both by extending the Zerocash zk-SNARK based approach and adding functionality that allows for customer due diligence 'at the gate'. Clients can do fully anonymous transactions up to a certain amount per time unit and larger transactions are forced to include verifiably encrypted transactions details that can only be opened by a select group of 'judges'.**

***Index Terms*—auditability, anonymous e-cash, blockchain, distributed ledger technologies, zero-knowledge proof, zk-SNARKs**

I. Introduction

While in the previous century most of our transactions were still made with cash, nowadays a large fraction of monetary transaction are made using a plastic bank card, mobile banking, or other digital transaction forms. These digital transactions give banks access to their clients' personal and sensitive information. In many cases this is undesirable and unnecessary.

A recent approach to return control from financial transactions to their clients, without requiring them to pack their wallets with cash, are so called cryptocurrencies. Of particular interest are cryptocurrencies with a strong focus on transactional privacy, so called privacy coins. Unfortunately, this focus on privacy is often in conflict with anti-money laundering (AML) regulations, which causes some regulatory bodies to take actions against privacy coins and the exchanges that list them [1].

Because of this, it might seem that most privacy coins are on the complete other side of the spectrum with respect to regulatory bodies and traditional financial institutions. However, this need not necessarily be the case. In this paper we present a privacy enhanced digital payment scheme that bridges this apparent gap.

We propose a digital payment scheme that is the first step towards an anonymous payment system that adheres to existing auditability regulations for (digital) financial transactions [2]. The proposed scheme provides complete transactional anonymity to its users, whilst also allowing for auditability. We achieve auditability through an approach based on both know-your-customer (KYC) 'at the gate' and enclosing verifiably encrypted transaction details to larger transactions. Our proposal is not a suggestion for yet another (permissionless) cryptocurrency, rather we propose a permissioned, decentralized approach backed by participating banks, with a direct connection to fiat currency.

We design our scheme in such a way, that it can be implemented on top of a permissioned blockchain. Access control and verification is provided by a group of administrators, consisting of e.g. participating banks, financial institutions and regulatory bodies.

a) Related work: Anonymous payment schemes exist in both centralised and decentralised forms. Centralised solutions [3], [4] offer anonymous transactions in which one central party, e.g. a bank, plays a key role and is in general responsible for assuring validity, thereby creating a potential single point of failure.

Decentralised schemes on the other hand do not suffer from the drawback of having a single point of failure and have gained increasing popularity recently. Examples of decentralized schemes are privacy coins such as **Monero** [5] and **Verge** [6], that achieve anonymity through e.g. traceable ring signatures [7]. A disadvantage of these schemes is that they only obfuscate part of the transaction data instead of fully hiding all transaction details.

However, schemes that hide all transaction details completely do exist. There are two well-known schemes of this type: **Zerocoin** [8] and **Zerocash** [9]. Zerocash can be seen as the more mature and more practical version of Zerocoin. Both approaches achieve their privacy through the use of zero-knowledge proofs.

Since we want to provide the highest possible level of anonymity for users of our digital payment scheme, and want to avoid the risks of a single point of failure, the zk-SNARK approach as used by Zerocash best fits our intended use. However, none of the schemes discussed above is auditable.

In recent literature we do find schemes that provide both anonymity and auditablity. One such scheme [10] is based on ring signatures, which do not fully hide all transaction

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

details. Other schemes use zk-SNARK based approaches [11], [12] for achieving auditability and anonymity. [12] focuses on decentralized identity management and therefore does not present auditability measures specific to payment systems. Our approach also differs from [11] in that we use an account-based approach to allow for AML measures that cover multiple transactions.

b) Our contribution: Because the Zerocash protocol already largely fits our problem setting, we choose to use an adaptation of this protocol as basis for our new protocol. To address the missing functionality for our use case, this paper introduces a number of contributions that are set out in more detail in Section III-A.

First, we present how to transform the decentralised anonymous payment scheme as defined in Zerocash [9] into an **account-based** and **permissioned** one. Secondly, we show how the **conversion** between existing fiat currency and its anonymous counterpart can be achieved. Thirdly, we add **auditability** functionalities to the decentralised anonymous payment scheme, without giving in on the provided level of anonymity. Auditability allows us to limit the amount of value that any user can transfer anonymously in a certain fixed time frame. In case a user wishes to spend more than this limit, he or she needs to enclose verifiably encrypted transaction details, that can be opened by a select group of judges if necessary.

Moreover, we introduce anonymous **timelocks**. Anonymity in the sense that only the sender and the receiver of a transaction are aware of the timelock being in place. A timelock can be used to make a transaction output only spendable after a given amount of time has passed.

Finally, we present the details on a **proof of concept** implementation of our scheme.

c) Structure: The remainder of this paper is structured as follows. In Section II the most relevant cryptographic building blocks are described. A description of the solution is provided in Section III. Implementation details are then discussed in Section IV. We conclude in Section V.

II. Preliminaries

The decentralized payment scheme that will be defined in the following pages makes use of existing cryptographic building blocks. In this section we explain the two most prominent building blocks: zk-SNARKs and SAVER.

a) zk-SNARKs: The most prominent cryptographic building block on our solution is the zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK), as introduced by Bitansky et al. [13]. A SNARK is a *succinct* non-interactive argument attesting to the fact the there exists a witness that evaluates a given statement to true, and moreover the prover also knows this witness. It is similar to a NIZK, in the sense that it only requires the additional extra condition of succinctness.

More precisely, one can define a zk-SNARK scheme with the following three polynomial-time algorithms, where a binary statement that we want to prove is encoded in the arithmetic circuit C.

- $\text{Setup}^{\text{zkp}}(1^\lambda, C) \rightarrow (\text{pk}, \text{vk})$ is the setup function that generates the proving pk and verifying vk key, given the security parameter and a circuit C over which proofs are to be generated.
- $\text{Prove}^{\text{zkp}}(\text{pk}, x, a) \rightarrow \pi$ constructs a zk-SNARK proof π given a proving key pk that encodes the circuit over which the proof is constructed. The function also requires the right amount of public inputs x and auxiliary inputs a.
- $\text{Verify}^{\text{zkp}}(\text{vk}, \pi, x) \rightarrow b$ verifies the proof π over the circuit that is encoded in the verifying key vk, given public inputs x. The result is $b = \texttt{true}$ if verification succeeds, otherwise $b = \texttt{false}$.

Such a zk-SNARK scheme should satisfy the four informal definitions[1] below, for any given security parameter λ. In these definitions $\mathcal{R}_C$ is defined as the set of all valid relations with respect to the circuit C.

Definition 1 (Completeness)**.** *An honest prover should be able to convince an honest verifier, given a true statement. Concretely, given* $(x, a) \in \mathcal{R}_C$*,*

$$\mathbb{P}\Big[\mathit{Verify}^{zkp}(vk, \pi, x) = 1 | (pk, vk) \leftarrow \mathit{Setup}^{zkp}(1^\lambda, C); \pi \leftarrow \mathit{Prove}^{zkp}(pk, x, a)\Big] \approx 1.$$

Definition 2 (Proof of knowledge)**.** *Any bounded prover can only convince an honest verifier, if the prover "knows" the witness for a given instance. Concretely, for every polynomial-time bounded adversary* $\mathcal{A}$*, there exists a polynomial-time bounded extractor* $\mathcal{E}_\mathcal{A}$ *with complete access to* $\mathcal{A}$*, such that*

$$\mathbb{P}\Big[(x, a) \notin \mathcal{R}_C \wedge \mathit{Verify}^{zkp}(vk, \pi, x) = 1 | (pk, vk) \leftarrow \mathit{Setup}^{zkp}(1^\lambda, C); (\pi, x) \leftarrow \mathcal{A}(pk, vk); a \leftarrow \mathcal{E}_A(pk, vk)\Big] \approx 0.$$

Definition 3 (Zero-knowledge)**.** *A proof generated by an honest prover does not leak any information other than the truth (or not) of the statement. Concretely, there exists a polynomial-time simulator* Sim^{zkp} *such that for any polynomially bounded adversary* $\mathcal{A}$ *the following holds:*

$$\mathbb{P}\Big[\mathcal{A}(pk, vk, \pi) | (pk, vk) \leftarrow \mathit{Setup}^{zkp}(1^\lambda, C); \pi \leftarrow \mathit{Prove}^{zkp}(pk, x, a) = 1\Big] \approx \mathbb{P}\Big[\mathcal{A}(pk, vk, \pi) | (pk, vk) \leftarrow \mathit{Setup}^{zkp}(1^\lambda, C); \pi \leftarrow \mathit{Sim}^{zkp}(pk, x, a) = 1\Big].$$

For the payment scheme that will be defined in the coming pages, we use the *Groth16* zk-SNARK scheme [14], due to its efficiency and availability of existing implementations.

[1]We provide the computational versions of all four statements. These can be easily converted to their perfect counterparts.

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

b) SAVER: To ensure auditability of conspicuous transactions we use a verifiable encryption scheme known as SNARK-friendly, additively-homomorphic, and verifiable encryption and decryption with rerandomization (SAVER) [15]. As far as the authors are aware, SAVER is the first succinct non-interactive argument of knowledge (SNARK)-friendly encryption scheme out there. Next to providing us with verifiable encryption, the scheme has more features. SAVER has verifiable decryption, rerandomization, and is additively homomorphic. We use and focus on verifiable encryption and decryption features. A verifiable encryption scheme is a scheme in which one can prove certain properties of a message m, when only given the encryption c of m. We use this property in our scheme, by encrypting transaction details such as address keys whilst simultaneously using these address keys in our zk-SNARK proof. The verifiable decryption property of SAVER allows us to proof valid decryption of a ciphertext without revealing the decryption key. This is especially useful in the case, that one only wants to show decryption of one message out of a large group.

SAVER builds upon the constructions of *Groth16* and only works with this and related zk-SNARK schemes. This also strengthens our choice for *Groth16* for our zk-SNARK scheme. We refer the reader to [15] for the definitions of the construction.

III. Solution construction

In order to clarify our payment scheme construction, we first explain the ideas behind it. In this section we first give an overview of all requirements that our final construction should satisfy. Subsequently, we present a detailed description of the inner workings of our construction and show how the requirements are incorporated step-by-step.

A. Requirements

Below, we present a set of nine requirements that our protocol adheres to. These requirements ensure that the solution follows the scope as set out above. In line with the definition of the decentralised payment scheme from Zerocash [9] we define the following four basis requirements.

a) Completeness: A digital payment scheme is complete when any received value that has not yet been spent can actually be spent or stored in the receiver's account balance. Moreover, it requires that any value from the account balance that was not yet spent can be used, i.e. transferred to another user.

b) Ledger indistinguishability: This requirement focuses on the fact that no new information should be leaked to an adversary from everything that is published on the distributed ledger. The term 'no new information' refers to the fact that no other information should be leaked from the publicly available values than is required for the payment scheme to work.

c) Transaction non-malleability: Transaction non-malleability requires valid transactions to be constructed in such a way that no adversary can adapt a transaction such that it is either no longer valid, or performs a different transfer than the original sender intended. To be a bit more precise, no adversary should be able to alter any of the data included in a valid transaction.

d) Balance: A digital payment system is said to be in balance, if no user can own or spend more value than what he or she received or converted from fiat currency into the system.

