Curve Trees: Practical and Transparent Zero-Knowledge Accumulators

Matteo Campanelli, *Protocol Labs;* Mathias Hall-Andersen and Simon Holmgaard Kamp, *Aarhus University, Denmark*

https://www.usenix.org/conference/usenixsecurity23/presentation/campanelli

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

Curve Trees: Practical and Transparent Zero-Knowledge Accumulators

Matteo Campanelli
Protocol Labs
matteo@protocol.ai

Mathias Hall-Andersen
Aarhus University
ma@cs.au.dk

Simon Holmgaard Kamp
Aarhus University
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 2.9KB, its proving takes 2s and its verification 40ms 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 4KB. 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 [10]).

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 1.1 where we compare to transparent polynomial commitments and ring signatures [39]) 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) [36] and in [18]. 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[1]. Other solutions, such as [10, 20], mitigate this problem by requiring a trusted setup 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 [25], 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

[1] https://z.cash/technology/paramgen/

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[2] 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 2.1). 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 [49].

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 2.9KB, its proving takes 2s and its verification 40ms on an ordinary laptop.

Using our construction as a building block we can construct a simple and concretely efficient anonymous payment system with full anonymity set[3] and *transparent setup*. We dub this payment system $\mathbb{V}$Cash[4]. 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 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 [2]. 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 [22]. See also Appendix D.

1.1 Related Work

1.1.1 Zero-Knowledge Sets.

Seminal work by Micali, Rabin and Kilian [43] introduce the notion of a "zero-knowledge set": a hiding commitment to a set, enabling membership and non-membership proofs. Note that this is exactly complementary to the goal of this paper: in zero-knowledge sets, the *set is hidden* and the *retrieved elements public*, here the *set is public* and the *retrieved element hidden*. In Camenisch-Stadler notation (Section 2.2.1) this relation is $\{(S,r) : c = \mathsf{Com}(S;r) \wedge x \in S\}$ instead of $\{(x,r) : c = \mathsf{Com}(x;r) \wedge x \in S\}$.

Highly efficient constructions of zero-knowledge sets are known under a range of assumptions, notably Chase et al. [24] generalize the original construction by Micali et al. using Mercurial commitments.

1.1.2 Accumulators from Groups of Unknown Order.

The original work by Benaloh and de Mare [9] introducing cryptographic accumulators provides a simple construction based on strong RSA: a set of prime integers are accumulated by iteratively exponentiation in an RSA group. Camenisch and Lysyanskaya [16] extended this accumulator to be dynamic, Baldimtsi et al. [5] generically obtaining an adaptively sound dynamic accumulator by combining 1) an adaptively sound positive additive accumulator and 2) a non-adaptively sound positive dynamic accumulator. Rather than RSA, these constructions can be can instantiated with class groups: which avoids the need for a trusted setup, ut incurs a sustantial $\approx 20\times$ computational overhead at the same security level.

1.1.3 Accumulators from Bilinear Pairings.

Nguyen constructed accumulator from bilinear pairings [46], this construction was subsequently extended by Damgård and Triandopoulos [26] to support non-membership proofs. More recently Ghosh et al. [32] showed how to prove membership in zero-knowledge. In the concurrent work Zapico

[2]In the sense of the commit-and-prove building blocks in LegoSNARK [19] and in the work by Lipmaa [41].

[3]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 if the set consists of the whole history of transactions.

[4]As a reference to both Zcash and $\mathbb{V}$eksel [20] from which it borrows part of its design.

et al. [50] reduces the computational cost of proving membership from $O(n)$ to $O(\log(n))$, by relying on an $O(n \log n)$ precomputation. Common for all these works is the reliance on a structured "powers-of-τ" style structured reference string (SRS): size of the public parameters is proportional to the (apriori bounded) maximum set size and knowledge of the trapdoor breaks binding.

1.1.4 Authenticated Hash Tables & Verkle Trees.

Charalapos, et al. [47] suggests using a tree of accumulators: where every internal node is a cryptographic accumulator containing all its children. Which allows a trade-off between membership proof size and cost of updating the accumulator. The same concept ("Verkle Trees") was subsequently independently rediscovered by industry-afflicated people [34]. We observed that Charalapos et al. holds a patent for this construction [23] when used for authenticated computation, it is unclear if this applies to the scheme as used in the Ethereum blockchain. Our work differs from these by not using an accumulator at each level, the compression function is a simple Pedersen commitment, furthermore these works do not allow/describe efficient zero-knowledge membership proofs.

1.1.5 Curve Trees and Algebraic Merkle Trees.

The closest related (zero-knowledge) accumulator is the approach taken in Zcash (Sapling and Orchard versions) [36], in which a Merkle tree is instantiated with a hash function admitting an efficient algebraic description. In case of Zcash this hash function is based on multi-scalar exponentation over specially chosen elliptic curves. For "Pedersen hashes" as used in Zcash Sapling, the resulting circuit requires ≈ 44000 constraints (multiplications) for memberships of size 2^{32}, Our approach, on the other hand, requires proving ≈ 4500 constraints in zero-knowledge; roughly an order of magnitude less. Merkle trees instantiated with "SNARK-friendly hash functions" (e.g. Poseidon [33]) has similar performance compared to ours (see Section 8), however the concrete security of these hash functions is less well understood [8] [1].

1.1.6 Halo2 and Recursive Proofs.

Halo2[5] is a transparent (zero-knowledge) proof system enabling efficient recursion using "atomic accumulation" and cycles of elliptic curves. For efficiency the curves used by Halo2 need to have a "smooth" multiplicative subgroup to perform FFT which rules out some curves, in particular the `secp256k1` / `secq256k1` cycle (instead supported by our Bulletproofs implementation). This requirement restricts Halo2's compatibility with systems using other curves.

Although both—curve trees and Halo2—rely on the special algebraic structure of a cycle of curves, their goals are orthogonal: Halo2 is a proof system, ours a specialized data structure for zero-knowledge for set membership. Our techniques rely on a commit-and-prove which we instantiate with Bulletproofs for easy comparison.[6] It is possible to instantiate our scheme with Halo2; Halo 2 is ultimately *not* a competing approach but a potential way to apply the Curve Tree framework. However Halo2's generalized PLONK-based arithmetization [31] enables a more complex set of potential optimizations, including custom gates and lookups, which makes an apple-to-apple comparison substantially harder. We believe that replacing Bulletproofs with Halo2 would improve concrete performance: via custom gates for ECC operations and tables of precomputed points.

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

2 Preliminaries

Familiarity with elliptic curves and non-interactive proof systems is a prerequisite for this paper and in this section we provide a brief (and incomplete) introduction to these subjects. Since our techniques will only apply to elliptic curves we do not generalize to other group structures.

2.1 Elliptic Curves

We denote by $\mathbb{E}[\mathbb{F}_q] \subseteq \mathbb{F}_q \times \mathbb{F}_q$ the set of points in $(\mathbb{X}, \mathbb{Y})$ on the elliptic curve $\mathbb{E}$ [45]. We denote points on elliptic curves using upper-case letters (e.g. G and H). Whenever clear from context we might omit the *base field* $\mathbb{F}_q$ and simply write $\mathbb{E}$. The curve points form an Abelian group $(\mathbb{E}[\mathbb{F}_q], +)$; we use "additive notation". Throughout this paper, the number of points on $\mathbb{E}[\mathbb{F}_q]$ denoted $p := |\mathbb{E}[\mathbb{F}_q]|$ will always be prime, hence the group is cyclic. We call the prime field $\mathbb{F}_p \cong \mathbb{Z}/(p\mathbb{Z})$ the *scalar field* of $\mathbb{E}[\mathbb{F}_q]$ and denote by $[s] \cdot G$, with $s \in \mathbb{F}_p$ and $G \in \mathbb{E}[\mathbb{F}_q]$, s acting on G in the $\mathbb{Z}$-module ("scalar multiplication"). We denote by $\langle \vec{s}, \vec{G} \rangle = \sum_i [s_i] \cdot G_i$ the "inner product" between a vector of scalars $\vec{s} \in \mathbb{F}_p^n$ and a list of group elements $\vec{G} \in \mathbb{E}[\mathbb{F}_q]^n$.

2.1.1 Assumption: Generalized Discrete-Log

We rely on a common variant of the discrete logarithm assumption for multiple generators over elliptic curves:

Assumption 1 (Generalized Discrete-Log). *Let $\mathcal{G}(1^\lambda)$ a procedure for sampling a new elliptic curve. For all PPT adversaries $\mathcal{A}$ and $m \geq 2$:*

$$\Pr\left[\begin{array}{c} \langle \vec{a}, \vec{G} \rangle = 0 \in \mathbb{E}[\mathbb{F}_q] \\ \wedge\, \vec{a} \neq \vec{0} \in (\mathbb{F}_p)^m \end{array} : \begin{array}{r} (\mathbb{E}, \mathbb{F}_q, \mathbb{F}_p) \leftarrow \mathcal{G}(1^\lambda) \\ \vec{G} \leftarrow\!\!\$\, \mathbb{E}[\mathbb{F}_q]^m \\ \vec{a} \leftarrow \mathcal{A}((\mathbb{E}, \mathbb{F}_q, \mathbb{F}_p), \vec{G}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

[6] The Bulletproof arithmetization is R1CS, hence comparing the number of constraints is easy

We refer to this assumption as DLOG throughout the paper. Note that generalized variant of DLOG has a tight reduction to the standard ($m = 2$) variant.

2.1.2 Pedersen Commitments

Throughout the paper we will rely on the ubiquitous Pedersen commitment scheme. The setup consists of $(\mathbb{E}, \mathbb{F}_p, \mathbb{F}_q, \ell, \vec{G}, H)$, with $\mathbb{F}_p = |\mathbb{E}[\mathbb{F}_q]|$, $G_1, \ldots, G_\ell, H \in \mathbb{E}[\mathbb{F}_q]$. The commitment to $\vec{v} \in \mathbb{F}_p^\ell$ with randomness r is computed as follows:

$$C = \mathsf{Com}(\vec{v}; r) = \langle \vec{v}, \vec{G} \rangle + [r] \cdot H \in \mathbb{E}[\mathbb{F}_q]$$

It is easy to see that computational binding follows from DLOG (Assumption 1). Hiding is perfect and follows from the observation that $[r] \cdot H$ with $r \leftarrow\$ \mathbb{F}_p$ is uniformly distributed over the group $\mathbb{E}[\mathbb{F}_q]$. Importantly, Pedersen commitments are *rerandomizable commitments*: sampling $\delta \leftarrow\$ \mathbb{F}_p$ and computing $C^* \leftarrow C + [\delta] \cdot H$ yields a commitment to the same $\vec{v}$ with randomness $r + \delta$, furthermore the distribution of C^* is independent of C: it is a "fresh" perfectly hiding commitment to the same value.

