μCash: Transparent Anonymous Transactions

Liam Eagen
liameagen@protonmail.com

August 25, 2022

Abstract

Zero Knowledge Set Membership Proofs (zkSMPs) allow efficiently, i.e. sublinearly in the size of the set, proving membership of a value in a set in zero knowledge with respect to the value. They have been used to construct anonymous cryptocurrencies such as ZCash [1], which uses a zero knowledge Merkle [2] proof to show that the inputs of a transaction belong to the Transaction Output (TXO) set. Using a Merkle tree instantiated with a pair of Pedersen [3] hash functions between an amicable cycle of elliptic curves [4], similarly to Curve Trees [5], and the Weil Elliptic Curve Inner Product (ECIPs) proofs of [6], I design a set membership protocol with substantially smaller witness sizes than other Merkle zkSMPs. This protocol uses a pair of communicating Bulletproofs [7], one over each curve, whose total proof size I am able to reduce by proving portions of each verifier inside the other proof. Using these techniques, along with an adaptation of the Bulletproofs++ [8] confidential transaction protocol, I design μCash a transparent, i.e. without a trusted setup, anonymous transaction protocol for a decentralized cryptocurrency, whose security argument is reducible to the discrete log problem over a pair of elliptic curves. Over a 256 bit field, these transactions are $1349 + 64n + 32\lceil\log_2 c\rceil$ bytes for n inputs, m outputs, d depth, and c proof capacity, which is bounded by a linear function of nd, n, and m and is equal to 1 for up to $m < 1000$ or $n < 37$ when $d = 48$. Proving complexity is quasilinear and verifier complexity is linear in both nd and m, and in practice verification will be dominated by the cost of two Bulletproof verifications of length 1536 and 1744 for $c = 1$. μCash supports efficient batch verification, user defined assets and multi-asset confidential transactions, privacy preserving multi-party proving, adaptor signatures [9], absolute and relative time locks, and a multiphase transaction structure to support scriptless scripts for private atomic swaps and payment channels [10]. This protocol is likely compatible with the Halo [11] accumulation scheme, although I do not investigate this.

1 Introduction

Bitcoin [12] was the first decentralized cryptocurrency. It works by using a novel consensus algorithm, retroactively named Nakamoto consensus, to establish consensus about the entire Bitcoin transaction history. Each transaction produces outputs, which are an amount of Bitcoin and a public key (hash), and consumes outputs of earlier transactions. The total set of outputs is referred to as the Transaction Output (TXO) set and is partitioned into the Spent TXO (STXO) and the Unspent TXO (UTXO) sets. To verify the history of all Bitcoin transactions, one checks that each transaction's output amounts sum to its input amounts and that each element of the TXO set is spent exactly once. Equivalently, one verifies that Bitcoin transactions form directed acyclic graph, where each node is a transaction and each edge is a spent output connecting the originating transaction to the consuming transaction.

Publishing all Bitcoin transactions like this allowed for the creation of a decentralized cryptocurrency, but it also had the effect of substantially eroding financial privacy for Bitcoin transactions compared existing alternatives. Financial transaction in the traditional financial system are not private from the system itself or the state, but they are typically private from the general public. Despite the inherent pseudonymity of Bitcoin transactions, it is still possible to infer a great deal of information about the users of the system from the transaction graph itself [13]. When combined with additional sources of information, it is often possible to comprehensively surveil all Bitcoin transactions [14]. Such techniques can be used to target users for theft, to discriminate against users based on their spending habits, and to facilitate political persecution. More fundamentally, the ability to surveil Bitcoin undermines the fungibility of the currency. This was a known problem early in the history of Bitcoin [15] and increasingly functions as a backdoor mechanism for exactly the powers cryptocurrencies broadly and Bitcoin specifically sought to displace to reassert their control.

1.1 Related Work

There have been many different techniques proposed to improve the privacy of Bitcoin and other cryptocurrencies. These each have different tradeoffs between ease of use, compatibility with existing technologies, degree of privacy, and cryptographic assumptions. While a comprehensive survey [16] of such technologies is outside the scope of this paper, several are worth mentioning in more detail, including CoinJoin [17] style protocols, ring signature [18] based protocols like Monero [19], and SNARK [20] based protocols like ZCash. These roughly fall on a spectrum of increasing complexity in exchange for stronger theoretical privacy guarantees. Of course, actual real world privacy will depend on things like anonymity set size, i.e. the number of users of the currency, communication metadata, i.e. the timing and originating IP address of the transaction, and general operational security of the transacting parties that mostly fall outside the scope of the paper.

