Curve Trees: Practical and Transparent Zero-Knowledge Accumulators

Matteo Campanelli[1], Mathias Hall-Andersen[2], and Simon Holmgaard Kamp[2]

[1] Protocol Labs
`matteo@protocol.ai`
[2] Aarhus University, Denmark
`{ma,kamp}@cs.au.dk`

Abstract. In this work we improve upon the state of the art for practical *zero-knowledge for set membership*, a building block at the core of several privacy-aware applications, such as anonymous payments, credentials and whitelists. This primitive allows a user to show knowledge of an element in a large set without leaking the specific element. One of the obstacles to its deployment is efficiency. Concretely efficient solutions exist, e.g., those deployed in Zcash Sapling, but they often work at the price of a strong trust assumption: an underlying setup that must be generated by a trusted third party.
To find alternative approaches we focus on a common building block: accumulators, a cryptographic data structure which compresses the underlying set. We propose novel, more efficient and fully transparent constructions (i.e., without a trusted setup) for accumulators supporting zero-knowledge proofs for set membership. Technically, we introduce new approaches inspired by "commit-and-prove" techniques to combine shallow Merkle trees and 2-cycles of elliptic curves into a highly practical construction. Our basic accumulator construction—dubbed *Curve Trees*—is completely transparent (does not require a trusted setup) and is based on simple and widely used assumptions (DLOG and Random Oracle Model). Ours is the first fully transparent construction that obtains concretely small proof/commitment sizes for large sets and a proving time one order of magnitude smaller than proofs over Merkle Trees with Pedersen hash. For a concrete instantiation targeting 128 bits of security we obtain: a commitment to a set of *any* size is 256 bits; for $|S| = 2^{40}$ a zero-knowledge membership proof is 3KB, its proving takes 40 ms and its verification 2s on an ordinary laptop.
Using our construction as a building block we can design a simple and concretely efficient anonymous cryptocurrency with full anonymity set, which we dub $\mathbb{V}$cash. Its transactions can be verified in $\approx$ 80ms or $\approx$ 5ms when batch-verifying multiple ($>$ 100) transactions; transaction sizes are 4.1KB. Our timings are competitive with those of the approach in Zcash Sapling and trade slightly larger proofs (transactions in Zcash Sapling are 2.8KB) for a completely transparent setup.

1 Introduction

Zero-knowledge proofs are a cryptographic primitive that allows one to prove knowledge of a secret without revealing it. In many applications the focus is on proofs that are short and with efficient running time. One of the rising applications of zero-knowledge is in *set-membership*: given a short digest to a set S, we want to later show knowledge of a member in the set without revealing the latter. This primitive is useful in domains such as privacy-preserving distributed ledgers, anonymous broadcast, financial identities and asset governance (see, e.g., discussion in [BCF$^+$21]).

Limitations of prior work. Our focus in this work is on solutions that are highly *practical*. That is, solutions with concretely short proving/verification time and short proofs. While efficient solutions to zero-knowledge set-membership already exist, we argue that they have limitations. In particular, either they still have a high computational/communication cost (we elaborate in Section 1.2 where we compare to transparent polynomial commitments and ring signatures [LRR$^+$19]) or they rely on proof systems that are *non-transparent*. The latter means that, in order for the system to be bootstrapped, it is necessary to invoke a trusted authority. This is true for example in Zcash (Sapling) [HBHW21] and in [CFH$^+$21]. While we can partly overcome this issue by emulating the trusted authority through a large-scale MPC, this is still highly expensive, both computationally and logistically[3]. Other solutions, such as [BCF$^+$21, CHA21], mitigate this problem by requiring a trusted setup

[3] `https://z.cash/technology/paramgen/`

for parameters that are reusable in other cryptographic settings (an RSA modulus). This, however, still requires invoking a trusted authority or arranging a parameter-generation ceremony [CHI+20], which may not always be viable. We then turn to solutions that are fully transparent and still very efficient.

Our contributions. Our main contribution is a concretely efficient construction for proving private set-membership with a fully transparent setup. Specifically we design a new data structure, Curve Trees, that supports concretely small commitment to a set and where we can show set membership in zero-knowledge and with a small proof.

The design of a curve tree is simple and relies on discrete logarithm and the random oracle model (ROM) for its security. A curve tree can be described as a shallow Merkle tree where the leaves are points over an elliptic curve (and so are internal nodes). To hash, at each level we use an appropriately instantiated Pedersen hash alternating curves at each layer (we require a 2-cycle of curves). To prove membership in zero-knowledge we use commit-and-prove[4] capabilities of Bulletproofs and leverage the algebraic nature of our data structure. Our curves can be instantiated with existing ones in literature (see "Supported Curves" in Section 2.2). While we focus on accumulators and set membership, our approach can straightforwardly be applied to opening of vectors rather than sets obtaining an "index-hiding" vector commitment [ZBK+22].

Using our construction as a building block we can construct a simple and concretely efficient anonymous payment system with full anonymity set[5] and *transparent setup*. We dub this payment system $\mathbb{V}$Cash[6]. In $\mathbb{V}$Cash, the constraint system used for the zero-knowledge proof of a "spend" transaction is 20x smaller than that in Zcash Sapling.

The main distinguishing feature of $\mathbb{V}$Cash is that it can be concretely efficient and still support *full anonymity sets*. The latter is roughly the subset of existing transactions a spent transaction can be narrowed down to (if a protocol supports a full anonymity set then this set consists of the whole history of transactions so far). For "two inputs/two outputs" settings and for anonymity sets of size 2^{32} (like in Zcash) our confidential transactions ($\mathbb{V}$cash) require participants to compute/verify two Bulletproofs proofs of < 5000 constraints each. Verifying each of the proofs in parallel (4 cores) in batches of at least 100 transactions (e.g. when verifying the validity of all transactions in a block) yields a very practical per-transaction verification time of ≈ 5 ms. Transaction sizes are 4.1 KB. Our timings are competitive with those of the approach in Zcash Sapling and trade slightly larger proofs for a completely transparent setup and simpler curve requirements.

As a side contribution, we provide the first optimized implementation of Bulletproofs that can be instantiated with arbitrary curves and supports vector commitments of arbitrary dimension and arbitrary computations at the same time. To the best of our knowledge, previous implementations were not written modularly to work with arbitrary curves or supported only specific computations, such as range proofs.

Structure-Preserving Features. From the theoretical side, one interesting feature of curve trees is their *structure-preserving* properties[ACD+16]. This means our construction never needs to use any combinatorial hash (e.g., SHA) to convert representation of elements at each level or use their bit decomposition, but it only relies on basic structural properties of groups. In this sense, this construction provides some nuances to the implications of the recent impossibility result in [CFGG22]. See also Remark 3.

1.1 Technical Overview

Preliminaries: elliptic-curves and SNARK-native relations In the following we assume that the reader is familiar with elliptic curves (see also Section 2.2 and notation in Section 2.1).

We informally say that a relation is "SNARK-native" to prove for a specific (SNARK) proof scheme if it can be "naturally represented in the constraint system" (a constraint systems is a representation of a relation we aim to prove). For example, we usually consider Pedersen hashing (and commitments) to be native to Bulletproofs (instantiated in the right curve). In fact we can prove we know the scalar representation $\vec{u}$ of a group element

[4] In the sense of the commit-and-prove building blocks in LegoSNARK [CFQ19] and in the work by Lipmaa [Lip16].

[5] An anonymity set can be seen as the subset of existing transactions a spent transaction can be narrowed down to. We say that a protocol supports a full anonymity set then it this set consists of the whole history of transactions so far.

[6] As a reference to both Zcash and $\mathbb{V}$eksel [CHA21] from which it borrows part of its design.

$U = \sum_i [u_i] G_i$ through roughly $|\vec{u}|$ constraints[7]. Notice that for this operation to be actually native, each of the scalars $[u_i]$ should be elements in the scalar field of the curve to which U and the G_i-s belong to.

Starting point: shallow Merkle trees As a warm up we will ignore zero-knowledge for most of this overview and then show how to account for it. Our starting point is *shallow* Merkle trees, i.e. Merkle trees with a general branching factor $\ell \geq 2$. Let us consider a balanced tree of depth D. This has $N = \ell^{\mathsf{D}}$ leaves (and it is encoding a set of an equal number of elements). One of the main advantages of trees with a high branching factor is that we may afford in practice a linear dependence on the depth. For a concrete vector size such as $N = 2^{32}$, we can choose $\ell = \sqrt[4]{N} = 256$ and obtain a depth $\mathsf{D} = 4$.

Given a tree, we can label its nodes as follows: each internal node v' is labeled with the hash $H(v_1, \ldots, v_\ell)$ of the concatenation of its children; each leaf is labeled by its own value v. The root of this Merkle tree is public and represents the commitment to the set of elements. An internal node $\mathsf{rt}^{(i)}$ at level i can be seen as a root to a subtree branching from it. We denote by D the "lowest" level, to which the leaves belong, and by 0 the level to which the root belongs to (so we denote it by $\mathsf{rt}^{(0)}$).

The straightforward approach to opening a leaf in a Merkle tree opening provides the specific leaf together with the $\mathsf{rt}^{(i)}$-s, the internal nodes along the path from $v^{(\mathsf{D})}$ to the root, and the sets $\mathsf{Siblings}(\mathsf{rt}^{(i)})$ of siblings of each $\mathsf{rt}^{(i)}$. This way, anybody can verify membership of the leaf by hashing the siblings at each level. The size of the opening certificate is roughly $\ell \cdot \mathsf{D}$.

We would like to compress the communication complexity even further using SNARKs. However we are looking for better tradeoffs than what we can obtain by "plugging the whole tree opening inside the SNARK" (we discuss this more in the related work).

Our basic blueprint Our high-level solution stems from this insight. Instead of providing all siblings at opening time, we can just provide the internal nodes $\mathsf{rt}^{(1)}, \ldots, \mathsf{rt}^{(\mathsf{D})}$ together with short SNARKs $\pi^{(1)}, \ldots, \pi^{(\mathsf{D})}$[8]. Each proof $\pi^{(i)}$ should show "knowledge of the appropriate siblings", namely of $v_1^{(i)}, \ldots, v_\ell^{(i)}$ such that $\mathsf{rt}^{(i-1)} = \mathcal{H}(v_1^{(i)}, \ldots, v_\ell^{(i)})$ and $\mathsf{rt}^{(i)}$ is among the $v^{(i)}$-s (denoting $v^{(\mathsf{D})}$ by $\mathsf{rt}^{(\mathsf{D})}$).

Our main challenge is to keep at bay the complexity of proving (and verifying) a Pedersen hash of size ℓ at each level. In order to go from this general construction to our final concrete one, there are three additional steps:

1. Going from generic hash-functions to towers of elliptic curves
2. Adding zero-knowledge (from hashes to commitments with "select-and-rerandomize")
3. Slashing communication and running time by: *(i)* going from D to 2 proofs by moving from towers to 2-cycles of elliptic curves; *(ii)* finding compressed representation of tree nodes.

We now elaborate on each. We stress that, for sake of clarity, we somewhat simplify our explanation and leave out several optimizations we carry out in our final construction. See Section 3 and Section 4 for the actual construction.

Efficient proofs through "tower hashing" In order to obtain an efficient proof of hashing, we would like to apply a hash that is SNARK-native in the sense defined above. Our target will be applying a simple Pedersen vector hashing that is native for Bulletproofs, which is a transparent proof system. We, however, quickly run into an issue: this does not work for more than one level because it is not *structure-preserving.* An intuition about what we mean by that is: hashing at a single level would be no problem but when, when applied at multiple levels, we would need to hash the output of an earlier hash function. This would not be "native" anymore for the proof system. The problem arises because a Pedersen hash maps *field elements* to *group elements* and because we have multiple levels: a parent x of a node—a hash image—will have to be hashed again to produce its own parent—thus being part of a preimage. Node x would need to be a point in the field and in the group at the same time.

In a general version of our solution, we solve this problem by using a different hash function at each level so that we can prove it natively inside Bulletproofs at each level. In order to do this we exploit a tower of curves,

[7] While this is an informally defined notion, we contrast the Bulletproof example with the approach applying JubJub in Zcash Sapling. The latter is not *native* for Groth16 instantiated with BLS12-381 as it requires additional constraints for bit decompositions rather than directly describing the multiexponentiation.

[8] **NB:** In our final solution this is reduced to two proofs instead of D proofs.

with a different curve at each level. We provide more details in Section 2.2. Through this solution we can simply produce a root $\mathsf{rt}^{(i-1)}$ of ℓ children $v_1^{(i)}, \dots, v_\ell^{(i)}$ by representing each child as a pair of coordinates $(\mathbb{x}, \mathbb{y}) \in \mathbb{F}_p$ in the base field and producing a Pedersen hash with 2ℓ generators in $\mathbb{E}_q(\mathbb{F}_p)$. We can thus produce a group element H lying in $\mathbb{E}_q(\mathbb{F}_p)$ and represent it (and its siblings) as pairs in $\mathbb{F}_q \times \mathbb{F}_q$. At the next (upper) level we can do the same using 2ℓ generators in $\mathbb{E}_r(\mathbb{F}_q)$ for another elliptic curve of order r. And so in the same way till we get to the root. In our concrete solution we will reduce this tower to a 2-cycle and use only two elliptic curves.

Adding zero-knowledge So far we have been concerned with solutions that do not hide which element in the vector we are opening. We actually provided the internal nodes we encounter along the opening path; therefore, in order to make our solution private, we need to modify it so to provide a *masking* of each of the internal nodes along the opening path. That is, instead of sending the actual internal node $\mathsf{rt}^{(i)}$ (the hash of its children), we let the prover sample some fresh randomness $[\rho']$ and send a rerandomization (a commitment) $\mathsf{cm}^{(i)} = \mathsf{rt}^{(i)} + [\rho'] \cdot H$ (where H is a group generator). What the verifier should be shown *in zero-knowledge* at each level is a slight variation on the relation we considered before. For level i, Given $\mathsf{cm}^{(i-1)}$ and $\mathsf{cm}^{(i)}$ (respectively, the rerandomized nodes along the path at the next and current level) the verifier should be guaranteed that the prover knows some $v_1^{(i)}, \dots, v_\ell^{(i)}, [\rho], [\rho']$ and index $j \in \{1, \dots, \ell\}$ such that *(a)* $\mathsf{cm}^{i-1} = \mathcal{H}(v_1^{(i)}, \dots, v_\ell^{(i)}, [\rho'])$ (the hash is again a Pedersen hash but now randomized through $[\rho']$) and *(b)* $\mathsf{cm}^{(i)} = v_j^{(i)} + [\rho] \, H$. What is important for efficiency is that relation *(a)*—a multiscalar multiplication—can again be proved natively as before. Relation *(b)*—a single multiplication—needs to be expressed as an arithmetic circuit. This is still relatively inexpensive for us since it is performed once per level in a shallow tree.

Optimizing with a 2-cycle and other refinements. We optimize our proof size further by applying an observation. We can move from a tower of D curves to only 2-cycle (two curves only). We can use this to reduce our communication complexity since instead of having D proofs—one per level each with a different curve—we can produce together two proofs—each referring to $\mathsf{D}/2$ of the levels. We describe the rest of our optimizations and design choices in Section 4.3.