In addition to the requirements from Zerocash we also add five new requirements to include our own goals and research questions.

e) Access control: In a digital transaction scheme with access control, the administrators of the system should be able to decide who does and who does not get access to the transaction system. In this case, access describes the possibility to perform transactions in the system and write something on the distributed ledger or blockchain. Moreover, it should also be possible to revoke access of certain users in case this is needed. Finally, access control also dictates that only administrators determine the state of the blockchain.

f) Conversion to/from fiat currency: This requirement describes the linkage between the transfer of value in the digital transaction scheme and transfer of value using fiat currency. In order to link these two types of value, a conversion between both should be possible in the payment scheme. This conversion might be an actual conversion of one into the other, alternatively the scheme might also contain certain notes that represent fiat currency without an actual conversion taking place. In our approach we use the first option.

g) Spend limit: The spend limit requires that the payment system provides a possibility to limit the amount of value any user can spend in a certain time frame. To make this more precise, it means that the the sum of the value of all outgoing transactions in time frame T, for any user u, may not be higher then a certain predetermined limit L.

h) Auditability: This requirement is defined in coherence with the *spend limit*. In our use case, we consider auditability to be only required for transactions that surpass the spend limit. These transactions should include the important transaction details in a special field: transferred value, sender address, receiver address. This field should be encrypted as to not leak information to other users. Decryption of the field should only be possible by a select group of actors referred to as judges.

i) Timelocks: It should be possible for a sender to define a transaction with a timelock. This timelock should make sure that the receiver can only spend the output value of the transaction after the set amount of time has passed. In order to prevent side-channel attacks, it should not be visible to actors other than the sender and receiver that a timelock is in place.

B. Zerocash basis

In this section we describe the parts of Zerocash [9] that are used as a basis for our further additions and improvements. Our basis does differ from [9] in some places. We refer the reader to the original paper for a more in-depth description.

Every user u in our payment scheme has an address key pair, consisting of a public address key $\mathrm{pk}_{\mathrm{addr}}$ and a secret

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

address key sk_{addr}. A new user first generates a random sk_{addr} and computes $\text{pk}_{\text{addr}} := \text{PRF}^{\text{addr}}_{\text{sk}_{\text{addr}}}(0)$, where $\text{PRF}^{\text{addr}}_{x}$ is a pseudo random function (PRF) with seed x. Subsequently, u generates an asymmetric encryption key pair $(\text{sk}_{\text{enc}}, \text{pk}_{\text{enc}})$ for a key-private encryption scheme (IK-CCA) [16]. These keys can be derived from the secret address key using a key derivation function (KDF). Both pk_{addr} and pk_{enc} are made publicly available, the secret keys are kept hidden.

The virtual currency in our scheme is called a note and consists of a value v_{note} and an owner with pk_{addr}. Next to this, each note has a commitment $\text{cm}_{\text{note}} := \text{Comm}^{\text{note}}_{s_{\text{note}}}(\text{pk}_{\text{addr}} \| v_{\text{note}})$ computed with the secret commitment trapdoor s_{note} and a nullifier η to prevent double spending.

Each note that is in circulation has its commitment cm_{note} included as a leaf in a Merkle Tree, that we will refer to as the Note Merkle Tree. The root of this Note Merkle tree rt_{note} is included on the blockchain, thereby indirectly including all note commitments. This root gets updated for every new note that comes into circulation.

Suppose, now that user u_a want to transfer its note $\text{note}^{\text{old}} := (s^{\text{old}}_{\text{note}}, v_{\text{note}}, \text{pk}^{a}_{\text{addr}}, \text{cm}^{\text{old}}_{\text{note}})$ to user u_b. To do this, u_a should follow the following steps.

Firstly, u_a creates a new note $\text{note}^{\text{new}} := (s^{\text{new}}_{\text{note}}, v_{\text{note}}, \text{pk}^{b}_{\text{addr}}, \text{cm}^{\text{new}}_{\text{note}})$, where $s^{\text{new}}_{\text{note}}$ is a random commitment trapdoor and $\text{cm}_{\text{note}} := \text{Comm}^{\text{note}}_{s_{\text{note}}}(\text{pk}_{\text{addr}} \| v_{\text{note}})$. Subsequently, u_a computes a nullifier $\eta := \text{PRF}^{\eta}_{\text{sk}_{\text{addr}}}(\text{pos}_{\text{note}})$, where pos_{note} is the position of note^{old} in the Note Merkle tree. PRF^{η}_{x} should be a collision-resistant PRF to ensure uniqueness of the nullifier.

In order to enable u_b to spend note^{new}, the commitment inputs should be transferred to u_b, without leaking them. u_a should thus compute the ciphertext $\text{data}^{\text{new}}_{\text{note}} := \text{Enc}_{\text{pk}^{b}_{\text{enc}}}(s^{\text{new}}_{\text{note}} \| v_{\text{note}})$, where $\text{Enc}_x(m)$ is the function that encrypts the message m under the public key x.

To ensure transaction integrity, any transaction posted on the blockchain should also be signed. Therefore, another cryptographic primitive is needed: a SUF-1CMA signature scheme [17]. During the construction of a transaction, user u_a randomly samples an appropriate asymmetric signature key pair $(\text{pk}_{\text{sig}}, \text{sk}_{\text{sig}})$ for this scheme and computes $k := \text{CRH}(\text{pk}_{\text{sig}})$ (where CRH is a collision-resistant hash function) and $\kappa := \text{PRF}^{\kappa}_{\text{sk}^{a}_{\text{addr}}}(k)$. The value of κ functions as a MAC and is needed to tie the signature key pair to this particular message.

Then, u_a constructs a zk-SNARK proof π for the following statement:

"I know s^{old}_{note}, v_{note}, pk^{a}_{addr}, sk^{a}_{addr}, cm^{old}_{note}, pos_{note}, pk^{b}_{addr}, s^{new}_{note} such that the following holds:

- *The public address key of the sender matches the sender's secret address key: $pk_{addr} = PRF^{addr}_{sk_{addr}}(0)$;*
- *Both commitments are constructed correctly: $cm^{old}_{note} = Comm_{s^{old}_{note}}(pk^{a}_{addr} \| v_{note})$ and $cm^{new}_{note} = Comm^{note}_{s^{new}_{note}}(pk^{b}_{addr} \| v_{note})$;*
- *The commitment of the old note cm^{old}_{note} appears at position pos_{note} in the Note Merkle tree;*
- *The nullifier to the old note is constructed correctly: $\eta = PRF^{\eta}_{sk_{addr}}(pos_{note})$;"*
- *The public signature key is tied to this message by means of κ: $\kappa = PRF^{\kappa}_{sk_{addr}}(k)$*

Using the above steps, value is transferred from one user to another by creating a new note with the same value. The nullifier to the old note is published to ensure that the old note becomes unusable. When u_a has performed all calculations, he or she now signs the public parts $\text{tx} := (\text{cm}^{\text{new}}_{\text{note}}, \eta, \pi, \text{data}^{\text{new}}_{\text{note}}, \kappa, k)$ using sk_{sig}, thus obtaining signature σ. Finally, u_a publishes pk_{sig}, tx and σ. The admin validates the proof of the transaction and if valid adds $\text{cm}^{\text{new}}_{\text{note}}$ to the Note Merkle Tree, such that it can be spent by u_b.

C. Towards an account-based model

In this section we present an updated version of our protocol that moves the protocol from the UTXO-model towards an hybrid account-based model, loosely based on [18]. This adaption introduces new data structures that are used to implement the remaining requirements.

In our basis version the user cannot get change back from a transaction, we solve this using a new data structure: private memory cells for the storage of an account balance. These memory cells can be stored in another Merkle tree, which will be called the Memory Merkle tree. Analogously to the Note Merkle tree it consists of commitments to memory cells, which contain the user's public address key and current account balance v_{mem}: $\text{cm}_{\text{mem}} := \text{Comm}^{\text{mem}}_{s_{\text{mem}}}(\text{pk}_{\text{addr}} \| v_{\text{mem}})$, where s_{mem} is a randomly sampled commitment trapdoor.

Every time a user makes a transaction, he or she should also update his or her own memory cell in order to process the change in account balance. This change in account balance is used to balance the difference between the value of the input note and that of the output note. To prevent linkability of transactions, these updates consist of publishing a new commitment to the updated account balance and same public address key and publishing a nullifier to the old memory cell such that it becomes invalidated, i.e. it cannot be used any more. We will refer to a memory cell with mem and to the memory nullifier with μ.

This addition gives rise to several additional and altered steps in the protocol. Where an old transaction had a single input note and a single output note, a new transaction has two inputs and two outputs. These inputs are an old, unused note and the latest memory cell. The outputs are a new note and a new, updated memory cell. Next to the unspent input note $\text{note}^{\text{old}} := (s^{\text{old}}_{\text{note}}, v^{\text{old}}_{\text{note}}, \text{pk}_{\text{addr}}, \text{cm}^{\text{old}}_{\text{note}})$, the user also has the latest version of its memory cell $\text{mem}^{\text{old}} := (s^{\text{old}}_{\text{mem}}, v^{\text{mem}}_{\text{old}}, \text{pk}_{\text{addr}}, \text{cm}^{\text{old}}_{\text{mem}})$ as input to a transaction. For the construction of the new note everything stays the same except for the fact that the value of this new note $v^{\text{new}}_{\text{note}}$ can now be any positive value, as long as the account balance does not become negative. This updated account balance is calculated as $v^{\text{new}}_{\text{mem}} := v^{\text{old}}_{\text{note}} + v^{\text{old}}_{\text{mem}} - v^{\text{new}}_{\text{note}}$. The commitment to the new memory cell is then computed as $\text{cm}^{\text{new}}_{\text{mem}} := \text{Comm}^{\text{mem}}_{s^{\text{new}}_{\text{mem}}}(\text{pk}_{\text{addr}} \| v^{\text{new}}_{\text{mem}})$, where $s^{\text{new}}_{\text{mem}}$ is a new randomly sampled trapdoor and pk_{addr} is the sender's public address key. Obviously, the old memory cell should also be

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

invalidated, hence we need to calculate and publish the nullifier to this. This nullifier μ is computed analogously to the nullifier of the old note as $\mu := \text{PRF}^{\mu}_{\text{sk}_{\text{addr}}}(\text{pos}_{\text{mem}})$, where pos_{mem} is the position of the commitment to the old memory cell in the Memory Merkle tree.

Finally, the user also needs to prove in the zk-SNARK proof π that the new commitments and nullifier have been constructed correctly and that the transaction values are in balance.

Additionally to what is already included in the transaction tx in previous versions, we now also publish (and sign) μ, $\text{cm}^{\text{new}}_{\text{mem}}$ and the updated proof π.

Remembering the commitment trapdoor that is used for the new memory commitment is not very practical. Therefore, the sender can include the encrypted version of the commitment trapdoor in the transaction, by encrypting it under the sender's own public encryption key. Concretely, the field $\text{data}^{\text{new}}_{\text{mem}} := \text{Enc}_{\text{pk}_{\text{enc}}}(s^{\text{new}}_{\text{mem}})$ should be added to tx.