2.1.3 Avoiding Bit Decomposition via 2-Cycles of Curves

A 2-cycle of elliptic curves consists of two elliptic curves $\{\mathbb{E}_{(\text{evn})}, \mathbb{E}_{(\text{odd})}\}$ and two prime fields $\{\mathbb{F}_p, \mathbb{F}_q\}$ such that:

$$p = |\mathbb{E}_{(\text{evn})}[\mathbb{F}_q]| \text{ And } q = |\mathbb{E}_{(\text{odd})}[\mathbb{F}_p]|$$

In other words: the base/scalar fields of the two curves are complementary. Crucial for our application will be the observation that a point $(\mathbb{X}, \mathbb{Y}) \in \mathbb{E}_{(\text{evn})}[\mathbb{F}_q]$ can be treated as a pair of scalars on $\mathbb{E}_{(\text{odd})}$, e.g. $[\mathbb{X}] \cdot G_1 + [\mathbb{Y}] \cdot G_2 \in \mathbb{E}_{(\text{odd})}[\mathbb{F}_p]$ for $G_1, G_2 \in \mathbb{E}_{(\text{odd})}[\mathbb{F}_p]$ is a well-defined operation. The observant reader will see that this defines Pedersen commitments in $\mathbb{E}_{(\text{odd})}$ to lists of points on $\mathbb{E}_{(\text{evn})}$, without relying on bit-decomposition for field elements or hashing, making it cheaper in zero-knowledge. Numerous instantiations of 2-cycles exists, e.g., the Pasta cycle [35] (used in this paper and Halo2) or the well known `secp256k1` / `secq256k1` cycle [7]. No known attacks make use of this additional structure, additionally we do not require any efficiently computable pairings on either curve.

2.2 Non-Interactive Zero-Knowledge Proofs

2.2.1 Camenisch-Stadler Notation

When expressing an NP relation $R(x, w)$ we use a variant of Camenisch-Stadler notation [17], the witness w is explicitly (enclosed in brackets) and the public statement x is defined by all remaining terms e.g. the "discrete log relation" $\mathcal{R} := \{(z) : y = [z] \cdot G\}$ – the witness is the scalar $z \in \mathbb{F}_p$, while group elements $G, y \in \mathbb{E}$ constitute the instance.

[7]With `secp256k1` being used by the Bitcoin blockchain.

2.2.2 NIZKAoKs

Definition 1. *A NIZKAoK 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.*

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 NIZKAoKs, i.e. whose algorithms use a reference string urs *sampled uniformly.*

We require a NIZKAoK 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* $\pi \leftarrow \mathsf{Prove}(\mathsf{urs}, R, x, w)$*. For security we require standard notions of knowledge-soundness and zero-knowledge:*

Knowledge-Soundness. *For all $\lambda \in \mathbb{N}$ and for all (non-uniform) efficient adversaries $\mathcal{A}$, there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that*

$$\Pr\left[\begin{array}{l} \mathsf{urs} \leftarrow\leftarrow\$ \{0,1\}^{\mathsf{poly}(\lambda)}; \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{urs}) \\ w \leftarrow \mathcal{E}(\mathsf{urs}) \end{array} : \begin{array}{r} R_\lambda(x, w) \neq 1 \wedge \\ \mathsf{Vfy}(\mathsf{urs}, x, \pi) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Note the order of quantifiers: the extractor $\mathcal{E}$ depends on $\mathcal{A}$.

Zero-Knowledge. *There exists a PPT simulator $\mathcal{S}$ such that for any $\lambda \in \mathbb{N}$, PPT $\mathcal{A}$, relation $R \in \mathfrak{R}$, $(x, w) \in R$, it holds $p_0 = p_1$ where:*

$$p_b := \Pr\left[\begin{array}{l} \quad \mathsf{urs}_1 \leftarrow\$ \{0,1\}^{\mathsf{poly}(\lambda)} \\ (\mathsf{urs}_0, \pi_0) \leftarrow \mathcal{S}(1^\lambda, x) \\ \pi_1 \leftarrow \mathsf{Prove}(\mathsf{urs}, x, w) \end{array} : \mathcal{A}(1^\lambda, \mathsf{urs}_b, \pi_b) = 1\right]$$

Remark 1 (Practical Efficiency)**.** *For a broad class of NIZKs the "cost" of the NZIK[8] scales with the number of multiplicative constraints in the relation. Hence when comparing/estimating how "expensive" a certain relation is prove using a NIZK, the number of multiplications is a broadly useful metric which translates to concrete performance for a wide range of NIZKs.*

[8]In prover time, verifier time or proof size, depending on the concrete NIZK.

2.2.3 Commit-and-Prove for Pedersen Commitments

The techniques in this paper rely heavily on efficient "commit-and-prove" NIZKs for Pedersen commitments (Section 2.1.2). A "commit-and-prove" (C&P) NIZKs for Pedersen commitments enable efficient proofs of relations in which (part of) the witness is additionally committed inside a pedersen commitment, i.e. relations of the form:

$$R^* := \{(\vec{w}, r) : C = \mathsf{Com}(\vec{w}; r) \in \mathbb{E}[\mathbb{F}_q] \wedge R(x, \vec{w}) = 1\}$$

Many efficient "commit-and-prove" NIZKs exists e.g. Bulletproofs [13], Compressed Σ-Protocols [4] and Halo2. All these schemes make black-box use of the group $\mathbb{E}[\mathbb{F}_q]$, i.e. avoid expressing the group operation as an NP relation over $\mathbb{F}_q$. Note that in the example above $\vec{w} \in \mathbb{F}_p^\ell$, where $\mathbb{F}_p$ is the scalar field of $\mathbb{E}[\mathbb{F}_q]$.

2.2.4 A Concrete C&P-NIZKAoK: Bulletproofs

We denote by $\mathsf{zk\text{-}BP}[\mathbb{E}]$ an instantiation of the Bulletproofs [13] NIZKAoK on the elliptic curve $\mathbb{E}$. The Bulletproofs scheme exhibits the following relevant properties: (1) It is a commit-and-prove for Pedersen commitments on $\mathbb{E}[\mathbb{F}_q]$ and an concretely efficient proof system for R1CS relations over $\mathbb{F}_p$. (2) The URS consists of a list of random group elements in $\mathbb{E}$ with size linear with the size of the relation being proved. (3) $\mathsf{zk\text{-}BP}[\mathbb{E}]$ is computationally (simulation) sound in the random oracle model under the DLOG assumptions (Assumption 1) on $\mathbb{E}[\mathbb{F}_q]$.

3 Zero-Knowledge Set Membership

In this section we describe a modular primitive for proving set memberships in zero-knowledge, which can be composed with commit-and-proof zero-knowledge proof system to prove additional properties about the member of the set. Informally, for a set of rerandomizable commitments (see Section 2.1.2) $S = \{C_1, \dots, C_n\}$ the primitive proves:

$$\{(r, i) : \hat{C} = \mathsf{Rerand}(C_i, r)\}$$

In other words, the commitment $\hat{C}$ is a rerandomization of *a commitment* in S, without revealing which. Additional properties about the opening of $\hat{C}$ can then be proved using commit-and-prove techniques (see Section 2.2.3).

Need for Compression. The relation outlined above has size n, as a result verifying a proof for the relation requires $O(n)$ work – to even read the statement. To reduce this cost, the set of commitments itself can be compressed using a commitment. When the set is fixed, or incrementally updated, this greatly reduces computation for both prover and verifier. We formalize this general primitive below. Our scheme achieves $O(\log(n))$ communication and $O(\sqrt[\mathsf{D}]{n})$ computation where D is a parameter of the scheme (D is both constant and small) and n is the size of the set. [9]

3.1 Select-and-Rerandomize Accumulators

Below, we fix the message space of the commitment scheme to $\mathbb{F}^k$ for some k and its randomness space to $\mathbb{F}$ – as is the case for Pedersen commitments (Section 2.1.2). Our definitions below can be generalized easily.

Definition 2 (Select-and-Rerandomize)**.** *A select-and-rerandomize accumulator scheme consists of six algorithms:*

$\mathsf{SelRerand.Setup}(1^\lambda) \to \mathsf{pp}$ *returns public parameters of the scheme. These parameters are transparent—no trusted party needs be invoked.*

$\mathsf{SelRerand.Comm}(\mathsf{pp}, v_{\text{leaf}}, o) \to C$ *commits to a string v_{leaf} with randomness o.*

$\mathsf{SelRerand.Rerand}(\mathsf{pp}, C, r) \to \hat{C}$ *rerandomizes committment C with randomness r.*

$\mathsf{SelRerand.Accum}(\mathsf{pp}, S) \to A$ *deterministically accumulates a set of commitments. We assume the set S to have a canonical order.*

$\mathsf{SelRerand.Prove}(\mathsf{pp}, S, C, r) \to \pi$ *returns a proof showing that $C \in S$ verifiable through a rerandomized commitment to c with randomness r.*

$\mathsf{SelRerand.Vfy}(\mathsf{pp}, A, \hat{C}, \pi) \to 0/1$ *verifies that $\hat{C}$ is a rerandomization of an element in the set.*

Correctness of Select-and-Rerandomize. For any $\lambda \in \mathbb{N}$, for any set $S = \{v_i\}_i$, $j^* \in [|S|]$, commitment randomness $(o_1, \dots, o_n)$ and commitment rerandomization r the verification always succeeds, i.e.

$$1 = \Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{SelRerand.Setup}(1^\lambda) \\ \forall i \in [n] : C_i \leftarrow \mathsf{SelRerand.Commit}(\mathsf{pp}, v_i, o_i) \\ A \leftarrow \mathsf{SelRerand.Accum}(\mathsf{pp}, \{C_1, \dots, C_n\}) \\ \hat{C} \leftarrow \mathsf{SelRerand.Rerand}(\mathsf{ck}, C_{j^*}, r) \\ \pi \leftarrow \mathsf{SelRerand.Prove}(\mathsf{pp}, S, C_{j^*}, r) \\ \mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, A, \hat{C}, \pi) = 1 \end{array}\right]$$

The above can be thought as a main correctness property. For it to be meaningful, it needs to be complemented by the following one, which specifically makes explicit what it means for commitments (output of Comm) to be rerandomizable: for any $\lambda \in \mathbb{N}$, for any message $m \in \mathbb{F}^k$, opening o and randomness $r \in \mathbb{F}$, it should hold that

[9] The circuit has $O(\sqrt[\mathsf{D}]{n})$ constraints for each layer, but as D is constant this does not affect the asymptotic complexity. Similarly the size of the proof is $\Theta(\log(\sqrt[\mathsf{D}]{n})) = \Theta(\log n)$

$\mathsf{SelRerand.Rerand}(\mathsf{pp},\mathsf{Comm}(\mathsf{pp},m;o)) = \mathsf{Comm}(\mathsf{pp},m;o+r)$ and $\mathsf{pp} \leftarrow \mathsf{SelRerand.Setup}(1^\lambda)$[10].