1.1.1 CoinJoin and Mixers

CoinJoin is a relatively simple measure for obscuring the transaction graph that works with Bitcoin and many other cryptocurrencies without any special modifications. Essentially, multiple users collaborate to construct a single large transaction that performs all the individual users' transactions at once. The structure of Bitcoin transactions ensures that the relationship between inputs and outputs within a single transaction is symmetric, i.e. all the inputs and all the outputs are fungible, so it is not possible to determine externally which inputs and outputs correspond to each subtransaction. This does reveal other information that can be used to infer the structure of the subtransactions, for example if arbitrary amounts are allowed and one input and one output have nearly the same value, an observer can infer that the source and destination are related. Depending on the protocol used to construct the transaction, it may also be possible for other participants to learn the internal structure. These sorts of protocols have been generalized into other "mixing" protocols, which may provide more privacy but may entail other forms of risk [21].

1.1.2 Ring Signatures and Monero

Ring signatures were suggested as an early solution to adding more private transactions to Bitcoin [22]. Although they never were in Bitcoin, they have been implemented in Monero. A ring signature allows a single signer to construct a signature over a message with one of n candidate public keys, without revealing which public key was used to construct the signature. In a cryptocurrency, as in Monero, a transaction can select the real input, as well as several additional inputs, and use a ring signature to obscure which input is real. Ring signatures can be understood as a kind of zero knowledge membership proof with witness size proportional to the size of the set. Their advantage lies in their relative simplicity, compared to SNARK based protocols, and historically their lack of a trusted setup and compatibility with general discrete log hard groups, i.e. they did not require a pairing or group of unknown order. Their disadvantages are their linear scaling, in the size of the set and the transaction, and their smaller anonymity set. They are also vulnerable to certain attacks when an adversary controls a large number of outputs [23].

1.1.3 SNARKs and Zcash

SNARKs or Succinct Non-interactive ARguments of Knowledge were initially deployed in Zcash using the protocol of Groth [20] and have since been generalized substantially. Zcash itself developed out of an earlier proposal Zerocoin [24], which used RSA accumulators, in an effort to support more ergonomic features like variable value notes and shorter proofs. Compared to ring signatures, ZCash transactions had much smaller proofs that were much more expensive to compute and required a trusted setup per circuit to be proved. Since this initial protocol, many other SNARK protocols, and zero knowledge proofs more generally, have been developed to relax the trusted setup requirements and improvements and to reduce the computational burden of constructing SNARKs [25]. Zcash also uses a related Pedersen hash based construction for their Merkle proofs, although it does not use a cycle of curves.

Recently, ZCash introduced the Orchard protocol [1] that uses the Halo 2 [26] proof system [27] and which does not require a trusted setup. This is not a traditional SNARK, in the sense that proof verification is not succinct, and instead supports a form verification where a small amount of work can be performed to transform multiple proofs into a form whose verification time is independent of the number of proofs that were transformed. This is intended to eventually be used with recursive proof composition to perform non-interactive transaction aggregation and achieve a very low amortized proof size and verifier time per proof.

1.1.4 Other Anonymous Payment Protocols

The original anonymous payments protocol on a decentralized blockchain was Zerocoin [24], which used RSA accumulators for its set membership proofs. Proof sizes were fairly large, and at the time use of an RSA accumulator required a trusted setup for the modulus. There have also been several Bulletproof based protocols including Lelantus [28] and Omniring [29] that are transparent and provide better transaction size scaling properties than ring signatures, but still have linear verification time in the size of the set. This allows much larger anonymity sets than Monero, and mostly avoids the aforementioned attack, but still limits the size of the anonymity set compared to Zcash. Protocols like Hyrax [30] and Dory [31] are transparent, but have proof sizes that are approximately ten times larger than μCash. There are also modern accumulator based protocols life Veksel [32] for anonymous account based payments, which can be instantiated with a variety of accumulators and proof systems. Account based payment systems are used by cryptocurrencies like Ethreum and are an alternative to TXO based systems.