D. Access control

Using the previous version of our protocol, we show how to enforce the *access control* requirement in the two new protocol versions below. [2]

In order to let the admin function as a gatekeeper to the digital transaction system, we introduce one more Merkle tree: the Account Merkle Tree. This Account Merkle tree stores commitments to all the accounts that are allowed to perform transactions in the system. Specifically, it stores commitments of the form $\text{cm}_{\text{cred}} := \text{Comm}^{\text{cred}}_{s_{\text{cred}}}(\text{pk}_{\text{addr}} \| \text{sk}_{\text{addr}})$ for every accepted user u, where $(\text{pk}_{\text{addr}}, \text{sk}_{\text{addr}})$ is u's address key pair, or credentials. By giving the admin control over which credentials are allowed in the Account Merkle tree, the admin can also control which users can perform transactions and which not. Additionally, the admin is needs an asymmetric key pair for a SUF-CMA signature scheme to be able to publish and sign messages as the admin.

Implementation-wise, this means that whenever a new user u wants to join the system, u should ask an admin to get an entry in the Account Merkle Tree. To do this, u first constructs $(\text{pk}_{\text{addr}}, \text{sk}_{\text{addr}})$ and $(\text{pk}_{\text{enc}}, \text{sk}_{\text{enc}})$, samples a random s_{cred} and computes $\text{cm}_{\text{cred}} := \text{Comm}^{\text{cred}}_{s_{\text{cred}}}(\text{pk}_{\text{addr}} \| \text{sk}_{\text{addr}})$. u also constructs a zk-SNARK proof π stating: *"I know s_{cred} and sk_{addr} such that pk_{addr} and cm_{cred} are constructed correctly."*. The user u then sends cm_{cred}, pk_{addr} and π to the admin. When everything (including customer due diligence (CDD)) is correct, the admin adds the commitment cm_{cred} to the Account Merkle tree. In addition, the admin publishes a message on the blockchain, in which the addition of the commitment is stated.

The admin can revoke an account, if this is needed, removing the commitment from the Merkle Tree and publishing a similar message on the blockchain.

[2]To clarify the explanation, we present the administrator as a single entity. This is easily extended to a group of administrators.

E. Proof of access

In this version we show how a transaction sender u constructs proof of having passed access control. u has credentials $(\text{pk}_{\text{addr}}, \text{sk}_{\text{addr}})$ and credential commitment cm_{cred} present in the Account Merkle tree. The only thing that really changes in this version is the zk-SNARK proof π. Some additional statements need to be proved there. These additional statements are: *"I know pk_{addr}, sk_{addr}, cm_{cred}, and s_{cred}, such that cm_{cred} is constructed correctly and is contained in the Account Merkle tree."*

Because of the inclusion of the public address key pk_{addr} in the credential commitment, one statement of the earlier version of this protocol can be removed from the proof. This concerns the statement: *"I know pk_{addr}, sk_{addr} such that $pk_{addr} = PRF^{addr}_{sk_{addr}}(0)$."* This statement has become redundant since it is already proved when gaining access.

F. Fiat currency to anonymous notes

In this version we describe how one should actually create a new anonymous note from fiat currency.

To obtain a new note with value $v^{\text{new}}_{\text{note}}$, u pays the amount $v^{\text{new}}_{\text{note}}$ in fiat currency to the admin. Subsequently, the admin constructs a new note with value $v^{\text{new}}_{\text{note}}$ destined for the public address key pk_{addr} of the user u. Explicitly, the admin computes $\text{cm}^{\text{new}}_{\text{note}} := \text{Comm}^{\text{note}}_{s^{\text{new}}_{\text{note}}}(\text{pk}_{\text{addr}} \| v^{\text{new}}_{\text{note}})$, with $s^{\text{new}}_{\text{note}}$ a randomly sampled commitment trapdoor. The admin also computes the encrypted commitment secrets $\text{data}^{\text{new}}_{\text{note}}$, analogously to a user-to-user transfer.

In addition to these computations, the admin constructs a zk-SNARK proof π to ensure correct construction. Finally, the admin publishes $\text{cm}^{\text{new}}_{\text{note}}$, π and $\text{data}^{\text{new}}_{\text{note}}$, all signed under the admin's secret signature key to the blockchain. The user u observes this transaction, and can decrypt $\text{data}^{\text{new}}_{\text{note}}$ to ascertain that the transaction was sent to u and use this note as input for a future transaction.

G. Anonymous notes to fiat currency

The only building block that is still missing in the basis protocol, is that of converting one's anonymous notes back to fiat currency.

If u wants to convert a received note of value $v^{\text{old}}_{\text{note}}$ to fiat currency, u should perform a normal transaction to the public address key of the admin, with one simple addition. The admin should know which user sent the note. This is achieved by adding u's public address key pk_{addr} to the field with encrypted secrets as $\text{data}^{\text{old}}_{\text{note}} := \text{Enc}_{\text{pk}^{\text{old}}_{\text{enc}}}(s^{\text{old}}_{\text{note}} \| v^{\text{old}}_{\text{note}} \| \text{info})$, where info contains pk_{addr} and possibly some extra information.

The rest of the conversion is now up to the admin who finds this transaction on the blockchain. The admin calculates the nullifier η to the received note in the usual way, and then computes a zero-knowledge proof π attesting to the correct construction of the commitment and nullifier, possession of the note and presence in the Note Merkle Tree.

Subsequently, the admin publishes η, π and the admin's public address key signed under the admin's public signature key. Once this message is accepted on the blockchain, the

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

CRED (Full zk-SNARK statement for creation of credentials)
Given $(\mathrm{cm}_{\mathrm{cred}}, \mathrm{pk}_{\mathrm{addr}})$, the prover knows $(\mathrm{sk}_{\mathrm{addr}}, \mathrm{s}_{\mathrm{cred}})$, such that the following statements hold:

- Correct derivation of public address key from secret address key: $\mathrm{pk}_{\mathrm{addr}} = \mathrm{PRF}^{\mathrm{addr}}_{\mathrm{sk}_{\mathrm{addr}}}(0)$;
- Correct computation of credential commitment: $\mathrm{cm}_{\mathrm{cred}} := \mathrm{Comm}^{\mathrm{cred}}_{s_{\mathrm{cred}}}(\mathrm{pk}_{\mathrm{addr}} \| \mathrm{sk}_{\mathrm{addr}})$.

CTO (Full zk-SNARK statement of a fiat-to-note conversion)
Given $(\mathrm{cm}^{\mathrm{new}}_{\mathrm{note}})$, the prover knows $(s^{\mathrm{new}}_{\mathrm{note}}, v^{\mathrm{new}}_{\mathrm{note}}, \mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}})$, such that the following statement holds:

- Correct computation of output note commitment: $\mathrm{cm}^{\mathrm{new}}_{\mathrm{note}} = \mathrm{Comm}^{\mathrm{note}}_{s^{\mathrm{new}}_{\mathrm{note}}}(\mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}} \| v^{\mathrm{new}}_{\mathrm{note}} \| 0)$.

CFROM (Full zk-SNARK statement of a note-to-fiat conversion)
Given $(\mathrm{rt}_{\mathrm{note}}, \eta, \mathrm{pk}_{\mathrm{addr}})$, the prover knows $(\mathrm{sk}_{\mathrm{addr}}, s^{\mathrm{old}}_{\mathrm{note}}, v^{\mathrm{old}}_{\mathrm{note}}, t^{\mathrm{old}}_{\mathrm{note}}, \mathrm{cm}^{\mathrm{old}}_{\mathrm{note}}, \mathrm{path}_{\mathrm{note}}, \mathrm{pos}_{\mathrm{note}})$, such that the following statements hold:

- Correct derivation of public address key from secret address key: $\mathrm{pk}_{\mathrm{addr}} = \mathrm{PRF}^{\mathrm{addr}}_{\mathrm{sk}_{\mathrm{addr}}}(0)$;
- $(\mathrm{pos}_{\mathrm{note}}, \mathrm{path}_{\mathrm{note}})$ is the valid Merkle Tree path from $CRH^{\mathrm{note}}(\mathrm{cm}^{\mathrm{old}}_{\mathrm{note}}, t^{\mathrm{old}}_{\mathrm{note}})$ to $\mathrm{rt}_{\mathrm{note}}$;
- Correct computation of input note commitment: $\mathrm{cm}^{\mathrm{old}}_{\mathrm{note}} = \mathrm{Comm}^{\mathrm{note}}_{s^{\mathrm{old}}_{\mathrm{note}}}(\mathrm{pk}_{\mathrm{addr}} \| v^{\mathrm{old}}_{\mathrm{note}} \| 0)$;
- Correct computation of input note nullifier: $\eta = \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\mathrm{addr}}}(\mathrm{pos}_{\mathrm{note}})$.

XFER (Full zk-SNARK statement of a note-to-note transaction)
Given $(\overline{\mathrm{pk}}_{\mathrm{sndr}}, \overline{\mathrm{pk}}_{\mathrm{rcvr}}, \overline{v}_{\mathrm{xfer}}, \mathrm{rt}_{\mathrm{mem}}, \mathrm{rt}_{\mathrm{note}}, \mathrm{rt}_{\mathrm{cred}}, k, \kappa, \eta, \mu, \mathrm{cm}^{\mathrm{new}}_{\mathrm{note}}, \mathrm{cm}^{\mathrm{new}}_{\mathrm{mem}}, t^{\mathrm{new}})$, the prover knows $(\mathrm{rt}^{\mathrm{real}}_{\mathrm{mem}}, \mathrm{rt}^{\mathrm{int}}_{\mathrm{mem}}, \mathrm{rt}^{\mathrm{int}}_{\mathrm{note}}, \eta^{\mathrm{int}}, \mathrm{pk}_{\mathrm{addr}}, \mathrm{sk}_{\mathrm{addr}}, s_{\mathrm{cred}}, \mathrm{cm}_{\mathrm{cred}}, v^{\mathrm{old}}_{\mathrm{note}}, s^{\mathrm{old}}_{\mathrm{note}}, t^{\mathrm{old}}_{\delta}, \mathrm{cm}^{\mathrm{old}}_{\mathrm{note}}, t^{\mathrm{old}}_{\mathrm{note}}, v^{\mathrm{old}}_{\mathrm{mem}}, s^{\mathrm{old}}_{\mathrm{mem}}, c^{\mathrm{old}}, \mathrm{cm}^{\mathrm{old}}_{\mathrm{mem}}, t^{\mathrm{old}}_{\mathrm{mem}}, v^{\mathrm{ceil}}_{\mathrm{mem}}, s^{\mathrm{ceil}}_{\mathrm{mem}}, c^{\mathrm{ceil}}, \mathrm{cm}^{\mathrm{ceil}}_{\mathrm{mem}}, t^{\mathrm{ceil}}_{\mathrm{mem}}, \mathrm{path}_{\mathrm{cred}}, \mathrm{path}_{\mathrm{note}}, \mathrm{path}_{\mathrm{mem}}, \mathrm{path}_{\mathrm{ceil}}, \mathrm{pos}_{\mathrm{note}}, \mathrm{pos}_{\mathrm{mem}}, \mathrm{pos}_{\mathrm{cred}}, \mathrm{pos}_{\mathrm{ceil}}, s^{\mathrm{new}}_{\mathrm{note}}, s^{\mathrm{new}}_{\mathrm{mem}}, v^{\mathrm{new}}_{\mathrm{note}}, v^{\mathrm{new}}_{\mathrm{mem}}, c^{\mathrm{new}}, \mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}}, b_{\mathrm{note}}, b_{\mathrm{mem}}, b_{\mathrm{saver}}, t_{\delta})$, such that the following statements hold:

- Correct computation of credential commitment: $\mathrm{cm}_{\mathrm{cred}} := \mathrm{Comm}^{\mathrm{cred}}_{s_{\mathrm{cred}}}(\mathrm{pk}_{\mathrm{addr}} \| \mathrm{sk}_{\mathrm{addr}})$;
- Correct computation of input note commitment: $\mathrm{cm}^{\mathrm{old}}_{\mathrm{note}} = \mathrm{Comm}^{\mathrm{note}}_{s^{\mathrm{old}}_{\mathrm{note}}}(\mathrm{pk}_{\mathrm{addr}} \| v^{\mathrm{old}}_{\mathrm{note}} \| t^{\mathrm{old}}_{\delta})$;
- Correct computation of previous memory cell commitment: $\mathrm{cm}^{\mathrm{old}}_{\mathrm{mem}} := \mathrm{Comm}^{\mathrm{mem}}_{s^{\mathrm{old}}_{\mathrm{mem}}}(\mathrm{pk}_{\mathrm{addr}} \| v^{\mathrm{old}}_{\mathrm{mem}} \| c^{\mathrm{old}})$;
- Correct computation of ceiling memory cell commitment: $\mathrm{cm}^{\mathrm{ceil}}_{\mathrm{mem}} := \mathrm{Comm}^{\mathrm{mem}}_{s^{\mathrm{ceil}}_{\mathrm{mem}}}(\mathrm{pk}_{\mathrm{addr}} \| v^{\mathrm{ceil}}_{\mathrm{mem}} \| c^{\mathrm{ceil}})$;
- $(\mathrm{pos}_{\mathrm{cred}}, \mathrm{path}_{\mathrm{cred}})$ is the valid Merkle Tree path from $\mathrm{cm}_{\mathrm{cred}}$ to $\mathrm{rt}_{\mathrm{cred}}$;
- $(\mathrm{pos}_{\mathrm{note}}, \mathrm{path}_{\mathrm{note}})$ is the valid Merkle Tree path from $\mathrm{CRH}^{\mathrm{note}}(\mathrm{cm}^{\mathrm{old}}_{\mathrm{note}}, t^{\mathrm{old}}_{\mathrm{note}})$ to $\mathrm{rt}^{\mathrm{int}}_{\mathrm{note}}$;
- $(\mathrm{pos}_{\mathrm{mem}}, \mathrm{path}_{\mathrm{mem}})$ is the valid Merkle Tree path from $\mathrm{CRH}^{\mathrm{mem}}(\mathrm{cm}^{\mathrm{old}}_{\mathrm{mem}}, t^{\mathrm{old}}_{\mathrm{mem}})$ to $\mathrm{rt}^{\mathrm{int}}_{\mathrm{mem}}$;
- $(\mathrm{pos}_{\mathrm{ceil}}, \mathrm{path}_{\mathrm{ceil}})$ is the valid Merkle Tree path from $\mathrm{CRH}^{\mathrm{mem}}(\mathrm{cm}^{\mathrm{ceil}}_{\mathrm{mem}}, t^{\mathrm{ceil}}_{\mathrm{mem}})$ to $\mathrm{rt}^{\mathrm{int}}_{\mathrm{mem}}$ `or` $c^{\mathrm{ceil}} = 0$;
- Correct computation of previous memory cell nullifier: $\mu = \mathrm{PRF}^{\mu}_{\mathrm{sk}_{\mathrm{addr}}}(\mathrm{pos}_{\mathrm{mem}})$;
- Correct computation of input note nullifier: $\eta^{\mathrm{int}} = \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\mathrm{addr}}}(\mathrm{pos}_{\mathrm{note}})$;
- If there is no previous memory cell set the related values accordingly: `if` $b_{\mathrm{mem}} =$ `false, then` $(v^{\mathrm{old}}_{\mathrm{mem}} = 0$ `and` $\mathrm{pos}_{\mathrm{mem}} = -1$ `and` $\mathrm{rt}_{\mathrm{mem}} = \mathrm{rt}^{\mathrm{real}}_{\mathrm{mem}}$ `and` $c^{\mathrm{ceil}} = 0$ `and` $c^{\mathrm{old}} = 0)$, `else if` $b_{\mathrm{mem}} =$ `true, then` $\mathrm{rt}_{\mathrm{mem}} = \mathrm{rt}^{\mathrm{int}}_{\mathrm{mem}}$;
- If there is no input note set the related values accordingly: `if` $b_{\mathrm{note}} =$ `false, then` $(v^{\mathrm{old}}_{\mathrm{note}} = 0$ `and` $\eta = 0$ `and` $\mathrm{rt}_{\mathrm{note}} = 0)$, `else if` $b_{\mathrm{note}} =$ `true, then` $(\eta = \eta^{\mathrm{int}}$ `and` $\mathrm{rt}_{\mathrm{note}} = \mathrm{rt}^{\mathrm{int}}_{\mathrm{note}})$;
- Correct calculation of new account balance in memory cell: $v^{\mathrm{new}}_{\mathrm{mem}} = v^{\mathrm{old}}_{\mathrm{note}} + v^{\mathrm{old}}_{\mathrm{mem}} - v^{\mathrm{new}}_{\mathrm{note}}$;
- The new total outgoing value is correct: $c^{\mathrm{new}} = c^{\mathrm{old}} + v^{\mathrm{new}}_{\mathrm{note}} \cdot (1 - b_{\mathrm{saver}})$;
- The output note value and new account balance lie in the correct range: $0 \le v^{\mathrm{new}}_{\mathrm{mem}}, v^{\mathrm{new}}_{\mathrm{note}} \le v_{\max}$;
- The transaction is not an empty transaction: $0 < v^{\mathrm{new}}_{\mathrm{note}} + v^{\mathrm{old}}_{\mathrm{note}}$.
- The ceiling memory cell is old enough: $t^{\mathrm{ceil}}_{\mathrm{mem}} < t^{\mathrm{new}} - T$;
- The spend limit is not surpassed: $c^{\mathrm{new}} - c^{\mathrm{ceil}} \le L$ `or` $b_{\mathrm{saver}} =$ `true`;
- The note is unlocked: $t^{\mathrm{old}}_{\mathrm{note}} + t^{\mathrm{old}}_{\delta} \le t^{\mathrm{new}}$.
- Correct computation of output note commitment: $\mathrm{cm}^{\mathrm{new}}_{\mathrm{note}} = \mathrm{Comm}^{\mathrm{note}}_{s^{\mathrm{new}}_{\mathrm{note}}}(\mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}} \| v^{\mathrm{new}}_{\mathrm{note}} \| t_{\delta})$;
- Correct computation of new memory cell commitment: $\mathrm{cm}^{\mathrm{new}}_{\mathrm{mem}} := \mathrm{Comm}^{\mathrm{mem}}_{s^{\mathrm{new}}_{\mathrm{mem}}}(\mathrm{pk}_{\mathrm{addr}} \| v^{\mathrm{new}}_{\mathrm{mem}} \| c^{\mathrm{new}})$;
- The public signature key is tied to this message with the sender's secret address key by means of κ: $\kappa = \mathrm{PRF}^{\kappa}_{\mathrm{sk}_{\mathrm{addr}}}(k)$;
- The saver encrypted values are defined correctly if $b_{\mathrm{saver}} =$ `true`: $\overline{\mathrm{pk}}_{\mathrm{sndr}} = \mathrm{pk}_{\mathrm{addr}} \cdot b_{\mathrm{saver}}$, $\overline{\mathrm{pk}}_{\mathrm{rcvr}} = \mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}} \cdot b_{\mathrm{saver}}$, and $\overline{v}_{\mathrm{xfer}} = v^{\mathrm{new}}_{\mathrm{note}} \cdot b_{\mathrm{saver}}$;
- $b_{\mathrm{note}}, b_{\mathrm{saver}}, b_{\mathrm{mem}} \in \{0, 1\}$.

Fig. 1. zk-SNARK statements

admin transfers the note value $v^{\mathrm{old}}_{\mathrm{note}}$ in fiat currency to u in the regular fashion.

H. Spend limit

The only algorithm that really needs adaptations is that of a regular transfer transaction, since this is the type of transaction we actually want to limit.

Say we want to limit the amount that any user can spend during any time span of size T to L. In that case, any sender u should prove that the value of u's new transaction $v^{\mathrm{note}}_{\mathrm{new}}$ plus the amount of all transactions u spent between now (t_{now}) and T time ago ($t_{\mathrm{now}} - T$) is not more than L. In order to prove this, u needs to add some more information to each memory cell and also make use of another old memory cell. We will denote this other old memory cell as the 'ceiling' memory cell $\mathrm{mem}_{\mathrm{ceil}}$.

Each memory cell gets the following additional fields: c for the total amount of value transferred anonymously up to and including the transaction that created this memory cell and t for the time this memory cell got updated. We add c to the commitment of the memory cell: $\mathrm{cm}_{\mathrm{mem}} = \mathrm{Comm}^{\mathrm{mem}}_{s_{\mathrm{mem}}}(\mathrm{pk}_{\mathrm{addr}} \| v_{\mathrm{mem}} \| c)$. On the other hand we add t to the Memory Merkle tree. Explicitly, we compute the Merkle leaf as $\mathrm{CRH}^{\mathrm{mem}}(\mathrm{cm}_{\mathrm{mem}} \| t)$ using a new hash function $\mathrm{CRH}^{\mathrm{mem}}$, instead of adding only the commitment.

When a user u wants to send a new note with value $v^{\mathrm{new}}_{\mathrm{note}}$ at time t^{new}, u needs to have two memory cells: the latest memory cell $\mathrm{mem}^{\mathrm{old}} = (s^{\mathrm{old}}_{\mathrm{mem}}, v^{\mathrm{old}}_{\mathrm{mem}}, \mathrm{pk}_{\mathrm{addr}}, \mathrm{cm}^{\mathrm{old}}_{\mathrm{mem}}, c^{\mathrm{old}}, t^{\mathrm{old}})$ and a memory cell $\mathrm{mem}^{\mathrm{ceil}} = (s^{\mathrm{ceil}}_{\mathrm{mem}}, v^{\mathrm{ceil}}_{\mathrm{mem}}, \mathrm{pk}_{\mathrm{addr}}, \mathrm{cm}^{\mathrm{ceil}}_{\mathrm{mem}}, c^{\mathrm{ceil}}, t^{\mathrm{ceil}})$, with $t^{\mathrm{ceil}} < t^{\mathrm{new}} - T$. Also, u needs to ensure that $c^{\mathrm{old}} - c^{\mathrm{ceil}} + v^{\mathrm{new}}_{\mathrm{note}} \le L$.

In this version of the protocol a memory cell is computed as $\mathrm{cm}^{\mathrm{new}}_{\mathrm{mem}} := \mathrm{Comm}^{\mathrm{mem}}_{s^{\mathrm{new}}_{\mathrm{mem}}}(\mathrm{pk}_{\mathrm{addr}} \| v^{\mathrm{new}}_{\mathrm{mem}} \| c^{\mathrm{new}})$, with $c^{\mathrm{new}} := c^{\mathrm{old}} + v^{\mathrm{new}}_{\mathrm{note}}$. Finally, u can construct the zk-SNARK proof π, proving the same statements as before, with the altered

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

commitment computations, and also proving satisfaction of limits with respect to T and L and correct computation of c.