For our application/instantiation we require the select-and-rerandomize scheme to satisfy the following security notions:

Select-and-Rerandomize Binding. This is the main security definition of our model. We say the select-and-rerandomize scheme is binding if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for any PPT adversary $\mathcal{A}$:

$$\mathsf{negl}(\lambda) \geq \Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{SelRerand.Setup}(1^\lambda) \\ (\vec{v},\vec{o},\hat{v},\hat{o},\pi) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \forall i \in [n] : C_i \leftarrow \mathsf{SelRerand.Commit}(\mathsf{pp},v_i,o_i) \\ \hat{C} \leftarrow \mathsf{SelRerand.Commit}(\mathsf{pp},\hat{v},o') \\ A \leftarrow \mathsf{SelRerand.Accum}(\mathsf{pp},\{C_1,\ldots,C_n\}) \\ \textbf{if } \hat{v} \notin \vec{v} \wedge \mathsf{SelRerand.Vf}(\mathsf{pp},A,\hat{C},\pi) = 1 \end{array}\right]$$

Perfect Hiding of Commitment. For all m, m', $\mathsf{pp} \leftarrow \mathsf{SelRerand.Setup}(1^\lambda)$ the following distributions are perfectly indistinguishable:

$$\{\mathsf{SelRerand.Comm}(\mathsf{pp},m,o) \mid o \leftarrow\!\!\$\ \mathbb{F}\}$$
$$\approx \{\mathsf{SelRerand.Comm}(\mathsf{pp},m',o') \mid o' \leftarrow\!\!\$\ \mathbb{F}\}$$

Select-and-Rerandomize Zero-Knowledge. A select-and-rerandomize is (perfect) zero-knowledge if there exists an efficient simulator $\mathcal{S}$, such that for any $\lambda \in \mathbb{N}$, any (stateful) adversary $\mathcal{A}$, any $j^* \in [n]$, it holds $p_0 = p_1$ where

$$p_b := \Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{SelRerand.Setup}(1^\lambda); \\ (v_1,\ldots,v_n,o_1,\ldots,o_n) \leftarrow \mathcal{A}(\mathsf{pp}) \\ S := \{C_i = \mathsf{SelRerand.Commit}(\mathsf{pp},v_i,o_i)\}_{i\in[n]} \\ r \leftarrow\!\!\$\ \mathbb{F}; \hat{C} = \mathsf{SelRerand.Rerand}(\mathsf{pp},C_{j^*},r) \\ \pi \leftarrow X_b(\mathsf{pp},S,C,\hat{C},r) \\ \mathcal{A}(\mathsf{pp},\hat{C},\pi) = 1 \end{array}\right]$$

With $X_0(\mathsf{pp},S,C,\hat{C},r) := \mathcal{S}(\mathsf{pp},S,\hat{C})$ and $X_1(\mathsf{pp},S,C,\hat{C},r) := \mathsf{SelRerand.Prove}(\mathsf{pp},S,C,r)$.

Remark 2. *Our formalization of* SelRerand *combines together commitments, accumulators (Definition 7) and zero knowledge properties. There are, of course, other possible way to model this primitive. We found this natural enough. We also observe that our definition of correctness and binding imply their counterparts in a standard accumulator as a special case (where the commitment and the rerandomization are trivial).*

[10]Notice that homomorphic commitments (and thus Pedersen commitments) satisfy this property.

4 Curve Trees as Accumulators

In this section we first define curve trees. We then describe some of its properties in terms of commitments (that are binding and hiding). Finally, we show how to traverse a tree to show membership of a an element in zero-knowledge. The latter represents our actual construction (Fig. 1).

4.1 Intro to $(\ell, \mathbb{E}_{(\text{evn})}, \mathbb{E}_{(\text{odd})})$-Curve Trees

Recall (from Section 2.1) the observation that $[\mathbb{X}] \cdot G_1 + [\mathbb{Y}] \cdot G_2 \in \mathbb{E}_{(\text{odd})}$ for any $(\mathbb{X},\mathbb{Y}) \in \mathbb{E}_{(\text{evn})}$[11] is a meaningful operation. This is generalizable to any number ℓ of $\mathbb{E}_{(\text{evn})}$ points: computing $\langle \vec{\mathbb{X}}, \vec{G}^{\mathsf{x}}_{\mathbb{E}_{(\text{odd})}} \rangle + \langle \vec{\mathbb{Y}}, \vec{G}^{\mathsf{x}}_{\mathbb{E}_{(\text{odd})}} \rangle \in \mathbb{E}_{(\text{odd})}$ for $i \in [\ell] : (\mathbb{X}_i, \mathbb{Y}_i) \in \mathbb{E}_{(\text{evn})}$. This is a compression function $f_{\mathbb{E}_{(\text{odd})}} : {\mathbb{E}_{(\text{evn})}}^{\ell} \mapsto \mathbb{E}_{(\text{odd})}$. At this point a natural strategy to obtain an accumulator is to use $f_{\mathbb{E}_{(\text{odd})}}$ to construct a Merkle tree from $f_{\mathbb{E}_{(\text{evn})}}$: a tree in which every parent (an $\mathbb{E}_{(\text{odd})}$ point) is the hash of its children ($\mathbb{E}_{(\text{evn})}$ points) using $f_{\mathbb{E}_{(\text{odd})}}$. However this encounters an obvious "type problem": the output of $f_{\mathbb{E}_{(\text{odd})}}$ is a point on $\mathbb{E}_{(\text{odd})}$, while the inputs are points on $\mathbb{E}_{(\text{evn})}$, preventing us from applying $f_{\mathbb{E}_{(\text{odd})}}$ to the resulting outputs. The solution to this "type mismatch" is to define $f_{\mathbb{E}_{(\text{evn})}} : {\mathbb{E}_{(\text{odd})}}^{\ell} \mapsto \mathbb{E}_{(\text{evn})}$ analogously to $f_{\mathbb{E}_{(\text{odd})}}$ and *alternate the compression function* at every level of the tree. We call this construction a *Curve Tree*, which can be seen as an "algebraically compatible" Merkle tree using Pedersen commitments alternating over $\mathbb{E}_{(\text{evn})}/\mathbb{E}_{(\text{odd})}$: a parent node on one curve will be the hash of its children, represented as points on the other curve. To refer more easily to curves alternating within a tree, we introduce the following piece of notation.

Remark 3 (Notation for alternating curves)**.** *As mentioned above, a curve tree alternates between curves at each level. If we are referring to a specific "current" level (obvious from context), we will denote the corresponding curve as $\mathbb{E}_{(\_)}$. The "other" curve will be denoted by $\mathbb{E}_{\text{other}(\_)}$. That is: if $\mathbb{E}_{(\_)}$ is $\mathbb{E}_{(evn)}$, then $\mathbb{E}_{\text{other}(\_)}$ is $\mathbb{E}_{(odd)}$, and vice versa. We extend this notation to subscripts for group elements in the natural way (see, e.g., usage in the following definition).*

In order to define a Curve Treewe adopt a variant of (standard) approaches to defining a tree as a recursive data structure: an internal node is a list of (a function of) its children. The function which maps children to parents that we adopt uses an intermediate "labeling" step. A label can be thought of as a group element succinctly describing the node.

Definition 3 (Curve Trees)**.** *A Curve Tree is parameterized by (I). a depth $\mathsf{D} \in \mathbb{N}$, (II). a branching factor $\ell \in \mathbb{N}$, (III). a 2-cycle of Elliptic curves $(\mathbb{E}_{(evn)}, \mathbb{E}_{(odd)}, \mathbb{F}_p, \mathbb{F}_q)$ (IV). 2ℓ points $\vec{G}^{\mathsf{x}}_{(evn)}, \vec{G}^{\mathsf{y}}_{(evn)} \in \mathbb{E}^{\ell}_{(evn)}$ (V). 2ℓ points $\vec{G}^{\mathsf{x}}_{(odd)}, \vec{G}^{\mathsf{y}}_{(odd)} \in \mathbb{E}^{\ell}_{(odd)}$.*

The tree is defined recursively over D a follows:

[11]Assuming the identity ("point-at-infinity") is represented in $\mathbb{F}_q \times \mathbb{F}_q$

Leaves: $(0, \ell, \mathbb{E}_{(\_)}, \mathbb{E}_{\mathsf{other}(\_)})-\mathsf{CurveTree}$*:*

A leaf node is completely described by a curve point $C \in \mathbb{E}_{(\_)}$. The label of a leaf is C .

