Matteo Campanelli and Mathias Hall-Andersen

𝕍eksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions

Abstract: We propose 𝕍eksel, a simple generic paradigm for constructing efficient non-interactive coin mixes. The central component in our work is a concretely efficient proof $\pi_{\text{1-many}}$ that a homomorphic commitment c^* is a rerandomization of a commitment $\mathsf{c} \in \{\mathsf{c}_1, \ldots, \mathsf{c}_\ell\}$ without revealing c. We formalize anonymous account-based cryptocurrency as a universal composability functionality and show how to efficiently instantiate the functionality using $\pi_{\text{1-many}}$ in a straightforward way (𝕍eksel). We instantiate and implement $\pi_{\text{1-many}}$ from Strong-RSA, DDH and random oracles targeting ≈ 112 bits of security. The resulting NIZK has constant size ($|\pi_{\text{1-many}}| = 5.3$KB) and constant proving/verification time (≈ 90ms), on an already accumulated set. Compared to Zerocash [5]—which offers comparable marginal verification cost and an anonymity set of every existing transaction—our transaction are larger (6.2 KB) and verification is slower. On the other hand, 𝕍eksel relies on more well-studied assumptions, does not require an expensive trusted setup for proofs and is arguably simpler (from an implementation standpoint). Additionally we think that $\pi_{\text{1-many}}$ might be interesting in other applications, e.g. proving possession of some credential posted on-chain.

Keywords: blockchains, UC, zero-knowledge, accumulators, implementation

1 Introduction

Cryptocurrencies allow for fully decentralized and publicly verifiable currency systems. An interesting problem in cryptocurrencies is that of guaranteeing some level of privacy by making impossible to an observer to learn anything about the "flow of money". While users in the network only require known pseudonyms to be identified, we know that they are not sufficient to achieve an acceptable level of privacy [27, 29].

Several prior works address this problem (ZeroCash, Monero, OmniRing). The main challenge they face is to build solutions that are private but can still scale. An important requirement for scalability are the efficiency of spending and verifying transactions as well as their size. Often this boils down to the efficiency and proof size of their underlying non-interactive zero-knowledge schemes (or NIZKs) these works rely on.

Some protocols (e.g., Monero[1] and OmniRing [25]) inherently trade efficiency against privacy requirements. They need to keep a relatively small anonymity set (a ring of signatures, in their specific constructions) for each transaction: a set of "coins" that a spending transaction can refer to. Other solutions (e.g. ZCash [22]) do not have this limitation, but have other caveats such as relying on an expensive trusted-setup and cryptographic assumptions that are not well-studied yet (e.g. the knowledge of exponent assumption).

In this work we address the question of *how to design cryptocurrencies whose efficiency does not degrade with privacy requirements.* Addressing this question, we also focus on solutions that rely on "minimal" cryptographic assumptions. In particular we want to design solutions that rely on *transparent* proof systems, i.e. that do not require a trusted setup and avoid the use of non-standard assumptions such as knowledge-of-exponent.

On the way, we study the problem of formalizing and obtaining privacy in account-based cryptocurrencies. We believe that this model is of interest because many existing cryptocurrencies (e.g. Ethereum) are account-based. Additionally, some of the approaches in literature to balance privacy and accountability are account-based [14]. In this setting where users maintain fixed accounts over time, however, we cannot hope to achieve the same levels of privacy of the UTXO (Unspent Transaction Output) model where each transfer can refer to freshly created pseudonyms. Though weaker, this privacy model is still interesting in some applications. To the best of our knowledge it has not been formally investigated before.

Matteo Campanelli: Aarhus University, matteo@cs.au.dk, supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM)
Mathias Hall-Andersen: Aarhus University, ma@cs.au.dk, Concordium Blockhain Research Center

1 https://www.getmonero.org/

1.1 Contribution

Our main contribution is a concrete construction for a cryptocurrency with privacy-preserving properties *that supports arbitrary-sized anonymity sets*. We obtain small concrete transaction sizes compared to other solutions in literature (see Figure 1) and efficient marginal costs of verification and spending (see Figure 2 and Section 6). Our construction relies on standard cryptographic assumptions and on transparent non-interactive zero-knowledge proofs (secure in the random oracle model). Our concrete efficiency relies on improvements on the state of art of zero-knowledge arguments over accumulators that may be of independent interests (more details below). We implement our construction in Rust; its code is open source and available at [1].

On the way we make the following contributions:

- We formalize privacy-preserving cryptocurrencies with accounts (Section 3) through a UC functionality.
- We provide a highly general and modular construction for this functionality (Section 4). By a modular description, our construction can be further improved by simply replacing some of its building blocks without having to prove its security again. Its more concrete version is described (in light of following sections) in Appendix A. Our solution support coins of arbitrary value and can be extended to the UTXO setting (see Appendix D).
- We describe a new concrete transparent NIZK to prove a one-out-of-many relation [21], to prove that one public commitment is rerandomized from a set of existing commitments. Our techniques rely on commit-and-prove[2] zero-knowledge proof accumulators in unknown-order groups [6] and on optimized relations in Bulletproofs [10]. One challenge we need to solve is how to commit (and accumulate) to coins. Since coins are also "commitments" it is not immediate to have an efficient proof system that supports this double level of commitments. In our solutions we adopt a new SNARK-friendly elliptic curve that is compatible with `Curve25519`. We believe this curve (which we dub Jabberwock) and its surrounding techniques can be of independent interest.

2 As in [12] we label as "commit-and-prove" a proof system that works efficiently over a commitment representation and can thus be composed with others of the same type.

1.2 Prior Works

Monero	Anonymity Set	Concrete Tx Size
QuisQuis	2^4	13 KB
Lelantus$_{2^{10}}$	2^{10}	2.7 KB
Lelantus$_{2^{14}}$	2^{14}	3.9 KB
Lelantus$_{2^{16}}$	2^{16}	5.6 KB
Omniring$_{2^{10}}$	2^{10}	1.0 KB
Omniring$_{2^{14}}$	2^{14}	1.3 KB
Omniring$_{2^{16}}$	2^{16}	1.4 KB
Zerocash	Any	$<$ 1 KB
Zerocoin	Any	45 KB
𝕍eksel	Any	$<$ 6.3 KB

Fig. 1. Tx. size for different anonymity sets

	Spend	Verify	Tx Size	Amounts
Monero	$O(n)$	$O(n)$	$O(n)$	Yes
QuisQuis	$O(n)$	$O(n)$	$O(n)$	Yes
Lelantus	$O(n)$	$O(n)$	$O(\log n)$	Yes
Omniring	$O(n)$	$O(n)$	$O(\log n)$	Yes
Zerocash	$O(\log n)$	$O(\log n)$	$O(1)$	Yes
Zerocoin	$O(1)$	$O(1)$	$O(1)$	No
𝕍eksel	$O(1)$	$O(1)$	$O(1)$	Yes

Fig. 2. Asymptotic marginal cost of verification / spending

Groth & Kohlweiss: In [20], Groth and Kohlweiss constructed an efficient proof with size $O(\log n)$ for the relation $\{r : \exists i \text{ st } \mathsf{Comm}(0; \mathsf{r}) = \mathsf{c}_i\}$ given Pedersen commitments $\mathsf{c}_1, \ldots, \mathsf{c}_n$. Using this they showed how to exploit the homomorphic property of the commitment scheme to create concretely efficient anonymous transactions. Similar techniques has since been widely explored in Omniring [25], RingCT3.0 [30] and Lelantus [23]. These works have different concrete efficiency, but share the same asymptotic efficiency: $O(\log n)$ transaction size and inherent $O(n)$ spending/verification time since the size of the statement proved in zero-knowledge is linear in the anonymity set. Due to the linear verification time these approaches only scale to anonymity sets of size $\approx 2^{16}$ in practice, even with batch verification techniques.

Zerocoin: Zerocoin [28] uses an RSA accumulator to 'compress' the set of coins, a coin is spend by opening the unique serial number of the coin and proving its

membership in the accumulator in zero-knowledge. This enables $O(1)$ spending/verification assuming the coins have been aggregated ahead of time. Unfortunately the "double discrete-log" proof in Zerocoin relies on cut-and-choose and is therefore concretely in-efficient: over 45 KB for 128-bits of security. Additionally Zerocoin does not enable coins to have arbitrary denominations.

Zerocash: Zerocash [5] applies the same technique of "compressing the statement" using an accumulator (Merkle tree) which enables it to achieve logarithmic marginal spending/verification cost. Zerocash verifies the Merkle path inside a zk-SNARK (Groth16 [19] in Zcash implementation), which hides the index of the coin to be spend and compresses the membership proof down to $O(1)$ with very small constants. In terms of concrete efficiency the Zcash implementation of Zerocash is currently the most efficient decentalized payment system with a "full" anonymity set.

QuisQuis: QuisQuis [17] seeks to mitigate the issue of an ever-growing set of 'spending tags' which must be maintained by the nodes in Zerocash to avoid double spending. This is achived by having the spender essentially do a shuffle locally: the spender picks n other unspend coins along with the coin he wishes to spend, then proves that she can spend one of the $n+1$ coins, correctly rerandomize the remaining n coins and post the n new rerandomized coins on the chain. Since QuisQuis relies on posting the new set of outputs to the chain, the transaction size of this approach is $O(n)$. The anonymity set also inherently consists (at best) of the set of unspend outputs (as oppose to the set of all coins created).

1.3 Technical Overview

1.3.1 Basic Setting

At a basic level our approach to decentralized payments is similar to that of Zerocoin[28], however we aim at supporting coins of arbitrary (hidden) denominations, in this sense we diverge from the simpler setting in [28] [21] where all coins have the same denomination. For sake of providing intuition, in this section we describe the account-based model where each party has a commitment $(\mathsf{bal}_i)_i$ stored on-chain and locally holds the *private* balance v_i that is the opening of bal.

To transfer a certain amount v, a sender party $\mathcal{S}$ will *create* a coin spendable by the recipient $\mathcal{R}$, that is a commitment to a triple consisting of: **a)** v the value of the coin (payment amount) **b)** the identity of the recipient **c)** a random spending tag t. The sender then broadcasts this commitment together with an encryption through $\mathsf{pk}_{\mathcal{R}}$ of its opening. Naturally the sender should also be able to show they can afford the transfer; we temporarily ignore this issue and discuss how to approach it later in this section. Once they have observed that someone created a new coin, all users keep track of it in a set S_{coins} of existing coins.

In order to claim the transfer—to *collect* the coin—user $\mathcal{R}$ will need to do two things: *(i)* show that it knows the opening of one among the existing coins; *(ii)* reveal its tag t so that the coin cannot be spent again. The first step requires some care because we want the transfer to be somewhat private, i.e. with the exception of $\mathcal{S}$ and $\mathcal{R}$, no observer of the system should learn anything about the coin being collected. To do that we need to apply a zero-knowledge proof showing we know the opening of some coin in S_{coins} such that this coin encodes tag t.

Because we require each coin to denote a custom transferred amount, we now have and additional challenge. When parties observe that $\mathcal{R}$ collects a coin, they should have a way to update $\mathcal{R}$'s balance $\mathsf{bal}_{\mathcal{R}}$ *without* having $\mathcal{R}$ reveal the value of the coin. While this could be done using the homomorphic properties of commitments by "adding" the coin[3] to the balance, we cannot reveal the coin itself either (that would, at the least, leak the sender!). Thus we let $\mathcal{R}$ produce a rerandomization[4] cn^* of the collected coin; parties can now use homomorphically add the latter to $\mathsf{bal}_{\mathcal{R}}$.

Following the approach outlined above, collecting a coin requires $\mathcal{R}$ to prove in zero-knowledge that the rerandomized coin cn^* opens to same amount as one of the coins in S_{coins}[5]. In the remainder of this section we describe our technical solutions to efficiently produce and verify this proof.

3 We temporarily ignore the issue that a coin also commits to other elements, such as the tag t, when performing this homomorphic operation; this can be simply addressed. We refer the reader to our main construction.

4 In an homomorphic commitment scheme we can always achieve rerandomization by adding a commitment to 0.

5 Obviously cn^* should also open to the same tag and recipient as the collected coin in S_{coins}. These are public values and we do not to include them in the "zero-knowledge" part of the proof. We solve this instead by exploiting the homomorphic properties of commitments. See main construction.

1.3.2 Our techniques

As a first step towards our goal, we let parties keep a compressed digest to the set S_{coins}, through algebraic accumulators [2, 6, 8, 11]. Thus, given a set S, we can produce $A = \mathsf{accum}(S)$, a binding (but usually not hiding) compressing commitments to S. An important features of accumulators is that it allows to prove membership of elements "inside" A efficiently (that is, with short certificates and fast verification). In order to efficiently prove the desired relation in zero-knowledge we adopt a modular approach and we split it in two components, set membership and rerandomization. We thus apply two proof systems that are *specifically efficient* for each of the two relation components. To ensure that they refer to the same content, we use a commit-and-prove approach, and link them through a hiding commitment to the coin cn_k we are collecting. We now describe this process in more detail.