I. Auditability: Enclosing encrypted transaction details

When a user u wants to send a transaction tx that does not adhere to the spend limit, he should be forced to include the transaction details of tx in such a way that a trusted authority can view these details in case of doubts regarding u's intentions. However, these details should not be visible to any other party. We propose that u encrypts the transaction details using SAVER and the public key pk_{svr} of the trusted authority.

Specifically, this means that during the setup of the payment scheme the trusted authority should generate a SAVER key triplet $(\text{pk}_{\text{svr}}, \text{vk}_{\text{svr}}, \text{sk}_{\text{svr}})$. The public key pk_{svr} and verification key vk_{svr} should be made publicly available. The secret SAVER key sk_{svr} should be kept secret at all times.

When u surpasses the spend limit on a transaction tx, the following fields: (1) sender (u) public key $\overline{\text{pk}}_{\text{sndr}} := \text{pk}_{\text{addr}}$; (2) receiver's public key $\overline{\text{pk}}_{\text{rcvr}} := \text{pk}^{\text{new}}_{\text{addr}}$; and (3) transferred value $\overline{v}_{\text{xfer}} := v^{\text{new}}_{\text{note}}$. If u does not surpass the spend limit, he should define these values to 0.

We also need to ensure that a transaction with valid enclosed details does not count towards the spend limit. Therefore, the user u should set $c^{\text{new}} := c^{\text{old}}$, instead of $c^{\text{old}} + v^{\text{new}}_{\text{note}}$, when u's encrypted details are added to the transaction. The zk-SNARK proof π should be updated to include statements regarding all the above mentioned restrictions.

With this new structure, the trusted authority can decrypt the contents of the data_{tx} field of a questionable transaction tx on the blockchain, whenever this is deemed necessary. This decryption is performed using the authority's secret key sk^{svr} and verification key vk^{svr}. These keys are used as input for the decryption algorithm Dec^{svr}, which returns the sender and receiver public key, as well as the transferred value in readable form. Next to this, the algorithm constructs a decryption proof ν which will come in handy in case of transferring the results to the relevant legal authority if this is required. Namely, ν proofs correctness of these values without having to hand over sk^{svr}, thereby not giving the legal authority free play to decrypt any message.

J. Key sharing over N judges

The single trusted authority can be replaced by a group of N judges.This can be achieved by executing the key generation $\text{KeyGen}^{\text{svr}}$ in a distributed setting. A distributed key generation protocol (in the honest-but-curious setting) can be constructed using straightforward secure multi-party computation techniques and are therefore not included in this paper.

K. Timelocks

The final version update of our protocol implements the only missing requirement: anonymous *timelocks*.

Suppose that the sender of a transaction with transaction time t_{new}, wants to lock the output note for time t_δ. In other words, the sender creates a transaction at time t_{new} with output note note_{new} and makes the note only spendable at time $t_{\text{new}} + t_\delta$ or later.

We can achieve this by slightly adapting the Note Merkle tree. Namely, we can alter the leaf of a Note Merkle tree to have the value of the hash of the transaction time together with the commitment to the new note $\text{cm}^{\text{new}}_{\text{note}}$. We call this hash function CRH^{mem}.

The sender should also add the value t_δ to both $\text{cm}^{\text{new}}_{\text{note}}$ and $\text{data}_{\text{note}}$. Thus each note commitment is now constructed as $\text{cm}_{\text{note}} := \text{Comm}^{\text{note}}_{s_{\text{note}}}(pk_{\text{addr}} \| v_{\text{note}} \| t_\delta)$. Similarly, the encrypted note data is now computed as $\text{data}_{\text{note}} := \text{Enc}_{\text{pk}_{\text{enc}}}(s_{\text{note}} \| v_{\text{note}} \| t_\delta \| \text{info})$.

When t_δ has passed and the receiver of note_{new} wants to spend this note in a new transaction, he or she also needs to prove that the note is allowed to be spent. For this we add an additional statement to the zk-SNARK proof π.

Together, these steps allow a sender to lock an output note for a certain amount of time. In the case that the sender of a transaction does not want to lock the output note for a certain amount of time, he or she simply sets $t_\delta := 0$. Furthermore, we force users and administrators to set $t_\delta := 0$ on creating a note as input for one of the conversion transactions, by fixing this in the respective proofs π.

L. Solution Definition

The algorithms and zk-SNARK statements that comprise the protocol as described above are defined explicitly in Figure 1, Figure 2, and Figure 3.

IV. Implementation

In this section we explain the concrete choices made for our cryptographic building blocks that provide a security level of at least 128 bits. These building blocks were used in a proof of concept implementation[3] of our solution. The performance of this implementation is also briefly discussed.

a) zk-SNARKs: We use the BLS12-381 elliptic curve for implementing our zk-SNARKs, following the *Groth16* scheme [14]. This curve is chosen due to its efficiency and provides us with the possibility of implementing elliptic curve arithmetic in an arithmetic circuit efficiently on an embedded curve inside BLS12-381, otherwise known as *Jubjub* [19].

b) CRH, PRF, COMM and Merkle trees: We define our Note and Memory Merkle trees to have depth 32, this allows for 2^{32} leafs, which should be more than sufficient for the foreseeable future. The position of a commitment in one of these Merkle tree can be represented by a 33-bit number (32 bits for the position and 1 bit extra for the case of using -1 as the position). The Credential Merkle tree has depth 20.

We use Blake2s [20] with a 512 bit input and 256 bits output as basis for the PRF, giving us the desired security properties. The Blake2s function is denoted as $\mathcal{H}$.

A Pedersen hash function is used as basis for the commitment and hash functions, due to its higher efficiency in zk-SNARK schemes [21].

[3]https://github.com/TariqTNO/anonymous-transactions

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

Setup
Input: security parameter λ
Output: public parameters pp

(1) Construct the arithmetic circuits C_{XFER},C_{CRED}, C_{CTO}, C_{CFROM} at security level λ.
(2) Generate the proving and verification key pairs for all four circuits $(\mathrm{pk}_{\mathsf{XFER}}, \mathrm{vk}_{\mathsf{XFER}}) := \mathrm{Setup}^{\mathrm{svr}}(1^\lambda, C_{\mathsf{XFER}}))$, $(\mathrm{pk}_{\mathsf{CRED}}, \mathrm{vk}_{\mathsf{CRED}}) := \mathrm{Setup}^{\mathrm{zkp}}(1^\lambda, C_{\mathsf{CRED}}))$, $(\mathrm{pk}_{\mathsf{CTO}}, \mathrm{vk}_{\mathsf{CTO}}) := \mathrm{Setup}^{\mathrm{zkp}}(1^\lambda, C_{\mathsf{CTO}}))$, and $(\mathrm{pk}_{\mathsf{CFROM}}, \mathrm{vk}_{\mathsf{CFROM}}) := \mathrm{Setup}^{\mathrm{zkp}}(1^\lambda, C_{\mathsf{CFROM}}))$.
(3) The admin(s) together compute $(\mathrm{pk}_{\mathrm{svr}}, \mathrm{sk}_{\mathrm{svr}}, \mathrm{vk}_{\mathrm{svr}}) := \mathrm{KeyGen}^{\mathrm{svr}}(\mathrm{pk}_{\mathsf{XFER}}, \mathrm{vk}_{\mathsf{XFER}})$ and store their individual shares of $\mathrm{sk}_{\mathrm{svr}}$.
(4) Create the public parameters for the encryption scheme $\mathrm{pp}_{\mathrm{enc}} := \mathrm{Setup}^{\mathrm{enc}}(1^\lambda)$.
(5) Create the public parameters for the signature scheme $\mathrm{pp}_{\mathrm{sig}} := \mathrm{Setup}^{\mathrm{sig}}(1^\lambda)$.
(6) Create the public parameters for the admin signature scheme $\mathrm{pp}_{\mathrm{asig}} := \mathrm{Setup}^{\mathrm{asig}}(1^\lambda)$.
(7) Set $\mathrm{pp} := (\mathrm{pk}_{\mathsf{XFER}}, \mathrm{vk}_{\mathsf{XFER}}, \mathrm{pk}_{\mathsf{CRED}}, \mathrm{vk}_{\mathsf{CRED}}, \mathrm{pk}_{\mathsf{CTO}}, \mathrm{vk}_{\mathsf{CTO}}, \mathrm{pk}_{\mathsf{CFROM}}, \mathrm{vk}_{\mathsf{CFROM}}, \mathrm{pp}_{\mathrm{enc}}, \mathrm{pp}_{\mathrm{sig}}, \mathrm{pp}_{\mathrm{asig}}, \mathrm{pk}_{\mathrm{svr}}, \mathrm{vk}_{\mathrm{svr}})$.
(8) Each admin calls **AddAdmin**(λ.pp)
(9) Publish pp.

AddAdmin
Input: security parameters λ; public parameters pp
Output: admin credentials $\mathrm{cred}_{\mathrm{adm}}$

(1) Parse pp as $(\mathrm{pp}_{\mathrm{enc}}, \mathrm{pp}_{\mathrm{asig}}, *)$
(2) Generate the public and secret key for the admin signature scheme as $(\mathrm{pk}_{\mathrm{asig}}, \mathrm{sk}_{\mathrm{asig}}) := \mathrm{KeyGen}^{\mathrm{asig}}(\mathrm{pp}_{\mathrm{asig}})$.
(3) Randomly sample an address secret key $\mathrm{sk}_{\mathrm{addr}}$ that is also a seed to $\mathrm{PRF}^{\mathrm{addr}}$.
(4) Define the address public key as $\mathrm{pk}_{\mathrm{addr}} := \mathrm{PRF}^{\mathrm{addr}}_{\mathrm{sk}_{\mathrm{addr}}}(0)$.
(5) Generate the encryption public-private key pair $(\mathrm{pk}_{\mathrm{enc}}, \mathrm{sk}_{\mathrm{enc}}) := \mathrm{KDF}^{\mathrm{enc}}(\mathrm{pp}_{\mathrm{enc}}, \mathrm{sk}_{\mathrm{addr}})$.
(6) Add $(\mathrm{pk}_{\mathrm{asig}}\mathrm{pk}_{\mathrm{addr}}, \mathrm{pk}_{\mathrm{enc}})$ to pp.
(7) Set $\mathrm{cred}_{\mathrm{adm}} = (\mathrm{pk}_{\mathrm{asig}}, \mathrm{sk}_{\mathrm{asig}}, \mathrm{sk}_{\mathrm{addr}}, \mathrm{pk}_{\mathrm{addr}}, \mathrm{pk}_{\mathrm{enc}}, \mathrm{sk}_{\mathrm{enc}})$.