Parents: $(\mathsf{D}, \ell, \mathbb{E}_{(\_)}, \mathbb{E}_{\mathsf{other}(\_)})-\mathsf{CurveTree}$*:*

An internal node C is a list of ℓ $(\mathsf{D}-1, \ell, \mathbb{E}_{\mathsf{other}(\_)}, \mathbb{E}_{(\_)})$-Curve Trees. Let $C_1 = (\mathbb{X}_1, \mathbb{Y}_1) \in \mathbb{E}_{\mathsf{other}(\_)}, \ldots, C_\ell = (\mathbb{X}_\ell, \mathbb{Y}_\ell) \in \mathbb{E}_{\mathsf{other}(\_)}$ be their respective labels. The label $C \in \mathbb{E}_{(\_)}$ for the internal node is then defined as:

$$C = \langle \vec{\mathbb{X}}, \vec{G}^{\mathsf{x}}_{(\_)} \rangle + \langle \vec{\mathbb{Y}}, \vec{G}^{\mathsf{y}}_{(\_)} \rangle \quad (1)$$

(Note that the curves are switched between levels)

Trees for Sets. When we say that a curve tree is built for a set $S \subseteq \mathbb{E}$ (of size ℓ^{D}) we mean the natural layer-by-layer algorithm inductively constructing a tree with S as the leafs: partitioning S into subsets of size ℓ in some fixed way, then computing a Curve Tree for each set in the partition and forming a parent for the resulting ℓ children.

4.2 Binding and Hiding within Curve Trees

The previous notion (Definition 3) uses 2ℓ points per curve ($\vec{G}^{\mathsf{x}}_{(\text{evn})}, \vec{G}^{\mathsf{y}}_{(\text{evn})} \in \mathbb{E}^{\ell}_{(\text{evn})}$ and $\vec{G}^{\mathsf{x}}_{(\text{odd})}, \vec{G}^{\mathsf{y}}_{(\text{odd})} \in \mathbb{E}^{\ell}_{(\text{odd})}$) in order to label parent nodes by compressing their children. This already achieves a form of binding. By sampling one additional point per curve– $H_{(\text{odd})} \in \mathbb{E}_{(\text{odd})}, H_{(\text{evn})} \in \mathbb{E}_{(\text{evn})}$– we can blind / rerandomize a Curve Tree in the natural way. The root of a tree (and of each subtree) thus becomes a Pedersen commitment that is both binding and hiding. We formalize these observations below:

Lemma 1. *Assuming DLOG (Assumption 1) on $\mathbb{E}_{(evn)}$ and $\mathbb{E}_{(odd)}$, the root $C \in \mathbb{E}_{(\_)}$ of a $(\mathsf{D}, \ell, \mathbb{E}_{(\_)}, \mathbb{E}_{\mathsf{other}(\_)})$–CurveTree is a (non-hiding) Pedersen commitment whose opening is the ℓ roots of its children (in $\mathbb{E}_{\mathsf{other}(\_)}$). Additionally, for the same C and a random scalar r, the group element $\hat{C} := C + [r] \cdot H_{(\_)}$ is a hiding Pedersen commitment to C's children.*

Proof. The first part is a direct implication of the definition above. Also, observe then any internal node is already a root to a subtree. Let r' be a scalar (in the appropriate field) and let $\hat{C} = C + [r'] \cdot H_{(\_)}$. From standard properties of Pedersen commitments, we can observe $\hat{C}$ is still bound to the children of C. Hiding follows immediately (see Section 2.1.2). □

4.3 Traversing $(\ell, \mathbb{E}_{\textbf{(evn)}}, \mathbb{E}_{\textbf{(odd)}})$-Curve Trees

We now extend upon the observation in Section 4.2 that a node in a tree can be rerandomized. A natural strategy which stems from this observations is that, to prove membership of a Curve Tree in zero-knowledge, we can descend the tree one layer at a time starting from the root and following this approach: open a (hiding) commitment to a $(\mathsf{D}, \ell, \mathbb{E}_{(\_)}, \mathbb{E}_{\mathsf{other}(\_)})$-Curve Tree, pick one of it children (in zero-knowledge), then rerandomize the child and "output" the resulting hiding commitment to the $(\mathsf{D}-1, \ell, \mathbb{E}_{\mathsf{other}(\_)}, \mathbb{E}_{(\_)})$-Curve Tree; apply recursion. In this section we formalize a more efficient version of this informal sketch.

4.3.1 Descending a Single Level of the Tree

Our central component is a simple construction for a select-and-rerandomize-like relation for a *single* level in a curve tree. We later apply this at many levels at once in order to obtain a full select-and-rerandomize (Fig. 1). Consider a curve tree whose internal nodes at layer $d-1$ are in $\mathbb{E}_{(\_)}$. The inputs to relation $\mathcal{R}^{(\mathsf{single\text{-}level},(\text{evn}))}$ (resp. $\mathcal{R}^{(\mathsf{single\text{-}level},(\text{odd}))}$) are:

- public inputs: a rerandomized commitment $\hat{C} \in \mathbb{E}_{(\text{odd})}$ (resp. $\mathbb{E}_{(\text{evn})}$); its alleged parent $C \in \mathbb{E}_{(\text{evn})}$ (resp. $\mathbb{E}_{(\text{odd})}$);
- witnesses: index i whose semantics is "$\hat{C}$ is the (rerandomized) i-th child of C"; Pedersen opening scalars $r, \delta, \vec{\mathbb{X}}, \vec{\mathbb{Y}}$.

At each layer, this relation opens the parent commitment C to $\vec{\mathbb{X}}, \vec{\mathbb{Y}}$ using a commit-and-prove over $\mathbb{E}_{(\_)}$, plus it shows rerandomization of one of the children. At each level, even or odd, it is defined as follows.

Definition 4 (Relation for Select-and-Rerandomize)**.**

$$\mathcal{R}^{(\mathsf{single\text{-}level},\mathbb{E}_{(\_)})} := \left\{ \begin{array}{c} \textit{// open parent} \\ \begin{pmatrix} i, r, \delta, \\ \vec{\mathbb{X}}, \vec{\mathbb{Y}} \end{pmatrix} : \begin{array}{c} C = \langle \left[\vec{\mathbb{X}}\right], \vec{G}^{\mathsf{x}}_{(\_)} \rangle \\ + \langle \left[\vec{\mathbb{Y}}\right], \vec{G}^{\mathsf{y}}_{(\_)} \rangle \\ + [r] \cdot H_{(\_)} \in \mathbb{E}_{(\_)} \end{array} \\ \textit{// randomize i'th child} \\ \hat{C} = (\mathbb{X}_i, \mathbb{Y}_i) + [\delta] \cdot H_{\mathsf{other}(\_)} \in \mathbb{E}_{\mathsf{other}(\_)} \end{array} \right\}$$

We can implement this efficiently because the "parent opening" constraints can be directly enforced using a commit-and-prove for Pedersen commitments (Section 2.2.3). The additional "child opening" requires a single, cheap fixed-based exponentiation explicitly expressed as constraints. We describe an optimized arithmetic circuit for the relation above in the full version.

The following property will be useful for correctness later. It states that the relation above expresses the parent-child relation in a curve tree and that this holds even if we rerandomize children or internal nodes.

Lemma 2. *Consider a set S and a single-level curve tree (one root immediately followed by leaves) built on it. Let $C_{leaf} = (\mathbb{X}_i, \mathbb{Y}_i)$ be one of the leaves. Then the above relation $\mathcal{R}^{(\mathsf{single\text{-}level},(\_))}$—for the only existing level $d = 1$—is satisfied for any rerandomization factor δ such that $\hat{C} = C_{leaf} + [\delta] \cdot H_{\mathsf{other}(\_)}$. This property still holds if the root of the tree is rerandomized by some scalar r.*

Proof. This is a straightforward implication of how curve trees are defined. More in detail: In a single-level curve tree, C will be the root and thus constructed with $r = 0$. The first equation will be trivially satisfied by $\vec{\mathbb{X}}, \vec{\mathbb{Y}}$ such that $((\mathbb{X}_j, \mathbb{Y}_j))_j$ are the leaves (i.e., the children of the root C). The second equation will be satisfied by our assumption on $\hat{C}$. We finally observe that we can pick an honestly generated root, rerandomize it by adding $[r] \cdot H_{(\_)}$ for a scalar r and use the latter to let the first equation check. This proves the last part of the lemma statement. □

4.3.2 Descending all D Layers of The Tree

So far we discussed proving membership zooming in on a single level of a curve tree. We now want an approach that works for multiple levels. One straightforward method works by providing a separate proof for each level (this would be a proof for the relation in Definition 4). We will do something better instead. We leverage two facts: *i)* that there are two algebraic groups we are working with (depending on the layer parity); *ii)* that we can produce a single proof *at once* and *for multiple layers* "working in the same group". This way we are able to reduce our relation to two proofs only, one for the parents at odd layers and one for parents at even layers.

These two proofs will show respectively two "multi-leveled" relations, one for odd layers and on for even layers. They are defined below.

$$\mathcal{R}^{(\text{evn-levels})} := \left\{ \bigwedge_{j \in \{0,2,\ldots,\mathsf{D}-2\}} \mathcal{R}^{(\mathsf{single\text{-}level},(\text{evn}))} \right\}$$

$$\mathcal{R}^{(\text{odd-levels})} := \left\{ \bigwedge_{j \in \{1,3,\ldots,\mathsf{D}-1\}} \mathcal{R}^{(\mathsf{single\text{-}level},(\text{odd}))} \right\}$$

The witnesses and public statements for these relations are respectively the concatenation of the witnesses and public statements in Definition 4. See also Supplementary Material and in particular Fig. 4. The full construction is in Fig. 1.

5 Correctness and Security

Theorem 1. *The construction in Fig. 1 is a transparent select-and-rerandomize (Section 3.1). Its security relies on DLOG (Assumption 1) in $\mathbb{E}_{(evn)}$ and $\mathbb{E}_{(odd)}$ and the security of Bulletproofs as a NIZKAoK. It has $O(\sqrt[\mathsf{D}]{n})$ prover/verifier complexity*[12] *and its proof consists of $\mathsf{D}-1$ group elements and two Bulletproofs (each of size $O(\log \frac{n}{\mathsf{D}})$).*

Proof. We first observe that, by the DLOG assumption on both curves $\mathbb{E}_{(\text{odd})}$ and $\mathbb{E}_{(\text{evn})}$, we can use the fact that, by the standard Fiat-Shamir transformation [30], zk-BP$[\mathbb{E}_{(\text{odd})}]$ (resp.

[12] In practice $\mathsf{D} \approx 4$.