$\mathbb{V}$cash: transparent practical anonymous transactions. We use Curve Trees as a main building block in $\mathbb{V}$cash(see Section 5)We refer the reader to [CHA21] for some high level ideas on the architecture (see in particular Section 1.3). We use the standard idea of having unspent coins in the systems stored as a set S of commitments in an accumulator. At the moment of transferring a coin, a user would prove in zero-knowledge that they own the coin (that is, they know the opening of some element in the set S) and provide a rerandomization of that coin (which also needs to be appropriately proved in zero-knowledge). We refer the reader to Section 1.3 in [CHA21] for a more thorough introduction.

While our high-level approach is not novel we make use of several specific aspects of our constructions—the homomorphic properties of coins and their structural compatibility with Curve Tree and the underlying proof system—and come up with new techniques for optimizations—see, e.g., techniques in Remark 5.

1.2 Related Work

When comparing to existing approaches to zero-knowledge for set membership we focus on succinctness and prover efficiency.

Some works with transparent setup do not achieve succinctness (that is, practically short proofs and a $o(|S|)$ verification time). For example, Monero [AJ18]—or, generally, approaches based on ring signatures—have proofs linear in the set and where the verifier's running time is linear in the size of the set $|S|$. Other approaches such as Omniring reduce the proof size to $O(\log(|S|))$ but still have linear verification time [LRR$^+$19].

Other approaches to accumulator with zero-knowledge properties do not involve general-purpose SNARKs. This includes for example the multilinear pairing-based polynomial commitment in [BCF$^+$21], the seminal KZG [KZG10] and the polynomial commitments in [BMM$^+$21]. They, however, all require knowledge-based assumptions and a trusted setup. Similar observations hold for the recent work in Caulk[ZBK$^+$22].

Other works apply asymptotically efficient polynomial commitments with a transparent setup, but their commitment and proof size are concretely large. This is the case of Hyrax [WTs$^+$18], where for large set sizes commitments can be $\gg 10KB$, and Dory [Lee21] where commitments are 190 bytes (6-7 times larger than ours). Proofs of single opening are also large (18 KB) in Dory, although the scheme can amortize this cost with batching (expect for very large opening batches this amortized proof size is still significantly higher than ours). The Spartan proof system has overall opening sizes, proving and verification time that are competitive with respect to ours (for sets up to approximately 2^{20} where Spartan starts to perform worse), but it has very high commitment sizes, e.g. $\geq 20KB$ for sets of size 2^{20} ($625\times$ worse than ours)[9]. Other transparent polynomial commitments include those based on Reed-Solomon IOPs [BBHR18] or on Diophantine ARguments of Knowledge (DARK) [BFS20]. As argued in [Lee21] (Section 1.1) they achieve worse concrete performances than the works above in practice.

Works that apply specialized proving techniques on accumulators in unknown-order groups: $\mathbb{V}$eksel, [CHA21, CHA22, BCF$^+$21, CFH$^+$21]. These works obtain concretely small proofs/verifier with an efficient proving time, but require an RSA modulus (non-transparent) for their efficient instantiations[10]. While the work in [CFH$^+$21] obtains concretely efficient proving time with a slightly larger proof size in Zcash it requires trusted setup to instantiate its proof system in addition to RSA modulus.

Using "friendlier" hash functions for Merkle trees mitigates the complexity of proving an opening. One such example is Poseidon [GKR$^+$21]. The limitation of these solutions is that they rely on hash functions which are quite new and have not received the proper cryptanalytic scrutiny yet.

Comparing our accumulator to transparent Algebraic Merkle Trees. The most interesting comparison to our (zero-knowledge) accumulator construction is a "transparent" version of that used in Zcash. Here, to show membership in a set we apply a specific type of Merkle tree. In it, the collision-resistant hash function we use at each level has an extra property and in particular we require it to be *algebraic*, i.e., one that can be expressed as a "simple enough" polynomial in a ring. A natural choice for this—and the one applied in Zcash—is Pedersen hash. In order to prove membership we apply a zkSNARK to the opening of the Merkle tree. For this approach to be efficient we need that the group in which we compute the hash is tied to the group in which the zkSNARK "functions"[11]. Zcash uses Groth16 on curve BLS12-381 [Gro16] as a zkSNARK and a specific curve for hashing, JubJub. Nonetheless, this approach could be made transparent by applying Bulletproofs on the Ristretto curve and choosing an appropriate elliptic curve for hashing (this includes for example Jabberwock in [CHA21]). In the remainder of this comparison we refer to this way of transparently instantiating Merkle trees with Pedersen hash as $\mathsf{AlgMT}_{\mathrm{BP}}$.

We now compare the approach in our work to that in $\mathsf{AlgMT}_{\mathrm{BP}}$. Let us denote by N the set size. First we observe that asymptotically $\mathsf{AlgMT}_{\mathrm{BP}}$requires performing $\log_2(N)$ hash computations with 2 elements each (one per level of tree, hence $\log_2(N)$) inside the SNARK. Concretely, for a representative choice of $N = 2^{32}$ to ≈ 45000 constraints. Our construction, on the other hand, requires ≈ 4500 constraints (roughly an order of magnitude less).

Comparing to Verkle Trees. At the *very* high-level, our approach resembles the currently explored "Verkle Tree" (VT) approach in Ethereum[12]. In both approaches an internal node represents a vector commitment to its children. The two approaches have a few substantial differences. First, that approach is currently not *structure-preserving* in the sense ours is. Each node is a commitment to a *hash* (e.g., SHA or Blake) of the children. This is required to solve a similar problem to that we approach with towers of curves. Currently VT do no account for zero-knowledge (our focus in this work). This is a source of additional efficiency challenges. A privacy-preserving VT would have to show in zero-knowledge that hashing the children has been performed correctly. For example, in the case of Blake this would require 20K additional constraints per level[13] (our solution, on the other hand, is in the ballpark of 5K constraint *in total*). Another difference is that the current concretly succinct implementation of VT relies on KZG polynomial commitments [KZG10] which require a trusted setup.

Curve trees and Halo2. Halo2[14] is a concrete transparent (zero-knowledge) proof system that uses recursion. It is concretely efficient and it obtains recursion by going back and forth in a cycle of two curves. Halo2 and curve trees have orthogonal goals: one is a full-fledged proof system, the other can be seen as a specialized

[9] See [Lee21] for numbers referred in this section.

[10] In all these works we can replace the RSA group with a transparent class-group [BH01] at a substantial efficiency cost. See, e.g., discussion in [DGS20].

[11] More specifically, this means that elliptic curve of the zkSNARK should be of order related to that of the definition field of the elliptic curve we use for Pedersen hashing.

[12] `https://dankradfeist.de/ethereum/2021/06/18/verkle-trie-for-eth1.html`

[13] `https://github.com/zcash/zcash/issues/2258`

[14] `https://electriccoin.co/blog/explaining-halo-2/`

data structure (and related constraint system) for zero-knowledge for set membership. We see, however, great potential in combining the techniques in these two systems and we are currently working in this direction.

Subsequent work. Recently Eagen has built upon our work to show how to design confidential transactions of smaller size seemingly at the cost of additional proving time [Eag22] through nested proving and other techniques. It is still unclear how to compare these extensions to our work: the current writeup in [Eag22] does not make all the assumptions behind its estimates concrete and the work does not have a complete implementation yet.

2 Preliminaries

2.1 Notation

We denote by $\mathbb{E}[\mathbb{F}]$ the elliptic curve $\mathbb{E}$ defined over the field $\mathbb{F}$, whenever clear from context we might omit the field of definition $\mathbb{F}$—also known as *base field*—and simply write $\mathbb{E}$. We call *scalar field* the field $\mathbb{F}_p$ where $p := |\mathbb{E}[\mathbb{F}]|$ is a prime. Whenever possible we explicitly mark scalar elements in equations through square brackets and group elements with upper case letters (we will occasionally break this convention if not particularly useful for clarity). In practice, when estimating performance, the number of multiplications (or constraints) is the primary metric when proving satisfiability of arithmetic circuits.

When expressing an NP relation $R(x, w)$ we make the private witness w explicit as such but we keep the public statement x implicit. For example, in the relation $\mathcal{R}$ below

$$\mathcal{R} := \{z : y = \mathsf{SHA}([z] \cdot G)\}$$

the only private witness is the scalar z, while group element G and y are considered publicly known inputs.

2.2 Towers of Elliptic Curves

We call a sequence of elliptic curves if $\mathbb{E}_0(\mathbb{F}_{p_0}), \ldots, \mathbb{E}_{\mathsf{D}}(\mathbb{F}_{p_{\mathsf{D}}})$ a $\mathsf{D}+1$-*tower* if the base field of each $\mathbb{E}_{i+1}(\mathbb{F}_{p_{i+1}})$ is the scalar field of $\mathbb{E}_i(\mathbb{F}_p)$. As an example, this allows us to make a point P_0^* on $\mathbb{E}_0$ from coordinates $(\mathtt{x}_1, \mathtt{y}_1) \in \mathbb{E}_1$ through $P_0^* := [\mathtt{x}_1] \cdot P_0 + [\mathtt{y}_1] \cdot P_0'$ where $P_0, P_0' \in \mathbb{E}_0$. Formally, in a tower it holds that for all $i \in \{0, 1, \ldots, \mathsf{D}-1\}$: $p_{i+1} = |\mathbb{E}_i(\mathbb{F}_{p_i})|$; in other words . We will generally let the field of definition be implicit to simplify notation. Towers of curves have previously been used to optimize the proving of cryptographic operations in zkSNARKs [KZM$^+$15] [HBHW21], which will also be the application in this paper. Additionally the same techniques has been applied for recursive proofs systems [BCTV14] [BGH19]. We will not require that any of the curves are pairing friendly.

Cycles of Elliptic Curves. Of particular interest are *m-cycles* of elliptic curves: infinitely long towers where $\mathbb{E}_i(\mathbb{F}_{p_i}) = \mathbb{E}_{i+m}(\mathbb{F}_{p_{i+m}})$ for all i. Most commonly $m = 2$, which will be the primary case of interest in this paper as well.

2.3 Commitments

We use the following syntax for commitments:

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 values of m 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 $\mathsf{Comm}(\mathsf{ck}, m; r) = \mathsf{Comm}(\mathsf{ck}, m'; r')$.

Remark 1 (Rerandomizable Commitments). We will use rerandomizable commitments, i.e., endowed with an algorithm $\mathsf{Rerand}(\mathsf{ck}, \mathsf{Comm}(\mathsf{ck}, m; r)) \to (c', r')$ such that $c' = \mathsf{Comm}(\mathsf{ck}, m; r + r')$. Notice that homomorphic commitments (and thus Pedersen commitments) satisfy this property.

2.4 Accumulators

Definition 2 (Accumulator scheme). *An accumulator scheme* Acc *over universe* $\mathcal{U}_\lambda(\mathsf{Acc})$ *(where* λ *is a security parameter) consists 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* $x \notin S$[15].

2.5 NIZKs

Non-Interactive Zero-Knowledge schemes (or NIZKs) require a reference string which can be either uniformly sampled (a urs), or structured (a srs). In the latter case it needs to be sampled by a trusted party. 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.
– $\mathsf{ZK.VerProof}(\mathsf{urs}, R, x, \pi) \to b \in \{0,1\}$ *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 proof* $\pi \leftarrow \mathsf{Prove}(\mathsf{urs}, R, x, w)$.

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 [BCFK19].

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 [BCFK19]. In our constructions we often assume prover and verifier to implicitly take as input the relation description[16].

In the proof of security of our protocol 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 [Gro06] for formal definitions.

Curve Friendliness While we present our constructions generically in the next sections, our purpose is to exploit some specific properties of proof systems to finally obtain a concretely efficient scheme. One of these properties has to do with how relations can be efficiently formalized in that proof system as arithmetic circuit. Given a curve $\mathbb{E}$ we say that a NIZK is $\mathbb{E}$-*friendly* if its relation (resp. its inputs) are represented as operations (resp. elements) in the scalar field of $\mathbb{E}$. Given a scheme we make this property explicit by denoting it as $\mathsf{ZK}[\mathbb{E}]$.

[15] These definitions are standard and we refer the reader to [BBF19] for a formal treatment. We also observe that our construction satisfies a "stronger" variant of the binding property which holds even when the adversary provides the target accumulator (rather than providing a set from which the accumulator is honestly computed).
[16] This parameter is usually short.

Modular NIZKs through Commit-and-Prove. We use the framework for black-box modular composition of commit-and-prove NIZKs (or CP-NIZKs) in [CFQ19] and [BCFK19]. Informally a CP-NIZK is a NIZK that can efficiently prove properties of committed inputs through some commitment scheme C. Let x be a public input and c a commitment. Such a scheme can for example prove knowledge of (u, ω, r) such that $c = \mathsf{Comm}(u; r)$ 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.

In our construction 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 $c \leftarrow \mathsf{Comm}(u, r)$; generates proofs π and π' from the respective schemes; it outputs combined proof $\pi^* := (c, \pi, \pi')$. The verifier checks each proof over respective inputs (x, c) and (x', c').

The following theorem (informally stated) is a direct consequence of Theorem 3.1 in [CFQ19].

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

We can see Bulletproof [BBB+18] as a CP-NIZK since it works efficiently over an implicit commitment representation (see discussion in [CFQ19]). We use this fact in our instantiations.

3 Curve Trees as Accumulators

3.1 Accumulator: $(\ell, \mathbb{E}_0, \ldots, \mathbb{E}_\mathsf{D})$-Curve Tree

Our accumulation scheme is a "Curve Tree": a Curve Tree can be seen as an "algebraically compatible" Merkle tree using Pedersen commitments over $\mathbb{E}_0, \ldots, \mathbb{E}_\mathsf{D}$ as the compression function at each respective level. Notice that below we use a "randomness" scalar $[r]$. This will be useful for several concrete aspects of our construction described in the next section. This scalar will intuitively be useful for efficiency by making sure the internal points are "nice" (see Section 4.3 and in particular "permissible points").

The following is the formal definition of a Curve Tree. We describe an operational way to build a Curve Tree in 4.