CreateAccount
Input: public parameters pp
Output: credentials cred

(1) Parse pp as $(\mathrm{pk}_{\mathsf{CRED}}, \mathrm{pp}_{\mathrm{enc}}, *)$.
(2) Randomly sample an address secret key $\mathrm{sk}_{\mathrm{addr}}$ that is also a seed to $\mathrm{PRF}^{\mathrm{addr}}$.
(3) Generate the encryption public and private key pair $(\mathrm{pk}_{\mathrm{enc}}, \mathrm{sk}_{\mathrm{enc}}) := \mathrm{KDF}^{\mathrm{enc}}(\mathrm{pp}_{\mathrm{enc}}, \mathrm{sk}_{\mathrm{addr}})$.
(4) Define the address public key $\mathrm{pk}_{\mathrm{addr}} := \mathrm{PRF}^{\mathrm{addr}}_{\mathrm{sk}_{\mathrm{addr}}}(0)$.
(5) Randomly sample a commitment trapdoor s_{cred}.
(6) Calculate the hiding commitment $\mathrm{cm}_{\mathrm{cred}}$ as $\mathrm{cm}_{\mathrm{cred}} := \mathrm{Comm}^{\mathrm{cred}}_{s_{\mathrm{cred}}}(\mathrm{pk}_{\mathrm{addr}} \| \mathrm{sk}_{\mathrm{addr}})$.
(7) Define $x := (\mathrm{cm}_{\mathrm{cred}}, \mathrm{pk}_{\mathrm{addr}})$ and $a := (\mathrm{sk}_{\mathrm{addr}}, s_{\mathrm{cred}})$.
(8) The user computes the proof $\pi_{\mathsf{CRED}} := \mathrm{Prove}^{\mathrm{zkp}}(\mathrm{pk}_{\mathsf{CRED}}, x, a)$.
(9) Send x and π_{CRED} to the admin, and wait until $\mathrm{cm}_{\mathrm{cred}}$ is included in the Credential Merkle Tree.
(10) Define $\mathrm{cred} := (\mathrm{pk}_{\mathrm{addr}}, \mathrm{sk}_{\mathrm{addr}}, \mathrm{pk}_{\mathrm{enc}}, \mathrm{sk}_{\mathrm{enc}}, s_{\mathrm{cred}}, \mathrm{cm}_{\mathrm{cred}})$.

AddAccount
Input: public parameters pp; public credential values x; credential proof π_{CRED}; Credential Merkle root $\mathrm{rt}_{\mathrm{cred}}$; admin credentials $\mathrm{cred}_{\mathrm{adm}}$
Output: tx

(1) Parse pp as $(\mathrm{vk}_{\mathsf{CRED}}, *)$.
(2) Parse $\mathrm{cred}_{\mathrm{adm}}$ as $(\mathrm{sk}_{\mathrm{asig}}, *)$.
(3) Parse x as $(\mathrm{cm}_{\mathrm{cred}}, *)$.
(4) If $\mathrm{Verify}^{\mathrm{zkp}}(\mathrm{vk}_{\mathsf{CRED}}, \pi_{\mathsf{CRED}}, x)$ outputs `false`, then return $\perp$.
(5) Define $m_{\mathrm{add}} := (\textit{"Add credential"}, \mathrm{cm}_{\mathrm{cred}})$.
(6) Compute the signature on the message as $\sigma_{\mathrm{add}} := \mathrm{Sign}^{\mathrm{asig}}_{\mathrm{sk}_{\mathrm{asig}}}(m_{\mathrm{add}})$.
(7) Publish $\mathrm{tx} := (m_{\mathrm{add}}, \sigma_{\mathrm{add}})$.
(8) Add $\mathrm{cm}_{\mathrm{cred}}$ to the Credential Merkle Tree and update $\mathrm{rt}_{\mathrm{cred}}$.

RevokeAccount
Input: credential commitment $\mathrm{cm}_{\mathrm{cred}}$; Credential Merkle root $\mathrm{rt}_{\mathrm{cred}}$; admin credentials $\mathrm{cred}_{\mathrm{adm}}$
Output: tx

(1) Parse $\mathrm{cred}_{\mathrm{adm}}$ as $(\mathrm{sk}_{\mathrm{asig}}, *)$.
(2) Define $m_{\mathrm{rvk}} := (\textit{"Revoke credential"}, \mathrm{cm}_{\mathrm{cred}})$.
(3) Compute a signature on the message as $\sigma_{\mathrm{rvk}} := \mathrm{Sign}^{\mathrm{asig}}_{\mathrm{sk}_{\mathrm{asig}}}(m_{\mathrm{rvk}})$.
(4) Publish $\mathrm{tx} := (m_{\mathrm{rvk}}, \sigma_{\mathrm{rvk}})$.
(5) Remove $\mathrm{cm}_{\mathrm{cred}}$ from the Merkle tree and update $\mathrm{rt}_{\mathrm{cred}}$.

ConvertToNote
Input: public parameters pp; conversion value $v^{\mathrm{new}}_{\mathrm{note}}$; public destination address $\mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}}$; destination encryption key $\mathrm{pk}^{\mathrm{new}}_{\mathrm{enc}}$; extra info info; admin credentials $\mathrm{cred}_{\mathrm{adm}}$
Output: transaction tx; $\mathrm{note}^{\mathrm{new}}$

(1) Parse pp as $(\mathrm{pk}_{\mathsf{CTO}}, *)$.
(2) Parse $\mathrm{cred}_{\mathrm{adm}}$ as $(\mathrm{sk}_{\mathrm{asig}}, *)$.
(3) Randomly sample the commitment trapdoor $s^{\mathrm{new}}_{\mathrm{note}}$ for the new note.
(4) Compute the commitment $\mathrm{cm}^{\mathrm{new}}_{\mathrm{note}}$ for the new note as $\mathrm{cm}^{\mathrm{new}}_{\mathrm{note}} := \mathrm{Comm}^{\mathrm{note}}_{s^{\mathrm{new}}_{\mathrm{note}}}(\mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}} \| v^{\mathrm{new}}_{\mathrm{note}}) \| 0$.
(5) Define $\mathrm{data}^{\mathrm{new}}_{\mathrm{note}} := \mathrm{Enc}_{\mathrm{pk}^{\mathrm{new}}_{\mathrm{enc}}}(s^{\mathrm{new}}_{\mathrm{note}} \| v^{\mathrm{new}}_{\mathrm{note}} \| \mathrm{info})$.
(6) Set $x := (\mathrm{cm}^{\mathrm{new}}_{\mathrm{note}})$.
(7) Set $a := (s^{\mathrm{new}}_{\mathrm{note}}, v^{\mathrm{new}}_{\mathrm{note}}, \mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}})$.
(8) Obtain a zero-knowledge proof for the transaction $\pi_{\mathsf{CTO}} := \mathrm{Prove}^{\mathrm{zkp}}(\mathrm{pk}_{\mathsf{XFER}}, x, a)$.
(9) Define $m_{\mathrm{cto}} := (x, \pi_{\mathsf{CTO}}, \mathrm{data}^{\mathrm{new}}_{\mathrm{note}})$.
(10) Compute a signature on the transaction message $\sigma_{\mathrm{cto}} := \mathrm{Sign}^{\mathrm{asig}}_{\mathrm{sk}_{\mathrm{asig}}}(m_{\mathrm{cto}})$.
(11) Set $\mathrm{note}^{\mathrm{new}} := (v^{\mathrm{new}}_{\mathrm{note}}, \mathrm{pk}^{\mathrm{new}}_{\mathrm{addr}}, s^{\mathrm{new}}_{\mathrm{note}})$.
(12) Publish $\mathrm{tx} := (m_{\mathrm{cto}}, \sigma_{\mathrm{cto}})$.

ConvertFromNote
Input: public parameters pp; input note $\mathrm{note}^{\mathrm{old}}$; Note Merkle root $\mathrm{rt}_{\mathrm{note}}$; admin credentials $\mathrm{cred}_{\mathrm{adm}}$
Output: transaction tx

(1) Parse pp as $(\mathrm{pk}_{\mathsf{CFROM}}, *)$.
(2) Parse $\mathrm{cred}_{\mathrm{adm}}$ as $(\mathrm{sk}_{\mathrm{addr}}, \mathrm{pk}_{\mathrm{addr}}, \mathrm{sk}_{\mathrm{asig}}, *)$.
(3) Parse $\mathrm{note}^{\mathrm{old}}$ as $(v^{\mathrm{old}}_{\mathrm{note}}, \mathrm{cm}^{\mathrm{old}}_{\mathrm{note}}, t^{\mathrm{old}}_{\mathrm{note}}, s^{\mathrm{old}}_{\mathrm{note}}, *)$.
(4) Determine the position $\mathrm{pos}_{\mathrm{note}}$ of $\mathrm{cm}^{\mathrm{old}}_{\mathrm{note}}$ in the Note Merkle Tree and its path $\mathrm{path}_{\mathrm{note}}$ to $\mathrm{rt}_{\mathrm{note}}$.
(5) Determine the nullifier to $\mathrm{note}^{\mathrm{old}}$ as $\eta := \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\mathrm{addr}}}(\mathrm{pos}_{\mathrm{note}})$.
(6) Set $x := (\mathrm{rt}_{\mathrm{note}}, \eta, \mathrm{pk}_{\mathrm{addr}})$.
(7) Set $a := (\mathrm{sk}_{\mathrm{addr}}, s^{\mathrm{old}}_{\mathrm{note}}, v^{\mathrm{old}}_{\mathrm{note}}.t^{\mathrm{old}}_{\mathrm{note}}, \mathrm{cm}^{\mathrm{old}}_{\mathrm{note}}, \mathrm{path}_{\mathrm{note}}, \mathrm{pos}_{\mathrm{note}})$.
(8) Create a zero-knowledge proof for the transaction $\pi_{\mathsf{CFROM}} := \mathrm{Prove}^{\mathrm{zkp}}(\mathrm{pk}_{\mathsf{CFROM}}, x, a)$.
(9) Define $m_{\mathrm{cfrom}} := (x, \pi_{\mathsf{CFROM}})$.
(10) Compute a signature on the transaction message $\sigma_{\mathrm{cfrom}} := \mathrm{Sign}^{\mathrm{asig}}_{\mathrm{sk}_{\mathrm{asig}}}(m_{\mathrm{cfrom}})$.
(11) Publish $\mathrm{tx} := (m_{\mathrm{cfrom}}, \sigma_{\mathrm{cfrom}})$.