$\mathsf{SelRerand.Setup}(1^\lambda) \to \mathsf{pp}$

Sample $\vec{G}^{(\text{evn})} \in \mathbb{E}_{(\text{evn})}{}^{N_{\text{urs}}}, H^{(\text{evn})} \in \mathbb{E}_{(\text{evn})}$

Sample $\vec{G}^{(\text{odd})} \in \mathbb{E}_{(\text{odd})}{}^{N_{\text{urs}}}, H^{(\text{odd})} \in \mathbb{E}_{(\text{odd})}$

Return all sampled elements as pp

$\mathsf{SelRerand.Comm}(\mathsf{pp}, v_{\text{leaf}} \in \mathbb{F}_{||\mathbb{E}_{(\text{evn})}|}, o \in \mathbb{F}_{||\mathbb{E}_{(\text{evn})}|}) \to C$

$C \leftarrow G_1^{(\text{evn})} \cdot [v_{\text{leaf}}] + H^{(\text{evn})} \cdot [o]$

return $C \in \mathbb{E}_{(\text{evn})}$

$\mathsf{SelRerand.Rerand}(\mathsf{pp}, C \in \mathbb{E}_{(\text{evn})}, r \in \mathbb{F}_{||\mathbb{E}_{(\text{evn})}|}) \to \hat{C}$

$\hat{C} \leftarrow C + H^{(\text{evn})} \cdot [r]$

return $\hat{C} \in \mathbb{E}_{(\text{evn})}$

$\mathsf{SelRerand.Accum}(\mathsf{pp}, S' = \{C_1, \ldots, C_n\}) \to \mathsf{rt}$

Return rt, root of a tree computed on S' as by Definition 3

$\mathsf{SelRerand}.\mathcal{P}(\mathsf{pp}, S, C_{\text{leaf}}, r^{(\mathsf{D})})$

Reconstruct tree from S; let rt be its root

Let $C^{(0)}, \ldots, C^{(\mathsf{D})}$ be the path elements to C_{leaf} in the tree

(with $C^{(0)}$ corresponding to rt, $C^{(\mathsf{D})} = C_{\text{leaf}}$)

Let $\hat{C}^{(0)} := \mathsf{rt}$ and $r^{(0)} := 0$

for $k = 1, \ldots, \mathsf{D}/2$ **do**

$j \leftarrow 2k - 1$ // $j = 1, 3, \ldots$

$j' \leftarrow 2(k-1)$ // $j' = 0, 2, \ldots$

Sample $r^{(j)} \leftarrow\$ \mathbb{F}_{|\mathbb{E}_{(\text{odd})}|}$

if $j' < \mathsf{D}$ **then** Sample $r^{(j')} \leftarrow\$ \mathbb{F}_{|\mathbb{E}_{(\text{evn})}|}$

$\hat{C}^{(j)} \leftarrow C^{(j)} + \left[r^{(j)}\right] \cdot H_{(\text{odd})}$

$\hat{C}^{(j')} \leftarrow C^{(j')} + \left[r^{(j')}\right] \cdot H_{(\text{evn})}$

endfor

$\pi_{(\text{evn})} \leftarrow \mathsf{zk\text{-}BP}[\mathbb{E}_{(\text{evn})}].\mathsf{Prove}\left(\mathsf{pp}, \mathcal{R}^{(\text{evn-levels})}, \mathsf{x}_{(\text{evn})}, \mathsf{w}_{(\text{evn})}\right)$

$\pi_{(\text{odd})} \leftarrow \mathsf{zk\text{-}BP}[\mathbb{E}_{(\text{odd})}].\mathsf{Prove}\left(\mathsf{pp}, \mathcal{R}^{(\text{odd-levels})}, \mathsf{x}_{(\text{odd})}, \mathsf{w}_{(\text{odd})}\right)$

Return $\pi^* := \left(\hat{C}^{(1)}, \ldots, \hat{C}^{(\mathsf{D}-1)}, \pi_{(\text{evn})}, \pi_{(\text{odd})}\right)$

$\mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, \mathsf{rt}, \hat{C}_{\text{leaf}}, \pi^*)$

Parse π^* as $\left(\hat{C}^{(1)}, \ldots, \hat{C}^{(\mathsf{D}-1)}, \pi_{(\text{evn})}, \pi_{(\text{odd})}\right)$

Let $\hat{C}^{(\mathsf{D})} := \hat{C}_{\text{leaf}}$

Let $\hat{C}^{(0)} := \mathsf{rt}$

$b_{(\text{evn})} \leftarrow \mathsf{zk\text{-}BP}[\mathbb{E}_{(\text{evn})}].\mathsf{VerProof}\left(\mathsf{pp}, \mathcal{R}^{(\text{evn-levels})}, \mathsf{x}_{(\text{evn})}, \pi_{(\text{evn})}\right)$

$b_{(\text{odd})} \leftarrow \mathsf{zk\text{-}BP}[\mathbb{E}_{(\text{odd})}].\mathsf{VerProof}\left(\mathsf{pp}, \mathcal{R}^{(\text{odd-levels})}, \mathsf{x}_{(\text{odd})}, \pi_{(\text{odd})}\right)$

Accept iff $b_{(\text{evn})} \wedge b_{(\text{odd})} = 1$

Figure 1: Construction of Curve Tree Select-and-Rerandomize for a set of size n, branching factor ℓ, depth D (which we assume to be even), on cycle $(\mathbb{E}_{(\text{evn})}, \mathbb{E}_{(\text{odd})})$.

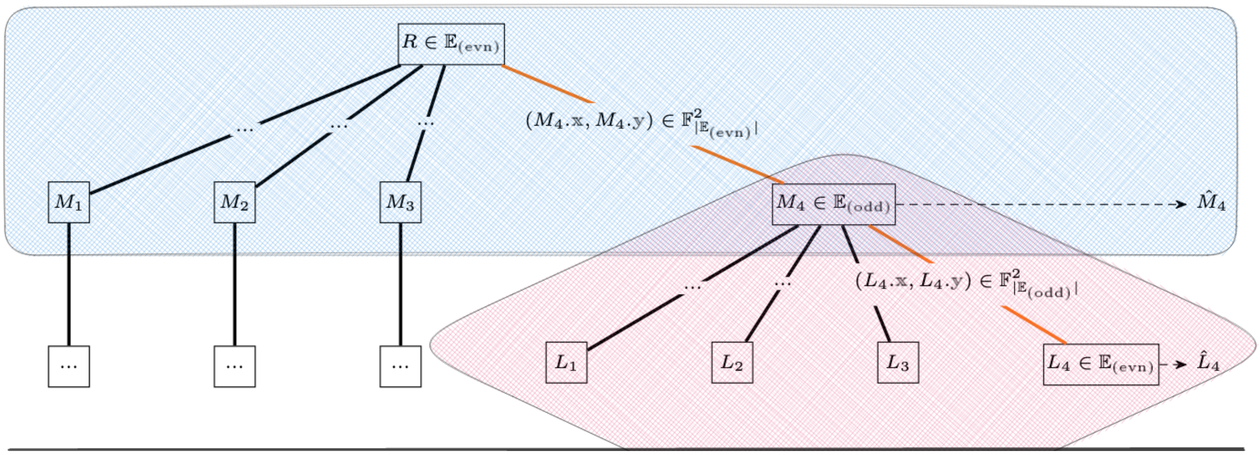

Figure 2: Illustrating proving select-and-rerandomize for a tree with $\mathsf{D} = 2$ and $\ell = 4$. Letters R, M, L hint respectively to commitments to root, a "middle" and "lower" layer respectively. The textured box and diamond areas denote the relation proven through Bulletproofs (on different curves, hence the different color). The dashed arrows going towards the right denote rerandomization.

zk-BP$[\mathbb{E}_{(\text{evn})}]$) is a correct, zero-knowledge and extractable NIZK. This will be useful in the remainder of the proof.

Correctness. Correctness of rerandomization is immediate: we are using standard Pedersen as a commitment, which is rerandomizable. That is if $C = G_1^{(\text{evn})} \cdot [v_{\text{leaf}}] + H^{(\text{evn})} \cdot [o]$ then its rerandomization by r is $\hat{C} = C + H^{(\text{evn})} \cdot [r] = G_1^{(\text{evn})} \cdot [v_{\text{leaf}}] + H^{(\text{evn})} \cdot [o + r]$.

To argue Select-and-Rerandomize correctness we will invoke these facts: that the output of Comm—i.e., leaves—are rerandomizable objects (observation from previous paragraph), the fact that internal nodes are rerandomizable (Lemma 1) and finally the correctness of Bulletproofs as NIZK. We can use the above to observe that, for an honestly generated commitment to a set, the honest prover will reconstruct a path, rerandomize each elements and then prove a conjunction of the level equation ($\mathcal{R}^{(\text{single-level},(\_))}$). We can invoke correctness of Bulletproofs if its prover is invoked with a statement satisfying those equations (see Lemma 2). Observing that a *conjunction* of satisfiable $\mathcal{R}^{(\text{single-level},(\_))}$-s is satisfiable (with corresponding witnesses) concludes the correctness proof.

Hiding and Zero-knowledge. Hiding is immediate from properties of Pedersen commitments. We describe a simulator $\mathcal{S}$ for the zero-knowledge which outputs π^* consisting of : $\hat{C}^{(1)}, \ldots, \hat{C}^{(\mathsf{D}-1)}$ fresh commitments to dummy values; $\pi_{(\text{evn})}$ and $\pi_{(\text{odd})}$ outputs of the respective simulators for the Bulletproofs NIZK on the respective relations. Notice that—by the definition of the game for select-and-rerandomize zero-knowledge and Lemma 2—the Bulletproofs simulators are invoked on *true* statements, crucially. To argue indistinguishability of the output of our simulator from that of the honest prover, we can just apply a hybrid argument where we invoke hiding of commitments and zero-knowledge of the underlying Bulletproofs.

Select-and-Rerandomize Binding. For sake of clarity and simplicity of notation, we first show our proof for the two-level case $\mathsf{D} = 2$ and then describe how it generalizes.

The verifier will then receive the following (see also definition of $\pi^\star$ in Fig. 1 for context as well as Fig. 2 for visual cues and an example):

- A rerandomized commitment to the leaf C_{leaf}
- A proof π^* consisting of: 1. a rerandomized commitment $\hat{C}_{\text{mid}}$ to the intermediate layer ($\hat{C}^{(1)}$ in Fig. 1); 2. an "upper-level" proof $\pi_\uparrow$, "linking" root and mid layer; 3. a "lower-level" proof $\pi_\downarrow$, "linking" mid and leaf layer.