1.1.5 Verkle Trees

Verkle trees [33] also use Pedersen commitments in a Merkle tree. Unlike an ordinary Merkle tree, which would require presenting the entire witness at each level, a Verkle tree commits to the hashes of its children in a Pedersen commitment, which allows proving that a particular hash is one of its children without opening the entire commitment. Ethereum plans to use Verkle trees in order to construct light clients [34]. Despite both being Merkle trees and using Pedersen hashes, Curve Trees and Verkle trees differ substantially in how they operate and how they are intended to be used. The most important distinction is that Verkle trees are typically not designed to support zkSMPs, as the generic hash functions used by Verkle trees are not efficient to prove in zero knowledge.

1.1.6 Curve Trees

Recently, the accumulator and zkSMP protocol of Curve Trees [5] was also introduced by the same authors as Veksel. A curve tree is a Merkle tree defined using a sequence of structure preserving Pedersen hash functions over a tower of elliptic curves $\#E_i(p_i) = p_{i+1}$. When instantiated over a cycle of curves, in particular a 2-cycle, this allows constructing trees of arbitrary depth and proving membership in the tree using a pair of zero knowledge proofs over each curve. The Curve Tree zkSMP protocol acheives zero knowledge by, at each level, selecting one of the children and "reblinding" it by adding a secret scalar multiple of the blinding point. These values are then published and opened in the proof over the other curve.

The Curve Tree zkSMP is used to build an anonymous payment system called $\mathbb{V}cash$. This is not an accomunt based system, but is based on the earlier work on Veksel. Transaction sizes are approxiamtely $2kb$ for a set of 2^{32} elements and support two inputs and two outputs. Certain details are left unspecified, like the nullifier protocol, but should not affect the over all prover or verifier complexity much.

This protocol is very similar to μCash, given the underlying accumulator structure is the same. However, the structure of the zkSMPs is different in the following way: rather than reblinding each child seperately, each zkSMP takes a random linear combination of the children together, along with blinding, and sends that to the other proof. In conjunction with Weil ECIPs and the nested Bulletproof verification, this allows μCash to support larger transactions with smaller proof sizes. It remains to be seen what the difference in performance between the two protocols is. I suspect that the techniques of this paper, specifically the zkSMP protocol and general use of Weil ECIPs, are highly compatible with $\mathbb{V}$cash.

1.2 Contributions

This paper is organized around fully specifying the anonymous transaction protocol. Transaction protocols can be organized according to whether or not they hide the internal structure of the transaction, like types and amounts of money being transferred, and whether or not the hide the external relationships between transactions. Protocols that do neither include most cryptocurrencies like Bitcoin and Ethereum, protocols that hide internal information but not external information are called confidential, and protocols that hide both are called anonymous. This is analogous to the relationship between encryption, which provides confidentiality, and anonymity networks like Tor, which hide metadata. There have been protocols that provide anonymity but not confidentiality like ecash [35] and Zerocoin [24], but it in order to prevent tracking amounts as a proxy for identity they typically require using tokens of fixed denomination.

The central component of the anonymous transaction protocol is the new Curve Tree (CT) zkSMP protocol. The zkSMP exploits the linearity of the hashes and uses two communicating Bulletproofs. In short, these Bulletproofs take a random linear combination of the Pedersen hashes at alternative levels and send the resulting curve points to the other proof, which opens it as a Pederesen commitment into the proof. This "aligns" the field structure of the proof with the field structure of the witness, and uses the Weil ECIP protocol to efficiently prove the elliptic curve operations to prepare the commitments. In order to avoid the need to publish both Bulletproofs separately, which would more than double the proof size as compared to the final proof size of the protocol, it is possible to use the same Weil ECIP protocol to perform portions of the Bulletproof verification in zero knowledge inside the complementary proof.