Definition 4 (Curve Trees). *A Curve Tree—parameterized by (i) a branching factor ℓ, (ii) a tower of Elliptic curves $\mathbb{E}_0, \ldots, \mathbb{E}_\mathsf{D}$ and group constants (iii) group elements $\vec{G}^{(i)}_\mathsf{x}, \vec{G}^{(i)}_\mathsf{y} \in \underbrace{\mathbb{E}_i \times \cdots \times \mathbb{E}_i}_{\ell}$, $H^{(i)} \in \mathbb{E}_i$ for $i \in \{0, \ldots, \mathsf{D}\}$—has the following recursive structure:*

Leaf: $(\ell, \mathbb{E}_\mathsf{D})-\mathsf{CurveTree}$: *A $(\ell, \mathbb{E}_\mathsf{D})$-Curve Tree (leaf node) is a 3-tuple $(C, 0, C)$ where $C \in \mathbb{E}_\mathsf{D}$ (intuition: a leaf is a tree itself holding value C and with root C).*

Parent: $(\ell, \mathbb{E}_0, \ldots, \mathbb{E}_\mathsf{D})-\mathsf{CurveTree}$: *A $(\ell, \mathbb{E}_0, \ldots, \mathbb{E}_\mathsf{D})$-Curve Tree (parent node) is a 3-tuple $(C, r, (T_1, \ldots, T_\ell))$, where $T_1, \ldots, T_\ell$ are $(\ell, \mathbb{E}_0, \ldots, \mathbb{E}_{\mathsf{D}-1})$-Curve Trees, i.e.*

$$T_1 = (\hat{C}_1 = (\mathbb{x}_1, \mathbb{y}_1), \hat{r}_1, \hat{T}_1), \ldots, T_\ell = (\hat{C}_\ell = (\mathbb{x}_\ell, \mathbb{y}_\ell), \hat{r}_\ell, \hat{T}_\ell)$$

The root C of the $(\ell, \mathbb{E}_0, \ldots, \mathbb{E}_\mathsf{D})$-Curve Tree is a Pedersen commitment to the coordinates of the children:

$$C = \langle \vec{\mathbb{x}}, \vec{G}^{(\mathsf{D})}_\mathsf{x} \rangle + \langle \vec{\mathbb{y}}, \vec{G}^{(\mathsf{D})}_\mathsf{y} \rangle + [r] \cdot H^{(\mathsf{D})}$$

NB: *We say such a tree has depth D and $\mathsf{D}+1$ levels (or layers).*

Remark 3 (Curve Tree as somewhat structure-preserving). The recent results of [CFGG22] on commitments to vectors that have linear verification show that (informally) it is not possible to have a short commitment and a short opening at the same time in a setting that makes no assumption on the underlying group (in Maurer's generic group model [Mau05]). One could think that the moral corollary of these results is a need for heavily destructuring or "non-algebraic" (e.g., SHA) operations in succinct vector commitments. However, the underlying approach in our work rules out this extreme conclusion: the basic Curve Tree construction uses algebraic operations at each step and a linear verification assuming only the representation of group elements as "pairs of scalars for a (distinct) group". This bypasses the stricter definition of "structure-preserving" in [CFGG22], which considers one *single* abstract group and black-box use of its addition. Curve trees, on the

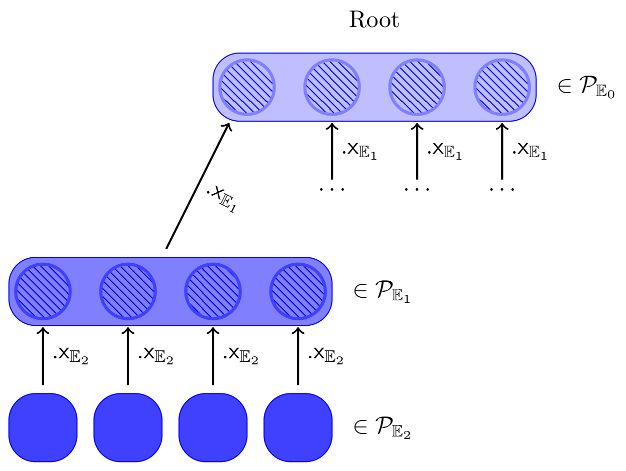


Fig. 1: Illustration of a $(4, \mathbb{E}_0, \mathbb{E}_1, \mathbb{E}_2)$-Curve Tree. Hatch pattern circles indicates that the point is represented as a field element, rounded rectangles represents Pedersen commitments to the field elements (circles) inside. Darker shades indicates lower levels in the tree. The commitments to y-coordinates are omitted as they are not needed in our optimized construction (Section 4.3). Permissible sets $\mathcal{P}_{\mathbb{E}}$ for a curve $\mathbb{E}$ are defined in Section 4.3.

other hand, exploit several groups (assumed to constitute a tower/cycle of groups of elliptic curves) while still making black-box use of their respective addition operations (after representing them as pairs of scalars as mentioned above). We stress that our claim is not that we can contradict the impossibility result in [CFGG22], nor is our intention to undermine it. Instead, we argue Curve Trees provides further nuances to the observations in [CFGG22]: they show that we can meaningfully go around them by only slightly weakening the algebraic requirements of the model. We finally remark that the above only refers to curve trees as an authenticated data structure (this section), but not to the privacy-preserving variant of its opening (Section 4).

3.2 Supported Curves.

Since we only require the hardness of discrete log, the techniques described in this paper are broadly applicable to any tower of curves and in particular existing cycles of elliptic curves, e.g. the "Pasta Cycle" (Pallas / Vesta curves) [Hop20], the "Tweedle Cycle" (Tweedledum / Tweedledee curves) [Hop19] or the Secp256k1 / Secq256k1 curves cycle [Poe18]. Even though our techniques do not use bilinear pairing, security of our zero-knowledge accumulator holds even in the presence of an efficiently computable bilinear map (type I, type II or type III) on one/both of the curves: our techniques are then compatible with proof systems over such cycles.

4 Zero-Knowledge Set Membership in Curve Trees

In this section we describe how to prove set membership in zero knowledge for our curve trees accumulators. We use a slightly more specific version of the relation for zero knowledge for set membership in, e.g., [BCF$^+$21, CFH$^+$21]. Our variant (dubbed "Select-and-Rerandomize") is described below. We chose this formalization to explicitly model the strong unlinkability properties we require later in Section 5[17].

Our *general* scheme achieves $O(\log n)$ communication and $O(\sqrt[\mathsf{D}]{n})$ computation where D is the number of levels in the tree and n is the size of the accumulated set. Our *final concrete* scheme (Section 4.3 and Appendix B) achieves morally "constant-size" proof sizes for the common case where D and ℓ are concretely very small.

[17] This is mostly a stylistic choice: we do not claim that straightforward variants of [BCF$^+$21, CFH$^+$21] cannot satisfy our model. Select-and-rerandomize just aims at stressing that we want *anonymity* vs *pseudonymity* only. This is reflected in our primitive explicitly rerandomizing a commitment in the accumulator. On the other hand, the formal models in [BCF$^+$21, CFH$^+$21] consider relations on a commitment that is already given (and is potentially not "fresh"). Moreover, several of the application domains discussed in these works are "commit-ahead-of-time" scenarios where a commitment could be reused through time.

4.1 "Select-and-Rerandomize" Accumulators

Here we define an auxiliary interface for a primitive we call a "select-and-rerandomize accumulator". Given an accumulator of committed values, I can provide you with a handle (a commitment) to a value in the accumulated set proving to you it is in the set but without revealing which one it is. We achieve this through rerandomization of commitments and zero-knowledge proofs over the set membership relation and commitment rerandomization. This primitive is natural in several settings, such as in anonymous payment systems, including the one in this work and in [CHA21]. Similar attempts at having hiding for accumulator witnesses (with different techniques) also appeared in [CFH$^+$21].

Below we assume an accumulator scheme and an accumulated set S whose elements are (rerandomizable) commitments. We also denote by pp the concatenation of the accumulator parameters and the commitment key.

$\mathsf{SelRerand}.\mathcal{P}(\mathsf{pp}, S, c) \to (c', \pi, r')$ returns a rerandomized commitment (of $c \in S$), a proof of membership and the (auxiliary) randomness used for rerandomization.
$\mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, A, c', \pi) \to 0/1$ verifies that c' is a rerandomization of an element in the set.

Correctness: For any set S, $c \in S$ (with c an honestly generated commitment), honestly generated parameters $\mathsf{pp} = (\mathsf{pp}_{\mathrm{acc}}, \mathsf{ck})$ the following holds

$$\mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, \mathsf{Accum}(\mathsf{pp}_{\mathrm{acc}}, S), c', \pi) = 1 \ \wedge\ c' = \mathsf{Rerand}(\mathsf{ck}, c; r')$$

where $\mathsf{SelRerand}.\mathcal{P}(\mathsf{pp}, S, c) \to (c', \pi, r')$.

Security (informal): we require the proof π to be an extractable NIZK (i.e., we require knowledge soundness and zero-knowledge) for the relation below:

$$\mathcal{R}^{(\mathsf{SelectRerand})} := \left\{ (W, c, r) : \begin{array}{c} c' = \mathsf{Rerand}(\mathsf{ck}, c; r') \\ \wedge\ \mathsf{Acc.VfyMem}(\mathsf{pp}_{\mathrm{acc}}, A, c, W) \end{array} \right\}$$

whenever the parameters and the accumulator have been generated honestly.

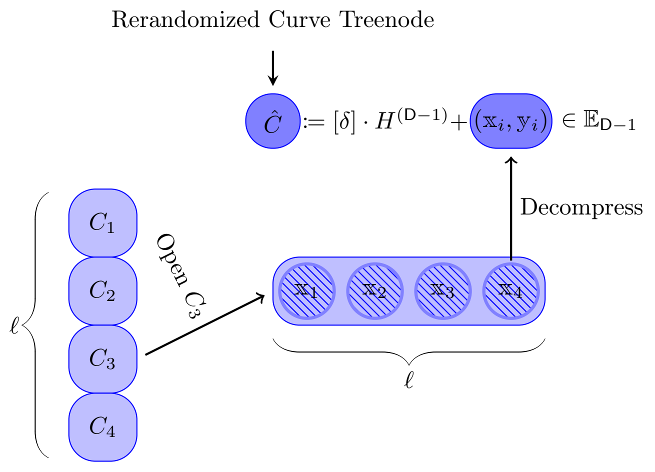
4.2 Constructing Select-and-Rerandomize in Curve Trees

The main intuition is as follows. Observe that the leafs of a $(\ell, \mathbb{E}_0, \ldots, \mathbb{E}_{\mathsf{D}})$-Curve Tree are curve points on $\mathbb{E}_1$. These leafs are going to be Pedersen commitments to secret vectors. We exploit the recursive algebraic structure of our tree to enable efficient zero-knowledge proofs of knowledge of these vectors. A crucial feature of Curve Trees is that the roots (and those of every sub-tree) are rerandomizable commitments to Curve Trees: by rerandomizing the root C as $C' \leftarrow C + [\delta] \cdot H^{(\mathsf{D})}$ we obtain a perfectly hiding commitment to the same set of children as the original tree, we exploit this observation to traverse the tree level-by-level using a simple zero-knowledge proof described in the next section. Another property of curve trees is that the height can dynamically increase as the number of elements in the tree grows: by using a cycle of curves the tree can grow "upward" while the leaves remain $\mathbb{E}_1$ points.

4.2.1 Single-Level Select-and-Rerandomize

The central component in our construction is a simple construction for a select-and-rerandomize-like relation for a *single* level in a curve tree (we later recursively invoke this to obtain a full select-and-rerandomize in Section 4.2.2). Its underlying relation takes as input a rerandomized commitment $\hat{C}$, an alleged parent C (the root of a $(\ell, \mathbb{E}_0, \ldots, \mathbb{E}_{\mathsf{D}})$-Curve Tree), and a secret index i whose semantics is "$\hat{C}$ is the (rerandomized) i-th child of C". Concretely, this is accomplished at each level d by opening the commitment C to $\vec{\mathbb{x}}, \vec{\mathbb{y}}$ using a Pedersen commit-and-prove over $\mathbb{E}_d$, then rerandomizing the i'th point $\hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d)} \in \mathbb{E}_d$. Slightly more formally we require a zero-knowledge argument of knowledge for the following relation for each non-top level $d \in [\mathsf{D}]$:

$$\mathcal{R}^{(\mathsf{single\text{-}level}, d)} := \left\{ \begin{pmatrix} i, r, \delta, \\ \vec{\mathbb{x}}, \vec{\mathbb{y}} \end{pmatrix} : \begin{array}{c} C = \langle [\vec{\mathbb{x}}], \vec{G}_{\mathsf{x}}^{(d-1)} \rangle \\ + \langle [\vec{\mathbb{y}}], \vec{G}_{\mathsf{y}}^{(d-1)} \rangle \\ + [r] \cdot H^{(d-1)} \\ \wedge\ \hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d)} \end{array} \right\}$$



Rerandomized Curve Treenode
$\hat{C} := [\delta] \cdot H^{(\mathsf{D}-1)} + (\mathbb{x}_i, \mathbb{y}_i) \in \mathbb{E}_{\mathsf{D}-1}$
Decompress
Open C_3
C_1
C_2
C_3
C_4
$\mathbb{x}_1$
$\mathbb{x}_2$
$\mathbb{x}_3$
$\mathbb{x}_4$
ℓ
ℓ

Fig. 2: Illustration of the "select and rerandomize relation" for one level. In the example illustrated above we have branching factor $\ell = 4$. Note that $\hat{C}$ is one layer deeper in the tree.

As noted the $C = \langle [\vec{\mathbb{x}}], \vec{G}_{\mathsf{x}}^{(d-1)} \rangle + \langle [\vec{\mathbb{y}}], \vec{G}_{\mathsf{y}}^{(d-1)} \rangle + [r] \cdot H^{(d-1)}$ constraint can be very efficiently enforced using a commit-and-prove for Pedersen commitments (e.g. Bulletproofs or Compressed Σ-Protocol) over $\mathbb{E}_{d-1}$. While $\hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d)}$ requires a single fixed-based exponentiation "inside the circuit". We describe an optimized arithmetic circuit for the relation above in Appendix A.

4.2.2 Recursive $\sqrt[\mathsf{D}]{n}$ Membership Proof. We can observe that $\mathcal{R}^{(\mathsf{single\text{-}level},d)}$ already yields a "one-level" select-and-rerandomize. We can apply it one level at the time and obtain a full construction. Note that in the single-level case, at a given level we can provide a (hiding) Pedersen commitment of one of the children. The latter in turn represents the root of a (sub-)Curve Tree. We can thus extend the technique presented to obtain membership proofs with $O(\sqrt[\mathsf{D}]{n})$ prover/verifier complexity: by letting $\ell = \sqrt[\mathsf{D}]{n}$ and using $\mathsf{D}-1$ individual proofs. The proofs then descends down the Curve Tree one level at a time. The general construction is described in Section 4.2.2.

```
$\mathsf{SelRerand}.\mathcal{P}(\mathsf{pp}, S, c)$
  Reconstruct tree from $S$; let $\mathsf{rt}$ be its root
  Let $c^{(0)}, \dots, c^{(\mathsf{D})}$ be the path elements to $c$ in the tree
     (with $c^{(0)}$ corresponding to $\mathsf{rt}$)
  Let $\hat{c}^{(0)} := \mathsf{rt}$ and $r^{(0)} := 0$
  for $j = 1, \dots, \mathsf{D}$ do
    $(\hat{c}^{(j)}, r^{(j)}) \leftarrow \mathsf{Rerand}(\mathsf{ck}, c^{(j)})$
    $\pi_j \leftarrow \mathsf{ZK}[\mathbb{E}_{j-1}]$
          $.\mathsf{Prove}(\mathsf{pp}, \mathcal{R}^{(\mathsf{single\text{-}level},j)}, \hat{c}^{(j-1)}, \hat{c}^{(j)}, r^{(j)}, r^{(j-1)})$
  endfor
  Let $c' := \hat{c}^{(\mathsf{D})}$ and $\pi^* := \left(\hat{c}^{(1)}, \dots, \hat{c}^{(\mathsf{D}-1)}, \pi^{(1)}, \dots, \pi^{(\mathsf{D})}\right)$
  Return $\left(c', \pi^*, r^{(\mathsf{D})}\right)$

$\mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, \mathsf{rt}, c', \pi^*)$
  Parse $\pi^*$ as $\left(\hat{c}^{(1)}, \dots, \hat{c}^{(\mathsf{D}-1)}, \pi^{(1)}, \dots, \pi^{(\mathsf{D})}\right)$
  Let $\hat{c}^{(\mathsf{D})} := c'$
  Let $\hat{c}^{(0)} := \mathsf{rt}$
  for $j = 1, \dots, \mathsf{D}$ do
    $b_j \leftarrow \mathsf{ZK}[\mathbb{E}_{j-1}].\mathsf{VerProof}(\mathsf{pp}, \mathcal{R}^{(\mathsf{single\text{-}level},j)}, \hat{c}^{(j-1)}, \hat{c}^{(j)})$
  endfor
  Accept iff $\bigwedge_{j=1,\dots,\ell} b_j = 1$
```

Fig. 3: Select-and-rerandomize for a $(\ell, \mathbb{E}_0, \dots, \mathbb{E}_{\mathsf{D}})-\mathsf{CurveTree}$. Recall that $\mathsf{ZK}[\mathbb{E}]$ denotes a NIZK for computations in the scalar field of $\mathbb{E}$.