Fig. 2. Algorithm definitions

Using $\mathcal{H}$ we define the PRFs as follows, with $[\cdot]_x$ being the function that truncates its input to a size of x bits:

- $\mathrm{PRF}^{\mathrm{addr}}_x(s) := \mathcal{H}(x\|00\|s)$, with $x \in \{0,1\}^{256}$, $s \in \{0,1\}^{254}$;
- $\mathrm{PRF}^{\eta}_x(s) := \mathcal{H}(x\|01\|0^{191}\|s)$, with $x \in \{0,1\}^{256}$, $s \in \{0,1\}^{33}$;
- $\mathrm{PRF}^{\mu}_x(s) := \mathcal{H}(x\|10\|0^{191}\|s)$, with $x \in \{0,1\}^{256}$, $s \in$

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

CreateTransaction

Input: public parameters pp; credentials cred; Note Merkle root rt_{note}; Memory Merkle root rt_{mem}; Credential Merkle root rt_{cred}; old note note^{old}; previous memory cell mem^{old}; ceiling memory cell mem^{ceil}; new note value $v_{\text{note}}^{\text{new}}$; new public address $\text{pk}_{\text{addr}}^{\text{new}}$; new public encryption key $\text{pk}_{\text{enc}}^{\text{new}}$; extra info info; new block time t^{new}; boolean value for including SAVER encrypted values b_{saver}; lock time new note t_δ

Output: transaction tx

(1) Parse pp as $(\text{pk}_{\text{XFER}}, \text{pp}_{\text{enc}}, \text{pp}_{\text{sig}}, \text{pk}_{\text{svr}}*)$.
(2) Parse cred to $(\text{sk}_{\text{addr}}, \text{pk}_{\text{addr}}, \text{cm}_{\text{cred}}, s_{\text{cred}})$.
(3) Determine the position pos_{cred} of cm^{cred} in the Account Merkle Tree and its path $\text{path}_{\text{cred}}$ to rt_{cred}.
(4) Randomly sample two commitment trapdoors $s_{\text{note}}^{\text{new}}$, $s_{\text{mem}}^{\text{new}}$ for the new note.
(5) If $\text{note}^{\text{old}} = \perp$, then set $b_{\text{note}} = 0, \text{pos}_{\text{note}} = 0, \text{path}_{\text{note}} = 0, v_{\text{note}}^{\text{old}} = 0, s_{\text{note}}^{\text{old}} = 0, \eta = -1, t_\delta^{\text{old}} = 0, t_{\text{note}}^{\text{old}} = t^{\text{new}}$, compute $\text{cm}_{\text{note}}^{\text{old}}$ and $\text{rt}_{\text{note}}^{\text{int}}$ accordingly, and go to step 8.
(6) Parse note^{old} as $(v_{\text{note}}^{\text{old}}, \text{cm}_{\text{note}}^{\text{old}}, t_\delta^{\text{old}}, t_{\text{note}}^{\text{old}}, s_{\text{note}}^{\text{old}})$.
(7) Determine the position pos_{note} of $\text{cm}_{\text{note}}^{\text{old}}$ in the Note Merkle Tree and its path $\text{path}_{\text{note}}$ to rt_{note} and set $\text{rt}_{\text{note}}^{\text{int}} = \text{rt}_{\text{note}}$.
(8) Determine the nullifier η^{int} to note^{old} as $\eta^{\text{int}} := \text{PRF}_{\text{sk}_{\text{addr}}}^{\eta}(\text{pos}_{\text{note}})$.
(9) Calculate the commitment $\text{cm}_{\text{note}}^{\text{new}}$ for the new note as $\text{cm}_{\text{note}}^{\text{new}} := \text{Comm}_{s_{\text{note}}^{\text{new}}}^{\text{note}}(\text{pk}_{\text{addr}}^{\text{new}} \| v_{\text{note}}^{\text{new}} \| t_\delta)$.
(10) Define $\text{data}_{\text{note}}^{\text{new}} := \text{Enc}_{\text{pk}_{\text{enc}}^{\text{new}}}(s_{\text{note}}^{\text{new}} \| v_{\text{note}}^{\text{new}} \| t_\delta \| \text{info})$.
(11) If $\text{mem}^{\text{old}} = \perp$, then set $b_{\text{mem}} = 0, \text{pos}_{\text{mem}} = -1, t^{\text{old}} = -(T+1), v_{\text{mem}}^{\text{old}} = 0, s_{\text{mem}}^{\text{old}} = 0, c_{\text{mem}}^{\text{old}} = 0, \text{path}_{\text{mem}} = 0, \text{rt}_{\text{mem}}^{\text{real}} = \text{rt}_{\text{mem}}$, and compute $\text{cm}_{\text{mem}}^{\text{old}}$ and $\text{rt}_{\text{mem}}^{\text{int}}$ accordingly and go to step 14.
(12) Parse mem^{old} as $(v_{\text{mem}}^{\text{old}}, \text{cm}_{\text{mem}}^{\text{old}}, s_{\text{mem}}^{\text{old}}, c_{\text{mem}}^{\text{old}}, t_{\text{mem}}^{\text{old}})$.
(13) Determine the position pos_{mem} of $\text{CRH}^{\text{mem}}(\text{cm}_{\text{mem}}^{\text{old}}, t_{\text{mem}}^{\text{old}})$ in the Memory Merkle Tree and its path path_{mem} to rt_{mem} and set $\text{rt}_{\text{mem}}^{\text{real}} = \text{rt}_{\text{mem}}^{\text{int}} = \text{rt}_{\text{mem}}$.
(14) If $\text{mem}^{\text{ceil}} = \perp$, then set $t^{\text{ceil}} = -(T+1), v_{\text{mem}}^{\text{ceil}} = 0, s_{\text{mem}}^{\text{ceil}} = 0, c_{\text{mem}}^{\text{ceil}} = 0$, and compute $\text{cm}_{\text{mem}}^{\text{ceil}}$ accordingly, and go to step 17.
(15) Parse mem^{ceil} as $(v_{\text{mem}}^{\text{ceil}}, \text{cm}_{\text{mem}}^{\text{ceil}}, s_{\text{mem}}^{\text{ceil}}, c_{\text{mem}}^{\text{ceil}}, t_{\text{mem}}^{\text{ceil}})$.
(16) Determine the position pos_{ceil} of $\text{CRH}^{\text{mem}}(\text{cm}_{\text{mem}}^{\text{ceil}}, t_{\text{mem}}^{\text{ceil}})$ in the Memory Merkle Tree and its path $\text{path}_{\text{ceil}}$ to rt_{mem}.
(17) Determine the nullifier μ to mem^{old} as $\mu := \text{PRF}_{\text{sk}_{\text{addr}}}^{\mu}(\text{pos}_{\text{mem}})$.
(18) Determine $v_{\text{mem}}^{\text{new}} := v_{\text{note}}^{\text{old}} + v_{\text{mem}}^{\text{old}} - v_{\text{note}}^{\text{new}}$.
(19) Determine $c^{\text{new}} := c^{\text{old}} + v_{\text{note}}^{\text{new}} \cdot (1 - b_{\text{saver}})$.
(20) Calculate the new memory commitment $\text{cm}_{\text{mem}}^{\text{new}} := \text{Comm}_{s_{\text{mem}}^{\text{new}}}^{\text{mem}}(\text{pk}_{\text{addr}} \| v_{\text{mem}}^{\text{new}} \| c^{\text{new}})$.
(21) Define $\text{data}_{\text{mem}}^{\text{new}} := \text{Enc}_{\text{pk}_{\text{enc}}}(s_{\text{mem}}^{\text{new}})$.
(22) Randomly generate a signature public and private key pair $(\text{pk}_{\text{sig}}, \text{sk}_{\text{sig}}) := \text{KeyGen}_{\text{sig}}(\text{pp}_{\text{sig}})$.
(23) Compute $k := \text{CRH}^{\text{sig}}(\text{pk}_{\text{sig}})$ and calculate $\kappa := \text{PRF}_{\text{sk}_{\text{addr}}}^{\kappa}(k)$, which ties the signature public key to the secret address key of the sender.
(24) Determine the inputs for the SAVER encryption: $\overline{\text{pk}}_{\text{sndr}} = \text{pk}_{\text{addr}} \cdot b_{\text{saver}}$, $\overline{\text{pk}}_{\text{rcvr}} = \text{pk}_{\text{addr}}^{\text{new}} \cdot b_{\text{saver}}$, and $\overline{v}_{\text{xfer}} = v_{\text{note}}^{\text{new}} \cdot b_{\text{saver}}$.
(25) Set $m_{\text{svr}} := (\overline{\text{pk}}_{\text{sndr}}, \overline{\text{pk}}_{\text{rcvr}}, \overline{v}_{\text{xfer}})$.
(26) Set $x := (\text{rt}_{\text{mem}}, \text{rt}_{\text{note}}, \text{rt}_{\text{cred}}, k, \kappa, \eta, \mu, \text{cm}_{\text{note}}^{\text{new}}, \text{cm}_{\text{mem}}^{\text{new}}, t^{\text{new}})$.
(27) Set $a := (\text{rt}_{\text{mem}}^{\text{real}}, \text{rt}_{\text{mem}}^{\text{int}}, \text{rt}_{\text{note}}^{\text{int}}, \eta^{\text{int}}, \text{pk}_{\text{addr}}, \text{sk}_{\text{addr}}, s_{\text{cred}}, \text{cm}_{\text{cred}}, v_{\text{note}}^{\text{old}}, s_{\text{note}}^{\text{old}}, t_\delta^{\text{old}}, \text{cm}_{\text{note}}^{\text{old}}, t_{\text{note}}^{\text{old}}, v_{\text{mem}}^{\text{old}}, s_{\text{mem}}^{\text{old}}, c^{\text{old}}, \text{cm}_{\text{mem}}^{\text{old}}, t_{\text{mem}}^{\text{old}}, v_{\text{mem}}^{\text{ceil}}, s_{\text{mem}}^{\text{ceil}}, c^{\text{ceil}}, \text{cm}_{\text{mem}}^{\text{ceil}}, t_{\text{mem}}^{\text{ceil}}, \text{path}_{\text{cred}}, \text{path}_{\text{note}}, \text{path}_{\text{mem}}, \text{path}_{\text{ceil}}, \text{pos}_{\text{note}}, \text{pos}_{\text{mem}}, \text{pos}_{\text{cred}}, \text{pos}_{\text{ceil}}, s_{\text{note}}^{\text{new}}, s_{\text{mem}}^{\text{new}}, v_{\text{note}}^{\text{new}}, v_{\text{mem}}^{\text{new}}, c^{\text{new}}, \text{pk}_{\text{addr}}^{\text{new}}, b_{\text{note}}, b_{\text{mem}}, b_{\text{saver}}, t_\delta)$.
(28) Obtain a zero-knowledge proof and ciphertext for the transaction $(\pi_{\text{XFER}}, \text{data}_{\text{saver}}) := \text{Enc}^{\text{svr}}(\text{pk}_{\text{XFER}}, \text{pk}_{\text{svr}}, m_{\text{svr}}, x, a)$.
(29) Define $m_{\text{xfer}} := (x, \pi_{\text{XFER}}, \text{data}_{\text{note}}^{\text{new}}, \text{data}_{\text{mem}}^{\text{new}}, \text{data}_{\text{saver}})$.
(30) Compute a signature on the transaction message $\sigma_{\text{xfer}} := \text{Sign}_{\text{sk}_{\text{sig}}}^{\text{sig}}(m_{\text{xfer}})$.
(31) Set $\text{note}^{\text{new}} := (v_{\text{note}}^{\text{new}}, \text{pk}_{\text{addr}}^{\text{new}}, s_{\text{note}}^{\text{new}})$.
(32) Set $\text{mem}^{\text{new}} := (v_{\text{mem}}^{\text{new}}, \text{pk}_{\text{addr}}, s_{\text{mem}}^{\text{new}})$.
(33) Publish $\text{tx} := (m_{\text{xfer}}, \text{pk}_{\text{sig}}, \sigma_{\text{xfer}})$.