Recall that coins are commitments to pairs of amounts and additional information (tags and recipient identity). We denote by $\mathsf{Comm}^{\circ}(\cdot;\cdot)$ the commitment procedure that produces coins and by $\mathsf{Comm}^{\square}(\cdot;\cdot)$ the commitment procedure we use to link the two proofs mentioned above. The first parameter in each is the message we are committing to (which possibly has additional structure) while the second parameter denotes the randomness. For our concrete case, the two commitment schemes can be thought of as variants of Pedersen commitments in different groups. In order to prove that a coin c_k is legit without revealing it, $\mathcal{R}$ first produces a commitment $c \leftarrow \mathsf{Comm}^{\square}(\mathsf{cn}_k, r_c)$ where r_c is some freshly sampled randomness. Then $\mathcal{R}$ broadcasts c together with two proofs $(\pi_{\text{set-mem}}, \pi_{\text{rerand}})$ with the following semantics:

- $\pi_{\text{set-mem}}$: "I know (cn_k, r_c) such that $\mathsf{cn}_k \in \mathsf{Set}(A)$ and $c = \mathsf{Comm}^{\square}(\mathsf{cn}_k, r_c)$".
- π_{rerand}: "I know (cn_k, r_c, r) such that $\mathsf{cn}^* = \mathsf{cn}_k + \mathsf{Comm}^{\circ}(0; r)$ and $c = \mathsf{Comm}^{\square}(\mathsf{cn}_k, r_c)$".

In addition to the above, user $\mathcal{R}$ needs to prove knowledge of an actual opening of cn^* that refers to the revealed tag, identity $\mathcal{R}$ and some secret value v. In our concrete construction we use standard sigma protocols to prove knowledge of (v, r) such that $\mathsf{cn}^* = f^v g^{\mathsf{H}(\mathsf{t}||\mathcal{R})} h^r$, where H is a collision resistant function that maps to a valid exponent for g. In a sense we prove only a partial opening of cn^* since $g^{\mathsf{H}(\mathsf{t}||\mathcal{R})}$ can be subtracted publicly from cn^*.

We now discuss how we efficiently instantiate $\pi_{\text{set-mem}}$ and π_{rerand}. We choose to efficiently instantiate $\pi_{\text{set-mem}}$ with some of the components in [6]—which describes efficient commit-and-prove zkSNARKs over accumulated sets—and π_{rerand} with Bulletproofs, a transparent zero-knowledge schemes with short proofs that are compatible with some instantiations of [6].

We need some care in applying these techniques. Notice that the proof of rerandomization involves two different types of commitments in the statements (Comm° and $\mathsf{Comm}^{\square}$). Since their output may correspond to different groups, this can make it hard to *efficiently* instantiate the rerandomize relation for Bulletproofs. To solve these efficiency challenges we restrict what coins we can use in our systems (what coins are *permissible*) and describe a new SNARK-friendly curve whose arithmetic can be efficiently described as field operations when instantiating Bulletproofs over Ristretto25519.

Figure 3 illustrates a simplified version of our approach. We refer the reader to Section 5 for details.

2 Preliminaries

2.1 Notation

When describing an NP relation that we prove through a zero-knowledge argument, we use a semicolon to distinguish between public input and private witness as in $R(x; w)$. In the context of commitments we use a semicolon to distinguish between the committed value and the masking randomness as in $\mathsf{Comm}(\mathsf{ck}, u; r)$ where u is the committed value.

We assume all cryptographic algorithms implicitly take an input their respective public parameters whenever this yields no unambiguity, For example we may write $\mathsf{Comm}(u; r)$ to denote $\mathsf{Comm}(\mathsf{ck}, u; r)$ whenever ck is obvious from the context.

Universal Composability: We denote by $\Diamond$ compositions of UC functionalities and protocols e.g. $\Pi_A \Diamond \mathcal{F}_B$ denotes the protocol A in the B-hybrid model. We denote by $A \geq B$ that 'A implements B', i.e. the exists an efficient simulator Sim_A st. $A \overset{c}{\approx} \mathsf{Sim}_A \Diamond B$ for any enviroment; thoughout this paper we only consider PPT enviroments.

2.2 Commitments

We use the following syntax for commitments:

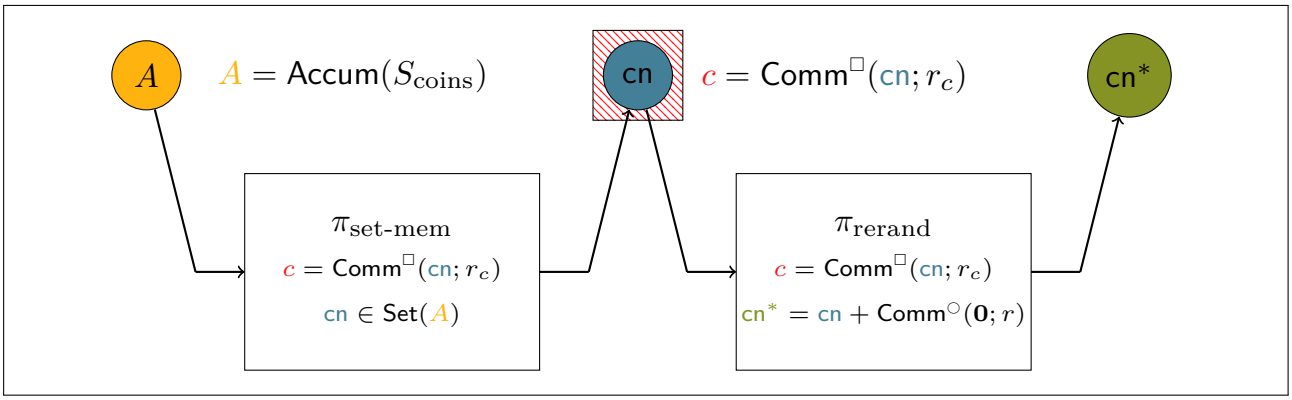

Fig. 3. The $\pi_{\text{1-many}} = (\pi_{\text{set-mem}}, \pi_{\text{rerand}})$ argument consists of two sub-proofs: a) A set membership proof, proving that $c = \mathsf{Comm}^{\square}(\mathsf{cn}; r_c)$ commits to $\mathsf{cn} \in A$. b) A rerandomization proof, showing that the same $c = \mathsf{Comm}^{\square}(\mathsf{cn}; r_c)$ commits to $\mathsf{cn} = \mathsf{cn}^* - \mathsf{Comm}^{\circ}(\mathbf{0}; r)$ where cn^* is a part of the statement (public).

Definition 1 (Commitments)**.** *A commitment scheme* C *is a pair of algorithms* $(\mathsf{Setup}, \mathsf{Comm})$ *with syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$: *generates a commitment key* ck*;*
- $\mathsf{Comm}(\mathsf{ck}, m; r) \to c_m$: *produces commitment* com_m *to message* m *with randomness* r*.*

As it is standard, we call *message space* the set of of m-s for which Comm is defined and commitment space its range, $\mathsf{Rng}(\mathsf{Comm})$. We require commitments to be *perfectly hiding*—the distribution of $\mathsf{Comm}(\mathsf{ck}, m; r)$ is identical to the uniform distribution over the commitment space—and *computationally binding*—no efficient adversary can produce two pairs $(m, r), (m', r')$ such that $m \neq m'$ and $\mathsf{Comm}(\mathsf{ck}, m; r) = \mathsf{Comm}(\mathsf{ck}, m'; r')$. Sometimes we want to require binding only with respect to messages from a set $\mathcal{P}$ of *permissible* messages, a subset of the message space. In that case we say the scheme is binding "with respect to set $\mathcal{P}$".

2.3 Accumulators

Definition 2 (Accumulator scheme)**.** *An accumulator scheme* Acc *over universe* $\mathcal{U}_\lambda(\mathsf{Acc})$ *(where* λ *is a security parameter) consists of a quadruple of PPT algorithms* $\mathsf{Acc} = (\mathsf{Setup}, \mathsf{Accum}, \mathsf{PrvMem}, \mathsf{VfyMem})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp})$ *generates public parameters* pp*.*

$\mathsf{Accum}(\mathsf{pp}, S) \to A$ *deterministically computes accumulator* A *for set* $S \subseteq \mathcal{U}_\lambda(\mathsf{Acc})$*.*

$\mathsf{PrvMem}(\mathsf{pp}, S, x) \to W$ *computes witness* W *that proves* x *is in accumulated set* S*.*

$\mathsf{VfyMem}(\mathsf{pp}, A, x, W) \to b \in \{0, 1\}$ *verifies through witness whether* x *is in the set accumulated in* A*. We do not require parameter* x *to be in* $\mathcal{U}_\lambda(\mathsf{Acc})$ *from the syntax.*

An accumulator scheme should satisfy correctness—the accumulator works as expected—and soundness—no efficient adversary can choose a set S *and then find a witness that checks on* $\mathsf{Acc.Accum}(\mathsf{pp}, S)$ *and* $u \notin S$[6]*.*

Remark 1 (Efficient Insertion)**.** *Throughout this work we assume an additional (deterministic) algorithm* $\mathsf{Acc.Add}$ *for a scheme* Acc *such that for all* $\lambda, x \in \mathcal{U}_\lambda(\mathsf{Acc}), S \subseteq \mathcal{U}_\lambda(\mathsf{Acc})$ $A' = \mathsf{Acc.Add}(A, x)$ *is such that, if* $A = \mathsf{Acc.Accum}(S)$ *then* $A' = \mathsf{Acc.Accum}(S \cup \{x\})$*.*

2.4 NIZKs

In this work we use and assume *transparent* NIZKs, i.e. whose algorithms use a reference string urs sampled uniformly.

Definition 3. *A NIZK for a relation family* $\mathfrak{R} = \{\mathfrak{R}_\lambda\}_{\lambda \in \mathbb{N}}$ *is a tuple of algorithms* $\mathsf{ZK} = (\mathsf{Prove}, \mathsf{VerProof})$ *with the following syntax:*

- $\mathsf{ZK.Prove}(\mathsf{urs}, R, x, w) \to \pi$ *takes as input a string* urs*, a relation description* R*, a statement* x *and a witness* w *such that* $R(x, w)$*; it returns a proof* π*.*

6 These definitions are standard and we refer the reader to [3] for a formal treatment.

– $\mathsf{ZK.VerProof}(\mathsf{urs}, R, x, \pi) \to b \in \{0,1\}$ *takes as input a string* urs*, a relation description* R*, a statement* x *and a proof* π*; it accepts or rejects the proof.*

We require a NIZK to be complete, that is, for any $\lambda \in \mathbb{N}, R \in \mathfrak{R}$ *and* $(x,w) \in R$ *it holds with overwhelming probability that* $\mathsf{VerProof}(\mathsf{urs}, R, x, \pi)$ *where* $\mathsf{urs} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{poly}(\lambda)}$ *and* $\pi \leftarrow \mathsf{Prove}(\mathsf{urs}, R, x, w)$.

We also require knowledge-soundness and zero-knowledge to hold. Informally, the former states we can efficiently "extract" a valid witness from a proof that passes verification; the latter states that the proof leaks nothing about the witness (this is modeled through a simulator that can output a valid proof for an input in the language without knowing the witness). We use variants of these notions with certain composability properties, e.g. requiring auxiliary inputs and relation generators. For a full formal treatment of these, we refer the reader to Sections 2.2 and 2.5 in [6].

Whenever the relation family is obviously defined, we talk about a "NIZK for a relation R".

Remark 2 (Relations and Public Inputs)**.** *In the algorithms above we have both a relation* R *and a public input* x *as inputs. The reason is that in a soundness experiment,* R *may be constrained to be from a certain distribution on* $\mathfrak{R}$ *whereas* x *can be be chosen arbitrarily by the adversary. See for example Section 2.2 in [6]. In our constructions we often assume prover and verifier to implicitly take as input the relation description*[7]*.*

In the proof of security of our construction we require an additional property for one of our NIZKs, *simulation-extractability*. Namely, extractability should hold even with respect to an adversary that has access to simulated proofs. We refer the reader to [18] for formal definitions.

Trusted Accumulator-Model. In our concrete constructions we will use NIZKs for relations parametrized by accumulators. This requires a tweak in the soundness definition: a malicious adversary should be able to select an arbitrary set, but the accumulator over that set should be computed honestly. Given an accumulator scheme Acc, we informally talk about this specific notion as "security under the Trusted Accumulator-Model for Acc". We do not provide formal details since this model corresponds to the notion of partial-extractable soundness in Section 5.2 in [6][8]; we refer the reader to this work for further details. This weaker model fits our applications where an accumulator of existing coins is maintained by the network.