Theorem 2 (informal). *The construction in Section 4.2.2 is select-and-rerandomize over curve trees as accumulators with $O(\sqrt[D]{n})$ prover/verifier complexity. If we instantiate the underlying NIZK with Bulletproofs this construction is transparent and is secure in the random-oracle model under the discrete-log assumption.*

The theorem above can be proved straightforwardly by invoking Theorem 1 at each level in the tree. Zero-knowledge follows straightforwardly by the rerandomization of the commitments and the zero-knowledge of the underlying NIZK.

4.3 Optimizations in Final Construction

In this section we describe optimizations we employ in our final construction.

Merging Proofs. The approach above which necessitates a total of D individual proofs $\pi_1, \ldots, \pi_{\mathsf{D}}$. However, when using a 2-cycle of curves, all the even/odd proofs are over the same curve/field and can therefore be combined in to a single larger statement; only requiring 2 proofs in total for any D.

Point Compression and Permissible Points. In the setting where the accumulator is guaranteed to have been computed honestly (e.g. in our confidential transactions application), we can reduce the number of exponentiations during committing and the size of the witness by only committing to the $\mathbb{x}$-coordinate of the children: this remains binding by ensuring that only one of $(\mathbb{x}, \mathbb{y})$ and $(\mathbb{x}, -\mathbb{y})$ is "allowed". One common choice is to take the numerically smallest between $\mathbb{y}$ and $-\mathbb{y}$, or discriminate based upon the parity (even/odd) over $\mathbb{Z}$, however neither of these constraints can be efficiently expressed as an arithmetic circuit; instead we use a universal hash function. Let $S(v) = 1$ iff. $v \in \mathbb{F}$ is a quadratic residue (i.e. there exists $w \in \mathbb{F}$ st. $w^2 = v$) and $S(v) = 0$ otherwise. Now consider the following family of 2-universal hash functions from any field to $\{0, 1\}$:

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0, 1\}$$
$$\mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta)$$

Observe that the constraint $\mathcal{U}_{\alpha,\beta}(v) = 1$ can be enforced using a circuit with multiplicative complexity 1, showing $\{(w) : w^2 = (\alpha \cdot v + \beta)\}$. We exploit this to efficiently define a set of "permissible points" on $\mathbb{E}$:

$$\mathcal{P}_{\mathbb{E}} = \{(\mathbb{x}, \mathbb{y}) \mid (\mathbb{x}, \mathbb{y}) \in \mathbb{E}(\mathbb{F}_p) \land \mathcal{U}_{\alpha,\beta}(\mathbb{y}) = 1 \land \mathcal{U}_{\alpha,\beta}(-\mathbb{y}) = 0\}$$

Note that $1/4$ of the points on $\mathbb{E}$ are permissible and any $(\mathbb{x}, \mathbb{y}) \in \mathcal{P}_{\mathbb{E}}$ is uniquely defined by its $\mathbb{x}$-coordinate – this is the case for any finite field of characteristic $\notin \{2, 3\}$. Any Pedersen commitment C can be "made permissible" by simply iteratively adding an additional generator H ("incrementing the randomness") until the point is permissible. Notice that H is a generator that was not used to commit to all the siblings to generate C (see $\mathsf{MakeInnerNode}$ in Fig. 4). Below $\mathbb{F}$ denotes the scalar field of $\mathbb{E}$; we denote by $\vec{G} \in \mathbb{E}^{\ell}$ the vector of generators used to commit all siblings to produce an internal node.

In expectation $\mathsf{MakePermissible}$ requires 4 curve additions and 8 square roots. By enforcing that only permissible points are added to the accumulator[18] so that the "decompression" is unique, we can reduce the complexity of $\mathcal{R}^{(\mathsf{SelectRerand})}$ slightly, instead showing the following optimized relation $\mathcal{R}^{(\mathsf{single\text{-}level}^\star, d)}$:

$$\mathcal{R}^{(\mathsf{single\text{-}level}^\star, d)} := \left\{ \begin{pmatrix} i, r, \delta, \\ \vec{\mathbb{x}}, \mathbb{y} \end{pmatrix} : \begin{array}{c} C = \langle [\vec{\mathbb{x}}], \vec{G}_{\mathsf{x}}^{(d-1)} \rangle \\ + [r] \cdot H^{(d-1)} \\ \land (\mathbb{x}_i, \mathbb{y}) \in \mathcal{P}_{\mathbb{E}^{(d)}} \\ \land \hat{C} = (\mathbb{x}_i, \mathbb{y}) + [\delta] \cdot H^{(d)} \end{array} \right\}$$

Note that the $(\mathbb{x}_i, \mathbb{y}) \in \mathcal{P}_{\mathbb{E}^{(d)}}$ constraint only requires a check that $(\mathbb{x}_i, \mathbb{y}) \in \mathbb{E}^{(d-1)}$ in addition to $\mathcal{U}_{\alpha,\beta}(\mathbb{y}) = 1$. We include the full circuit description in the appendix in Fig. 7.

[18] In case of our anonymous cryptocurrency application, this is enforced by the network of block validators: as a condition for a transaction being valid.

$\mathsf{MakeInnerNode}_{\mathbb{E}}\left(\vec{G}, H, \vec{\mathbb{x}}\right) \to \mathcal{P}_{\mathbb{E}}$

```
1 :  $C \leftarrow \langle \vec{G}, [\vec{\mathbb{x}}] \rangle$
2 :  $(C, r_{\mathcal{P}}) \leftarrow \mathsf{MakePermissible}_{\mathbb{E}}\left(\vec{G}, H, C\right)$
3 :  return $C$
```

$\mathsf{MakePermissible}_{\mathbb{E}}\left(\vec{G}, H, C\right) \to (\mathcal{P}_{\mathbb{E}}, \mathbb{F})$

```
1 :  $r_{\mathcal{P}} \leftarrow 0 \in \mathbb{F}$
2 :  while $C \notin \mathcal{P}_{\mathbb{E}}$ :
3 :    $C \leftarrow C + H$
4 :    $r_{\mathcal{P}} \leftarrow r_{\mathcal{P}} + 1$
5 :  return $(C, r_{\mathcal{P}})$
```

$\mathsf{BuildTree}\,(u_1, \dots, u_N \in \mathcal{P}_{\mathbb{E}_{\mathsf{D}}}) \to \mathcal{P}_{\mathbb{E}_0}$

```
1 :  // We assume $N = \mathsf{D}^{\ell}$ for simplicity
2 :  Let curLayer and nxtLayer be two empty lists
3 :  curLayer $\leftarrow (u_1.\mathbb{x}, \dots, u_N.\mathbb{x})$
4 :  for $d = \mathsf{D}, \dots, 1$ :
5 :    Divide curLayer into tuples $\vec{v}_1, \vec{v}_2, \dots$ each of size $\ell$
6 :    nxtLayer $\leftarrow \emptyset$
7 :    for each $\vec{v}_i$ :
8 :      $(C, r_{\mathcal{P}}) \leftarrow \mathsf{MakeInnerNode}_{\mathbb{E}_d}(\vec{G}^{(d)}_{\mathsf{x}}, H^{(d)}, \vec{v}_i)$
9 :      // The $\mathbb{x}$ coord. of $C$ is handled in the next iteration
10 :     nxtLayer $\leftarrow$ nxtLayer$||C.\mathbb{x}$
11 :     // Scalar $r_{\mathcal{P}}$ will be useful at proving time
12 :     Appropriately store $r_{\mathcal{P}}$ for later use
13 :   curLayer $\leftarrow$ nxtLayer
14 : // Only the root is left after last iteration; we return it
15 : return curLayer[0]
```

Fig. 4: Pseudocode (including optimizations) for creating a curve tree over a set of N (permissible) group elements in the base field of $\mathbb{E}_{\mathsf{D}}$. It is straightforward to modify it to support directly sets of scalars.

5 $\mathbb{V}$Cash: Transparent and Efficient Anonymous Payment System

In this section we informally describe our anonymous payment system, which we dub $\mathbb{V}$Cash. The techniques and model here follow mostly prior work.

5.1 Model

A formal description of our model is in the appendix in Appendix C. The ideal functionality in the appendix describes the simple expected behavior of an anonymous payment system: parties hold values; they can transfer part of these values to other parties; an adversary can observe transactions but it cannot tamper them or learn anything about the sender/receiver/value of the transaction. This functionality, in particular, supports the largest possible anonymity set at every transaction like ZCash.

5.2 A high-level view of our protocol

The flow of our protocol roughly follows known blueprints. We refer the reader to, e.g., the technical overview and Section 3 in [CHA21, CHA22] for further background.

Intuition about our construction. At any given moment in time, each party holds a certain number of coins[19]. Coins are the fundamental concept in a transaction. During a transaction we *pour* a certain amount from user to user by using two (unspent) input coins and producing two new output coins.

Each user is also holding a public state (the ledger $\mathcal{L}$) roughly containing all the transfers occurred so far. Through the state any user can verify the validity of each transfer. In addition to the public state, users hold a private state containing information as: the aforementioned auxiliary information to spend their coins, signing keys, etc.

In order to implement an anonymous payment system we thus require four algorithms that are run locally by each party in the system:

Setup The setup algorithm produces the initial parameters of the system. We emphasize that it does not require being run by a trusted setup.

Pour A sender $\mathcal{S}$ can "pour" the value of two *input coins* into two new *output coins* nullifying the input ones. The recipients of the two new coins can be distinct. It is possible for $\mathcal{S}$ itself to be one or both of the recipients. We require that the total value of input and output coins is the same. The algorithm Pour has two outputs: a new transaction that is publicly broadcasted and a private auxiliary opening that is sent to the respective recipients of the new output coins.

Verify A verifying algorithm allows any party to check a transaction is valid. It takes as a input the public parameters and the public state observed so far.

Process By a processing algorithm parties can update their public and private state after observing a transaction.

5.3 Our protocol in more detail

We describe our protocol in Fig. 6 and in the rest of this section.

A transaction consist of the creation of output coins from input coins. A coin roughly consists of a commitment to its amount and other information that ensures it will be used only once and by its intended recipient. For a transaction to be valid it must be the case that:

1. Output coins are in an appropriate non-negative range (we want to *give* money and not take it in a transaction). This corresponds to the Mint in Fig. 5.
2. Input coins "exist" and are valid themselves. This corresponds to the Spend in Fig. 5.
3. The total value of input and output coins is the same. This is handled by π_{bal} in Fig. 6.

[19] "Holding" a coin requires knowing a certain secret key associated to the user. In this section we ignore the aspect of registering with a new key to the system, but we stress it is straightforward to add.

We use zero-knowledge proofs to ensure the above. The first and third property can be ensured respectively by range proofs and homomorphic properties of Pedersen commitments & proving knowledge of appropriate discrete logarithms. The second property is where we use our select-and-rerandomize constructions from the previous sections: all coins are stored in an accumulator (a Curve Tree) and whenever they aim to spend an input coin, they can select-and-rerandomize it obtaining a rerandomized version of that input coin. This is included in the transaction together with a proof that it refers to the rerandomization of something existing in the accumulator.

Further details on our building blocks follow.

Breakdown of public parameters: – public parameters for $\mathsf{SelRerand}$ – urs (uniform reference string) for zero-knowledge – generators $\left(G_v, G_t, \hat{H}\right)$ for Pedersen commitments whose semantics we explain below.

Structure of a coin: A coin is a Pedersen commitment to: 1) the amount v transferred through the coin; 2) the tag/nullifier t, i.e. the (rerandomized) public key of the recipient. Hence each coin c is of the form $\mathsf{c} = [v] \cdot G_v + [t] \cdot G_t + [r] \cdot \hat{H}$ where r is the randomness we use for masking the polynomial.

Additional cryptographic primitives:

– Digital signatures with rerandomizable keys (see, e.g., [FKM+16]). The key property we require is that we can rerandomize a public key and correspondingly update a signing key. We use this feature in Mint in Fig. 5.
– Non-Interactive zero-knowledge for different relations:
  - Relation R_{dlog}, which shows knowledge of discrete logarithm for given generators for an input group element c. We use this relation to show zero-balance among input and output coins and to show knowledge of values in the input coins. Whenever we use relation R_{dlog} we also explicitly describe with respect to what tuple of generators. For instance, if we write $R_{\text{dlog}}\left(G_t, \hat{H}\right)$ it means that we are showing knowledge of (t, r) so that a certain commitment equals $[t] \cdot G_t + [r] \cdot \hat{H}$. The last example is instructive in one more way: that relation is equivalent to stating that the "transferred value v" inside a certain commitment (a coin) is zero. We use this fact to assert that the values of input and output coins is balanced overall.
  - Relation $R_{\geq 0}$, which shows knowledge of discrete logarithms for a coin plus that the value of the coin is in a positive range. That is it shows knowledge of (v, t, r) such that $\mathsf{c} = [v] \cdot G_v + [t] \cdot G_t + [r] \cdot \hat{H} \wedge v \in [0, 2^{64})$.
– We denote by $\mathcal{H}_{\mathbb{F}}$ a collision resistant hash function mapping group elements—the public keys of the users—to the appropriate scalar field $\mathbb{F}$. We use this hash function to be able to commit to the public keys as tags. Notice that we do not need to prove this hash function in zero-knowledge.

Other components of public state (i.e., the ledger):

– Set of coins S_{coins} (from which it is possible to compute the corresponding Curve Tree root $\mathsf{rt}_{\text{coins}}$)
– Set of seen "tags". Tags are (rerandomized) public keys of recipients. These are revealed every time an input coin is spent. We stress that they are unlinkable to the actual input coins they refer to because of the select-and-rerandomize proof.