As in the definition of binding (Section 3.1), we denote by $\hat{v}$ a malicious value not in the honestly generated set (but which the adversary "will claim" it's in the set).
We mark in *blue* elements that are extracted from the proofs.

Step 1. Apply knowledge-soundness to extract from the upper proof:

$$\begin{aligned}\hat{C}_{\text{root}} = &\langle \ldots \mathbb{X}(C_{\text{mid}}) \ldots, \vec{G}^{\mathtt{x}}_{(\_)} \rangle \\ &+ \langle \ldots \mathbb{Y}(C_{\text{mid}}) \ldots, \vec{G}^{\mathtt{y}}_{(\_)} \rangle + [r_{\text{root}}] \cdot H_{(\_)}\end{aligned} \tag{2}$$

$$\hat{C}_{\text{mid}} = C_{\text{mid}} + [\delta_{\text{mid}}] \cdot H_{\text{other}(\_)} \tag{3}$$

Observation a). We observe that above that the extracted C_{mid} will be the same as in the honest construction step of the tree (w.l.o.g. we can ignore the specific index on the path for it—this holds for all indices). If this were not the case we would be violating Lemma 1: C_{root} is an internal node of the tree and so it is a binding commitment to its children (see statement of Lemma 1). This observation will be useful later since we know the discrete logarithm of C_{mid} in $\vec{G}^{\mathsf{x}}_{\text{other}(\_)}, \vec{G}^{\mathsf{y}}_{\text{other}(\_)}, H_{\text{other}(\_)}$.

Step 2. Apply knowledge-soundness to extract from the lower proof:

$$\begin{aligned}\hat{C}_{\text{mid}} =& \langle \ldots \mathbb{X}(C_{\text{leaf}}) \ldots, \vec{G}^{\mathsf{x}}_{\text{other}(\_)} \rangle \\ &+ \langle \ldots \mathbb{Y}(C_{\text{leaf}}) \ldots, \vec{G}^{\mathsf{y}}_{\text{other}(\_)} \rangle + [r_{\text{mid}}] \cdot H_{\text{other}(\_)} \end{aligned} \tag{4}$$

$$\hat{C}_{\text{leaf}} = C_{\text{leaf}} + [\delta_{\text{leaf}}] \cdot H_{(\_)} \tag{5}$$

In addition to the group elements above, we will also extract, i^*, the index C_{leaf} refers to (see first witness in Definition 4).

Observation b). Because the adversary is successful in the binding experiment (through some claimed $\hat{v} \notin S = \{v_i\}_i$), we can conclude that i^* such that $C_{\text{leaf}} \neq \mathsf{Comm}(v_{i^*}, o_{i^*})$. (Otherwise we would have $\mathsf{Comm}(v_{i^*}, o_{i^*}) + [\delta_{\text{leaf}}] \cdot H_{(\_)} = \hat{C}_{\text{leaf}} = \mathsf{Comm}(\hat{v}, \hat{o})$ which would break DLOG) This is equivalent to saying that $\mathbb{X}(C_{\text{leaf}}) \neq \mathbb{X}_{i^*}$ or $\mathbb{Y}(C_{\text{leaf}}) \neq \mathbb{Y}_{i^*}$, where $\mathbb{X}_{i^*} := \mathbb{X}(\mathsf{Comm}(v_{i^*}, o_{i^*})), \mathbb{Y}_{i^*} := \mathbb{Y}(\mathsf{Comm}(v_{i^*}, o_{i^*}))$.

Step 3. Combine equations Eq. (3) and Eq. (4): Now, combining the equations, we can observe that:

$$\begin{aligned}&[r_{\text{mid}} - \delta_{\text{mid}}] \cdot H_{\text{other}(\_)} - C_{\text{mid}} + \\ &\langle \ldots \mathbb{X}(C_{\text{leaf}}) \ldots, \vec{G}^{\mathsf{x}}_{\text{other}(\_)} \rangle + \langle \ldots \mathbb{Y}(C_{\text{leaf}}) \ldots, \vec{G}^{\mathsf{y}}_{\text{other}(\_)} \rangle = 0\end{aligned}$$

This allows an adversary to break DLOG (Assumption 1) by using the following facts. As we observed (obs. (a)), C_{mid} is the same as in the honest tree construction, which implies its discrete logarithms can be derived knowing the original honest set. If $\mathbb{X}(C_{\text{leaf}}) \neq \mathbb{X}_{i^*}$, the adversary can then break DLOG for the generator $G^{\mathsf{x}}_{i^*,\text{other}(\_)}$. This becomes clear when rewriting the equation above like this:

$$\begin{aligned}G^{\mathsf{x}}_{i^*,\text{other}(\_)} = &\big(\underbrace{\mathbb{X}_{i^*} - \mathbb{X}(C_{\text{leaf}})}_{\neq 0}\big)^{-1} \cdot \big([r_{\text{mid}} - \delta_{\text{mid}}] \cdot H_{\text{other}(\_)} + \\ &\langle \vec{\mathbb{X}}^{(\text{leaf})}_{\neq i^*}, \vec{G}^{\mathsf{x}}_{\neq i^*,\text{other}(\_)} \rangle + \langle \ldots \mathbb{Y}(C_{\text{leaf}}) \ldots, \vec{G}^{\mathsf{y}}_{\text{other}(\_)} \rangle\big)\end{aligned}$$

where $\vec{\mathbb{X}}^{(\text{leaf})}_{\neq i^*} := \big(\mathbb{X}(\mathsf{Comm}(v_i, o_i))\big)_{i \neq i^*}$. If $\mathbb{Y}(C_{\text{leaf}}) \neq \mathbb{Y}_{i^*}$, we can modify the above accordingly to apply to $\mathbb{Y}$. This concludes the proof.

To generalize the proof to $\mathsf{D} \geq 2$. First, we recursively apply Step 1 and observation (a), i.e., we repeatedly apply Lemma 1 to argue that is the same as in the honestly constructed tree for each internal node C_{mid} on the path. Then, as we did above, we apply step 2 and step 3 for the last two layers, as well as observation b). (Notice that, in order to extract the equations, we will still use two proofs but now each of them will allow us to extract multiple levels. There are still only two proofs—even and odd—but now they refer to multiple disjoint levels of the tree instead of just two). □

6 Final Construction: Curve Trees with Compressed Points

In this section we describe some optimizations we employ in our final construction. Our initial observation is that a curve tree (as defined in Definition 3) uses both $\mathbb{X}$ and $\mathbb{Y}$ coordinates to represent a node (leaf or internal). This requires 2ℓ generators at each level. The factor 2 will become a cost at commitment, proving and verification time, as well in proof size. Here we discuss how to remove this factor.

The starting point of our idea are folklore approaches to point compression which rely on encoding a point through the $\mathbb{X}$ coordinate. We need to take extra care though. Where we need to take extra care is in: a) making sure, through appropriate checks. that a malicious prover cannot exploit this compression; b) making sure the latter checks are efficient constraints-wise when we prove/verify them in zero-knowledge. In order to do this we exploit the fact that the leaves in the tree are agreed on publicly (we remind that in our model as well in confidential transactions, the whole set of points is public; the item we prove membership on is hidden). This way, we can make sure at commitment time that each leaf is represented through pairs of points of a certain form. We call these points *permissible*. We modify our definition of curve trees to explicitly take compression and permissibility into account (Definition 5). To efficiently prove/verify this we rely on 2-universal hash functions (see rest of this section and Eq. (6)). Their algebraic nature allows us to not to employ bit decomposition. As a consequence, these techniques have nearly no impact on any additional complexity of the relation proved in zero-knowledge.

When we to plug in these additional tricks, our construction (Fig. 1) stays essentially the same: we can describe its changes in a modular fashion (see Section 6.3). The same holds for security and correctness proofs.

6.1 Point Compression and Permissible Points

In order to reduce the number of exponentiations during commitment and the size of the witness we rely on committing only to the $\mathbb{X}$-coordinate of children node. To guarantee that our construction remains binding we ensure 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 (which does not require bit decomposition). 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 $\{0,1\}$:

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0,1\} \quad \mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta) \tag{6}$$

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 $\mathbb{E}$:

$$\mathcal{P}_{\mathbb{E}} = \{(\mathbb{X},\mathbb{Y}) \mid (\mathbb{X},\mathbb{Y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathbb{Y}) = 1 \wedge \mathcal{U}_{\alpha,\beta}(-\mathbb{Y}) = 0\}$$

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 $\notin \{2,3\}$.

We make sure at commitment time that nodes are converted to permissible points by adding appropriate randomness. This is formalized in the supplementary material in Fig. 3 in the procedure AsPermissible as well in the "compressed points" definition of curve trees (Definition 5), which invokes it. In expectation, procedure AsPermissible requires 4 curve additions and 8 square roots.

6.2 Curve Trees with Compressed Points

The following definition simply adapts a curve tree to the setting where leaves are required to be permissible and internal children nodes are compressed through AsPermissible before committing to them in their parent.

Definition 5 (Curve Trees with compressed points)**.** *A Curve Tree with compressed points follows the same basic inductive definition as Definition 3, but with the following differences: first, the tree is also parametrized by two permissible sets* $\mathcal{P}_{(evn)} \subseteq \mathbb{E}_{(evn)}$ *and* $\mathcal{P}_{(odd)} \subseteq \mathbb{E}_{(odd)}$*. Second, Eq. (1) (root label* C *of an internal node) becomes*

$$C = \langle \vec{\mathbb{X}}', \vec{G}^{\times}_{(\_)} \rangle \in \mathbb{E}_{(\_)} \tag{7}$$

where for each $i \in [\ell]$*,* $\mathbb{X}'_i$ *is such that* $(\mathbb{X}'_i, \dots) \leftarrow \mathsf{AsPermissible}_{\mathbb{E}_{(\_)}}(\mathbb{X}_i, \mathbb{Y}_i)$*, and* $(\mathbb{X}_i, \mathbb{Y}_i)$ *are as in Eq. (1). Third, leaves are required to be permissible.*

Since the definition above makes a tree only out of permissible points[13] this gives a "decompression" that is unique. This in turn reduces the the complexity single-level relations. We thus define a new optimized relation $\mathcal{R}^{(\mathsf{single\text{-}level}^\star,(\_))}$:

Definition 6 (Optimized single-level relation)**.**

$$\mathcal{R}^{(\mathsf{single\text{-}level}^\star,(\_))} := \left\{ \begin{pmatrix} i,r,\delta, \\ \vec{\mathbb{X}},\mathbb{Y} \end{pmatrix} : \begin{array}{c} C = \langle \left[\vec{\mathbb{X}}\right], \vec{G}^{\times}_{(\_)} \rangle \\ + [r] \cdot H_{(\_)} \\ \wedge (\mathbb{X}_i, \mathbb{Y}) \in \mathcal{P}_{\mathsf{other}(\_)} \\ \wedge \hat{C} = (\mathbb{X}_i, \mathbb{Y}) + [\delta] \cdot H_{\mathsf{other}(\_)} \end{array} \right\}$$

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

Note that the constraint $(\mathbb{X}_i, \mathbb{Y}) \in \mathcal{P}_{\mathsf{other}(\_)}$ only requires a check that $(\mathbb{X}_i, \mathbb{Y}) \in \mathbb{E}_{\mathsf{other}(\_)}$ in addition to $\mathcal{U}_{\alpha,\beta}(\mathbb{Y}) = 1$. See full version for additional details.