Modular NIZKs through Commit-and-Prove. We use the framework for black-box modular composition of commit-and-prove NIZKs (or CP-NIZKs) in [12] and [6]. Informally is a CP-NIZK is a NIZK that can efficiently prove properties of committed inputs through some commitment scheme $\mathsf{C}^{\square}$. Let x be a public input and $c_{\square}$ a commitment. Such a scheme can for example prove knowledge of (u, ω, r) such that $c_{\square} = \mathsf{Comm}^{\square}(u; r)$ and that relation $R_{\text{inner}}(x; u, \omega)$ holds. We can think of ω as a non-committed part of the witness. Besides the proof, the verifier's inputs are x and $c_{\square}$.

In Section 5.2 we will make use of the following folklore composition to obtain efficient NIZKs from CP-NIZKs. Fixed a commitment scheme and given two CP-NIZKs $\mathsf{CP}, \mathsf{CP}'$ respectively for two "inner" relations R and R', we can prove their conjunction (for a shared witness u) $R^*(x, x', u, \omega, \omega') = R(x, u, \omega) \wedge R'(x', u, \omega')$ like this: the prover commits to u as $c_{\square} \leftarrow \mathsf{Comm}^{\square}(u, r)$; generates proofs π and π' from the respective schemes; it outputs combined proof $\pi^* := (c_{\square}, \pi, \pi')$. The verifier checks each proof over respective inputs $(x, c_{\square})$ and $(x', c'_{\square})$.

The following theorem (informally stated) is a direct consequence of Theorem 3.1 in [12].

Theorem 1 (Black-Box Composition of CP-NIZKs)**.** *The construction above is a secure NIZK for the conjunction relation* R^**.*

We can see Bulletproof [10] as a CP-NIZK since it works efficiently over an implicit commitment representation (this is further discussed in [12]). We use this fact in our instantiations in Section 5.

7 This parameter is usually short. For example, in Section 5.1 we let relations be described by a specific accumulator.

8 We notice that their model uses a slightly different language and formalizes accumulators as (binding-only) commitments for commit-and-prove NIZKs.

3 A Functionality for Anonymous Account-Based Payments

Here we describe our functionality for account-based payments with privacy requirements.

The functionality in Figure 4 captures both notions of anonymity (the adversary cannot link honest create/collects), 'balance conservation' (no money can be created in the system) and security of balances (the adversary cannot 'steal money' from honest parties) and "Faerie gold"-type attacks where a corrupted party can create multiple coins accepted by honest parties of which only a subset can be spent.

In our construction we will make use of an idealized communication $\mathcal{F}_{Comm}$ functionality described in Figure 5.

Remark 3 (Simplifications in $\mathcal{F}_{Comm}$)**.** *In practice the simultaneously delivery in $\mathcal{F}_{Comm}$ is impossible to implement, however we deliberately simplify the functionality since the omitted details in the modelling of the distributed ledger seem unlikely to affect the security of our anonymous transactions and it simplifies explication.*

Remark 4 (Implementation of $\mathcal{F}_{Comm}$)**.** *In practice the anonymous message delivery (of M) in $\mathcal{F}_{Comm}$ can be achieved by having the sender encrypt the message to the receiver using a public-key encryption scheme wherein the correct public key for a ciphertext is indistinguishable from a random public key. The same technique is used in Zerocash. Standard Elgamal encryption is one such scheme. The authenticated broadcast (of B) can be achieved by using digital signatures.*

The use of port ids in Figure 5 to identify which player sends a message avoids the explicit use of public keys in the constructions, however in practice $\mathcal{F}_{Comm}$ will be instantiated by identifying a peer by its public key and signing the broadcast messages.

$\mathcal{F}_{Anon}$

Initialize: On input $Balance, \mathcal{C}$ on infl:
1. Corrupt the players in $\mathcal{C}$.
2. Set $Coins \leftarrow [], Events \leftarrow []$.
3. Store the initial balances $Balance$.
4. Assert $\texttt{MAX-MONEY} \geq \sum_{i=1}^{n} Balance[i]$.

Create coin: Input (Create, i, v) on P_j:
1. Assert $Balance[j] \geq v$.
2. Set $Balance[j] \leftarrow Balance[j] - v$.
3. If $i \notin \mathcal{C}$ send (Create, j) on leak.
4. If $i \in \mathcal{C}$ send $(\mathsf{Create}, j, i, v)$ on leak.
5. Receive fresh id on infl.
6. Set $Events[id] \leftarrow (\mathsf{Create}, j, i, v)$

Collect coin: Input $(\mathsf{Collect}, id')$ on P_i:
1. Assert (id', v, j, i) in $Coins$.
2. Remove (id', v, j, i) from $Coins$.
3. If $j \notin \mathcal{C}$ send $(\mathsf{Collect}, i)$ on leak.
4. If $j \in \mathcal{C}$ send $(\mathsf{Collect}, id', j, i, v)$ on leak.
5. Receive fresh id on infl.
6. Set $Events[id] \leftarrow (\mathsf{Collect}, j, i, v)$

Process: Input $(\mathsf{Process}, id)$ on infl:
1. If $Events[id] = (\mathsf{Create}, j, i, v)$
   (a) Output (id, v) on P_i
   (b) Add (id, v, j, i) to $Coins$
2. If $Events[id] = (\mathsf{Collect}, j, i, v)$
   (a) Set $Balance[i] \leftarrow Balance[i] + v$.
   (b) Output $Balance[i]$ on P_i
3. Remove $Event[id]$

Fig. 4. Ideal functionality for account based anonymous transactions. The functionality enables the enviroment to learn when an account creates a transaction, but not the link between create/pickup unless the sender is corrupted. If an assertion is violated, the message is ignored and the state of functionality reverts to before receiving the message.

4 A Construction for Anonymous Account-Based Payments

In this section we provide our main construction. Our description tries to be as general as possible and to push all features that can be seen as optimizations to our instantiations in Section 5. The bulk of our construction is

$\mathcal{F}_{Comm}$

Initialize: Given $\mathcal{C}$ on infl
1. Set $msg \leftarrow []$
2. Corrupt the players in $\mathcal{C}$

Message: Input (Msg, B, M, d) on P_i:
1. If $d \in \mathcal{C}$ output $(\mathsf{Msg}, i, B, M, d)$ on leak.
2. If $d \notin \mathcal{C}$ output $(\mathsf{Msg}, i, B, |M|)$ on leak.
3. Receive fresh id on infl.
4. Add (id, i, B, M, d) to msg

Deliver: Input $(\mathsf{Deliver}, id)$ on infl:
1. Assert $(id, i, B, M, d) \in msg$
2. For $j \in [1, n]$:
   (a) If $d = j$ output (B, M) on P_j
   (b) If $d \neq j$ output B on P_j
3. Remove (id, i, B, M, d) from msg

Fig. 5. Models broadcast and private messages. The model of totally ordered broadcast above is very simplistic, in particular it assumes that every player receives the messages simultaneously and that messages have instant 'finality' (no 'rollbacks').

in Figure 6. We describe a more concrete and optimized version in Appendix A.

4.1 An Auxiliary Interface

We present our construction using the following intermediate syntax for a "Decentralized Unlinkable-Payments" scheme (DUP).

Definition 4. *A DUP scheme consists of a tuple of PPT algorithms* $\mathsf{DUP} = (\mathsf{Setup}, \mathsf{CreateCoin}, \mathsf{CollectCoin}, \mathsf{Vfy}, \mathsf{Process})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathcal{L}_0, (\mathsf{st}_i)_{i\in[m]})$ *Generates public parameters, an initial ledger* $\mathcal{L}_0$ *and the initial private state of all users;*

$\mathsf{CreateCoin}(\mathsf{pp}, \mathsf{st}_\mathcal{S}, \mathsf{pk}_\mathcal{R}, v) \to (\mathsf{st}'_\mathcal{S}, \mathsf{aux}_{\mathbf{coin}}, \mathsf{tx}_{\mathbf{create}})$ *Makes a coin* c *of value* v *payable to user* $\mathcal{R}$*; it embeds the coin in a public transaction* tx_{create}*;* aux_{coin} *contains information sent privately to* $\mathcal{R}$*; it also outputs a new private state* $\mathsf{st}'_\mathcal{S}$*.*

$\mathsf{CollectCoin}(\mathsf{pp}, \mathsf{st}_\mathcal{R}, \mathsf{aux}_{\mathbf{coin}}, \mathcal{L}) \to (\mathsf{st}'_\mathcal{R}, \mathsf{tx}_{\mathbf{clct}})$ *It takes as input a a private state* $\mathsf{st}_\mathcal{R}$*, a string* aux_{coin} *and a ledger* $\mathcal{L}$*; it outputs a new private state* $\mathsf{st}'_\mathcal{R}$ *and a "collect" transaction* tx_{clct}*.*

$\mathsf{Vfy}(\mathsf{pp}, \mathsf{tx}, \mathcal{L}) \to \mathsf{accept}/\mathsf{reject}$ *It verifies a transaction* tx *with respect to ledger* $\mathcal{L}$*.*

$\mathsf{Process}(\mathsf{pp}, \mathsf{tx}, \mathcal{L}) \to \mathcal{L}'$ *It processes a transaction* tx *with respect to ledger* $\mathcal{L}$ *and returns a new ledger.*

4.2 Building Blocks

Commitment schemes We assume a commitment scheme $\mathsf{C}^\bigcirc = (\mathsf{Setup}^\bigcirc, \mathsf{Comm}^\bigcirc)$. We concretely instantiate it later in Figure 8 as a Pedersen commitment over pairs. We also make the following assumptions:

- We assume that we can commit to pairs and they are homomorphic with respect to pairs, that is $\mathsf{Comm}((a,b);r) + \mathsf{Comm}((c,d);r') = \mathsf{Comm}((a+c, b+d); r+r')$.
- Given a value v, a tag t and an identity $\mathcal{R}$ we assume that the concatenation $t||\mathcal{R}$ is such that $(v, t||\mathcal{R})$ is always a pair in the message space of the commitment scheme.
- We assume all commitment invocations take as input the public commitment key ck although not explicitly included. We assume the same for zero-knowledge proofs over commitment in the protocol.

Zero-Knowledge Arguments We assume the following zero-knowledge arguments. We assume the commitment key to be part of the relation description. Although we keep it implicit, one should think of the following relations as parametrized by it.

We use three zero knowledge proofs for the following tasks:

- *Knowledge of opening:* at collection time, we prove knowledge of opening of a coin we are collecting. For technical reasons (see construction) we require the second component to be zero, that is this argument shows knowledge of opening to a pair $(v, 0)$.
- *Ranges:* whenever we transfer an amount ($\mathsf{CreateCoin}$), we prove that we can afford the transfer. We also prove that the transferred amount is non-negative (so that we are not subtracting a negative value from our balance increasing it!). This involves proving that the opening of two distinct commitments—an updated balance and a coin—are both in a range $[0, B_{\max}]$ where we consider $B_{\max}$ a parameter of the construction. Although the commitments we assume here bind to pairs of values (a, b) (rather than single value), we are interested only in ensuring that the first component a is in range.
- *One coin out of many:* whenever we claim an amount ($\mathsf{CollectCoin}$), we also need to prove that the coin we are collecting actually exists. We want to do

this without revealing that coin we are collecting, thus we prove that a rerandomized commitment c^* (which we revealed publicly) actually refers to "one out of many" in the set of existing coins.

Formally we assume the following NIZKs:
ZKOpen is an extractable NIZK argument for the relation

$$R_{\text{opn}}(\mathsf{c}; v, r) = 1 \iff \mathsf{c} = \mathsf{Comm}^{\circlearrowleft}((v, 0); r)$$

ZKDblRange is a NIZK argument for the relation

$$\begin{aligned} &R_{\text{DRng}}(\mathsf{c}, \mathsf{c}'; (a, b), r, (a', b'), r') = 1 \iff \\ &a \in [0, B_{\max}] \wedge a \in [0, B_{\max}] \\ &\wedge\ \mathsf{c} = \mathsf{Comm}^{\circlearrowleft}((a, b); r) \\ &\wedge\ \mathsf{c}' = \mathsf{Comm}^{\circlearrowleft}((a', b'); r') \end{aligned}$$

ZK-1-many is a simulation-extractable NIZK argument for the relation[9]

$$\begin{aligned} &R_{\text{1-many}}(\mathsf{c}^*, S; \mathsf{c}, r^*) = 1 \iff \\ &\mathsf{c} \in S \wedge\ \mathsf{c}^* = \mathsf{c} + \mathsf{Comm}^{\circlearrowleft}((0, 0); r^*) \end{aligned}$$

For simplicity we assume a single uniform reference string urs for all of them that can be sampled from a space large enough parametrized by the security parameter λ and the maximum balance $B_{\max}$

4.3 Construction Description

Ledger, states and transactions We assume a ledger to be structured as a triple $\mathcal{L} = (S_{\text{null}}, S_{\text{coins}}, (\mathsf{bal})_{i\in[m]})$ containing: a set S_{null} of "nullified" coin tag–recipient identity $t||\mathcal{R}$; an set S_{coins} of coins created so far; a tuple of commitments $(\mathsf{bal})_{i\in[m]}$ to the balances of parties. Within the construction we implicitly parse the ledger according to this syntax. We let states contain the opening of their committed balances. Notice that, for technical reasons, bal_i is not a commitment to a single scalar B representing the balance amount. Instead a private state st_i—the opening of bal_i—is a triple (B, aux_t, r_i) such that $\mathsf{bal}_i = \mathsf{Comm}((B, \mathsf{aux}_t), r_i)$, that is it opens to the pair (v, aux_t) where aux_t depends on the transfers that user i carried out till any given moment in time (see construction for details). Transactions can be of two types, Create or Collect; we prepend a type description to each transaction.