To build the transaction protocol, each input to the transaction uses a CT zkSMP for some fixed, public TXO root commitment. These proofs are handled separately, to support multiple distrusting provers. Each proof construct a Weil ECIP over a number of hashes approximately half the depth of the Merkle tree, and at tree depths of $32-48$, which is likely to be sufficient for cryptocurrencies in the near to medium term, each ECIP witness will occupy approximately 2000 scalars per Bulletproof over three rounds. By depth, I refer to $\lceil \log_2 N \rceil$ for a set of N leaves. The same proof structure will also be used for outputs. The outputs of a transaction will be stored in a local CT commitment, and the proof will use a modified zkSMP to open them all in zero knowledge. The output tree can then be inserted into the global TXO tree without ever being publicly opened. Each output costs approximately 100 scalars to prove, as this tree is much shallower than the global tree and interior nodes can be amortized across all the leaves. Since proof capacity is largely fungible within rounds, this means that the transactions of this protocol can easily support over 10 times as many outputs as inputs at a fixed proof size. This extra capacity may be useful for adding scripting functionality to the output tree, perhaps using a variant of Taproot [36].

Besides the zkSMP protocol, anonymous transactions also require a nullifier protocol and a confidential transaction protocol. A nullifier, also called a key image in Monero, is a deterministically derived quantity from a UTXO that is hard for an external observe to associated with the UTXO and serves to prevent double spending. The nullifier protocol used by the transactions is structurally similar to the protocol used by Monero. The confidential transaction protocol, which as mentioned before handles the conservation of money proofs, is an adaptation of Bulletproofs++. This protocol supports multi-party, multi-asset confidential transactions efficiently as well as large base range proofs to reduce the witness size. These features are not strictly necessary for an anonymous transaction protocol, but are useful in practice and found in many non-confidential transactions protocols. Since the real world privacy of a protocol comes from the number of people who use the protocol, supporting useful features will hopefully increase the number of users and thereby increase the anonymity set for all transactions. Anonymity set size creates a feedback loop dependent on some threshold, above which more people will use the protocol increasing the anonymity set size and below which people will not use the protocol as it provides limited anonymity decreasing the set size.

To that end, μCash has also been designed to support adaptor signatures and scriptless scripts. The way such protocols often work [37] has the parties to a contract, like an atomic swap or a payment channel, prepare transactions liquidating a multisig output before the output has been funded. This ensures they can recover their money if the protocol fails or the counterparty defects. In a zkSMP based transaction protocol, this creates a cyclic dependency, as a transaction spending an output must prepare a zkSMP for the output in the TXO set, but the users will not fund the multisig output until this transaction has been created. I resolve this paradox through the use of a multi-phase transaction protocol. The first phase, the transaction body phase, proves the confidential portion of the transaction, and after this phase the amounts and types of currency are not needed to complete the transaction. The second phase, called the membership phase, shows that all the inputs belong to a Merkle root using CT zkSMPs. This phase can be completed by any collection of provers where that collectively know all the Merkle witness information, in particular it can be completed by one party of a swap or payment channel without the cooperation of the counterparty. The final phase of the transaction, called the Bulletproof phase, performs the nested Bulletproof verification described above and can be completed by anyone, including a miner or untrusted third party prover.

The first two phases support privacy preserving multi-party proving using the same techniques as other Bulletproof protocols. That is, provers only ever need to perform linear operations over secret values, and it is never necessary for two provers to multiply two mutually secret values. To protect mutual privacy, it must be the case that the value, as group elements, of the input commitments never be revealed to the other provers in the transaction body or membership proof phases. This is because the sender of transaction will know the input commitment's value as a commitment. These features of the protocol, like support for multi-party, privacy preserving proving and scriptless scripts, can be removed to reduce proof size and improve the efficiency of the prover. However, support for features like these incurs a relatively, to the input zkSMP and nested Bulletproof, small cost at the benefit of many useful features.

As far as I am aware, this protocol has the smallest standalone proof size for an anonymous transaction protocol without a trusted setup, at $1344 + 64n + 32 \lceil \log_2 c \rceil$ bytes over a curve with 128 bits of security, plus 5 to store signs for the curve points. That is a protocol that provides perfect anonymity for each input among the entire anonymity set. The quantity c is bounded by a linear function in (dn, n, m) and is equal to 1 for most transactions in practice. Additionally, I am not aware of any anonymous transaction protocols that support a super set of the features of μCash including scriptless scripts for atomic swaps and payment channels, multi-party multi-asset confidential transactions, and very large numbers of outputs. While current Orchard transactions seem to be much larger, direct comparisons are complicated by the planned introduction of recursive proof composition. When implemented, this will bring the amortized cost per transaction in Orchard arbitrarily low not counting nullifiers and other un-prunable data. I expect, given that congruity of the underlying proof system, that the transactions of this paper will also be compatible with a Halo type accumulation scheme although I do not investigate this question as part of this paper. Additionally, given that recursive proof composition in Halo protocols also requires proofs of EC operations, it may be the case that elements of this protocol can be used in Halo.