We describe setup and processing algorithm at a very high level since they are almost immediate from the rest of the protocol. The setup algorithm generates all the public parameters described above; it should also provide an initial distribution of coins to users (the mechanism of this initial distribution is unimportant for our focus). The processing algorithm consists in keeping the public state above up to date after each (valid) transaction. It simply updates the set of coins with the new observed output coins and the set of seen tags with those in the latest transaction.

Remark 4 (Optimizations). The construction in Fig. 6 shows a separate proof for each of the relations of interest. This is for clarity only. Our final construction produces a single Bulletproof proof whenever possible, thus avoiding a linear overhead in the number of relations. The final numbers are those stated in Section 6.2 and consist of two Bulletproofs lying on two different curves.

Remark 5 (Full security through efficient PRF). The scheme in Fig. 6 is a slightly simplified version of our final protocol for didactic purposes. The simplification has to do with how we generate new tags ($G^{(j)}_{\text{nll,out}}$). The scheme in the figure, as it is, has an additional leakage: a party $\mathcal{S}$ sending a transaction tx to a party $\mathcal{R}$ can learn when R will spend the coins received in tx (but not to whom). Only sender S can infer this. Additionally the scheme suffers from "Faerie's Gold Attack", which enables an adversary to create two distinct transactions of which only one can be spent by the honest receiver. Our final scheme mitigates both of these issues using a PRF. This solution is *similar* to that used in Zcash. Differently than Zcash we can exploit a more efficient way to prove the PRF computation—thanks to our choice of PRF and groups. However, in order to avoid bloating the circuit to be proven in ZK, we use a "commit-and-prove friendly" PRF with bounded-query security. The fact that we need to require this bound beforehand is not a problem since we can use a bound on the number of transactions we expect in the system (e.g. a very conservative bound of 2^{32} transactions per-user). We give a concrete instantiation based on Diffie-Hellman Inversion Assumption (DHI) using a PRF is based upon Dodis and Yampolskiy [DY05] where $\mathsf{PRF}_K(x) = \left[(K + x)^{-1}\right] \cdot G$. Security of this extensions follows from the well-studied Diffie-Hellman Inversion (DHI) assumption [MSK02]. More details are in Appendix D. **NB:** differently from [DY05], our instantiation group is pairing-free and thus we can instead obtain an evaluation proof through an additional opening of a group element in Bulletproof (alternatively one could use a Sigma-protocol).

```
$\mathsf{Spend}\left(\mathsf{aux}_{\text{in}}^{(j)}\right)$
    // Reconstruct input coin
    Parse $\mathsf{aux}_{\text{in}}^{(j)}$ as $\left(v_{\text{in}}^{(j)}, \mathcal{S}_{\text{rr}}^{(j)}, r_{\text{in}}^{(j)}\right)$
    $G_{\text{nll,in}}^{(j)} \leftarrow \mathcal{H}_{\mathbb{F}}\left(\mathcal{S}_{\text{rr}}^{(j)}\right) \cdot G_t$ // reconstruct input tag
    $\mathsf{c}_{\text{in}}^{(j)} \leftarrow \left[v_{\text{in}}^{(j)}\right] \cdot G_v + G_{\text{nll,in}}^{(j)} + \left[r_{\text{in}}^{(j)}\right] \cdot \hat{H}$ // reconstruct coin
    // Select-and-Rerandomize input coin
    $\left(\mathsf{c}_{\text{rr}}^{(j)}, \pi_{\text{SR}}(j), r_{\text{rr}}^{(j)}\right) \leftarrow \mathsf{SelRerand}.\mathcal{P}\left(\mathsf{pp}_{\text{SR}}, S_{\text{coins}}, \mathsf{c}_{\text{in}}^{(j)}\right)$
    // Prove knowledge of opening of input coin
    $\pi_{\text{spnd}}^{(j)} \leftarrow \mathsf{ZK.Prove}\left(\mathsf{urs}, R_{\text{dlog}}\left(G_v, \hat{H}\right), \mathsf{c}_{\text{rr}}^{(j)} - G_{\text{nll,in}}^{(j)}; \mathsf{aux}_{\text{in}}^{(j)}, r_{\text{rr}}^{(j)}\right)$
```

```
$\mathsf{Mint}\left(\mathcal{R}^{(j)}, v_{\text{out}}^{(j)}\right)$
    $r_{\text{out}}^{(j)} \leftarrow\!\!\$\ \mathbb{F}$ // to mask coin
    $r_{\text{pk}}^{(j)} \leftarrow\!\!\$\ \mathbb{F}$ // to rerandomize pk
    $\mathcal{R}_{\text{rr}}^{(j)} \leftarrow \left[r_{\text{pk}}^{(j)}\right] \cdot \mathcal{R}^{(j)}$ // rerandomized pk
    $G_{\text{nll,out}}^{(j)} \leftarrow \mathcal{H}_{\mathbb{F}}\left(\mathcal{R}_{\text{rr}}^{(j)}\right) \cdot G_t$ // make output tag
    $\mathsf{c}_{\text{out}}^{(j)} \leftarrow \left[v_{\text{out}}^{(j)}\right] \cdot G_v + G_{\text{nll,out}}^{(j)} + \left[r_{\text{out}}^{(j)}\right] \cdot \hat{H}$ // make coin
    $\mathsf{aux}_{\text{out}}^{(j)} \leftarrow \left(v_{\text{out}}^{(j)}, \mathcal{R}_{\text{rr}}^{(j)}, r_{\text{out}}^{(j)}\right)$ // opening of coin
    // Proves value of coin $\geq 0$
    $\pi_{\geq 0}^{(j)} \leftarrow \mathsf{ZK.Prove}\left(\mathsf{urs}, R_{\geq 0}, \mathsf{c}_{\text{out}}^{(j)}; \mathsf{aux}_{\text{out}}^{(j)}, \mathcal{R}^{(j)}\right)$
```

Fig. 5: Auxiliary algorithms for algorithm Pour. We assume all variables have the same scope as Pour.

6 Implementation and Evaluation

We implement select-and-rerandomize and $\mathbb{V}$Cash in Rust on top of the `dalek` Bulletproofs library [20]. The Bulletproof implementation has been extended with support for vector commitments and elliptic curves implemented using the arkworks[21] curve traits.

Code. All our code is available and released as open source at

`https://github.com/simonkamp/curve-trees`.

Experimental Setting and Instantiations. Our benchmarks were run on a C6i.2xlarge[22] instance with 8 vCPUs, which corresponds to 4 physical cores on an Intel Xeon 8375C processor with 2.9 GHz clock speed. Benchmarks for amortized batch verification were run from a similar machine with 96 vCPUs (C6i.24xlarge). We use Curve Trees of even depth D in our evaluation and instantiate the two underlying elliptic curves through those in the Pasta cycle [Hop20]. We use Schnorr signatures for $\mathbb{V}$Cash.

[20] `https://github.com/dalek-cryptography/bulletproofs`
[21] `https://github.com/arkworks-rs`
[22] `https://aws.amazon.com/ec2/instance-types/c6i/`

```
$\mathsf{Pour}\left(\mathsf{pp}, \mathsf{st}_{\mathcal{S}}, \left(\mathcal{S}^{(j)}, \mathsf{aux}_{\text{in}}^{(j)}, \mathcal{R}^{(j)}, v_{\text{out}}^{(j)}\right)_{j\in[2]}\right)$
  $//$ Create output coins
  for $j \in [2]$ :
    $\mathsf{Mint}\left(\mathcal{R}^{(j)}, v_{\text{out}}^{(j)}\right)$
  $//$ Show we are using existing coins
  for $j \in [2]$ :
    $\mathsf{Spend}\left(\mathsf{aux}_{\text{in}}^{(j)}\right)$
  $//$ Show that $v_{\text{in}}^{(1)} + v_{\text{in}}^{(2)} = v_{\text{out}}^{(1)} + v_{\text{out}}^{(2)}$
  $\mathsf{c}_{\text{bal}} \leftarrow \mathsf{c}_{\text{out}}^{(1)} + \mathsf{c}_{\text{out}}^{(2)} - \mathsf{c}_{\text{rr}}^{(1)} - \mathsf{c}_{\text{rr}}^{(2)}$
  $\pi_{\text{bal}} \leftarrow \mathsf{ZK.Prove}(\mathsf{urs}, R_{\text{dlog}}\left(G_t, \hat{H}\right), \mathsf{c}_{\text{bal}};$
          $\mathsf{aux}_{\text{in}}^{(j)}, r_{\text{rr}}^{(j)}, \mathsf{aux}_{\text{out}}^{(j)}, \mathcal{R}^{(j)})$
  $\mathsf{tx} := \left(\left(\mathcal{S}_{\text{rr}}^{(j)}, \mathsf{c}_{\text{rr}}^{(j)}, \mathsf{c}_{\text{out}}^{(j)}, \mathcal{S}_{\text{rr}}^{(j)}\right)_{j\in[2]}, \text{proofs } \pi_\star\right)$
  Double sign $\mathsf{tx}$ with $\mathsf{sk}$-s for $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$
  Privately send $\left(\mathsf{aux}_{\text{out}}^{(j)}\right)_{j\in[2]}$; Broadcast $\mathsf{tx}$
```

```
$\mathsf{Vfy}\left(\mathsf{pp}, \mathsf{tx} := \left(\left(\mathcal{S}_{\text{rr}}^{(j)}, \mathsf{c}_{\text{rr}}^{(j)}, \mathsf{c}_{\text{out}}^{(j)}, \mathcal{S}_{\text{rr}}^{(j)}\right)_{j\in[2]}, \text{proofs } \pi_\star\right), \mathcal{L}\right)$
  for $j \in [2]$ :
    check $\mathsf{SelRerand.Vfy}\left(\mathsf{pp}_{\text{SR}}, \mathsf{rt}_{\text{coins}}, \mathsf{c}_{\text{rr}}^{(j)}, \pi_{\text{SR}}^{(j)}\right)$
    $G_{\text{nll,in}}^{(j)} \leftarrow \mathcal{H}_{\mathbb{F}}\left(\mathcal{S}_{\text{rr}}^{(j)}\right) \cdot G_t$ $//$ reconstruct tags
    Reject if $G_{\text{nll,in}}^{(j)}$ has been seen before already
    check $\mathsf{ZK.Vfy}\left(\mathsf{urs}, R_{\text{dlog}}\left(G_v, \hat{H}\right), \mathsf{c}_{\text{rr}}^{(j)} - G_{\text{nll,in}}^{(j)}, \pi_{\text{spnd}}^{(j)}\right)$
    check $\mathsf{ZK.Vfy}\left(\mathsf{urs}, R_{\geq 0}, \mathsf{c}_{\text{out}}^{(j)}, \pi_{\geq 0}^{(j)}\right)$
  $\mathsf{c}_{\text{bal}} \leftarrow \mathsf{c}_{\text{out}}^{(1)} + \mathsf{c}_{\text{out}}^{(2)} - \mathsf{c}_{\text{rr}}^{(1)} - \mathsf{c}_{\text{rr}}^{(2)}$
  Check $\mathsf{ZK.Vfy}\left(\mathsf{urs}, R_{\text{dlog}}\left(G_t, \hat{H}\right), \mathsf{c}_{\text{bal}}, \pi_{\text{bal}}\right)$
  Verify signatures on $\mathsf{tx}$ with public keys for $\mathcal{S}_{\text{rr}}^{(j)}$-s
  Accept iff all checks above succeed
```

Fig. 6: Pour and Verification algorithms in $\mathbb{V}$Cash.

6.1 Zero-Knowledge for Set-Membership

The results in Table 1 summarize the efficiency of our scheme (Section 4) for different set sizes—modest, medium and large. Given a choice of parameters—the branching factor ℓ and (even) depth D—the total number of constraints to prove in zero-knowledge amounts to $\mathsf{D}(912 + \ell - 1)$ (half per even/odd layers respectively). We heuristically choose for these set size ($|S| = \ell^{\mathsf{D}}$) in order to optimize the running time by obtaining number of constraints "not overflowing" powers of two if possible. This is illustrated the benchmarks for sets of size 2^{32} and 2^{40}: despite the gap between the set sizes we are able to obtain similar performances.

(D, ℓ)	Set size	# Constraints	Proof size (kb)	Proving time (s)	Verification time (ms)	Amort. batch verification time (ms)
$(2, 1024)$	2^{20}	3870	2.6	1	24.03	1.43
$(4, 256)$	2^{32}	4668	3	1.94	41.78	2.36
$(4, 1024)$	2^{40}	7740	3	1.96	42.88	2.69

Table 1: Benchmarks of the select and rerandomize primitive with depth D and branching factor ℓ. The amortized batch verification time refers to a batch of 100 proofs.

6.2 $\mathbb{V}$Cash

Table 2 compares $\mathbb{V}$Cash with various anonymous payment systems. The following statements refer to anonymity sets sizes if size $\geq 2^{20}$. $\mathbb{V}$Cash shows the best proving time (which corresponds to "pouring" time for transactions) among the transparent schemes. Otherwise, the only other better proving time is that of $\mathbb{V}$eksel. (We do not have concrete estimates for Omniring for larger anonymity sets but their proving/verification time will roughly scale linearly with the set size) When used for batch verification, $\mathbb{V}$Cash outperforms other schemes, sometimes of orders of magnitude (for the same anonymity sets). These results also hold for less parallelized implementations of $\mathbb{V}$Cash (see timings in appendix). Non-batched verification time is highly competitive when compared to transparent constructions, but $3\times$ slower than *Zcash* Sapling (which mainly consists of a few pairing operations). The only other better transaction size among transparent constructions is that of Omniring(we estimate Zcash to be less than $2\times$ larger for same anonymity sets).

Concretely, a ”pour” in $\mathbb{V}$Cash for two inputs/two outputs and anonymity sets of 2^{32} (like in Zcash) our confidential transactions ($\mathbb{V}$cash) require participants to compute/verify two Bulletproofs proofs of < 5000 constraints each. We can contrast that to another approach supporting large anonymity sets, Zcash Sapling [23], compared to which our circuit for the zero-knowledge proof of a "spend" transaction is 20x smaller.

We remark that, in the table, we only compare to approaches with concretely small transaction size (of a few kilobytes for large enough anonymity sets). Solutions not in the table because of their large transaction size include: the original approach in Zerocoin [MGGR13] (45KB for full security [CHA22]); Quisquis [FMMO19] (13KB for $|S| = 2^4$); Monero [AJ18] (whose transaction grows linearly with $|S|$ and is already at 1.3KB for $|S| < 2^4$).