6.3 Adapting Construction in Fig. 1 to Compressed Points

Our final construction essentially remains the same as in Fig. 1 with two exceptions.

- In order to accumulate a set (SelRerand.Accum) we generate a root through the procedure derived from Definition 5 instead of the one for Definition 3.
- The proofs $\pi_{(\mathrm{evn})}$ and $\pi_{(\mathrm{odd})}$ are for slightly different relations: they prove/verify relations for $\mathcal{R}^{(\text{evn-levels})}$ and $\mathcal{R}^{(\text{odd-levels})}$ but defined in terms of $\mathcal{R}^{(\mathsf{single\text{-}level}^\star,(\_))}$ from Definition 6 (instead of Definition 4).

This variant construction is also correct and secure:

Theorem 2. *The variant of the construction of Fig. 1 described in this section is a transparent select-and-rerandomize primitive (under the same assumptions as in Theorem 1).*

The proof for theorem above follows the same blueprint as the one in Theorem 1. Zero-knowledge/hiding is trivially untouched by the changes in the construction. Binding is clearly still guaranteed since the relation we prove ($\mathcal{R}^{(\mathsf{single\text{-}level}^\star,(\_))}$) is now *stricter* than the one in $\mathcal{R}^{(\mathsf{single\text{-}level},(\_))}$. Observing correctness only requires observing that a variant of Lemma 2 also holds (easily) for definition Definition 5.

7 Anonymous Currency from Curve Trees

As an application of Curve Trees we propose a simple anonymous payment scheme: $\mathbb{V}$Cash. Full model/description/proof of $\mathbb{V}$Cash can be found in the full version of this paper, here we provide an overview/intuition for the construction. We rely on terminology introduced in the Zerocash paper [7]: $\mathbb{V}$Cash is very similar in design, but with a slightly simpler construction. In $\mathbb{V}$Cash users are identified by 1. a public-key $\mathsf{pk}_{\mathsf{sig}}$ for a signature scheme with rerandomizable keys, such as Schnorr signatures. 2. a public-key $\mathsf{pk}_{\mathsf{enc}}$ for an IK-CPA encryption scheme (where ciphertexts for different keys are indistinguishable; the known El Gamal scheme is an example). The balance of a party in $\mathbb{V}$Cash is defined by the total value of previously unspent "notes"[14] belong to the party. A $\mathbb{V}$Cash "transaction bundle" consists of a number of "pour" and "spend" operations, which create new notes and spend previously created notes respectively.

POUR. Let v be the value to spend. The sender rerandomizes the receiver's public key $\mathsf{pk}_{\mathsf{sig}}$ with randomness r_{sig} to obtain $\mathsf{pk}'_{\mathsf{sig}}$, computes the hash $h = \mathcal{H}(\mathsf{pk}'_{\mathsf{sig}})$. Then forms a

[14] Also "unspent transaction outputs" (UTXO) in Bitcoin.

Pedersen commitment to (v,h) using randomness $r_{\mathsf{com}} \leftarrow\$ \mathbb{F}_p$ and obtains the commitment C. The sender proves (using a Bulletproof) that $\mathsf{v} \in [0,2^{32})$, obtains π_{range}. The sender then encrypts $(r_{\mathsf{sig}}, r_{\mathsf{com}}, \mathsf{v})$ using $\mathsf{pk}_{\mathsf{enc}}$ to obtain a ciphertext ct. The tuple $(C, \pi_{\mathsf{range}}, \mathsf{ct})$ is posted to the network, which verifies π_{range} and adds C to a global Curve Tree.

SPEND. Obtain ciphertext ct from the network and decrypt it using $\mathsf{sk}_{\mathsf{enc}}$ to obtain $(r_{\mathsf{sig}}, r_{\mathsf{com}}, \mathsf{v})$. The receiver proves that $\hat{C} = \mathsf{Commit}((\mathsf{v},h); \hat{r}_{\mathsf{com}})$ is a rerandomization of a commitment C in the global Curve Tree using the techniques covered in this paper, obtaining a membership proof $\pi_{\mathsf{CurveTree}}$. The receiver computes $h' = \mathcal{H}(\mathsf{pk}'_{\mathsf{sig}})$ and proves $h = h'$ using a Bulletproof. Finally it signs the entire transaction using $\mathsf{sk}'_{\mathsf{sig}}$ obtained by rerandomzing $\mathsf{sk}_{\mathsf{sig}}$ using r_{sig}, obtaining a signature σ. The result $(\pi_{\mathsf{CurveTree}}, \pi_{\mathsf{open}}, \mathsf{pk}'_{\mathsf{sig}}, \hat{C}, \sigma)$ is posted to the network, which recomputes h', verifies both proofs and the signature. If h' has been seen by the network previously the spend operation is rejected.

BALANCE CHECK. To avoid money being created, the spender proves that the values of spend/pour operations cancel out: $\left\{(\Delta, r) : \sum_i C_i - \sum_j \hat{C}_j = \mathsf{Commit}(([0],[\Delta]); r)\right\}$ where Δ is the sum of the hashes of the rerandomized public keys. For efficiency all the Bulletproofs are combined, as a result a $\mathbb{V}$Cash transaction always consists of just two Bulletproofs of moderate size. We benchmark this scheme in Section 8.

8 Implementation and Evaluation

We implement select-and-rerandomize and $\mathbb{V}$Cash in Rust on top of the `dalek` Bulletproofs library[15]. The Bulletproof implementation has been extended with support for vector commitmentsand elliptic curves implemented using the arkworks[16] 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[17] instance with 8 vCPUs, which corresponds to 4 physical cores on an Intel Xeon 8375C processor with 2.9 GHz clock speed[18]. When possible (and unless otherwise explicitly specified) we have benchmarked alternative schemes on the same hardware. We use Curve Trees of even depth D in our evaluation and instantiate the two underlying elliptic curves through both those in the Pasta cycle [35] and the `secp256k1` / `secq256k1` cycle. We use Schnorr signatures for $\mathbb{V}$Cash.

[15] https://github.com/dalek-cryptography/bulletproofs

[16] https://github.com/arkworks-rs

[17] https://aws.amazon.com/ec2/instance-types/c6i/

[18] While we tabulate only results for this architecture, we also performed benchmarks on a common laptop.

Curves	(D,ℓ)	$\lvert S\rvert$	# Constraints	Proof (kb)	Prove (s)	Verify (ms)	Verify batch (ms)
Pasta	$(2,1024)$	2^{20}	3870	2.6	0.88	23.17	1.44
	$(4,256)$	2^{32}	4668	2.9	1.71	39.63	2.35
	$(4,1024)$	2^{40}	7740	2.9	1.74	40.41	2.73
Secp/Secq	$(2,1024)$	2^{20}	3870	2.6	0.97	26.81	1.61
	$(4,256)$	2^{32}	4668	2.9	1.89	47.39	2.64
	$(4,1024)$	2^{40}	7740	2.9	1.92	48.40	3.02

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

Scheme	# Constraints	Prove (s)	Verify (ms)	Verify batch (ms)
Curve Trees (Pasta)	3565	1.5	31	1.8
Curve Trees (Secp/Secq)	3565	1.7	37	2
Poseidon 4:1	4515	8.8	651	-
Poseidon 8:1	4180	8.5	825	-

Table 2: Benchmarks of accumulators over sets of size 2^{30} based respectively on curves trees and on Merkle trees with Poseidon (Appendix G). Batch verification time is for the amortized time for a batch of size 100.

8.1 Zero-Knowledge for Set-Membership

The results in Table 1 summarize the efficiency of our select and rerandomize scheme (Section 3) using the final construction in Section 6.3 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 the set size ($|S| = \ell^{\mathsf{D}}$) in order to optimize the running time by obtaining a number of constraints which "does not overflow" powers of two if possible. This is illustrated by the benchmarks for sets of size 2^{32} and 2^{40}: despite the gap between the set sizes they show similar performance.

If only proofs of membership of field elements are needed, these can be achieved by using the select and rerandomize scheme on vector commitments of ℓ' elements obtaining a scheme which uses only $\mathsf{D}(912+\ell-1)+(\ell'-1)$ constraints to show membership of a set with $\ell^{\mathsf{D}} \cdot \ell'$ elements. Using the parameters $\mathsf{D}=3$, $\ell=256$, and $\ell'=64$ we get a direct comparison ($|S| = 2^{30}$) with [33] in which they use bulletproofs to show membership in Poseidon based Merkle trees with 2^{30} leaves and $\ell = 2$, 4, or 8. The best performing instances in [33] are using branching factors of 4 and 8 on the ed25519 curve: one results in slightly fewer constraints and faster proving time, while the other verifies faster. The results in Table 2 show that the accumulator based on Curve Trees is > 5 times faster at proving and > 20 times faster at verifying compared to the fastest instances of Poseidon-based Merkle trees.