1.3 Future Work

I have not completed an implementation of this protocol. I expect that above a certain minimal transaction size, both the prover and verifier will be faster as the underlying circuit and witness size is significantly smaller than that of Curve Trees and Orchard. Additionally, while I believe that all the witness size calculations are correct, in the event that some unforeseen additional costs arise upon further review or in the course of implementing the protocol, it is possible to substantially increase the witness sizes in exchange for a very small increase in proof size, i.e. increasing the outer witness length to $w_0 = 2048$ and the inner proportionally adds one curve point to the proof size. I have tried to make the witness sizes as small as possible to reduce the cost for the verifier, but if necessary they can be increased to fit the protocol.

I also do not attempt to provide a security proof for this protocol. While I do attempt to fully specify the protocol, there are many individual components and it is still necessary to actually show that the resulting protocol satisifes SHVZK, CWEE, and Soundness. There are several pieces of this protocol that merit more comprehensive proofs of security. Implementors should be aware that there may be bugs in the protocol, and that this paper is intended only to communicate the structure of the protocol and demonstrate that such a protocol, at the given proof sizes, is feasible. The protocol itself is fairly modular, and so can be modified to accomodate for errors or as needed in the course of proving its security.

1.3.1 Succinct Verifier

The transactions of this paper do not have a succinct verifier, as the arithmetic circuit representation requires verifier work proportional in size to the circuit. There are alternative ways of representing arithmetic circuits used by SNARKs, in particular PLONK style protocols, that allow making the verifier work independent of the circuit structure. For smaller transactions, this would introduce a large cost. However, for larger transactions it may be possible to adapt these techniques to make the verifier fully succinct, that is logarithmic in the size of the transaction, up to nullifier verification.

The proposed method of supporting larger transaction sizes simply increases the length of the Bulletproof witness vectors. The maximum transaction capacity increases with the product of the length increases, for example quadratically if the lengths maintain a fixed ratio. If instead the prover increased the level of nesting of the proofs, the transaction capacity per proof size would increase exponentially, as each level of nesting incurs a fixed cost but multiplies the proof capacity by the factor of increase in the witness length. Since the circuit representation requires proportionate work from the verifier, the resulting proof system would not have a succinct verifier. However, using an alternative arithmetization scheme it may be possible to have a verifier with complexity linear in the witness length per commitment and the degree of nesting, i.e. succinct in the total proof capacity.

2 Preliminaries

The security paradigm of the protocols of this paper is the same as that of Bulletproofs and other Bulletproofs based protocols: reduction to the discrete log problem in the random oracle model. In this case, as in Halo, this is assumed over both elliptic curves in the cycle. The security arguments proceed in the same manner, using the generalized forking lemma [38] to show Computational Witness Extended Emulation, and demonstrating that there are enough degrees of freedom in the blinding values for the protocol to show Special Honest Verifier Zero Knowledge. This argument is not tight as a proof of security, but is essentially of the same structure as all other zero knowledge proof protocols based on ordinary elliptic curves, i.e. not using pairings.

The nullifier protocol requires additionally the Computational Diffie-Hellman problem (CDH) to ensure that the sender of an output cannot construct the nullifier for the output and the Decisional Diffie-Hellman problem to ensure that the sender cannot decide if a given nullifier corresponds to a particular output. Given that DDH reduces to CDH, it is sufficient to assume the hardness of the DDH problem. These problems are believed to be difficult in elliptic curves without efficiently computable pairings. It should also be noted that while DDH is necessary to protect the privacy of a transaction, solving DDH or CDH is not sufficient to steal an output. To do that would require forging a Schnorr signature, which is reducible to the discrete log problem in the random oracle model.