	Anonymity set size	Transparent setup	Tx size (kb)	Proving time (S)	Verification time (ms)	Amort. batch verification time (ms)
Zcash	2^{32}	✗	2.8	2.38	7	-
$\mathbb{V}$eksel	Any	✗⋆	5.3	0.44	61.88	-
Lelantus	2^{10}	✓	2.7	0.27†	-	6.8†
	2^{14}	✓	3.9	2.35†	-	10.2†
	2^{16}	✓	5.6	4.8†	-	52†
Omniring	2^{10}	✓	1	$\approx$ 1.5‡	$\approx$ 130‡	-
$\mathbb{V}$Cash	2^{20}	✓	3.6	1.98	42.75	2.82
	2^{32}	✓	4.1	3.85	81.27	4.94
	2^{40}	✓	4.1	3.91	82.83	5.66

Table 2: Benchmarks of $\mathbb{V}$Cash against other anonymous payment schemes. The $\mathbb{V}$Cash schemes are instantiated with Curve Trees with the corresponding set size in Table 1. The batch verification time is measured as the cost per proof of verifying a batch of 100 proofs. If batch verification is empty, it means it is not available as an option for that specific construction or not possible to estimate from the related work.

⋆ $\mathbb{V}$eksel only needs setup if using accumulators instantiated with RSA (which provide the smallest tx size), but not for zero-knowledge.

† Lelantus was benchmarked on an Intel i7-4870HQ (4 cores, 2.5GHz).[Jiv19]

‡ Omniring was benchmarked on an Intel i7-7600U (2 cores, 2.8GHz).[LRR+19].

[23] While Zcash has recently (June 2022) updated to a new version called Orchard, which removes trusted setup, we focus on Sapling for comparison for a few reasons. First, except for the setup requirements—Orchard uses the transparent Halo2—Sapling is the "hardest competitor" for its superior performances: transaction size is 2KB smaller; verification at roughly $5\times$; proving time is comparable. Also, the approach in ZCash Sapling (algebraic Merkle Trees with Groth16) and thus highly representative. See also `https://electriccoin.co/blog/technical-explainer-halo-on-zcash/`.

References

ACD$^+$16. Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. *Journal of Cryptology*, 29(4):833–878, October 2016.

AJ18. Kurt M. Alonso and Jordi Herrera Joancomartí. Monero - privacy in the blockchain. Cryptology ePrint Archive, Report 2018/535, 2018. `https://eprint.iacr.org/2018/535`.

BBB$^+$18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BBF19. Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

BCF$^+$21. Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 393–414, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

BCFK19. Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255, 2019. `https://eprint.iacr.org/2019/1255`.

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. `https://eprint.iacr.org/2019/1021`.

BH01. Johannes Buchmann and Safuat Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2001.

BMM$^+$21. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 65–97. Springer, 2021.

CFGG22. Dario Catalano, Dario Fiore, Rosario Gennaro, and Emanuele Giunta. On the impossibility of algebraic vector commitments in pairing-free groups. *Cryptology ePrint Archive*, 2022.

CFH$^+$21. Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. Succinct zero-knowledge batch proofs for set accumulators. Cryptology ePrint Archive, Report 2021/1672, 2021. `https://ia.cr/2021/1672`.

CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

CHA21. Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. Cryptology ePrint Archive, Report 2021/327, 2021. `https://ia.cr/2021/327`.

CHA22. Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 652–666, 2022.

CHI$^+$20. Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. Cryptology ePrint Archive, Report 2020/374, 2020. `https://eprint.iacr.org/2020/374`.

DGS20. Samuel Dobson, Steven D. Galbraith, and Benjamin Smith. Trustless groups of unknown order with hyperelliptic curves. Cryptology ePrint Archive, Report 2020/196, 2020. `https://eprint.iacr.org/2020/196`.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

Eag22. Liam Eagen. μcash: Transparent anonymous transactions. Cryptology ePrint Archive, Paper 2022/1104, 2022. `https://eprint.iacr.org/2022/1104`.

FKM$^+$16. Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 301–330. Springer, Heidelberg, March 2016.

FMMO19. Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 649–678. Springer, Heidelberg, December 2019.

GKR$^+$21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. pages 519–535. USENIX Association, 2021.

Gro06. Jens 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.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

GW20. Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. `https://eprint.iacr.org/2020/315`.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

HBHW21. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, version 2021.2.16 [nu5 proposal], 2021.

Hop19. Daira Hopwood, 2019. `https://github.com/daira/tweedle`.

Hop20. Daira Hopwood, 2020. `https://github.com/zcash/pasta`.

Jiv19. Aram Jivanyan. Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Paper 2019/373, 2019. `https://eprint.iacr.org/2019/373`.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

KZM$^+$15. Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015. `https://ia.cr/2015/1093`.

Lee21. Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography Conference*, pages 1–34. Springer, 2021.

Lin16. Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. `https://eprint.iacr.org/2016/046`.

Lip16. Helger Lipmaa. Prover-efficient commit-and-prove zero-knowledge SNARKs. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 185–206. Springer, Heidelberg, April 2016.

LRR$^+$19. Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, November 2019.

Mau05. Ueli Maurer. Abstract models of computation in cryptography. In *IMA International Conference on Cryptography and Coding*, pages 1–12. Springer, 2005.

MGGR13. Ian Miers, Christina Garman, Matthew Green, and Aviel 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.

MSK02. Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE Transactions*, E85-A(2):481–484, February 2002.

Poe18. Andrew Poelstra, 2018. `https://moderncrypto.org/mail-archive/curves/2018/000992.html`.

WTs$^+$18. Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

ZBK$^+$22. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. Cryptology ePrint Archive, Paper 2022/621, 2022. `https://eprint.iacr.org/2022/621`.

A Circuit Specifications

Remark 6 (Custom Gates). We keep the explication of our techniques as broadly applicable as possible: working for any elliptic curve on short Weierstrass form and any commit-and-proof system for Pedersen commitments.

However, the circuits in this section can be further optimized for particular curves (e.g. with non-trivial efficient endomorphisms) and proof systems (e.g. Plonk [GWC19] with custom gates for elliptic curve operations, and/or, Plookup [GW20]).

We provide all circuit specifications as Rank-1 constraints systems (R1CS): the left side of any constraint consists of a product ($\times$) of affine combinations, while the right side consists of an affine combination.

A.1 2-Set Membership

To constrain $w \in \{v_1, v_2\}$, enforce the following R1CS constraint:

$$(w - v_1) \times (w - v_2) = 0 \tag{1}$$

Most commonly $w \in \{0, 1\}$ (i.e. $v_1 = 0$ and $v_2 = 1$).

A.2 Not Zero

To enforce $v \neq 0$, introduce $\mathbb{t}_1$ and constrain:

$$\mathbb{t}_1 \times v = 1 \tag{2}$$

A.3 Curve Check

For a point $P = (\mathbb{x}, \mathbb{y}) \in \mathbb{E}(\mathbb{F})$, introduce $\mathbb{t}_1, \mathbb{t}_2$ and constraints:

$$\begin{aligned} \mathbb{x} \times \mathbb{x} &= \mathbb{t}_1 && (3) \\ \mathbb{x} \times \mathbb{t}_1 &= \mathbb{t}_2 && (4) \\ \mathbb{y} \times \mathbb{y} &= \mathbb{t}_2 + A\mathbb{x} + B && (5) \end{aligned}$$

A.4 Incomplete Curve Addition

We denote by $\mathbin{⸭}$ incomplete addition on the short Weierstrass curve $\mathbb{E}$, formally:

$$\begin{aligned} \mathbb{E} \cup \{\perp\} \mathbin{⸭} \mathbb{E} \cup \{\perp\} &\to \mathbb{E} \cup \{\perp\} \\ \perp \mathbin{⸭} \_ &\mapsto \perp \\ \_ \mathbin{⸭} \perp &\mapsto \perp \\ 1 \mathbin{⸭} \_ &\mapsto \perp \\ \_ \mathbin{⸭} 1 &\mapsto \perp \\ P \mathbin{⸭} -P &\mapsto \perp, P \in \mathbb{E} \\ P \mathbin{⸭} P &\mapsto \perp, P \in \mathbb{E} \\ P \mathbin{⸭} Q &\mapsto P + Q, P \in \mathbb{E}, Q \in \mathbb{E}, \text{ Otherwise} \end{aligned}$$

In other words: for points $(\mathbb{x}_1, \mathbb{y}_1), (\mathbb{x}_2, \mathbb{y}_2) \in \mathbb{E}(\mathbb{F})$ the operation is undefined when $\mathbb{x}_1 = \mathbb{x}_2$ (and undefined on points not on the curve) or when one of the operands is the point at infinity. For three points (witnesses) $(\mathbb{x}_1, \mathbb{y}_1), (\mathbb{x}_2, \mathbb{y}_2), (\mathbb{x}_3, \mathbb{y}_3)$ we enforce $(\mathbb{x}_3, \mathbb{y}_3) = (\mathbb{x}_1, \mathbb{y}_1) \mathbin{⸭} (\mathbb{x}_2, \mathbb{y}_2)$, by introducing a free variable for the slope δ and the 3 constraints:

$$\begin{aligned} \delta \times (\mathbb{x}_2 - \mathbb{x}_1) &= \mathbb{y}_2 - \mathbb{y}_1 && (6) \\ \delta \times (\mathbb{x}_3 - \mathbb{x}_1) &= -\mathbb{y}_3 - \mathbb{y}_1 && (7) \\ \delta \times \delta &= \mathbb{x}_3 + \mathbb{x}_1 + \mathbb{x}_2 && (8) \end{aligned}$$

A.5 Checked Curve Addition

When exceptional cases may occur, we can check for these by enforcing distinct $\mathbb{x}$-coordinates. i.e. to enforce:

$$(\mathbb{x}_3, \mathbb{y}_3) = (\mathbb{x}_1, \mathbb{y}_1) + (\mathbb{x}_2, \mathbb{y}_2)$$

Enforce the constraints:

$$\mathbb{x}_1 \neq \mathbb{x}_2 \tag{9}$$

$$(\mathbb{x}_3, \mathbb{x}_3) = (\mathbb{x}_1, \mathbb{y}_1) \dotplus (\mathbb{x}_2, \mathbb{y}_2) \tag{10}$$

A.6 Secret 3-Bit Lookup

An n-dimensional secret lookup in a constant table, i.e. $v = T[b_0 + 2 \cdot b_1 + 2^2 \cdot b_2]$ for secret $b_0, b_1, b_2 \in \{0, 1\} \subseteq \mathbb{F}$ and $v \in \mathbb{F}^n$ with $T : \mathbb{N}_8 \to \mathbb{F}^n$. For a table $T : \mathbb{N}_8 \to \mathbb{F}$ the lookup requires 5 R1CS constraints:

$$b_0 \in \{0, 1\} \tag{11}$$

$$b_1 \in \{0, 1\} \tag{12}$$

$$b_2 \in \{0, 1\} \tag{13}$$

$$b_\& = b_1 \times b_2 \tag{14}$$

$$b_0 \times \begin{pmatrix} -T_0 \cdot b_\& + T_0 \cdot b_2 + T_0 \cdot b_1 - T_0 + T_2 \cdot b_\& \\ -T_2 \cdot b_1 + T_4 \cdot b_\& - T_4 \cdot b_2 - T_6 \cdot b_\& \\ +T_1 \cdot b_\& - T_1 \cdot b_2 - T_1 \cdot b_1 + T_1 - T_3 \cdot s_\& \\ +T_3 \cdot b_1 - T_5 \cdot b_\& + T_5 \cdot b_2 + T_7 \cdot b_\& \end{pmatrix}$$
$$= \begin{matrix} v - T_0 \cdot b_\& + T_0 \cdot T_2 + T_0 \cdot b_1 - T_0 + T_2 \cdot b_\& \\ -T_2 \cdot b_1 + T_4 \cdot b_\& - T_4 \cdot b_2 - T_6 \cdot b_\& \end{matrix}$$

In general, for tables $T : \mathbb{N}_8 \to \mathbb{F}^n$ the technique above requires $4 + n$ constraints: repeating the last constraint for each additional coordinate.

A.7 Circuit for Fixed-Base Exponentiation and Rerandomization

Abusing notation, we write $(\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = (\mathbb{x}, \mathbb{y}) \dotplus T$ for the constraint: $(\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = (\mathbb{x}, \mathbb{y}) \dotplus (\hat{\mathbb{x}}, \hat{\mathbb{y}})$ and $(\hat{\mathbb{x}}, \hat{\mathbb{y}}) \in T$. Multiplying a constant curve point by a secret scalar is implemented by decomposing the scalar into 3-bit windows (b_0, b_1, b_2) and defining the tables T st. the exceptional cases does not occur (except for the last table – where we use the checked version). Let $m = \lfloor \lambda/3 \rfloor + 1$, for for $i \in 1, \ldots, m-1$, define the table T_i as:

$$T_i = \left\{ \left[j \cdot 2^{3 \cdot (i-1)} + 2^{3 \cdot i)} \right] \cdot H \;\middle|\; j \in 0, \ldots, 2^3 - 1 \right\}$$

Define T_m as follows:

$$T_m = \left\{ \left[j \cdot 2^{3 \cdot (m-1)} - \sum_{i=1}^{m-1} 2^{3 \cdot i} \right] \cdot H \;\middle|\; j \in 0, \ldots, 2^3 - 1 \right\}$$

To enforce $(\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = [r] \cdot H + (\mathbb{x}, \mathbb{y})$, we express it as:

$$\begin{matrix} (\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = \\ \mathsf{Rerand}(\mathbb{x}, \mathbb{y}) \end{matrix} := \left\{ \begin{matrix} (\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = (\mathbb{x}, \mathbb{y}) + T_m + \\ (T_{m-1} \dotplus T_{m-2} \dotplus \ldots \dotplus T_1) \end{matrix} \right\} \tag{15}$$

Note the use of incomplete addition except for two curve additions. And decompose with witness $r \in \mathbb{Z}_{|\langle H \rangle|}$ as

$$r = \sum_i v_i \cdot 2^{3i}$$

A.8 Range Check

A range check for $v \in [0, 2^i)$ requires i constraints:

$$\forall\ b_i \in \{0,1\} \tag{16}$$

$$v = \sum_i 2^i \cdot b_i \tag{17}$$

A.9 Selection

Selecting a single secret entry (hidden index) from a secret vector:

$$\mathsf{Select}(\vec{\mathbb{X}}) := \left\{ (\mathbb{x}, \vec{\mathbb{X}}) : 0 = \prod_{i=1} (\mathbb{X}_i - \mathbb{x}) \right\}$$

We shall occasionally write the relation check above—selecting $\mathbb{x}$ among all siblings $\vec{\mathbb{X}}$—as follows:

$$\mathbb{x} = \mathsf{Select}(\vec{\mathbb{X}})$$

A.10 Permissibility

As discussed in Section 4.3, when checking permissibility condition at proving time, we can just verify a weaker condition: in an honestly generated tree, this will imply the stronger condition. Specifically in our circuits we check permissibility as:

$$(\mathbb{x}, \mathbb{y}) \in \mathcal{P}_{\mathbb{E}} \iff (\mathbb{x}, \mathbb{y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathbb{y}) = 1$$

That is, we do not check $\mathcal{U}_{\alpha,\beta}(-\mathbb{y}) = 0$ as in the strong definition.

B Our Concrete Construction: Two-Proofs Select-and-Rerandomize

This is a more detailed description of the construction of Section 4.2.2 instantiated with a two-cycle of curves. We can break the relation down into two parts, for odd even levels respectively. At each level we use the version of our construction with the optimizations described in Section 4.3.