4.3.1 Implementing $\Pi_{Anon}\Diamond\mathcal{F}_{Comm}$ through the Interface in Figure 6

We do not formally describe the initialization stage. We assume that honest parties receive initial public parameters, ledger for a common initial balance v_0 and initial private states as described in Setup in Figure 6. The rest of the protocol looks as follows:

Create Coin: On input (Create, j, v) on $\Pi_{Anon}.\mathsf{P}_i$
1. Run $(\mathsf{st}_i, \mathsf{aux}_{\text{coin}}, \mathsf{tx}_{\text{create}}) \leftarrow \mathsf{CreateCoin}(\mathsf{pp}, \mathsf{st}_i, j, v)$
2. Broadcast $\mathsf{tx}_{\text{create}}$ and send coin privately by outputting $(\mathsf{Msg}, \mathsf{tx}_{\text{create}}, \mathsf{aux}_{\text{coin}}, \mathcal{R})$ on $\mathcal{F}_{Comm}.\mathsf{P}_{\mathcal{S}}$

Receive Coin: On input $(\mathsf{Msg}, j, \mathsf{tx}_{\text{create}}, \mathsf{aux}_{\text{coin}})$ on $\mathcal{F}_{Comm}.\mathsf{P}_i$:
1. Assert $\mathsf{Vfy}(\mathsf{pp}, \mathsf{tx}, \mathcal{L}) = 1$
2. Update $\mathcal{L} \leftarrow \mathsf{Process}(\mathsf{pp}, \mathsf{tx}_{\text{create}}, \mathcal{L})$
3. Parse $\mathsf{aux}_{\text{coin}}$ as (v, t, r_c)
4. If $t \in \mathsf{MyTags}$ return early.
5. Add t to MyTags.
6. Sample local id randomly.
7. Store $\mathsf{mycoins}[id] \leftarrow \mathsf{aux}_{\text{coin}}$
8. Output (id, v) on P_i

Collect Coin: On input $(\mathsf{Collect}, id')$ on $\Pi_{Anon}.\mathsf{P}_i$:
1. Assert $\exists$ entry $\mathsf{mycoins}[id']$
2. Update $(\mathsf{st}_i, \mathsf{tx}_{\text{clct}}) \leftarrow \mathsf{Collect}(\mathsf{st}_i, \mathsf{mycoins}[id'], \mathcal{L})$
3. Remove $\mathsf{mycoins}[id']$
4. Output $(\mathsf{Msg}, \mathsf{tx}_{\text{clct}}, \perp, \perp)$ on $\mathcal{F}_{Comm}.\mathsf{P}_{\mathcal{R}}$

Process Tx: On input $(\mathsf{Msg}, \mathcal{S}, \mathsf{tx})$ on $\mathcal{F}_{Comm}.\mathsf{P}_i$:
1. Assert $\mathsf{Vfy}(\mathsf{pp}, \mathsf{tx}, \mathcal{L}) = 1$
2. Update $\mathcal{L} \leftarrow \mathsf{Process}(\mathsf{pp}, \mathsf{tx}, \mathcal{L})$

Completeness of the construction above follows by observation; we prove its security in the Appendix in Section B.

5 Efficient Instantiations of Our Arguments

In this section we describe how to instantiate our construction from the previous section through transparent

9 Notice that, in contrast with the first two relations, relation $R_{\text{1-many}}$ does not require showing any opening of the commitments c and c^*. This implies that a honest prover does not need to know these openings. Although we do not use this property in our construction, this could be useful in efficiently delegating to a service (such as a wallet) that, for example, we trust enough not to publicly reveal which coin we are collecting, but enough not to steal our coin.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathcal{L}_0, (\mathsf{st}_i)_{i\in[m]})$

$\mathsf{ck} \leftarrow \mathsf{Setup}^\circ(1^\lambda)$
$\mathsf{urs} \leftarrow\$\, U_{\mathrm{ZK}}(1^\lambda, B_{\max})$
for $i = 1..m$ **do**
  // Create balances for all parties at default v_0
  $r_i \leftarrow\$\, \mathbb{F}$
  $\mathsf{bal}_i \leftarrow \mathsf{Comm}^\circ(v_0; r_i)$
  $\mathsf{st}_i := (v_0, 0, r_i)$
$\mathcal{L}_0 := \big(\emptyset, \emptyset, (\mathsf{bal}_i)_{i\in[m]}\big)$
return $\big(\mathsf{pp} := \mathsf{ck}, \mathcal{L}_0, (\mathsf{st}_i)_{i\in[m]}\big)$

$\mathsf{CreateCoin}(\mathsf{pp}, \mathsf{st}_\mathcal{S}, \mathcal{R}, v) \to (\mathsf{st}'_\mathcal{S}, \mathsf{aux}_{\mathrm{coin}}, \mathsf{tx}_{\mathrm{create}})$

Parse $\mathsf{st}_\mathcal{S}$ as $(B, \mathsf{aux}_t, r_\mathcal{S})$
$t \leftarrow\$\, \{0,1\}^\lambda;\ r_c \leftarrow\$\, \mathbb{F}$
$\mathsf{c} \leftarrow \mathsf{Comm}^\circ((v, t||\mathcal{R}); r_c)$
$o_c := ((v, t||\mathcal{R}), r_c)$
// Update balance subtracting coin from current balance
$\mathsf{bal}'_i \leftarrow \mathsf{Comm}^\circ((B, \mathsf{aux}_t); r_\mathcal{S}) - \mathsf{c}$
$o' := ((B - v, \mathsf{aux}_t - t||\mathcal{R}), r_\mathcal{S} - r_c)$
// Prove coin amount is positive and that $\mathcal{S}$ can afford it
$\pi_{\mathrm{create}} \leftarrow \mathsf{ZKDblRange.Prove}(\mathsf{urs}, \mathsf{c}, \mathsf{bal}'; o_c, o')$
$\mathsf{st}'_\mathcal{S} := (B - v, \mathsf{aux}_t - t||\mathcal{R}, r_\mathcal{S} - r_c)$
Let $\mathsf{aux}_{\mathrm{coin}} := (v, t, r_c)$
$\mathsf{tx}_{\mathrm{create}} := (\mathsf{Create}, \mathsf{c}, \pi_{\mathrm{create}})$
return $(\mathsf{st}'_\mathcal{S}, \mathsf{aux}_{\mathrm{coin}}, \mathsf{tx}_{\mathrm{create}})$

$\mathsf{CollectCoin}(\mathsf{pp}, \mathsf{st}_\mathcal{R}, \mathsf{aux}_{\mathrm{coin}}, \mathcal{L}) \to (\mathsf{st}'_\mathcal{R}, \mathsf{tx}_{\mathrm{clct}})$

Parse $\mathsf{st}_\mathcal{S}$ as $(B, \mathsf{aux}_t, r_\mathcal{R})$
// reconstruct coin
Parse $\mathsf{aux}_{\mathrm{coin}}$ as (v, t, r_c)
$\mathsf{c} \leftarrow \mathsf{Comm}^\circ((v, t||\mathcal{R}); r_c)$
// Rerandomize coin
$r^* \leftarrow\$\, \mathbb{F};\ \mathsf{c}^* \leftarrow \mathsf{c} + \mathsf{Comm}^\circ((0,0); r^*)$
// Prove one out of many w.r.t. accumulator
$\pi_{\text{1-many}} \leftarrow \mathsf{ZK\text{-}1\text{-}many.Prove}(\mathsf{urs}, \mathsf{c}^*, S_{\mathrm{coins}}; r^*)$
// Prove "partial" opening of c^*
$c_v \leftarrow \mathsf{c}^* - \mathsf{Comm}^\circ((0, t||\mathcal{R}); 0)$
$\pi_{\mathrm{opn}} \leftarrow \mathsf{ZKOpen.Prove}(\mathsf{urs}, c_v; v, r_c + r^*)$
$\mathsf{st}'_\mathcal{R} := (B + v, \mathsf{aux}_t + t||\mathcal{R}, r_\mathcal{R})$
$\mathsf{tx}_{\mathrm{clct}} := (\mathsf{Collect}, \mathsf{c}^*, t, \mathcal{R}, \pi_{\mathrm{clct}} := (\pi_{\text{1-many}}, \pi_{\mathrm{opn}}))$
return $(\mathsf{st}'_\mathcal{R}, \mathsf{tx}_{\mathrm{clct}})$

$\mathsf{Process}(\mathsf{pp}, \mathsf{tx}, \mathcal{L}) \to \mathcal{L}'$

Run $\mathsf{Vfy}(\mathsf{pp}, \mathsf{tx}, \mathcal{L})$ and abort if it fails
if $\mathsf{type}(\mathsf{tx}) = \mathsf{Create}$ **then**
  Parse tx as $(\mathsf{Create}, \mathsf{c}, \pi_{\mathrm{create}})$
  // add coin to set
  $S'_{\mathrm{coins}} \leftarrow S_{\mathrm{coins}} \cup \{\mathsf{c}\}$
  // homomorphically update balance of sender
  $\mathsf{bal}'_\mathcal{S} \leftarrow \mathsf{bal}_\mathcal{S} - \mathsf{c}$
elseif $\mathsf{type}(\mathsf{tx}) = \mathsf{Collect}$ **then**
  Parse tx as $(\mathsf{Collect}, \mathsf{c}^*, t, \mathcal{R}, \pi_{\mathrm{clct}})$
  // add tag to nullifier set
  $S'_{\mathrm{null}} \leftarrow S_{\mathrm{null}} \cup \{t||\mathcal{R}\}$
  // homomorphically update balance of receiver
  $c_v \leftarrow \mathsf{c}^* - \mathsf{Comm}^\circ((0, t||\mathcal{R}); 0)$
  $\mathsf{bal}'_\mathcal{R} \leftarrow \mathsf{bal}_\mathcal{R} + c_v$
Let $\mathcal{L}'$ be $\mathcal{L}$ updated with new set and balance
return $\mathcal{L}'$

$\mathsf{Vfy}(\mathsf{pp}, \mathsf{tx}, \mathcal{L}) \to \mathsf{accept/reject}$

if $\mathsf{type}(\mathsf{tx}) = \mathsf{Create}$ **then**
  Parse tx as $(\mathsf{Create}, \mathsf{c}, \pi_{\mathrm{create}})$
  Assert $\mathsf{ZKDblRange.Vfy}(\mathsf{urs}, \mathsf{c}, \mathsf{bal}_\mathcal{S} - \mathsf{c}, , \pi_{\mathrm{create}}) = 1$
elseif $\mathsf{type}(\mathsf{tx}) = \mathsf{Collect}$ **then**
  Parse tx as $(\mathsf{Collect}, \mathsf{c}^*, t, \mathcal{R}, \pi_{\mathrm{clct}} := (\pi_{\text{1-many}}, \pi_{\mathrm{opn}}))$
  Assert $\mathsf{ZK\text{-}1\text{-}many.Vfy}(\mathsf{urs}, \mathsf{c}^*, S_{\mathrm{coins}}, \pi_{\text{1-many}}) = 1$
  $c_v \leftarrow \mathsf{c}^* - \mathsf{Comm}^\circ((0, t||\mathcal{R}); 0)$
  Assert $\mathsf{ZKOpen.Vfy}(\mathsf{urs}, c_v) = 1$
  Assert $t||\mathcal{R} \notin S_{\mathrm{null}}$

Fig. 6. Procedures describing the bulk of our construction; we use them as auxiliary syntax when we show our construction in more detail in Section 4.3.1. These procedures have no side-effects (except for sampling randomness) and return pure functions of their inputs.

and efficient proof systems. We describe this at different levels of abstraction.

First we replace a set (of coins) with a compressed representation, an accumulator. Thus we replace relation $R_{\text{1-many}}$ with $R^A_{\text{1-many}}cc$ that works over an accumulator A. We then proceed how to decompose the latter efficiently through a commit-and-prove approach. In the rest of the section we describe our specific instantiations using: RSA accumulators, zero-knowledge techniques on them from [6], and Bulletproofs.

Our main technical challenge is how to have commitments over coins (which are themselves commitments) that support efficient proofs over them. We do this introducing a new SNARK-friendly curve (in the pairing-free group of `Curve25519`) and embedding its arithmetic in a Bulletproof relation in an optimized manner (see also Appendix C).