VerifyTransaction

Input: public parameters pp; transaction tx; current blockchain B

Output: boolean value for correctness b

(1) Parse pp as $(\text{vk}_*, \text{pk}_{\text{svr}}*)$.
(2) Parse tx as $(m, \text{pk}_{\text{sig}}, \sigma)$, if transaction type is *not* regular transfer transaction get the public admin key pk_{asig}.
(3) Parse m as $(x, \pi, \text{data}_{\text{saver}}, *)$.
(4) If the transaction type is *not* a regular transfer transaction, go to step 10.
(5) Parse x as $(\text{rt}_{\text{cred}}, \text{rt}_{\text{mem}}, \text{rt}_{\text{node}}, k, \kappa, \eta, \mu, t^{\text{new}}, *)$.
(6) If rt_{mem}, rt_{cred} and rt_{node} do not appear in the same block on B, then output `false`.
(7) If t^{new} is not close to the current block time, then output `false`.
(8) If η or μ does appear on B, then output `false`.
(9) If k does not equal $\text{CRH}^{\text{sig}}(\text{pk}_{\text{sig}})$, then output `false`.
(10) If $\text{Verify}_{\text{pk}_{(\text{a})\text{sig}}}^{(\text{a})\text{sig}}(m, \sigma)$ outputs `false`, then output `false`.
(11) Define $b := \text{VerifyEnc}^{\text{svr}}(\text{vk}_*, \text{pk}_{\text{svr}}, \pi, \text{data}_{\text{saver}}, x)$ if the transaction type is a regular transaction, otherwise $b := \text{Verify}^{\text{zkp}}(\text{vk}_*, \pi, x)$.

ReceiveTransaction

Input: public parameters pp; new transaction tx; credentials cred; current blockchain B

Output: received note note^{new}

(1) If transaction type is not a regular transfer transaction, then output $\perp$.
(2) If VerifyTransaction(pp, tx, B) outputs `false`, then output $\perp$.
(3) Parse cred as $(\text{sk}_{\text{enc}}, \text{pk}_{\text{addr}}, *)$.
(4) Parse tx as $(m_{\text{xfer}}, *)$.
(5) Parse m_{xfer} as $(x, \text{data}_{\text{note}}, *)$.
(6) Parse x as $(\text{cm}_{\text{note}}^{\text{new}}, *)$.
(7) Compute $(s_{\text{note}}^{\text{new}}, v_{\text{note}}^{\text{new}}, t_\delta^{\text{new}}, \text{info}) = \text{Dec}_{\text{sk}_{\text{enc}}}(\text{data}_{\text{note}}^{\text{new}})$, if the output was $\perp$, then output $\perp$.
(8) If $\text{cm}_{\text{note}}^{\text{new}}$ does not equal $\text{Comm}_{s_{\text{note}}^{\text{new}}}^{\text{note}}(\text{pk}_{\text{addr}} \| v_{\text{note}}^{\text{new}})$, then output $\perp$.
(9) Set $\text{note}^{\text{new}} := (v_{\text{note}}^{\text{new}}, s_{\text{note}}^{\text{new}}, \text{info}, \text{cm}_{\text{note}}^{\text{new}})$.

Fig. 3. Algorithm definitions

$\{0,1\}^{33}$;
- $\text{PRF}_x^{\kappa}(s) := \mathcal{H}(x \| 11 \| [s]_{254})$, with $x \in \{0,1\}^{256}$, $s \in \{0,1\}^{256}$.

Consistently with the above defined PRFs, we take our public and secret address key to have bit length 256. Using this fact we can define the credential commitment using the Pedersen hash.

We will denote the Pedersen hash function as $\mathcal{P}$. Our Pedersen hash function takes as input a bit string of arbitrary length and outputs an element in the Jubjub curve as defined in the Zcash protocol [22].

Given this Pedersen hash function and three pre-defined elements on the Jubjub Curve J_1, J_2, and J_3 we define our commitment functions, with $s_* \in \{0,1\}^{252}$, as:

- $\text{COMM}_{s_{\text{cred}}}^{\text{cred}}(\text{pk}_{\text{addr}} \| \text{sk}_{\text{addr}}) := \mathcal{P}(\text{pk}_{\text{addr}} \| \text{sk}_{\text{addr}}) \cdot J_1^{s_{\text{cred}}}$;
- $\text{COMM}_{s_{\text{note}}}^{\text{note}}(\text{pk}_{\text{addr}} \| v_{\text{note}}) := \mathcal{P}(\text{pk}_{\text{addr}} \| v_{\text{note}} \| t_\delta) \cdot J_2^{s_{\text{note}}}$;
- $\text{COMM}_{s_{\text{mem}}}^{\text{mem}}(\text{pk}_{\text{addr}} \| v_{\text{mem}}) := \mathcal{P}(\text{pk}_{\text{addr}} \| v_{\text{mem}} \| c_{\text{mem}}) \cdot J_3^{s_{\text{mem}}}$.

Moreover, we define $v_{\text{max}} := 2^{64} - 1$ for convenience.

Similarly, we define the Merkle tree hash functions as:

- $\text{CRH}^{\text{mem}}(\text{cm}_{\text{mem}}, t) = \mathcal{P}(\text{cm}_{\text{mem}} \| t)$

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.

- $\mathrm{CRH}^{\mathrm{note}}(\mathrm{cm}_{\mathrm{note}}, t) = \mathcal{P}(\mathrm{cm}_{\mathrm{note}} \| t)$.

On the other hand, we define our collision-resistant hash as $\mathrm{CRH}^{\mathrm{sig}} = \mathcal{H}$. Because this hash function needs not be encoded inside the arithmetic circuit, efficiency does not matter as much and we simply choose the more secure option.

c) Signatures: Both signature schemes can be instantiated using EdDSA. Normally, EdDSA only provides EUF-CMA security, however if one introduces an additional check on the domain of the signature [23] it provides SUF-CMA (and thus also SUF-1CMA) security.

d) Encryption and KDF: For the key private encryption scheme we choose to use ECIES [24], with any appropriate key derivation to derive the user's private key. Specifically, we use a SHA-512 derived Hash KDF to generate X25519 key pairs. The used encryption algorithm is CHACHA20-POLY1305 [25]. We choose to adopt ECIES since it is a standardised and widely adopted scheme with available implementations and it fits our security requirement.

e) Performance: A brief analysis of the proof of concept reveals that the total size of a *Transfer* transaction in our implementation is 2640 bytes, whereas the other transaction types have a size of around 350 bytes. Next to this we see that computation times on a modern desktop PC (6 core CPU @4.0GHz and RAM at 3600MHz) take around 2 to 3 seconds for the *Transfer* transaction and under a second for the other transaction types.

V. Conclusions

We propose a decentralized, permissioned payment scheme that combines auditability with anonymity without making large compromises. Any user of the scheme can make transfer value to other users without any details of any transaction being visible to any other parties. However, this anonymity does not get in the way of accountability and regulation.

Participating financial institutions can perform KYC 'at-the-gate' and a distributed group of judges can, if necessary, view transaction details of suspiciously large transactions.

Future work: Many of the features presented in this paper can be seen as a first step towards an auditable and anonymous payment scheme with a wide range of functionalities. Ideas of future improvements can be found in many ranges. It would be interesting to investigate how anonymous timelocks can be used for second layer solutions such as the Lightning Network.

References

[1] FATF, "International Standards on Combating Money Laundering and the Financing of Terrorism & Proliferation," 2012, report. [Online]. Available: http://www.fatf-gafi.org/recommendations.html

[2] ——, "Guidance for a Risk-Based Approach to Virtual Assets and Virtual Asset Service Providers," 2019, report. [Online]. Available: www.fatf-gafi.org/publications/fatfrecommendations/documents/Guidance-RBA-virtual-assets.html

[3] D. Chaum, "Blind Signatures for Untraceable Payments," in *Advances in Cryptology*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Boston, MA: Springer US, 1983, pp. 199–203.

[4] T. Sander and A. Ta-Shma, "Auditable, Anonymous Electronic Cash," in *Advances in Cryptology — CRYPTO' 99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed. Springer, 1999, pp. 555–572.

[5] K. M. Alonso and koe, "Zero to Monero: First Edition," Jun. 2018, online article. [Online]. Available: https://web.getmonero.org/it/library/Zero-to-Monero-1-0-0.pdf

[6] CryptoRekt, "Blackpaper Verge Currency," Jan. 2019, blackpaper. [Online]. Available: https://vergecurrency.com/static/blackpaper/verge-blackpaper-v5.0.pdf

[7] E. Fujisaki and K. Suzuki, "Traceable Ring Signature," in *Public Key Cryptography – PKC 2007*, ser. Lecture Notes in Computer Science, T. Okamoto and X. Wang, Eds. Berlin, Heidelberg: Springer, 2007, pp. 181–200.

[8] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 397–411, ISSN: 1081-6011.

[9] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE, May 2014, pp. 459–474.

[10] W. Li, Y. Wang, L. Chen, X. Lai, X. Zhang, and J. Xin. Fully auditable privacy-preserving cryptocurrency against malicious auditors. [Online]. Available: https://www.semanticscholar.org/paper/Fully-Auditable-Privacy-preserving-Cryptocurrency-Li-Wang/a68a6c1ea1ad83b298cc47dc8c3646fdeec0c8e5

[11] C. Garman, M. Green, and I. Miers, "Accountable privacy for decentralized anonymous payments," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, J. Grossklags and B. Preneel, Eds. Springer, 2017, pp. 81–98.

[12] I. Damgård, C. Ganesh, H. Khoshakhlagh, C. Orlandi, and L. Siniscalchi, "Balancing privacy and accountability in blockchain identity management," in *Topics in Cryptology – CT-RSA 2021*, ser. Lecture Notes in Computer Science, K. G. Paterson, Ed. Springer International Publishing, 2021, pp. 552–576.

[13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS '12*. ACM Press, 2012, pp. 326–349.

[14] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, vol. 9666, pp. 305–326.

[15] J. Lee, J. Choi, J. Kim, and H. Oh, "SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization," IACR, Tech. Rep. 1270, 2019. [Online]. Available: https://eprint.iacr.org/2019/1270

[16] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-Privacy in Public-Key Encryption," in *Advances in Cryptology — ASIACRYPT 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed. Berlin, Heidelberg: Springer, 2001, pp. 566–582.

[17] N. P. Smart, *Cryptography made simple*, ser. Information security and cryptography. Springer, 2016, oCLC: 934618628.

[18] D. Khovratovich and M. Vladimirov, "Full privacy in account-based cryptocurrencies v 0.12," Jul. 2019, online draft. [Online]. Available: https://dusk.network/uploads/Private-account-based-model.pdf

[19] Electric Coin Company. What is jubjub? [Online]. Available: https://z.cash/ja/technology/jubjub/

[20] M.-J. O. Saarinen and J.-P. Aumasson. The BLAKE2 cryptographic hash and message authentication code (MAC). Num Pages: 30. [Online]. Available: https://datatracker.ietf.org/doc/rfc7693

[21] E. Tromer, "[Sapling] specify Pedersen hashes for a collision-resistant hash function inside the SNARK · Issue #2234 · zcash/zcash," Apr. 2017. [Online]. Available: https://github.com/zcash/zcash/issues/2234\#issuecomment-292419085

[22] S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification," Feb. 2020. [Online]. Available: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf

[23] P. Wuille, "Dealing with malleability," Mar. 2014, bIP 62. [Online]. Available: https://github.com/bitcoin/bips

[24] D. R. L. Brown, "Standards for Efficient Cryptograhpy 1 (SEC 1)," May 2009. [Online]. Available: https://www.secg.org/sec1-v2.pdf

[25] Y. Nir and A. Langley. ChaCha20 and poly1305 for IETF protocols. Num Pages: 46. [Online]. Available: https://datatracker.ietf.org/doc/rfc8439

Authorized licensed use limited to: University of Groningen. Downloaded on February 13,2023 at 11:00:01 UTC from IEEE Xplore. Restrictions apply.