B.1 Some Preliminaries and Notation

Recall that we denote by ℓ the branching factor at each level and that a tree has a total of $N := |S| = \ell^{\mathsf{D}}$.

- **Number of Layers:** Below we assume an even depth D, that is: we assume layers going from 0 to D (with the root at layer 0 and the selected leaf at even layer D as before).
- **Curves:** We denote by $\mathbb{E}_{(\text{evn})}, \mathbb{E}_{(\text{odd})}$ the two curves in the 2-cycle. Curve $\mathbb{E}_{(\text{evn})}$ is for nodes at even-indexed layers, e.g., the root is a point on $\mathbb{E}_{(\text{evn})}$. Sometimes we make explicit the group in which we are performing addition as $+_e$ or $+_o$ respectively. We denote the generators for each curve as $G_1^\star, \ldots, G_\ell^\star, H^\star$ where $\star \in \{(\text{evn}), (\text{odd})\}$. We will assume two proof systems that are friendly with respect to respectively $\mathbb{E}_{(\text{odd})}$ and $\mathbb{E}_{(\text{evn})}$. As in our concrete construction with Bulletproofs, this could also be the same proof system instantiated on different curves.
- **Children:** Given an internal commitment node in the tree $C \in \mathbb{E}_{(\text{evn})}$ (resp. $\mathbb{E}_{(\text{odd})}$), we denote its children commitments in the tree by $\mathsf{Children}(C) \in \mathbb{E}_{\text{evn}}^\ell$ (resp. $\mathbb{E}_{\text{odd}}^\ell$).
- **Path indices:** If we are selecting-and-rerandomizing leaf commitment C_{leaf} at index j_{leaf}, we denote by path be tuple of indices selected at each layer to select the j_{leaf}-th leaf. For example, if $\ell = \mathsf{D} = 4$ and we are selecting the sixth leaf then $\mathsf{path} = (1, 1, 2, 1)$.
- **Commitments selected along the path:** Given a vector path we can denote the commitments selected along the path by $\mathsf{SelComms}(\mathsf{path}) = (C_{\mathsf{sel},1}, \ldots, C_{\mathsf{sel},\mathsf{D}}) \in \mathbb{E}_{(\text{odd})} \times \mathbb{E}_{(\text{evn})} \times \cdots \times \mathbb{E}_{(\text{evn})}$, where $C_{\mathsf{sel},\mathsf{D}} = C_{\text{leaf}}$ More concretely: consider a leaf C_{leaf} and the path to it $\mathsf{path}_{\text{leaf}} = \big(p_1, \ldots, p_{\mathsf{D}}\big) \in [\ell]^{\mathsf{D}}\big)$. Along its path we are then selecting: at level 1 commitment $C_{\mathsf{sel},1}$, the p_1-th among $\mathsf{Children}(C_0)$ where C_0 is the root; at level 2 commitment $C_{\mathsf{sel},2}$,the p_2-th among $\mathsf{Children}(C_{\mathsf{sel},1})$, and so on and so forth.

– **Coordinates:** Given a node along the path $C_{\mathsf{sel,i}} = (\mathbb{x}, \mathbb{y})$, we use the following syntax to denote its coordinates: $\mathbb{x} = \mathsf{x}(C_{\mathsf{sel,i}})$ and $\mathbb{y} = \mathsf{y}(C_{\mathsf{sel,i}})$.
– **Groups of siblings:** Each node along the path $C_{\mathsf{sel,i}}$ will have $\ell - 1$ siblings. We denote the tuple of ℓ nodes (ordered as by tree constructions) from which $C_{\mathsf{sel,i}}$ has been selected—and which includes its siblings and the node $C_{\mathsf{sel,i}}$ itself—as $\mathsf{SiblingsPlusSelf}(C_{\mathsf{sel,i}})$. (that is, $\mathsf{SiblingsPlusSelf}(C_{\mathsf{sel,i}}) = \mathsf{Children}(\mathsf{parent}(C_{\mathsf{sel,i}}))$)

B.2 A Construction Based on Two-Cycles of Curve

Our construction is described in Fig. 7.

We use two sub-relations—described in Fig. 8—that are defined on their respective curves: one is for selection of nodes on odd layers and one for selection of nodes on even layers. That is, if the nodes along the path are $\mathsf{SelComms}(\mathsf{path}) = (C_{\mathsf{sel},1}, \ldots, C_{\mathsf{sel,D}}) \in \mathbb{E}_{(\mathrm{odd})} \times \mathbb{E}_{(\mathrm{evn})} \times \cdots \times \mathbb{E}_{(\mathrm{evn})}$, then the "odd" (resp., "even") relation considers selection of $(C_{\mathsf{sel},1}, C_{\mathsf{sel},3}, \ldots, C_{\mathsf{sel,D\text{-}1}}) \in \mathbb{E}^{\mathsf{D}/2}_{(\mathrm{odd})}$ (resp., $(C_{\mathsf{sel},2}, C_{\mathsf{sel},3}, \ldots, C_{\mathsf{sel,D}}) \in \mathbb{E}^{\mathsf{D}/2}_{(\mathrm{evn})}$).

The witnesses used for proving these two relations are, for each (even or odd) layer $i \in [\mathsf{D}]$:

– The commitment $C_{\mathsf{sel,i}}$ selected at layer i;
– path index p_i such that $C_{\mathsf{sel,i}}$ is the p_i-th in $\mathsf{SiblingsPlusSelf}(C_{\mathsf{sel,i}})$;
– To show rerandomization we include the masking scalar $\hat{\rho}_i$;
– To show the opening of $C_{i-1} = \mathsf{parent}(C_{\mathsf{sel,i}})$, we include:
  • A vector of $\vec{\mathbb{X}}_i$, the $\mathbb{x}$ coordinates of all the points $\mathsf{SiblingsPlusSelf}(C_{\mathsf{sel,i}})$;
  • "adjustment" scalar for permissibility $r_{\mathcal{P},i-1}$ (see also Fig. 4).

We assume the public parameters contains all the information about the curves and the generators.

B.3 Implications for $\mathbb{V}$Cash Instantiation

The relations proved in the $\mathbb{V}$Cash protocol (Section 5) relies on commitments (e.g., coins) and proofs about them. These commitments are stored in a curve tree as leaves and as a consequence belong to curve $\mathbb{E}_{(\mathrm{evn})}$. The operations to apply the PRF-based techniques in Remark 4 are in $\mathbb{E}_{(\mathrm{odd})}$, in particular $c^*_{\mathsf{sk}} \in \mathbb{E}_{(\mathrm{odd})}$.

C Anonymous Payments Formalized

In Fig. 9 we formally describe our model for UTXO-based payments with privacy requirements through a *functionality*. The functionality describes the ideal behavior of the system as "a trusted party would execute it". Proving that our construction is secure, intuitively requires showing that any attack against the protocol *was already possible* in the case of parties interacting with the functionality. This is usually tantamount to showing the existence of a simulator that, by interacting with functionality, can produce an output that is indistinguishable by that of an adversary against the protocol. We defer the reader to Section 6 in [Lin16] for further details.

Below, we refer to our concrete construction the protocol described in Section 5.3 and Fig. 6 with the 2-cycle instantiation of select-and-rerandomize (Appendix B).

Theorem 3 ($\mathbb{V}$Cash security, Informal). *Our concrete construction securely computes the functionality $\mathcal{F}_{AnonUTXO}$ in Fig. 9 in the presence of static malicious adversaries in the random-oracle model, under DLOG for the groups of $\mathbb{E}_{(evn)}$ and $\mathbb{E}_{(odd)}$ and under the simulation extractability of Bulletproofs.*

We can also obtain a stronger version of our protocol without the leakage mentioned in Remark 5 under one additional assumption, Diffie-Hellman Inversion (or DHI). We refer the reader to Appendix D for further details on the extension.

Theorem 4 ($\mathbb{V}$Cash security with PRF, Informal). *Our concrete construction securely computes the functionality $\mathcal{F}_{StrongAnonUTXO}$ in Fig. 9 in the presence of static malicious adversaries in the random-oracle model, under the same assumptions as Theorem 3 and under the hardness of the B-Diffie-Hellman Inversion problem (Section 3.1 in [DY05]) for $\mathbb{E}_{(odd)}$ where B is a bound on the total number of transactions per user throughout the history of the payment system.*

$\underline{\mathsf{SelRerand}.\mathcal{P}\,(\mathsf{pp}, S, C_{\text{leaf}})}$:

- **Reconstruct** from S the curve tree the tree with depth D and root rt. (Notice that in a concrete implementation this step can naturally be preprocessed)
- Let $\mathsf{path}_{\text{leaf}}$ be the opening path (as defined above) for leaf C_{leaf}.
- **Rerandomize** the commitments along the path. This step produces rerandomized commitments $\hat{C}_1, \ldots, \hat{C}_{\mathsf{D}}$ and respective masking elements $\hat{\rho}_1, \ldots, \hat{\rho}_{\mathsf{D}}$. Formally, for each $i \in [\mathsf{D}/2]$:
  - Sample $\hat{\rho}_{2i-1} \leftarrow\!\$\, \mathbb{F}_{|\mathbb{E}_{(\text{odd})}|}$ and $\hat{\rho}_{2i} \leftarrow\!\$\, \mathbb{F}_{|\mathbb{E}_{(\text{evn})}|}$
  - Let $\hat{C}_{2i-1} \leftarrow C_{\mathsf{sel,2i\text{-}1}} +_e [\hat{\rho}_{2i-1}] \cdot H^{(\text{evn})}$ and $\hat{C}_{2i} \leftarrow C_{\mathsf{sel,2i}} +_o [\hat{\rho}_{2i}] \cdot H^{(\text{odd})}$
- **Prove** in zero-knowledge with $\mathsf{ZK}[\mathbb{E}_{(\text{odd})}]$ the odd layers constraints described for $\mathsf{SelRerand}^{(\text{odd})}\left(\mathsf{rt}, \left(\hat{C}_i\right)_{i=1,3,\ldots,\mathsf{D}-1}\right)$. Call this proof $\pi_{(\text{odd})}$.
- **Prove** in zero-knowledge with $\mathsf{ZK}[\mathbb{E}_{(\text{evn})}]$ the event layers constraints described for $\mathsf{SelRerand}^{(\text{evn})}\left(\left(\hat{C}_i\right)_{i=2,4,\ldots,\mathsf{D}}\right)$. Call this proof $\pi_{(\text{evn})}$.
- Return:
  - $C' := \hat{C}_{\mathsf{D}}$ // rerandomization of the selected leaf C_{leaf}
  - $\pi^* := \left(\hat{C}_1, \ldots, \hat{C}_{\mathsf{D}-1}, \pi_{(\text{odd})}, \pi_{(\text{evn})}\right)$

$\underline{\mathsf{SelRerand}.\mathcal{V}\left(\mathsf{pp}, \mathsf{rt}, C', \pi^* = \left(\hat{C}_1, \ldots, \hat{C}_{\mathsf{D}-1}, \pi_{(\text{odd})}, \pi_{(\text{evn})}\right)\right)}$:

- **Verify** $\pi_{(\text{odd})}$ for relation/public parameters $\mathsf{SelRerand}^{(\text{odd})}\left(\mathsf{rt}, \left(\hat{C}_i\right)_{i=1,3,\ldots,\mathsf{D}-1}\right)$ with $\mathsf{ZK}[\mathbb{E}_{(\text{odd})}]$.
- **Verify** $\pi_{(\text{evn})}$ for relation/public parameters $\mathsf{SelRerand}^{(\text{evn})}\left(\hat{C}_2, \hat{C}_4, \ldots, \hat{C}_{d-2}, C'\right)$ with $\mathsf{ZK}[\mathbb{E}_{(\text{evn})}]$.

Fig. 7: Our concrete construction for Select-and-Rerandomize with 2-cycle of curves. For reference, see Section 4.2.2 for its generic counterpart.