On notation. We will use and describe two commitments schemes in this section, C° and $\mathsf{C}^{\square}$. The scheme C° is the scheme we use in our construction for payments in the previous section; we can think of its output as coins and we denote them by a circle as in $c_{\circ}$. The elements of the accumulated set are the output of Comm°. The commitment scheme $\mathsf{C}^{\square}$ is the one we use for commit-and-prove NIZKs (see also construction for Theorem 1). We denote its output as $\mathsf{Comm}^{\square}$.

5.1 One-out-of-many Relations over Accumulators

Here we define a variant of the one-out-of-many relation $R_{\text{1-many}}$ introduced in Section 4.2. Instead of taking as input a set we let the relation be parametrized by an accumulator, a binding commitment to the set. Thus we can reduce prover and verifier's complexity to that of proving PrvMem and VfyMem which both run in constant time in our instantiation.

Given an accumulator scheme Acc and an accumulator A, the relation $R^A_{\text{1-many}}$ is defined as:

$$
\begin{aligned}
&R^A_{\text{1-many}}(c^*_{\circ}; c_{\circ}, r^*_{\circ}, W) = 1 \iff \\
&\mathsf{Acc.VfyMem}(A, \mathsf{c}, W) \\
&\wedge\ c^*_{\circ} = c_{\circ} + \mathsf{Comm}^{\circ}((0,0); r^*_{\circ})
\end{aligned}
$$

5.2 One-out-of-many from Commit-and-Prove NIZKs

Here we use the construction in Theorem 1. For that we need CP-NIZKs that work over commitments to $c_{\circ}$. As usual we denote the commitment scheme for CP-NIZKs as $\mathsf{C}^{\square}$.

Permissible Set. We assume a permissible set $\mathcal{P}$ of coins $c_{\circ}$. This allows us to model security requirements in a fine-grained way, e.g. we assume computational binding of $\mathsf{C}^{\square}$ to hold only for coins in $\mathcal{P}$ and similarly the soundness of the accumulator (we ensure this implicitly; see Figure 7). In this section we keep the permissible set abstract but we specify it completely in Section 5.3.1.

Breaking Down $R^A_{\mathbf{setmem}}$. We can decompose the above through two commit-and-prove schemes for the following two relations. The first one proves set membership, but does not guarantee that the coin is permissible (this is for technical reasons we explain in Section 5.4). The other relation guarantees that we can open (in $\mathsf{C}^{\square}$) to a rerandomized permissible commitment (in C°).

$$
\begin{aligned}
&R^A_{\text{setmem}}(c_{\square}; c_{\circ}, r_{\square}, W) = 1 \iff \\
&((\mathsf{Acc.VfyMem}(A, c_{\circ}, W) \\
&\vee\ c_{\circ} \notin \mathcal{P})) \\
&\wedge\ c_{\square} = \mathsf{Comm}^{\square}(c_{\circ}; r_{\square})
\end{aligned}
$$

$$
\begin{aligned}
&R_{\text{rrnd\&prms}}(c_{\square}; c_{\circ}, r_{\square}, r^*_{\circ}) = 1 \iff \\
&c^*_{\circ} = c_{\circ} + \mathsf{Comm}^{\circ}((0,0); r^*_{\circ}) \\
&\wedge\ c_{\circ} \in \mathcal{P} \\
&\wedge\ c_{\square} = \mathsf{Comm}^{\square}(c_{\circ}; r_{\square})
\end{aligned}
$$

We can now obtain a proof scheme for $R^A_{\text{1-many}}$ by composing $\mathsf{ZKCP}^A_{\text{setmem}}$ and $\mathsf{ZKCP}_{\text{rrnd\&prms}}$ and applying Theorem 1 in the Trusted-Accumulator Model[10].

Corollary 1. *Let* Acc *be an accumulator scheme and* $\mathsf{C}^{\square}$ *a commitment scheme that is computationally binding w.r.t. set* $\mathcal{P}$*. Then the composition of* ZKCP^A_{setmem} *and* $\mathsf{ZKCP}_{rrnd\&prms}$ *as for Theorem 1 is a NIZK for* $R^A_{1\text{-}many}$ *in the Trusted-Accumulator Model for* Acc*.*

10 The latter requires the composition results for "partially-extractable" NIZKs in [6].

5.3 Instantiating Accumulators and Commitment Schemes

We first describe our accumulator and commitment schemes construction and then describe the concrete groups in which they operate in Section 5.3.1.

We assume a group of unknown order $\mathbb{G}_?$ for our accumulator construction. For our commitments we assume two groups $\mathbb{G}^\circ$ and $\mathbb{G}^\square$. The two groups are related as we assume we can represent $\mathbb{G}^\circ$ elements as pairs $(\mathbb{F}_{|\mathbb{G}^\square|}, \mathbb{F}_{|\mathbb{G}^\square|})$ (see also Section 5.3.1).

Our accumulator schemes supports sets of $\mathbb{G}^\circ$ elements with a special structure (the first component should be prime, the standard encoding for elements in accumulators in groups of unknown order). The commitment scheme Comm° has as message space pairs $(\mathbb{F}_{|\mathbb{G}^\circ|}, \mathbb{F}_{|\mathbb{G}^\circ|})$ and commitment space $\mathbb{G}^\circ$. The scheme $\mathbb{G}^\square$ has as message space elements in $\mathbb{G}^\circ$.

Constraints on Permissible Set. The permissible set $\mathcal{P}$ is a set of pairs in $(\mathbb{F}_{|\mathbb{G}^\square|}, \mathbb{F}_{|\mathbb{G}^\square|}$. We require that the permissible set $\mathcal{P}$ is such that there are no "collisions in the second components" that is: for all $(\mathbb{x}, \mathbb{y}) \in \mathcal{P}$ there exists no $\mathbb{y}' \neq \mathbb{y}$ such that $(\mathbb{x}, \mathbb{y}') \in \mathcal{P}$. This is the case for permissible sets over elliptic curves such as the one we define in Section 5.3.1.

Constructions. We now proceed to describe our constructions for accumulators and commitments. We denote by $\mathcal{G}$ a group generation function which we assume returns a group description together with a generator.

In Figure 7 we describe our accumulator instantiation. This construction is secure under under the Strong-RSA assumption[11] and is based on the construction from Barić and Pfitzmann [2], later used in the context of efficient proofs in [6, 8, 11] among other works. In the accumulator construction we describe explicitly the structure of the messages (elements in $\mathbb{G}^\circ$) as pairs of components and we accumulate using first component only. Notice that we describe the construction through a variant of the syntax in the preliminaries (Definition 2): we define only an insertion algorithm and let the setup return an accumulator A_0 to an empty set. We assume the Strong-RSA property holds for $\mathbb{G}_?$ (and its group generation algorithm).

The commitment schems C° and $\mathsf{C}^\square$ are described in Figure 8. They are both standard Pedersen commitments, but we make the following tweaks: in C° the messages are pairs; in $\mathsf{C}^\square$ we describe explicitly the structure of the messages (elements in $\mathbb{G}^\circ$) as pairs of components and we commit to the first component discarding the second. Recall that we can do this in light of the constraint on the second component from $\mathcal{P}$. We assume that the discrete-log assumption holds for $\mathbb{G}^\circ$ and $\mathbb{G}^\square$.

We do not prove security of the schemes in Figures 7 and 8 since it is standard.

Theorem 2 (Security of Schemes in Figures 7 and 8)**.**

- *If the Strong-RSA assumption holds for $\mathcal{G}_?$ then the construction in Figure 7 is a secure accumulator for sets $S \subset \mathcal{P}$ where all the $(\mathbb{x}, \mathbb{y}) \in S$ have all distinct primes $\mathbb{x}$.*
- *If the DLOG assumption holds for $\mathcal{G}_\circ$ (resp. $\mathcal{G}_\square$) then C° (resp. $\mathsf{C}^\square$) is a computationally binding (resp. binding w.r.t $\mathcal{P}$) and perfectly hiding commitment scheme with message space $(\mathbb{F}_{|\mathbb{G}^\circ|}, \mathbb{F}_{|\mathbb{G}^\circ|})$ (resp. $\mathbb{G}^\circ$) and commitment space $\mathbb{G}^\circ$ (resp. $\mathbb{G}^\square$).*

Remark 5 (Accumulators without Trapdoors)**.** *We observe that our accumulator scheme construction can be instantiated in class groups [9] or constructions based on hyperelliptic curve constructions [15, 26] assuming the Low-Order Assumption holds for $\mathcal{G}_?$. We also refer the reader to Appendix E in [6].*

Remark 6 (Trapdoors in RSA Groups and MPC)**.** *We note that there exist practical MPC protocols to securely construct RSA moduli, e.g., [13].*

5.3.1 Group Instantiations and Set of Permissible Coins

We now describe concrete instantiations targeting 128-bits of security.

Group $\mathbb{G}^\square$ The group $\mathbb{G}^\square$—used in our commit-and-prove NIZKs— is `Ristretto25519`, the Ristretto subgroup of `Curve25519`[12].

Group $\mathbb{G}^\circ$ and the Jabberwock Curve The group $\mathbb{G}^\circ = \mathbb{E}(\mathbb{F}_{|\mathbb{G}^\square|})$—used to represent coins and other commitments in our constructions— is derived from an elliptic curve over the scalar field $\mathbb{F}^\circ = \mathbb{F}_{|\mathbb{G}^\square|}$ of the curve

11 See [8], Definition 2.

12 https://ristretto.group/

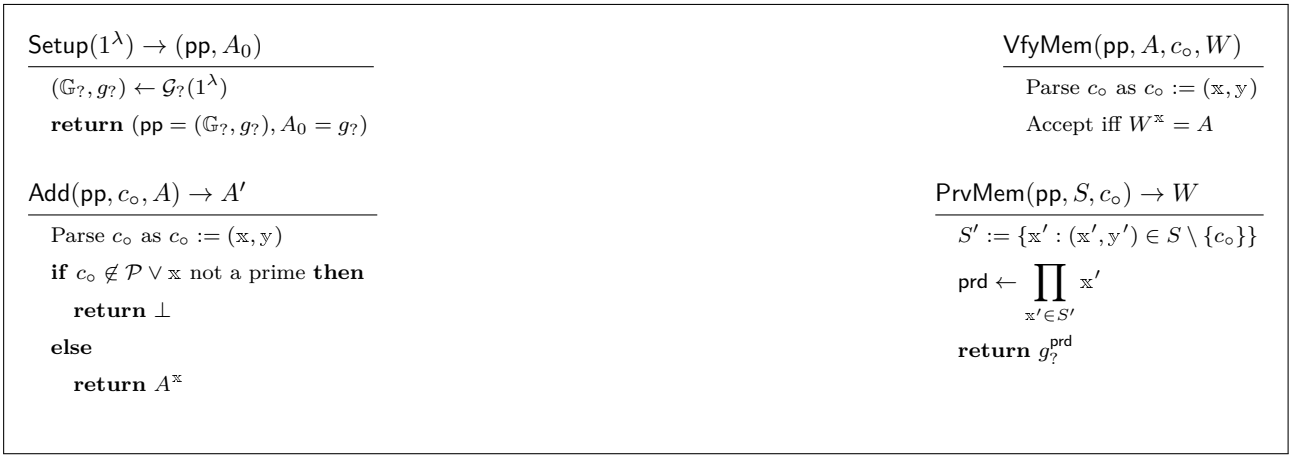

Fig. 7. Accumulator Instantiation for Acc.

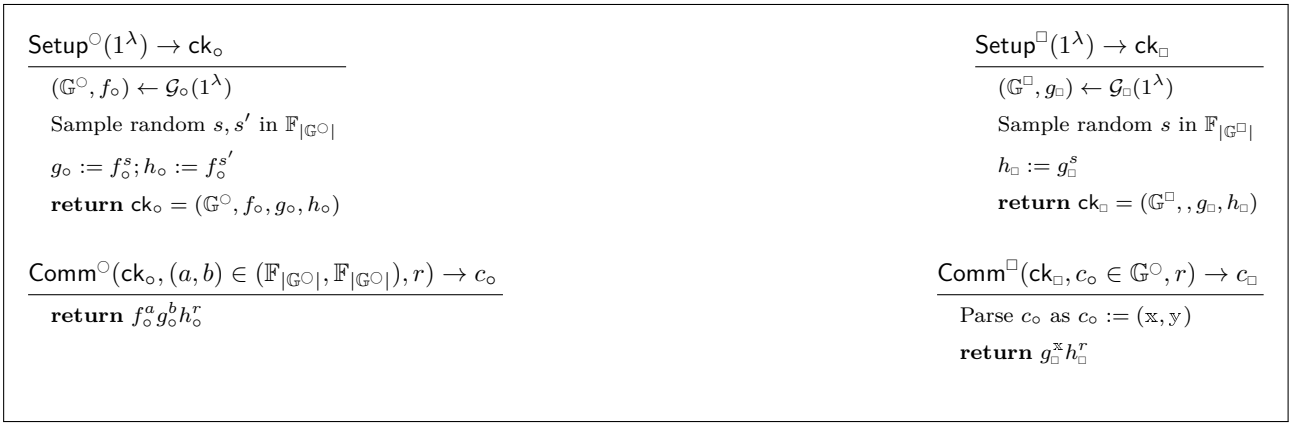

Fig. 8. Commitment Instantiations for C° and $\mathsf{C}^\square$.