8.2 $\mathbb{V}$Cash

Table 3 compares $\mathbb{V}$Cash with various anonymous payment systems. When used for batch verification, $\mathbb{V}$Cash outperforms other schemes, sometimes by orders of magnitude (for the same anonymity sets). Non-batched verification time is highly competitive when compared to transparent constructions, but $10\times$ slower than the non-transparent Zcash Sapling (which mainly consists of a few pairing operations). Orchard—the recent transparent version of Zcash based on Halo2 and Pasta (see also Appendix G—achieves a $5\times$ faster verification time than $\mathbb{V}$Cash. We believe that basing $\mathbb{V}$Cash on a Curve Tree using Halo2 would outperform Orchard. On the other hand this would come at the price, as it is the case for Orchard, of not supporting arbitrary 2-cycles of curves (see Section 1.1.6). The transaction size in Orchard is roughly twice as large as in $\mathbb{V}$Cash. The only other better transaction size among transparent constructions is that of Omniring (we estimate $\mathbb{V}$Cash 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, compared to which our circuit for "spend" transaction is 20x smaller. The cost of the set membership proofs dominate the combined transaction circuit. For instance the $\mathbb{V}$Cash combined circuit (over both fields) has 9464 constraints of which 9336 are used for the proof of membership and the Orchard action circuit has 2^{11} rows and 40 columns while the membership by itself uses 2^{11} rows and 35 columns.

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 [44] (45KB for full security [21]); Quisquis [29] (13KB for $|S| = 2^4$); Monero [3] (whose transaction grows linearly with $|S|$ and is already at 1.3KB for $|S| < 2^4$).

	$\lvert S\rvert$	Transparent setup	Tx size (kb)	Proving time (S)
Omniring [39]	2^{10}	✓	1	≈ 1.5‡
Lelantus [37]	2^{10}	✓	2.7	0.27†
	2^{14}	✓	3.9	2.35†
	2^{16}	✓	5.6	4.8†
$\mathbb{V}$Cash (Pasta)	2^{20}	✓	3.4	1.76
	2^{32}	✓	4	3.43
	2^{40}	✓	4	3.48
$\mathbb{V}$Cash (Secp/Secq)	2^{20}	✓	3.4	1.95
	2^{32}	✓	4	3.80
	2^{40}	✓	4	3.86
Zcash Orchard	2^{32}	✓	7.6	1.77
$\mathbb{V}$eksel [21]	Any	✗⋆	5.3	0.44
Zcash Sapling	2^{32}	✗	2.8	2.38

	$\lvert S\rvert$	Verification time (ms)	Batch verification time (ms)
Omniring [39]	2^{10}	≈ 130‡	-
Lelantus [37]	2^{10}	-	6.8†
	2^{14}	-	10.2†
	2^{16}	-	52†
$\mathbb{V}$Cash (Pasta)	2^{20}	41.40	2.87
	2^{32}	78.40	4.98
	2^{40}	80.52	5.77
$\mathbb{V}$Cash (Secp/Secq)	2^{20}	48.27	3.15
	2^{32}	90.40	5.60
	2^{40}	91.97	6.32
Zcash Orchard	2^{32}	15	-
$\mathbb{V}$eksel [21]	Any	61.88	-
Zcash Sapling	2^{32}	7	-

Table 3: 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). [37]

‡ Omniring was benchmarked on an Intel i7-7600U (2 cores, 2.8GHz). [39].

Acknowledgements

Mathias Hall-Andersen and Simon Holmgaard Kamp are funded by the Concordium Foundation.

References

[1] Report on the security of stark-friendly hash functions (version 2.0), 2020.

[2] 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.

[3] 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.

[4] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.

[5] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 301–315, 2017.

[6] 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.

[7] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[8] Eli Ben-Sasson, Lior Goldberg, and David Levit. STARK friendly hash – survey and recommendation. Cryptology ePrint Archive, Report 2020/948, 2020. https://eprint.iacr.org/2020/948.

[9] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002.

[17] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. 1997.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] Nikolaos Triandopoulos Charalampos Papamanthou, Roberto Tamassia. U.S Patent. US9098725B2, Cryptographic accumulators for authenticated hash tables, 2014.

[24] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 422–439. Springer, Heidelberg, May 2005.

[25] 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.

[26] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. https://eprint.iacr.org/2008/538.

[27] 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.

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

[29] 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.

[30] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[31] 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.

[32] Esha Ghosh, Olga Ohrimenko, Dimitrios Papadopoulos, Roberto Tamassia, and Nikos Triandopoulos. Zero-knowledge accumulators and set algebra. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 67–100. Springer, Heidelberg, December 2016.

[33] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

[34] Vitalik Buterin Guillaume Ballet, Dankrad Feist. Verkle tree eip, 2021.

[35] Daira Hopwood, 2020. https://github.com/zcash/pasta.

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

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

[38] 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.

[39] 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.

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

[41] 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.

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

[43] Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *44th FOCS*, pages 80–91. IEEE Computer Society Press, October 2003.

[44] 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.

[45] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, Heidelberg, August 1986.

[46] Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer, Heidelberg, February 2005.

[47] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 437–448. ACM Press, October 2008.

[48] 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.

[49] 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.

[50] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3121–3134. ACM Press, November 2022.

Supplementary Material

A Accumulators

For reference and to make easier a comparison to our primitive in Definition 2, we provide the more standard definition of accumulators [11].

Definition 7 (Accumulator scheme)**.** *An accumulator scheme* Acc *over universe* $\mathcal{U}_\lambda(\mathsf{Acc})$ *(for a security parameter* λ*) 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.

Correctness: For any set $S = \{v_i\}_i$, $j^* \in [|S|]$ the following holds

$$\Pr\left[\begin{array}{r} \mathsf{pp}_{\mathrm{acc}} \leftarrow \mathsf{Acc.Setup}(1^\lambda) \\ A = \mathsf{Acc.Accum}(\mathsf{pp}, S) \\ \pi \leftarrow \mathsf{Acc.PrvMem}(\mathsf{pp}_{\mathrm{acc}}, S, v_{j^*}) \end{array} : \right] = 1$$

Security: For any PPT adversary $\mathcal{A}$ the following holds:

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Acc.Setup}(1^\lambda) \\ (S, v', \pi) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathrm{acc}}) \\ A = \mathsf{Acc.Accum}(\mathsf{pp}, S) \end{array} : \begin{array}{l} \mathsf{Acc.VfyMem}(\mathsf{pp}, A, v', \pi) \\ \wedge\ v' \notin S \end{array}\right] \leq \mathsf{negl}(\lambda)$$

B How to Obtain Permissible Points

Auxiliary pseudocode for building a Curve Tree with compressed points is in Fig. 3.

$\mathsf{AsPermissible}_{\mathbb{E}_{(\_)}}(C) \to (\mathcal{P}_{(\_)}, \mathbb{F})$

```
1: r_P ← 0 ∈ F // Scalar field of E_(_)
2: while C ∉ P_(_) :
3:     C ← C + H_(_)
4:     r_P ← r_P + 1
5: return (C, r_P)
```

Figure 3: Explicit algorithm for permissible compression. Any Pedersen commitment C can be "made permissible" by simply iteratively adding an additional generator $H_{(\_)}$ ("incrementing the randomness") until the point is permissible. The algorithm returns the $\mathbb{X}$ coordinate and the permissibility scalar.

C Public Inputs/Witnesses for relations $\mathcal{R}^{\text{(evn-levels)}}/\mathcal{R}^{\text{(odd-levels)}}$

$$\mathsf{x}_{(\mathrm{evn})} := \left(\hat{C}^{(j-1)}, \hat{C}^{(j)}\right)_{1 \leq j < \mathsf{D},\, j \text{ odd}}$$

$$\mathsf{x}_{(\mathrm{odd})} := \left(\hat{C}^{(j)}, \hat{C}^{(j+1)}\right)_{1 \leq j < \mathsf{D},\, j \text{ odd}}$$

$$\mathsf{w}_{(\mathrm{evn})} := \Big(i_j, r^{(j-1)}, r^{(j)}, \mathbb{X}(\mathsf{pathChildren}_j), \mathbb{Y}(\mathsf{pathChildren}_j)\Big)_{1 \leq j < \mathsf{D},\, j \text{ odd}}$$

$$\mathsf{w}_{(\mathrm{odd})} := \Big(i_{j+1}, r^{(j)}, r^{(j+1)}, \mathbb{X}(\mathsf{pathChildren}_{j+1}), \mathbb{Y}(\mathsf{pathChildren}_{j+1})\Big)_{1 \leq j < \mathsf{D},\, j \text{ odd}}$$

i_j : index of node along the path at layer j (see Definition 4)

$\mathsf{pathChildren}_j$: sibling nodes along the path at layer j (see Definition 4)

Figure 4: Public input and witness for relations $\mathcal{R}^{\text{(evn-levels)}}/\mathcal{R}^{\text{(odd-levels)}}$ (used in Fig. 1.

D Curve Tree as somewhat structure-preserving

The recent results of [22] 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 [42]). 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 [22], which considers one *single* abstract group and black-box use of its addition. Curve trees, on the other hand, exploit several groups (assumed to constitute a 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 [22], nor is our intention to undermine it. Instead, we argue Curve Trees provides further nuances to the observations in [22]: 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 3).

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 More Related Work on Transparent Zero-Knowledge for Set Membership

Some works with transparent setup do not achieve succinctness (that is, practically short proofs and a $o(|S|)$ verification time). For example, Monero [3]—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 [39].

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 [10], the seminal KZG [38] and the polynomial commitments in [15]. They, however, all require knowledge-based assumptions and a trusted setup. Similar observations hold for the recent work in Caulk [49].

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 [48], where for large set sizes commitments can be $\gg 10KB$, and Dory [40] 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)[19]. Other transparent polynomial commitments include those based on Reed-Solomon IOPs [6] or on Diophantine ARguments of Knowledge (DARK) [14]. As argued in [40] (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, [20, 21, 10, 18]. These works obtain concretely small proofs/verifier with an efficient proving time, but require an RSA modulus (non-transparent) for their efficient instantiations[20]. While the work in [18] 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.

F.1 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 [28] through nested proving and other techniques. It is still unclear how to compare these extensions to our work: the current writeup in [28] does not make all the assumptions behind its estimates concrete and the work does not have a complete implementation yet.

G Other Implementations Used in Experimental Comparison

The Poseidon implementation can be found at https://github.com/lovesh/ursa/tree/zmix/libzmix/bulletproofs_amcl

The Orchard protocol was benchmarked using the implementation at https://github.com/zcash/orchard.

[19] See [40] for numbers referred in this section.

[20] In all these works we can replace the RSA group with a transparent class-group [12] at a substantial efficiency cost. See, e.g., discussion in [27].