$$\begin{aligned}
&\mathsf{SelRerand}^{(\mathrm{odd})}\Big(\mathsf{rt}, \big(\hat{C}_i\big)_{i=1,3,\dots,\mathsf{D}-1}\Big) \coloneqq \Big\{ \\
&\quad\Big(\Big(C_{\mathsf{sel,i}} := (\mathbb{x}_{\mathsf{sel},i}, \mathbb{y}_{\mathsf{sel},i}), \vec{\mathbb{X}}_i, \hat{\rho}_i\Big)_{i=1,3,\dots,\mathsf{D}-1}, \\
&\qquad \mathsf{path}^{(\mathrm{odd})}_{\mathrm{leaf}} := (p_1, p_3, \dots, p_{\mathsf{D}-1}), \\
&\qquad \vec{r}^{(\mathrm{odd})}_{\mathcal{P}} := (r_{\mathcal{P},1}, r_{\mathcal{P},3}, \dots, r_{\mathcal{P},\mathsf{D}-1})\Big) : \\
&\qquad C_0 := \langle \vec{G}^{(\mathrm{odd})}, \Big[\vec{\mathbb{X}}_1\Big]\rangle +_e [r_{\mathcal{P},0}] \cdot H^{(\mathrm{evn})} && /\!/ \text{ Open commitment to vector of } \mathbb{x}\text{-coordinates} \\
&\qquad \mathsf{rt} = C_0 && /\!/ \text{ Check against root value} \\
&\qquad \mathbb{x}_{\mathsf{sel},1} = \mathsf{Select}(\vec{\mathbb{X}}_1) && /\!/ \text{ Select a coordinate (using path index } p_1\text{)} \\
&\qquad (\mathbb{x}_{\mathsf{sel},1}, \mathbb{y}_{\mathsf{sel},1}) \in \mathcal{P}_{\mathbb{E}_{(\mathrm{odd})}} && /\!/ \text{ Decompress to ``permissible'' point.} \\
&\qquad \hat{C}_1 = \mathsf{Rerand}(\mathbb{x}_{\mathsf{sel},1}, \mathbb{y}_{\mathsf{sel},1}) && /\!/ \text{ Rerandomize inner commitment using } \hat{\rho}_1. \\
&\qquad \vdots \\
&\qquad C_{\mathsf{D}-2} := (\mathbb{x}_{\mathsf{D}-2}, \mathbb{y}_{\mathsf{D}-2}) = \langle \vec{G}^{(\mathrm{evn})}, \Big[\vec{\mathbb{X}}_{\mathsf{D}-1}\Big]\rangle +_e [r_{\mathcal{P},\mathsf{D}-2}] \cdot H^{(\mathrm{evn})} && /\!/ \text{ Open commitment to vector of } \mathbb{x}\text{-coordinates} \\
&\qquad \mathbb{x}_{\mathsf{sel},\mathsf{D}-1} = \mathsf{Select}(\vec{\mathbb{X}}_{\mathsf{D}-1}) && /\!/ \text{ Select a coordinate (using path index } p_{\mathsf{D}-1}\text{)} \\
&\qquad (\mathbb{x}_{\mathsf{sel},\mathsf{D}-1}, \mathbb{y}_{\mathsf{sel},\mathsf{D}-1}) \in \mathcal{P}_{\mathbb{E}_{(\mathrm{odd})}} && /\!/ \text{ Decompress to ``permissible'' point.} \\
&\qquad \hat{C}_{\mathsf{D}-1} = \mathsf{Rerand}(\mathbb{x}_{\mathsf{sel},\mathsf{D}-1}, \mathbb{y}_{\mathsf{sel},\mathsf{D}-1}) && /\!/ \text{ Rerandomize inner commitment using } \hat{\rho}_{\mathsf{D}-1}. \\
&\qquad \Big\}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{SelRerand}^{(\mathrm{evn})}\Big(\mathsf{rt}, \big(\hat{C}_i\big)_{i=2,4,\dots,\mathsf{D}}\Big) \coloneqq \Big\{ \\
&\quad\Big(\Big(C_{\mathsf{sel,i}} := (\mathbb{x}_{\mathsf{sel},i}, \mathbb{y}_{\mathsf{sel},i}), \vec{\mathbb{X}}_i, \hat{\rho}_i\Big)_{i=2,4,\dots,\mathsf{D}}, \\
&\qquad \mathsf{path}^{(\mathrm{evn})}_{\mathrm{leaf}} := (p_2, p_4, \dots, p_{\mathsf{D}}), \\
&\qquad \vec{r}^{(\mathrm{evn})}_{\mathcal{P}} := (r_{\mathcal{P},2}, r_{\mathcal{P},4}, \dots, r_{\mathcal{P},\mathsf{D}})\Big) : \\
&\qquad C_1 := \langle \vec{G}^{(\mathrm{evn})}, \Big[\vec{\mathbb{X}}_2\Big]\rangle +_e [r_{\mathcal{P},1}] \cdot H^{(\mathrm{odd})} && /\!/ \text{ Open commitment to vector of } \mathbb{x}\text{-coordinates} \\
&\qquad \mathbb{x}_{\mathsf{sel},2} = \mathsf{Select}(\vec{\mathbb{X}}_2) && /\!/ \text{ Select a coordinate (using path index } p_2\text{)} \\
&\qquad (\mathbb{x}_{\mathsf{sel},2}, \mathbb{y}_{\mathsf{sel},2}) \in \mathcal{P}_{\mathbb{E}_{(\mathrm{evn})}} && /\!/ \text{ Decompress to ``permissible'' point.} \\
&\qquad \hat{C}_2 = \mathsf{Rerand}(\mathbb{x}_{\mathsf{sel},2}, \mathbb{y}_{\mathsf{sel},2}) && /\!/ \text{ Rerandomize inner commitment using } \hat{\rho}_2. \\
&\qquad \vdots \\
&\qquad C_{\mathsf{D}-1} := (\mathbb{x}_{\mathsf{D}-1}, \mathbb{y}_{\mathsf{D}-1}) = \langle \vec{G}^{(\mathrm{odd})}, \Big[\vec{\mathbb{X}}_{\mathsf{D}}\Big]\rangle +_e [r_{\mathcal{P},\mathsf{D}-1}] \cdot H^{(\mathrm{odd})} && /\!/ \text{ Open commitment to vector of } \mathbb{x}\text{-coordinates} \\
&\qquad \mathbb{x}_{\mathsf{sel},\mathsf{D}} = \mathsf{Select}(\vec{\mathbb{X}}_{\mathsf{D}}) && /\!/ \text{ Select a coordinate (using path index } p_{\mathsf{D}}\text{)} \\
&\qquad (\mathbb{x}_{\mathsf{sel},\mathsf{D}}, \mathbb{y}_{\mathsf{sel},\mathsf{D}}) \in \mathcal{P}_{\mathbb{E}_{(\mathrm{evn})}} && /\!/ \text{ Decompress to ``permissible'' point.} \\
&\qquad \hat{C}_{\mathsf{D}} = \mathsf{Rerand}(\mathbb{x}_{\mathsf{sel},\mathsf{D}}, \mathbb{y}_{\mathsf{sel},\mathsf{D}}) && /\!/ \text{ Rerandomize inner commitment using } \hat{\rho}_{\mathsf{D}}. \\
&\qquad \Big\}
\end{aligned}$$

Fig. 8: Auxiliary relations for even/odd layers.

```
$\mathcal{F}_{\text{AnonUTXO}}$

Setup: $(setup, \mathsf{UTXO}) \overset{\mathsf{recv}}{\leftarrow} \mathrm{port}.\mathsf{infl}$
1 :  assert $\sum_{(\ldots,v)\in\mathsf{UTXO}} v \leq \texttt{MAX-MONEY} \land$  2 : $\left|\bigcup_{(\mathsf{id},\ldots)\in\mathsf{UTXO}} \mathsf{id}\right| \leq |\mathsf{UTXO}|$

Corrupt: $(corrupt, p) \overset{\mathsf{recv}}{\leftarrow} \mathrm{port}.\mathsf{infl}$
1 :  $\mathsf{Corrupt} \leftarrow \mathsf{Corrupt} \cup \{p\}$

Transfer: $(tx, \mathsf{id}_1, \mathsf{id}_2, (v'_1, \mathsf{t}'_1), (v'_2, \mathsf{t}'_2)) \overset{\mathsf{recv}}{\leftarrow} \mathrm{port}.\mathsf{P}_{\mathsf{p}}$
    // Check that outputs were sent to $\mathsf{p}$ and balances match
1 :  assert $\mathsf{tx}_1 = (\mathsf{id}_1, \mathsf{f}_1, \mathsf{p}, v_1) \in \mathsf{UTXO}$
2 :  assert $\mathsf{tx}_2 = (\mathsf{id}_2, \mathsf{f}_2, \mathsf{p}, v_2) \in \mathsf{UTXO}$
3 :  assert $v_1 + v_2 = v'_1 + v'_2$
    // Corrupted party created the output: learns when it is spent
4 :  $\boxed{\textbf{if } \mathsf{f}_1 \in \mathsf{Corrupt} : \mathrm{port}.\mathsf{leak} \overset{\mathsf{send}}{\leftarrow} \mathsf{id}_1}$
5 :  $\boxed{\textbf{if } \mathsf{f}_2 \in \mathsf{Corrupt} : \mathrm{port}.\mathsf{leak} \overset{\mathsf{send}}{\leftarrow} \mathsf{id}_2}$
    // Update UTXO set
6 :  $\mathsf{id}'_1 \overset{\mathsf{recv}}{\leftarrow} \mathrm{port}.\mathsf{infl}; \mathsf{id}'_2 \overset{\mathsf{recv}}{\leftarrow} \mathrm{port}.\mathsf{infl};$   // fresh id's.
7 :  $\mathsf{tx}'_1 \leftarrow (\mathsf{id}'_1, \mathsf{p}, \mathsf{t}'_1, v'_1); \mathsf{tx}'_2 \leftarrow (\mathsf{id}'_2, \mathsf{p}, \mathsf{t}'_2, v'_2)$
8 :  $\mathsf{UTXO} \leftarrow \mathsf{UTXO} \setminus \{\mathsf{tx}_1, \mathsf{tx}_2\}$
9 :  $\mathsf{UTXO} \leftarrow \mathsf{UTXO} \cup \{\mathsf{tx}'_1, \mathsf{tx}'_2\}$
    // Notify recipients
10 : $\mathrm{port}.\mathsf{P}_{\mathsf{t}'_1} \overset{\mathsf{send}}{\leftarrow} \mathsf{tx}'_1; \mathrm{port}.\mathsf{P}_{\mathsf{t}'_2} \overset{\mathsf{send}}{\leftarrow} \mathsf{tx}'_2$
```

Fig. 9: Ideal Functionality $\mathcal{F}_{\text{AnonUTXO}}$ for Anonymous Payments. A stronger version $\mathcal{F}_{\text{StrongAnonUTXO}}$ (see also Remark 5) is obtained by removing leakages marked in blue inside frameboxes.

D Rerandomization of Key with PRF

At a high level, the ameliorated scheme works as follows:

- The receiver's public key is a rerandomizable commitment c_{sk} to a PRF key sk; the sender creates an output by sending $\mathsf{tx} = (c^*_{\mathsf{sk}}, \mathsf{c}^{(1)}_{\mathrm{out}}, \mathsf{c}^{(2)}_{\mathrm{out}}, \dots)$ to the network, where c^*_{sk} is a rerandomization of the receivers public key and $\mathsf{c}_{\mathrm{out}}$-s are homomorphic commitments (as described earlier). For each $\mathsf{c}_{\mathrm{out}}$, the network homomorphically adds $\mathcal{H}(\mathsf{c}_{\mathrm{out}})$ and c^*_{sk} to $\mathsf{c}_{\mathrm{out}}$ and obtains $\mathsf{c}'_{\mathrm{out}}$, which is added to the accumulator as before (this should be a permissible point).
- To spend $\mathsf{c}^*_{\mathrm{out}}$ (rerandomization of $\mathsf{c}'_{\mathrm{out}}$) the receiver proves $\mathsf{t} = \mathsf{PRF}_{\mathsf{sk}}(\mathcal{H}(\mathsf{c}_{\mathrm{out}}))$ without revealing sk or $\mathcal{H}(\mathsf{c}_{\mathrm{out}})$, where t acts as a spending tag.
- The PRF key is $\mathsf{sk} \in \mathbb{F}_{|\mathbb{E}|}$. One can commit to the PRF key using a Pedersen commitment:

$$c_{\mathsf{sk}} \leftarrow [\mathsf{sk}] \cdot G + [r] \cdot H \in \mathbb{E}$$

- The network computes:

$$c_{\mathsf{sk}+\mathcal{H}(\mathsf{c}_{\mathrm{out}})} \leftarrow c^*_{\mathsf{sk}} + [\mathcal{H}(\mathsf{c}_{\mathrm{out}})] \cdot G$$

  and adds $c_{\mathsf{sk}+\mathcal{H}(\mathsf{c}_{\mathrm{out}})}$ to $\mathsf{c}_{\mathrm{out}}$ "in the exponent" (we abuse notation and letting $[X]$ be the encoding of $X \in \mathbb{E}$ as a scalar). That is, we rerandomize as in the select-and-rerandomize proof; notice that $\mathsf{c}_{\mathrm{out}}$ has a proof of well-formedness. The network computes: $\mathsf{c}'_{\mathrm{out}} \leftarrow \mathsf{c}_{\mathrm{out}} + \left[c_{\mathsf{sk}+\mathcal{H}(\mathsf{c}_{\mathrm{out}})}\right] \cdot \hat{G}_{\mathsf{PRF}} \in \hat{\mathbb{E}}$.
- To spend, the receiver extracts and rerandomizes the commitment $c_{\mathsf{sk}+\mathcal{H}(\mathsf{c}_{\mathrm{out}})}$ in the exponent using the same technique as select-and-rerandomize to obtain $c^*_{\mathsf{sk}+\mathcal{H}(\mathsf{c}_{\mathrm{out}})}$ and proves:

$$c^*_{\mathsf{sk}+\mathcal{H}(\mathsf{c}_{\mathrm{out}})} = [x] \cdot G + [r^*] \cdot H \wedge \mathsf{t} = \left[x^{-1}\right] \cdot G$$

  where t is the tag of the spent coin.
- All additional items are added to the signature for validation.

E Dynamic Sets with Curve Tree

In our exposition in the main text we described a construction for a static set. In many applications, including $\mathbb{V}$Cash, we will require dynamically updating the accumulator.

An easy solution is to represent all uninitialized leaf positions with a conventional dummy value. Whenever we insert a new leaf, it is easy to update Curve Trees without holding the whole set, as for Merkle Trees. This can be done by storing a "frontier" of internal nodes (of size $O(\mathsf{D})$) to the group of leaves we are updating. We then update each one of these internal nodes through group operations removing the dummy value, removing the permissibility masking, adding the new value in the appropriate generator and then making the node permissible again. This consists of $O(\mathsf{D})$ group operations.

Using this solution in concrete applications we should naturally make sure that one cannot exploit the dummy value to convincingly open to that element (which is supposed to be absent from the set). A simple solution is to choose a dummy value that is not permissible.

F Benchmarks of Multicore Batch Verification

The benchmarks provided in Section 6 are indicative of the performance on a laptop, and thus a useful metric for judging the performance of the system from the perspective of a user sending and receiving transactions. But for a server validating all transactions in the system it is relevant to investigate how throughput scales with additional parallelism. In Table 3 and Table 4 we provide more detailed benchmarks of batch verification to illustrate how performance scales with the size of batches and number of cores. Concretely we run the single core and 4 core benchmarks on the same c6i.2xlarge instance used for the rest of our benchmarks, while the other benchmarks are run on a c6i.4xlarge (16 vCPUs, 8 cores) and a c6i.24xlarge (96 vCPUs, 48 cores) respectively.

Set size	Batch size	Verification time single core (ms)	Verification time 4 cores (ms)	Verification time 8 cores (ms)	Verification time 48 cores (ms)
	1	102.84	24.03	14.49	6.73
	2	107.91	25.21	14.78	7.20
	10	147.40	35.51	20.99	9.63
2^{20}	50	351.04	83.52	48.77	18.65
	100	609.24	143.36	81.69	30.18
	150	877.20	205.50	114.37	43.20
	200	1131.73	264.44	146.74	55.17
	1	186.50	41.78	23.22	11.44
	2	194.58	43.43	23.53	12.12
	10	259.84	60.84	33.14	16.62
2^{32}	50	588.76	139.41	80.41	32.40
	100	999.50	236.60	135.98	53.52
	150	1412.85	335.90	192.32	76.36
	200	1847.01	440.13	252.76	98.83
	1	187.90	42.88	24.20	12.24
	2	197.34	44.20	24.93	13.14
	10	273.67	64.64	36.85	17.97
2^{40}	50	657.67	155.78	91.65	34.92
	100	1140.45	269.76	156.45	57.28
	150	1630.22	384.77	221.17	82.43
	200	2136.75	507.10	293.53	108.07

Table 3: Batch verification of the select and rerandomize relation.

Anonymity set size	Batch size	Verification time single core (ms)	Verification time 4 cores (ms)	Verification time 8 cores (ms)	Verification time 48 cores (ms)
	1	188.35	42.75	24.57	12.43
	2	198.26	44.31	27.82	14.31
	10	278.25	65.10	38.96	17.89
2^{20}	50	685.56	161.20	92.04	35.77
	100	1192.73	282.03	160.11	60.03
	150	1717.02	403.55	228.26	84.01
	200	2252.22	531.83	301.29	109.61
	1	346.82	81.27	47.56	22.69
	2	364.07	84.72	51.63	24.67
	10	499.59	120.10	71.59	32.02
2^{32}	50	1181.85	285.35	165.57	65.02
	100	2050.01	494.47	286.31	107.86
	150	2924.37	703.80	406.29	150.73
	200	3808.79	911.80	528.20	193.42
	1	350.46	82.83	48.46	24.42
	2	370.48	87.22	52.45	27.43
	10	529.79	128.42	77.39	35.28
2^{40}	50	1331.38	321.68	185.14	70.24
	100	2342.38	565.64	323.12	116.89
	150	3382.79	810.16	461.78	163.34
	200	4417.54	1053.20	600.38	212.68

Table 4: Batch verification of the pour relation.