2.1 Amicable Cycles of Elliptic Curves

There exist certain triples of an elliptic curve and two primes (E, p, q) such that $\#E(F_p) = q$ and $\#E(F_q) = p$. While a general discussion of such triples [4] is outside the scope of this paper, these curves have been used in cryptographic applications before [5, 11, 39] and are the basis of the set membership proof protocol. The construction of these triples in practice is closely linked to complex multiplication, and uses the method described here [40]. This same method, of finding a curve with prescribed order given small CM discriminant, is used in the construction of many families of pairing friendly curves. The security implications of using curves with small CM discriminant are well understood, and the discrete log problem in these curves is believed to be only be slightly easier then in general elliptic curve groups.

As a brief overview of the technique, let $\#E(F_p) = q$ be a prime order curve over a prime order field and define the complex multiplication discriminant $\Delta(E)$ as the squarefree part of the polynomial discriminant of the trace of Frobenius:

$$\Delta(E)s^2 = (p - q + 1)^2 - 4p = p^2 + q^2 + 1 - 2(pq + p + q)$$

Note that this quantity is symmetric in p and q, so if there were another elliptic curve $\#E'(F_q) = p$ it would have the same CM discriminant. It is a theorem that for every order N within the Hasse bound for a given field there exists an elliptic curve with that order over a prime field. Therefore, if p is within the Hasse bound for F_q, the curve E' must exist. By CRT, both of these local curves can be lifted to one curve over $\mathbb{Q}$, so I will simply refer to both curves as E.

Over the complex numbers $\mathbb{C}$, elliptic curves can be parameterized by the elements of the upper half plane $\tau \in \mathbb{H}$, $\text{im}(\tau) > 0$ up to isomorphism, and each isomorphism class can be identified with its j invariant $j(\tau)$. It is known that for quadratic integers in the upper half plane, $\tau^2 + a\tau + b = 0$ for integer a and b, $j(\tau)$ is algebraic and is a root of the Hilbert polynomial H_Δ associated to the quadratic field $\mathbb{Q}(\tau)$ with discriminant Δ.

The details of why this is the case are not important for the purposes of this explanation. It suffices to note that the degree and coefficients of the Hilbert polynomial are proportional to the order of the class group of the CM field and that the coefficients are integral. This allows factoring the polynomial over the field F_q to find the j invariants of local curves with prescribed CM discriminant for discriminants that are sufficiently small, i.e. negligible in the security parameter.

$$H_\Delta(x) = \prod_{\Delta(E)=\Delta} (x - j(E))$$

For each candidate j invariant and each twist, it is straightforward to construct an elliptic curve with that j invariant and to use Schoof's algorithm [41] to determine the structure of the curve group. Since it has already been established that there exists a curve over F_q with order p, and since such a curve must have CM discriminant Δ, one of the candidate curves must have order p.

As the size of the Hilbert polynomial is proportional to the size of the class group, which is proportional to the square root of the CM discriminant, this procedure will not work for general elliptic curves.

However, many curves already used in practice have small CM discriminant, like SECP256k1, which is used by Bitcoin, Ethereum, and many other cryptocurrencies for signatures, as well as many families of pairing friendly curves. The curves used by ZCash with Halo 2 are also constructed in this manner [42], with the additional structure requirement that they have large power of 2 order subgroups for efficient FFT computations.

2.2 Notation

Curve points will be denoted via capital, non-italicized words and scalars via lowercase italicized words. This creates an ambiguity in juxtaposition of scalars denoting multiplication or a multi-character name. Typically, scalars will use one character and the latter interpretation should be preferred. The group operation of point addition will be written additively and scalar multiplication via juxtaposition.

$$(x + y)(\mathrm{G} + \mathrm{H}) = x\mathrm{G} + x\mathrm{H} + y\mathrm{G} + y\mathrm{H}$$

The coordinates of curve points will be denoted via accessor functions x and y, and any expression in $x, y \in F(E) \subset E \to F$ will be a function in the curve points as well. For example, for any point $P \in E_{A,B}$

$$y(\mathrm{P})^2 - x(\mathrm{P})^3 - Ax(\mathrm{P}) - B = 0$$

Throughout the protocol, there will be two elliptic curves in use over two different fields of amicable characteristics. The fields will be denoted by F_0 and F_1 with curves $E_0 = E(F_0)$ and $E_1 = E(F_1)$. I have tried to organize the paper as much as possible to avoid discussing both elliptic curves at the same time to limit ambiguity. When this is unavoidable, membership in a curve or field will typically be denoted by the parity of subscripts, so $\mathrm{P}_{2k} \in E_0$.