$\mathbb{G}^\square$ (`Ristretto25519`) with:

$|\mathbb{G}^\square| = 2^{252} + 27742317777372353535851937790883648493$

In particular, we instantiate $\mathbb{G}^\circ$ as the Edwards curve [7, 16] with equation:

$$x^2y^2 = 1 - 698x^2y^2$$

The curve has a cofactor of 4 and a prime order group of $2^{250} - 28148165643402996844773726717916548891$.

Similar techniques has previously been used in the CØCØ [24] framework and Zcash [22] (JubJub curve[13]).

Group $\mathbb{G}_?$ The group $\mathbb{G}_?$ is a 2048-bit RSA group.

Permissible Set The set $\mathcal{P}$ of commitments, parametrized by an integer μ, consists of points on $\mathbb{G}^\circ$, where the $\mathtt{x}$-coordinate is a μ-bit prime and the $\mathtt{y}$-coordinate is the "canonically chosen" square root so that the point can be described by its $\mathtt{x}$-coordinate alone.

$$\mathcal{P} = \{(\mathtt{x}, \mathtt{y}) \in \mathbb{G}^\circ \subseteq (\mathbb{F}_{|\mathbb{G}^\square|}, \mathbb{F}_{|\mathbb{G}^\square|}) \mid \mathtt{x} \in [2^{\mu-1}, 2^\mu) \wedge \mathtt{y} \equiv 0 \mod 2\}$$

For our concrete instantiations we use $\mu = 251$ bits. We note that the results in Section 5.2 hold for any definition of $\mathcal{P}$ (with the collision constraint on the second component described earlier). Other choices of μ are also possible if one appropriately changes other parameters in the instantiations.

13 https://z.cash/technology/jubjub/

5.4 Instantiating Our ZK Building Blocks

5.4.1 $\mathsf{ZKCP}_{\mathsf{rrnd\&prms}}$

We instantiate $\mathsf{ZKCP}_{\mathrm{rrnd\&prms}}$ (Section 5.2) with a Bulletproof relation described in Appendix in Section C.

5.4.2 $\mathsf{ZKCP}^{\mathcal{A}}_{\mathsf{setmem}}$

We instantiate $\mathsf{ZKCP}^{A}_{\mathrm{setmem}}$ (Section 5.2) through a simplified variant of the (commit-and-prove) SNARK for set membership in [6], described below.

$\mathsf{ZKCP}^{A}_{\mathrm{setmem}}.\mathsf{Prove}(c_{\square}; c_{\circ}, r, W) \to \pi^*$

Parse $c_{\circ}$ as $c_{\circ} := (\mathtt{x}, \mathtt{y})$

// Make integer commitment to $\mathtt{x}$

Sample randomness r'

$c_{\mathrm{int}} \leftarrow g_{?}^{\mathtt{x}} h_{?}^{r'}$

$\pi_{\mathsf{modEq}} \leftarrow \mathsf{CP}_{\mathsf{modEq}}.\mathsf{Prove}(c_{\mathrm{int}}, c_{\square}; \mathtt{x}, r', r)$

$\pi_{\mathsf{root}} \leftarrow \mathsf{CP}^{A}_{\mathsf{root}}.\mathsf{Prove}(c_{\mathrm{int}}; \mathtt{x}, r, W)$

return $\pi^* := (c_{\mathrm{int}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{root}})$

The corresponding verifier checks both proofs using c_{int} and the rest of the public input.

Above we use an integer commitment in the RSA group $\mathbb{G}_?$ using an appropriately sampled element $h_?$. The proof system $\mathsf{CP}_{\mathsf{modEq}}$ roughly shows knowledge of integers x, r' and of $\mathtt{x}, r$ such that $x \equiv \mathtt{x} \mod |\mathbb{F}^{\square}|$, $c_{\mathrm{int}} = g_{?}^{\mathtt{x}} h_{?}^{r'}$ and $c_{\square} = g_{\square}^{\mathtt{x}} h_{\square}^{r}$. The scheme $\mathsf{CP}^{A}_{\mathsf{root}}$ proves knowledge of $W \in \mathbb{G}_?$, an integer x that opens c_{int} as above and such that W is a x-root for the accumulator A (this is roughly $\mathsf{Acc.VfyMem}$), that is $W^x = A$.

Above we skip some technical details from that are not relevant to understand our construction at a high-level. We however elaborate on one of them that is important in our larger context: the full scheme in [6] crucially relies on $\mathtt{x}$ being in some correct range. Without this guarantee on range, the construction above does not prove set membership w.r.t. A for elements that are not permissible (that is why we have "$\vee c_{\circ} \notin \mathcal{P}$" in R^{A}_{setmem}). On the other hand, once we prove $c_{\circ} = (\mathtt{x}, \mathtt{y})$ is permissible through $\mathsf{ZKCP}_{\mathrm{rrnd\&prms}}$, we ensure $\mathtt{x}$ is in range, and our scheme is secure as of the analysis in [6]. For further details and a proof of the following theorem, we refer to Section 4 in [6].

Theorem 3. *The construction above is a NIZK for the relation R^{A}_{setmem} (Section 5.2) in the Trusted-Accumulator Model for accumulator scheme* Acc *in Figure 7.*

5.4.3 ZKOpen

We instantiate ZKOpen from Section 4.3.1 with a Schnorr proof. On public input $c_{\circ}$ (the coin) and a commitment key $\mathsf{ck}_{\circ}$ for C° containing $f_{\circ}$ and $h_{\circ}$, the schemes proves (in zero-knowledge) knowledge of (v, r) such that $c_{\circ} = f_{\circ}^{v} h_{\circ}^{r}$. This protocol is very standard and we do not describe it in further details here.

5.4.4 ZKDblRange

We instantiate $\mathsf{ZKDblRange}$ with Bulletproofs.

6 Evaluation

6.1 Performance of ZK-1-many

We implemented our instantiation of $\mathsf{ZK\text{-}1\text{-}many}$ in Rust and experimentally evaluate its performance. Our code is open source and available at [1]. The performance of our implementation of $\mathsf{ZK\text{-}1\text{-}many}$ is shown in Figure 9.

Proof Size	5309 B
Proving Time	460 ms
Verification Time	93 ms

Fig. 9. Concrete performance of ZK-1-many from our Rust implementation. All benchmarking is done on a single core of AMD EPYC 7601 (@ 2.2 GHz).

6.2 Size of 𝕍eksel

$\pi_{\mathsf{1\text{-}many}}$	5309 B
π_{create}: Bulletproof with 128 constrains.	736 B
π_{opn}: Schnorr proof with 2 generators.	128 B
t: Spending tag	16 B
$\mathsf{Enc}_{\mathcal{R}}(\mathsf{t}\|\mathsf{v}\|\mathsf{r})$: Elgamal encrypted spending info	72 B
Total	6261 B

Fig. 10. Breakdown of estimated transaction (Create and Collect) size in 𝕍eksel.

We estimate the transaction size of 𝕍eksel to be 6261 bytes (breakdown shown in Figure 10), based on: implementation of $\pi_{\text{1-many}}$, the formula for the size of a Bulletproof (used as π_{create}) and the size of a generalized Schnorr for two generators used to 'partially open' the coin (used as π_{opn}). We note that both $\pi_{\text{1-many}}$ and π_{opn} can be used as signatures of knowledge.

References

[1] Veksel implementation. https://github.com/matteocam/veksel.

[2] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*, pages 480–494. Springer, 1997.

[3] G. Barthe, S. Belaïd, P.-A. Fouque, and B. Grégoire. maskVerif: a formal tool for analyzing software and hardware masked implementations. Cryptology ePrint Archive, Report 2018/562, 2018. https://eprint.iacr.org/2018/562.

[4] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, Dec. 2001.

[5] E. Ben-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*, pages 459–474. IEEE Computer Society Press, May 2014.

[6] D. Benarroch, M. Campanelli, D. Fiore, K. Gurkan, and D. Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255, 2019. https://eprint.iacr.org/2019/1255.

[7] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 29–50. Springer, Heidelberg, Dec. 2007.

[8] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Annual International Cryptology Conference*, pages 561–586. Springer, 2019.

[9] J. Buchmann and S. Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2011.

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[11] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Annual International Cryptology Conference*, pages 61–76. Springer, 2002.

[12] M. Campanelli, D. Fiore, and A. Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, Nov. 2019.

[13] M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, abhi shelat, M. Venkitasubramaniam, and R. Wang. Diogenes: Lightweight scalable rsa modulus generation with a dishonest majority. Cryptology ePrint Archive, Report 2020/374, 2020. https://eprint.iacr.org/2020/374.

[14] I. Damgård, C. Ganesh, H. Khoshakhlagh, C. Orlandi, and L. Siniscalchi. Balancing privacy and accountability in blockchain transactions. Cryptology ePrint Archive, Report 2020/1511, 2020. https://eprint.iacr.org/2020/1511.

[15] S. Dobson, S. D. Galbraith, and B. Smith. Trustless groups of unknown order with hyperelliptic curves. Cryptology ePrint Archive, Report 2020/196, 2020. https://eprint.iacr.org/2020/196.

[16] H. M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(03):393–423, Apr. 2007.

[17] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 649–678. Springer, Heidelberg, Dec. 2019.

[18] J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 444–459. Springer, 2006.

[19] J. Groth. On the size of pairing-based non-interactive arguments. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[20] J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. Cryptology ePrint Archive, Report 2014/764, 2014. http://eprint.iacr.org/2014/764.

[21] J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 253–280. Springer, 2015.

[22] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification, version 2020.1.15, 2020.

[23] A. Jivanyan. Lelantus: Towards confidentiality and anonymity of blockchain transactions from standard assumptions. Cryptology ePrint Archive, Report 2019/373, 2019. https://eprint.iacr.org/2019/373.

[24] A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. Papamanthou, R. Pass, a. shelat, and E. Shi. How to use SNARKs in universally composable protocols. Cryptology ePrint Archive, Report 2015/1093, 2015. http://eprint.iacr.org/2015/1093.

[25] R. W. F. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang. Omniring: Scaling private payments without trusted setup. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, Nov. 2019.

[26] J. Lee. The security of groups of unknown order based on jacobians of hyperelliptic curves. Cryptology ePrint Archive, Report 2020/289, 2020. https://eprint.iacr.org/2020/289.

[27] S. Meiklejohn and C. Orlandi. Privacy-enhancing overlays in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 127–141. Springer, 2015.

[28] 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*, pages 397–411. IEEE Computer Society Press, May 2013.

[29] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[30] T. H. Yuen, S. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu. RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security. In J. Bonneau and N. Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 464–483. Springer, Heidelberg, Feb. 2020.

A Explicit Account-Based Construction

For the sake of completeness, we describe a concrete and optimized version of our construction in Section 4. We assume all the instantiations described in Section 5.3. The main differences with Figure 6 are:

Set of coins: Rather than explicitly maintaining S_{coins} each party keeps the currenct (group of unknown order) accumulator A_{coins} containing all coins S_{coins} and a membership proof of all coins that she posseses (an accumulator of all coins except hers). Hence Collect transactions can be generated in $O(|\mathsf{mycoins}|)$ time by proving $R^{A_{\text{coins}}}_{\text{1-many}}$ using Corollary 1. Similarly Collect transactions can be verified in $O(1)$ time.

Identies: In Figure 6 the abstract identities of the parties $\mathcal{R} = (\text{pk}_\sigma, \text{pk}_e)$ takes the concrete form of public keys $(\text{pk}_\sigma, \text{pk}_e)$ for a strongly unforgable signature scheme (e.g. Schnorr) and IND-CPA + IK-CPA [4] (key-privacy) secure public key encryption pk_e (e.g. Elgmal) respectively.

$\mathcal{F}_{\text{Comm}}$: To send $(\mathsf{Msg}, B, M, \mathcal{R})$ interpret $\mathcal{R} = (\text{pk}_\sigma^{(\mathcal{R})}, \text{pk}_e^{(\mathcal{R})})$, encrypt $c \leftarrow \mathsf{Enc}(\text{pk}_\mathsf{e}^{(\mathcal{R})}, \mathsf{M})$, then sign $\sigma \leftarrow \mathsf{Sign}(\text{pk}_\sigma^{(\mathcal{S})}, \text{pk}_\sigma^{(\mathcal{S})} \| B \| c)$, broadcast $(\sigma, \text{pk}_\sigma^{(\mathcal{S})}, B, c)$. Hence in the explicit construction $\mathsf{aux}_{\text{coin}}$ is encrypted with $\text{pk}_e^{(\mathcal{R})}$ of the reciever and the ciphertext is broadcast on a public bulletin board ('blockchain') with a signature from the sender $\mathcal{S}$. The network checks the signatures σ on the transactions.

Compressing $t\|\mathcal{R}$: Since $|(t\|\mathcal{R})| > \log_2(|\mathbb{G}^\bigcirc|)$ in general, we use a collision resistance function $\mathsf{H} : \{0,1\}^* \to \mathbb{F}_{|\mathbb{G}^\bigcirc|}$ before committing to the second component through $\mathsf{Comm}^\bigcirc$ i.e. compute $\mathsf{Comm}^\bigcirc(v, \mathsf{H}(\mathsf{t}\|\mathcal{R}); \mathsf{r})$. Since these fields are revealed during 'collect' this hash can be recomputed by the verifier and is never proven in zero-knowledge (does not affect the efficiency of the proof schemes). Given the last two items, a coin with value v and tag t to recipient $\mathcal{R}$ is concretely computed as $\mathsf{c} \leftarrow f_\bigcirc^v g_\bigcirc^{\mathsf{H}(\mathsf{t}||\text{pk}_\mathcal{R})} h_\bigcirc^r$ where r is the randomness and $\text{pk}_\mathcal{R}$ is the public key of the recipient;

Permissibility A sender must produce a permissible coin c, to ensure that $c \in \mathcal{P}$ she keep sampling new randomness r until she obtains such a coin (e.g., the $\mathbb{x}$ component should be a prime). When validating a transactions, the network checks that $c \in \mathcal{P}$ and adds it the accumulator (see Figure 7).

B Security of Our Construction

Theorem 4 ($\Pi_{Anon} \Diamond \mathcal{F}_{Comm} \geq \mathcal{F}_{Anon}$)**.** *For the three different NIWI/NIZK arguments in the construction we require:*

ZK-1-many: *Simulation sound and zero-knowledge.*

ZKOpen: *Witness indistinguishable.*

ZKDblRange: *Witness indistinguishable.*

Proof. Construct Sim_{Anon} as follows: Get initial balances (genesis block) $(v_1, \ldots, v_n)$.

Initialize: Input $Balance = (v_1, \ldots, v_n)$ to $\mathcal{F}_{Anon}$. Initialize the empty ledger $\mathcal{L} \leftarrow \epsilon$.

Create coin (honest sender, honest receiver): On input (Create, j) on $\mathcal{F}_{Anon}$.leak where $j \notin \mathcal{C}$. Pick $r \leftarrow\!\$\, \mathbb{F}$, pick $t \leftarrow\!\$\, \{0,1\}^\lambda$, let $v = 0$, let $i = 0$, let $\mathsf{c} \leftarrow \mathsf{Comm}^\bigcirc((v, t\|i); r)$. Run ZKDblRange on the statement $(\mathsf{c}, \mathsf{bal}' = \mathsf{bal} - \mathsf{c})$, where $\mathsf{bal} = \mathsf{Comm}^\bigcirc(v_i; r_i)$ or $\mathsf{bal} = \mathsf{Comm}^\bigcirc(v_i; 0)$[14] using v, v_i, i, t, r, r_i as the witness, obtain π_{create}. Output $(\mathsf{Msg}, j, (\mathsf{Create}, \mathsf{c}, \pi_{\text{create}}), |(t, v, r)|)$ on $\mathcal{F}_{Comm}$.leak (pick random id for $\mathcal{F}_{Anon}$)

Collect coin (honest sender, honest receiver): On input $(\mathsf{Collect}, i)$ on $\mathcal{F}_{Anon}$.leak where $i \notin \mathcal{C}$. Pick $r^* \leftarrow\!\$\, \mathbb{F}$, pick $t \leftarrow\!\$\, \{0,1\}^\lambda$, let $v = 0$, simulate the ZK-1-many proof for the commitment $\mathsf{c}^* \leftarrow \mathsf{Comm}^\bigcirc((v, t\|i); r)$ using τ, get $\pi_{\text{1-many}}$. Run ZKOpen.Prove with statement $(i, t, \mathsf{c}_\Delta = \mathsf{c}^* - \mathsf{Comm}^\bigcirc((0, t\|i); 0))$ and witness (r, v), get π_{opn} Output $(\mathsf{Msg}, i, (\mathsf{Collect}, \mathsf{c}^*, t, (\pi_{\text{1-many}}, \pi_{\text{opn}})), 0)$ on $\mathcal{F}_{Comm}$.leak (pick random id for $\mathcal{F}_{Anon}$)

14 First transaction after genesis.

Create/collect coin (corrupt sender/receiver): Since the contents of the coin (id', j, i, v) is revealed during both Create and Collect, simulation is trivial.

Deliver On input $(\mathsf{Deliver}, id)$ on $\mathcal{F}_{Comm}.\mathsf{infl}$. Lookup id (from $\mathcal{F}_{Comm}$) and retrieve the associated broadcast message B. If $\mathsf{Vfy}(\mathsf{pp}, B, \mathcal{L}) = \mathsf{reject}$ ignore the message. If $\mathsf{Vfy}(\mathsf{pp}, B, \mathcal{L}) = \mathsf{accept}$, input $(\mathsf{Process}, id)$ on $\mathcal{F}_{Anon}.\mathsf{infl}$, update the ledger $\mathcal{L} \leftarrow \mathsf{Process}(\mathsf{pp}, B, \mathcal{L})$ and set of coins S_{coins}.

Note that the simulator only simulates proofs for ZK-1-many, for the other proofs a random statement and witness is sampled, which is intuitively why witness indistinguishablity is sufficient. Now formally showing $\mathsf{Sim}_{Anon} \Diamond \mathcal{F}_{Anon} \overset{c}{\approx} \Pi_{Anon} \Diamond \mathcal{F}_{Comm}$ using a sequence of hybrids:

1. Consider the hybrid $\mathcal{H}^{(\mathsf{Create})}$ which extracts (v, i) from $\mathcal{F}_{Anon}$ during Create (rather than fixing $v = 0$ and $i = 0$) and creates coins $\mathsf{c} \leftarrow \mathsf{Comm}^{\circlearrowleft}((t, v, i); r)$ for honest parties with the real denomination and destination. Observe that $\mathcal{H}^{(\mathsf{Create}, v, i)} = \mathsf{Sim}_{Anon} \Diamond \mathcal{F}_{Anon}$ by the perfect hiding of the commitment and witness indistinguishablity of ZKRange.
2. Consider $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+Sim)}$, which additionally extracts (id', v, j, i) from $\mathcal{F}_{Anon}$ during Collect and retrieves the generated tag t and r (created during simulated Create) associated with id'. Reconstructs the coin $\mathsf{c} \leftarrow \mathsf{Comm}^{\circlearrowleft}((t, v, i); r)$ (rather than creating a new randomly generated coin), samples $r^* \leftarrow\!\!\$\, \mathbb{F}$, defines $\mathsf{c}^* \leftarrow \mathsf{c} + \mathsf{Comm}^{\circlearrowleft}((0, 0); r^*)$ and simulates ZK-1-many with the statement $(\mathsf{c}^*, S_{\text{coins}})$. Observe that $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+Sim)} = \mathcal{H}^{(\mathsf{Create}, v, i)}$, by the perfect hiding of C (distribution over statements c^* are the same), note also that the distribution over witnesses and statements for ZKOpen NIWI is unchanged.
3. Observe that in $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}, Sim)}$, $w = (v, t \| i, r + r^*)$ is a witness for the statement $x = (\mathsf{c}^*, S_{\text{coins}})$. Define $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect})}$ which generates $\pi_{\text{1-many}}$ by running the prover (rather than simulation) with the witness w. Observe that $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}, Sim)} \approx \mathcal{H}^{(\mathsf{Create}+\mathsf{Collect})}$ by simulation indistinguishablity and simulation soundness of ZK-1-many.
4. Consider $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS})}$ which extends $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect})}$ by sampling the common reference string for the ZK-1-many NIZK without a simulation trapdoor. Observe $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect})} \approx \mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS})}$ by reference string indistinguishablity of the ZK-1-many NIZK.

Now use soundness of ZK-1-many, ZKOpen, ZKDblRange to argue that the outputs on $\{\mathsf{P}_i\}_{i \notin \mathcal{C}}$ are indistinguishablity betwen $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS})}$ and the real world $\Pi_{\mathcal{F}_{Anon}} \Diamond \mathcal{F}_{Comm}$: Define $Balance_{sim}$ with the initial entries $(v_1, \ldots, v_n)$. Maintain a set $\mathcal{C} = \{(\mathsf{c}, o, r_c)\}$ of created coins computed by extracting from ZKDblRange in every Create broadcast message B:

In **Deliver**, if $B = (\mathsf{Create}, \mathsf{c}, \pi_{\text{create}})$

1. Extract $o_c = ((v, t \| j), r_{\mathsf{c}})$ from π_{create} (with statement c)
2. Update $Balance_{sim}[i] \leftarrow Balance_{sim}[i] - v$.
3. Add $(\mathsf{c}, (v, t \| j), r_{\mathsf{c}})$ to $\mathcal{C}$

The set of coins c in $\mathcal{C}$ is exactly S_{coins}. Additionally maintain a set $\mathcal{O} = \{(\mathsf{c}, o, r_c)\}$ of opened coins computed by extracting from ZK-1-many and ZKOpen in every Collect broadcast message B:

In **Deliver**, if $B = (\mathsf{Collect}, \mathsf{c}^*, t, (\pi_{\text{1-many}} \pi_{\text{opn}}))$:

1. Extract r^* from $\pi_{\text{1-many}}$ (with the statement c^* and S_{coins}).
2. Extract v and r' from π_{opn} (with the statement c^*, c_v, j, t).
3. Compute $r_{\mathsf{c}} = r' - r^*$
4. Recompute $\mathsf{c} \leftarrow \mathsf{Comm}^{\circlearrowleft}((v, t \| j), r_{\mathsf{c}})$
5. Define $o = (v, t \| j)$
6. If $(\mathsf{c}, o, \cdot) \notin \mathcal{C}$, this violates soundness of $\pi_{\text{1-many}}$ or π_{opn}: since r^* is the re-randomization and r' is the randomness of c^*, it follows that c is uniquely defined.
7. If $(\mathsf{c}, o', r'_{\mathsf{c}}) \in \mathcal{O}$ for some o', r'_{c} with $o' \neq o$, this breaks computational binding of the commitment: stop and output $(\mathsf{c}, o', o, r_{\mathsf{c}}, r'_{\mathsf{c}})$ as a collision in the binding game of the commitment scheme C.
8. Update $Balance_{sim}[j] \leftarrow Balance_{sim}[j] + v$.
9. Add $(\mathsf{c}, o, r_{\mathsf{c}})$ to $\mathcal{O}$

Conclude that every coin $\mathsf{c} \in S_{\text{coins}}$ can occure at most once in the computation of $\mathcal{C}$ and that $\mathcal{O} \subseteq \mathcal{C}$. Construct hybrids:

1. $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v)}$, which on Process of $Events[id] = (\mathsf{Create}, j, i, v)$ outputs (id, v') from $\mathcal{O}$ on $\mathcal{F}_{Anon}.\mathsf{P}_i$ rather than (id, v) from the functionality $\mathcal{F}_{Anon}$. By the previous observation that the commited values claimed $\mathcal{O}$ is a subset of $\mathcal{O}$ we get that:
   (a) When the simulator inputs $(\mathsf{Process}, id)$ on $\mathcal{F}_{Anon}.\mathsf{infl}$ it always leads to an output (id, v) on $\mathcal{F}_{Anon}.\mathsf{P}_j$
   (b) By $\mathcal{O} \subseteq \mathcal{C}$ and the observation that every $(c, (v, t \| j), r_c) \in \mathcal{C}$ corresponds to a coin $(id', v, j, i) \in Coins$ for some j.

In conclusion: $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS})} \overset{c}{\approx} \mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v)}$

2. $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v+Balance)}$, which on Process of $Events[id] = (\mathsf{Collect}, j, i, v)$ outputs $Balance_{sim}[i]$ on $\mathcal{F}_{Anon}.\mathsf{P}_i$ rather than $Balance[i]$ from the functionality $\mathcal{F}_{Anon}$. Since the balance is just the sum of outputs for the player this follows from $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS})} \overset{c}{\approx} \mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v)}$ and by inspection of the 'bookkeeping' done in the functionality and hybrid.