2.2.1 Linear Algebra

Vectors will be denoted via bold letters with the same rules as their elements. That is, vectors of points will be capitalized and vectors of scalars will be lowercase. Vectors are zero indexed and elements of a vector will be written using subscripts and the syntax of the elements, i.e. not bold. Concatenation will be written using $\oplus$ or parenthesized lists and vectors are padded with the identity element on the right as necessary.

$$\mathbf{x} = (0, 1) \oplus \mathbf{y} \qquad x_1 = 1$$

Inner products are defined over any field and vector space over the field, which includes scalars with scalars and scalars with points. Inner products are written

$$\langle \mathbf{x}, \mathbf{G} \rangle = \sum_i x_i \mathrm{G}_i$$

Matrices will be written using italicized capital letters and will only ever include scalars. Matrices can multiply any vector of objects that form a vector space over the scalars in the matrix including scalars, curve points, or vectors of each.

2.2.2 Commitments

Pedersen commitments [3] will be written written using the function Com_b with the subscript indicating which curve they are mapping into. Blinding will be handled explicitly by the caller and integrated into the input vector. Both commitment functions have a maximum witness vector length w_0 and w_1 are therefore functions

$$\mathrm{Com}_0 \in F_0^{w_0} \to E_1 \qquad \mathrm{Com}_1 \in F_1^{w_1} \to E_0$$

Commitments can be extended to return vectors of commitments for larger inputs that exceed w_b in length. These function will be written using the vector form $\mathbf{Com}_b$ and will break the input into concatenated chunks of size w_b or smaller to commit. I will also use the "function" $\mathbf{wit}_b$ which is the inverse of Com_b. This is convenient for referring to the opening of a linear combination of commitments but is not actually a computable function. When it is clear from context, the subscript may be dropped.

$$\mathbf{wit}\left(\mathrm{Com}\left(\mathbf{x}\right)\right) = \mathbf{x} \qquad \mathrm{Com}\left(\mathbf{wit}\left(C\right)\right) = C$$

Merkle commitments will be written using the function, where $*$ exponentiation denotes an arbitrary list of inputs

$$\text{MrklCom} : E_0^* \to E_0$$

Note that the elliptic curves for Merkle commitments are fixed to be E_0, as this will canonically be the curve over which commitments to TXOs will be defined. However, the construction is equally valid over either curve.

2.2.3 Protocol Objects

To keep track of the information associated to various protocols in this paper, protocol objects will be written in fixed width font and will have a unique set of functions defined over the protocol object. For example, a protocol that includes a vector of blinding commitments would be written

$$\mathbf{Bl}(\texttt{Obj})$$

The convention for these functions follows the same rules as commitments and values with respect to points, scalars, and vectors depending on their return value. That is, a function is written in the syntactic style of its return type. For a particular name, the upper case form will denote a commitment and the lowercase vector form will denote the opening of the commitment

$$\mathbf{wit}(\text{Divs}(\texttt{Obj})) = \mathbf{divs}(\texttt{Obj}) \qquad \text{Com}(\mathbf{divs}(\texttt{Obj})) = \text{Divs}(\texttt{Obj})$$

Multiple protocol objects with the same structure can be composed in a natural manner by concatenating the witness vectors in the same round. As a short hand for this, rather than redefining new objects with the same interface, one can use concatenation on the objects directly

$$\mathbf{divs}\left(\bigoplus_i \texttt{Mem}_i\right) = \bigoplus_i \mathbf{divs}(\texttt{Mem}_i)$$

It is important to note that this is mostly a syntactic convention to make reading the protocols easier, and functions may not be defined on a given protocol object until a certain point in the execution of the protocol. The various commitments and witnesses associated to a protocol object should be understood to implicitly include the arithmetic circuits associated with the witnesses. This hopefully abstracts the fine grained details of the protocol enough to reveal the important structural properties while retaining sufficient specificity to eliminate ambiguity.