Note that all the ports (input/output) of $\mathcal{F}_{Anon}$ are now completely controlled by the simulator in $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v+Balance)}$. Lastly show that $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v+Balance)} \overset{p}{=} \Pi_{Anon} \Diamond \mathcal{F}_{Comm}$: the $\mathcal{F}_{Comm}$ functionality ensures that every message to every player is delivered in the same order and simultanously for all honest players. Hence the local state of the ledger $\mathcal{L}_i$ of every honest player P_i is exactly the same at all times. The only distinction between $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v+Balance)}$ and $\Pi_{Anon} \Diamond \mathcal{F}_{Comm}$ is that every player maintains its own local ledger state, where $\mathcal{H}^{(\mathsf{Create}+\mathsf{Collect}+\mathtt{CRS}+v+Balance)}$ mainstains a single ledger state for simulation. □

C Bulletproof Relation for π_{rerand}

Here we describe how we instantiate $\mathsf{ZKCP}_{\text{rrnd\&prms}}$ from Section 5.2. We use Bulletproofs over group $\mathbb{G}^{\square} = \mathtt{Curve25519}$ for a relation (described below) equivalent to $R_{\text{rrnd\&prms}}$. Let $g_{\square}, h_{\square}$ from $\mathsf{ck}_{\square}$ and $h_{\circ}$ from $\mathsf{ck}_{\circ}$ (Figure 8). Given randomness $r^*_{\circ}$ we parametrize the relation by a fixed group element $\tilde{h} = h_{\circ}^{r^*_{\circ}}$. The family of relations we consider is then:

$$R^{\tilde{h}} = \{(c_{\square}, (x, y), r) : c_{\square} = g_{\square}^{x} h_{\square}^{r} \wedge \ (x, y) \in \mathcal{P}\}$$

We consider permissible set $\mathcal{P}$ from Section 5.3.1 with $\mu = 251$. Below we use $\cdot$ to denote multiplication by constant (linear operation) and $\times$ to denote multiplication of two free variables. More details follow.

Statement: The statement is defined by a commitment $c_{\square} = g_{\square}^{\mathbb{x}} h_{\square}^{r} \in \mathbb{G}^{\square}$ and two field elements $(\mathbb{x}', \mathbb{y}') \in \mathbb{F}^{\square} \times \mathbb{F}^{\square}$.

C.1 Witness

The witness consists of: a bit-decomposition of the rerandomization scalar a bitwise decomposition of x (in $c_{\square}$), a bitwise decomposition of y:

1. A bit-decomposition $(r_0, \ldots, r_{251}) \in \{0_{\mathbb{F}^{\square}}, 1_{\mathbb{F}^{\square}}\}^{252}$ of the rerandomization randomness $r = \sum_{i=0}^{251} r_i \cdot 2^i \in \mathbb{F}_{|\mathbb{G}^{\square}|}$.
2. A bit-decomposition $(x_0, \ldots, x_{249}) \in \{0_{\mathbb{F}^{\square}}, 1_{\mathbb{F}^{\square}}\}^{250}$ of the $\mathbb{x}$-coordinate $\mathbb{x} = \sum_{i=0}^{249} x_i \cdot 2^i \in \mathbb{F}^{\square}$ of the 'input commitment' $(\mathbb{x}, \mathbb{y}) \in \mathcal{P}$.
3. A bit-decomposition $(y_1, \ldots, y_{255}) \in \{0_{\mathbb{F}^{\square}}, 1_{\mathbb{F}^{\square}}\}^{251}$ of the $\mathbb{y}$-coordinate $\mathbb{y} = \sum_{i=1}^{255} y_i \cdot 2^i \in \mathbb{F}^{\square}$ of the 'input commitment' $(\mathbb{x}, \mathbb{y}) \in \mathcal{P}$.

C.2 Relation

We denote by $\times$ a product between (linear combinations) of variables and by $\cdot$ a linear combination of variables. Every line represents a single multiplicative constraint. The relation has a total of 1514 constraints.

Part 1: Permissibility (758 constrains). Check that the point $(\mathbb{x}, \mathbb{y}) \in \mathcal{P}$ (i.e. is 'permissible').

$$\begin{aligned}
\forall i \in [0, 251] : 0 &= (1 - r_i) \times r_i && /\!/ \text{ Bit range check} \\
\forall i \in [1, 255] : 0 &= (1 - y_i) \times y_i && /\!/ \text{ Bit range check} \\
\forall i \in [0, 249] : 0 &= (1 - x_i) \times x_i && /\!/ \text{ Bit range check} \\
\mathbb{x} &= \sum_{i=0}^{249} x_i \cdot 2^i && /\!/ \text{ Range check of } \mathbb{x} \\
\mathbb{y} &= \sum_{i=1}^{255} y_i \cdot 2^i && /\!/ \text{ Check } \mathbb{y} \text{ is 'even'} \\
q_{\mathbb{x}} &= \mathbb{x} \times \mathbb{x} && /\!/ \text{ Curve check} \\
q_{\mathbb{y}} &= \mathbb{y} \times \mathbb{y} && /\!/ \text{ Curve check} \\
q_{\mathbb{x}} \times q_{\mathbb{y}} &= 1 - d \cdot q_{\mathbb{x}} \times q_{\mathbb{y}} && /\!/ \text{ Curve check}
\end{aligned}$$

Part 2: Rerandomization (756 constrains). Rerandomization of $(\mathbb{x}, \mathbb{y}) \in \mathbb{G}^{\circ}$ is done by repeated conditional Edwards addition of $h_{\circ}^{2^i} \in \mathbb{G}^{\circ}$ (constants in the circuit). Recall the group law for addition on Edwards curves:

$$(\mathbb{x}_1, \mathbb{y}_1) +_{\mathbb{G}^{\circ}} (\mathbb{x}_2, \mathbb{y}_2) = \left(\frac{\mathbb{x}_1\mathbb{y}_2 + \mathbb{x}_2\mathbb{y}_1}{1 + d\mathbb{x}_1\mathbb{x}_2\mathbb{y}_1\mathbb{y}_2}, \frac{\mathbb{y}_1\mathbb{y}_2 - \mathbb{x}_1\mathbb{x}_2}{1 - d\mathbb{x}_1\mathbb{x}_2\mathbb{y}_1\mathbb{y}_2} \right)$$

When d is not a square (as in our case), then the formula is complete. The circuit is optimized by borrowing techniques from the Zcash specification ([22], sec. A.3.3.7), employing 'limb-wise addition' with 3-bit limbs. The scalar $r = \sum_{i=0}^{251} r_i \cdot 2^i$ is split into 84 'windows' $j \in [0, 83)$ of 3 bits $(b_0, b_1, b_2) = (r_{3j}, r_{3j+1}, r_{3j+2})$, the table of points $T^{(j)} = [(\mathbb{u}_i, \mathbb{v}_i) = h_{\circ}^{2^{3j+i}}]_{i \in [0,8)}$ is precomputed for each window and the circuit does a lookup in T: verifying $(\mathbb{x}^{(j+1)}, \mathbb{y}^{(j+1)}) = T^{(j)}[b^{(j)}] \ +_{\mathbb{G}^{\circ}} \ (\mathbb{x}^{(j)}, \mathbb{y}^{(j)})$ where $b^{(j)} = \sum_{i=0}^{2} 2^i \cdot b_i = r_{3j} + 2 \cdot r_{3j+1} + 4 \cdot r_{3j+2}$.

Define the constraint $\mathcal{R}^{(j)}(b_0, b_1, b_2, \mathtt{x}_{in}, \mathtt{y}_{in}, \mathtt{x}_{out}, \mathtt{y}_{out})$, consisting of two parts, the table lookup (enforcing $(\mathtt{u}_b, \mathtt{v}_b) = T[b]$) and the point addition (enforcing $(\mathtt{x}_{out}, \mathtt{y}_{out}) = (\mathtt{u}_b, \mathtt{v}_b) \ +_{\mathbb{G}^\circ} \ (\mathtt{x}_{in}, \mathtt{y}_{in})$):

Part 2.1: Table Lookup (3 constraints). Let $(\mathtt{u}_i, \mathtt{v}_i) = h_\circ^{2^{3j+i}}$. Constrain $(\mathtt{u}_b, \mathtt{v}_b) = T^{(j)}[b]$ where $b = \sum_{i=0}^{2} 2^i \cdot b_i$:

$$
\begin{aligned}
b_\& &= b_1 \times b_2 \\
b_0 \times (\ & -\mathtt{u}_0 \cdot b_\& + \mathtt{u}_0 \cdot b_2 + \mathtt{u}_0 \cdot b_1 - \mathtt{u}_0 + \mathtt{u}_2 \cdot b_\& \\
& - \mathtt{u}_2 \cdot b_1 + \mathtt{u}_4 \cdot b_\& - \mathtt{u}_4 \cdot b_2 - \mathtt{u}_6 \cdot b_\& \\
& + \mathtt{u}_1 \cdot b_\& - \mathtt{u}_1 \cdot b_2 - \mathtt{u}_1 \cdot b_1 + \mathtt{u}_1 - \mathtt{u}_3 \cdot s_\& \\
& + \mathtt{u}_3 \cdot b_1 - \mathtt{u}_5 \cdot b_\& + \mathtt{u}_5 \cdot b_2 + \mathtt{u}_7 \cdot b_\&) = \\
& \mathtt{u}_b - \mathtt{u}_0 \cdot b_\& + \mathtt{u}_0 \cdot \mathtt{u}_2 + \mathtt{u}_0 \cdot b_1 - \mathtt{u}_0 + \mathtt{u}_2 \cdot b_\& \\
& \quad - \mathtt{u}_2 \cdot b_1 + \mathtt{u}_4 \cdot b_\& - \mathtt{u}_4 \cdot b_2 - \mathtt{u}_6 \cdot b_\& \\
b_0 \times (\ & -\mathtt{v}_0 \cdot b_\& + \mathtt{v}_0 \cdot b_2 + \mathtt{v}_0 \cdot b_1 - \mathtt{v}_0 + \mathtt{v}_2 \cdot b_\& \\
& - \mathtt{v}_2 \cdot b_1 + \mathtt{v}_4 \cdot b_\& - \mathtt{v}_4 \cdot b_2 - \mathtt{v}_6 \cdot b_\& \\
& + \mathtt{v}_1 \cdot b_\& - \mathtt{v}_1 \cdot b_2 - \mathtt{v}_1 \cdot b_1 + \mathtt{v}_1 - \mathtt{v}_3 \cdot s_\& \\
& + \mathtt{v}_3 \cdot b_1 - \mathtt{v}_5 \cdot b_\& + \mathtt{v}_5 \cdot b_2 + \mathtt{v}_7 \cdot b_\&) = \\
& \mathtt{v}_b - \mathtt{v}_0 \cdot b_\& + \mathtt{v}_0 \cdot \mathtt{v}_2 + \mathtt{v}_0 \cdot b_1 - \mathtt{v}_0 + \mathtt{v}_2 \cdot b_\& \\
& \quad - \mathtt{v}_2 \cdot b_1 + \mathtt{v}_4 \cdot b_\& - \mathtt{v}_4 \cdot b_2 - \mathtt{v}_6 \cdot b_\&
\end{aligned}
$$

Part 2.2: Point Addition (6 constraints). Constrain $(\mathtt{x}_{out}, \mathtt{y}_{out}) = (\mathtt{u}_b, \mathtt{v}_b) \ +_{\mathbb{G}^\circ} \ (\mathtt{x}_{in}, \mathtt{y}_{in})$:

$$
\begin{aligned}
(\mathtt{x}_{in} + \mathtt{y}_{in}) \times (\mathtt{v}_b - \mathtt{u}_b) &= T \\
\mathtt{x}_{in} \times \mathtt{v}_b &= A \\
\mathtt{y}_{in} \times \mathtt{u}_b &= B \\
(d \cdot A) \times B &= C \\
(1 + C) \times \mathtt{x}_{out} &= (A + B) \\
(1 - C) \times \mathtt{y}_{out} &= (T - A + B)
\end{aligned}
$$

Where $T, A, B, C, \mathtt{u}_b, \mathtt{v}_b, b_\&$ are otherwise free 'intermediate' variables, local to $\mathcal{R}^{(j)}$ (not used anywhere else).

Part 2.3: Constrain windows (84 × 9 constraints) Constrain every 3-bit window: define $\mathtt{x}^{(0)} = \mathtt{x}$, $\mathtt{y}^{(0)} = \mathtt{y}$, $\mathtt{x}^{(84)} = \mathtt{x}'$, $\mathtt{y}^{(84)} = \mathtt{y}'$ and add the 84 relations $\forall j \in [0, 84) : \mathcal{R}_j(r_{3j}, r_{3j+1}, r_{3j+2}, \mathtt{x}^{(j)}, \mathtt{y}^{(j)}, \mathtt{x}^{(j+1)}, \mathtt{y}^{(j+1)})$

D UTXO Construction

Our construction can be adapted to the UTXO setting obtaining stronger unlinkability properties: rather than having one $\pi_{\text{1-many}}$ proof every transaction would reference two previous coins[15] and create two new coins. In a UTXO instantiation the π_{create} and π_{opn} can be combined into a single Bulletproof. The two $\pi_{\text{1-many}}$ proofs can be optimized as well: the π_{rerand} can be extended to rerandomize two commitments in parallel, the two π_{modEq} proofs can be combined to check congruency of a random linear combination. We estimate the size of such a construction to be ≈ 10 KB.

15 Where one of them may be a dummy.