2.3 Discrete Log Problem

The discrete log problem can be defined over any abelian group. Over an elliptic curve, it asks the adversary to find x given points G and H such that $\text{H} = x\text{G}$. There are a large number of reductions of the discrete log problem to other, related problems. In particular, it is equivalent to the problem of finding any vector $\mathbf{x}$ for a vector of points $\mathbf{G}$ such that $\langle \mathbf{x}, \mathbf{G} \rangle = 0$. The discrete log problem is famously [43] solvable by a quantum computer, via Shor's algorithm, and therefore the protocols of this paper and all other Bulletproof-based protocols are not quantum resistant.

2.3.1 Diffie-Hellman Problems

The Diffie-Hellman problems, also believed to be hard in elliptic curves, are closely related to the discrete log problem. The CDH asks the adversary to compute $xy\text{G}$ given (A, B) where $\text{A} = x\text{G}$ and $\text{B} = y\text{G}$. This is clearly reducible to the discrete log problem, but it is not known to be equivalent to the discrete log problem over general elliptic curves. The decision version of the problem asks the adversary whether or not $(\text{A}, \text{B}, \text{C}) = (x\text{G}, y\text{G}, xy\text{G})$ for some unknown x and y. This problem is clearly reducible to the computational version of the problem, since given a CDH oracle one can simply compute the correct C given the other two points, but it is not known to be equivalent over general elliptic curves. In curves with pairings, the DH problems are not necessarily hard even though the discrete log problem is believed to be.

2.4 Zero Knowledge Proofs

Zero knowledge proofs and zero knowledge arguments of knowledge are protocols that allow a prover to convince a verifier that some witness $\mathbf{w}$ belongs to an NP-language L, either unconditionally or up to some computational assumptions, without revealing any information about the witness other than its membership in L. Zero knowledge proofs in which the verifier actively participates in the protocol are called interactive proofs, and such proofs can be constructed using a very large class of problems. Proofs in which the space of verifier responses can be constructed as a function of uniformly random, public values, also called public coin protocols, can be made non-interactive via the Fiat-Shamir heuristic, where each random value supplied by the verifier is supplied by a random oracle, in practice a hash function. This transformation feeds as input to the oracle all of the prover responses up until that point in the protocol. Failing to has all of the prover responses is a common source of implementation bugs in such protocols.

Each NIZK protocol must show three properties to be proven correct in the random oracle model: Special Honest Verifier Zero Knowledge, Computational Witness Extended Emulation, and Soundness. The first, SHVZK, essentially shows that a proof leaks no information about the witness, the second, CWEE, shows that the prover must know a witness in order to construct a proof, and the third shows that the prover can construct a proof if and only if the witness is valid. The soundness property is also sometimes broken down into an additional property Completeness, which states that the prover can construct a proof for a valid witness, and soundness refers to the property that a prover cannot construct a proof for an invalid witness. These properties are defined precisely by Bulletproofs, and this paper will use those definitions.

The first two properties follow almost mechanically for protocols of the form of this paper in the random oracle model. To show SHVZK, one shows that, given access to the challenge values in advance, a simulator is able to produce a proof transacript from the same distribution as the distribution of actual proofs. Typically, this follows from sampling all commitments and the final openings from the adversary and then computing a number of commitments equal to the number of verification equations synthetically from the result, as in other Bulletproofs protocols. To show CWEE, an emulator constructs a tree of proofs by rewinding a valid proof and sampling new challenge values. Using multiple challenge, the prover is able to solve for the witness. Typcially this follows from the linear independence of the coefficients of the linear combinations of commitments as polynomials or rational functions in the challenges. Soundness is somewhat less mechanical, as it depends non-trivially on the language to which the witness must belong, although it typically follows via repeated application of the Schwartz-Zippel lemma and from the hardness of the discrete log problem.

2.4.1 Random Oracle and Fiat-Shamir

This protocol uses the Fiat-Shamir transformation [44] to produce a non-interactive proof. While the description of the protocol is given as a sequence of interactions between a prover and verifier, in practice this is not how the protocol will work. Instead, each call to the verifier for a challenge will be replaced with a call to a hash function, which is modelled as a random oracle. This hash function is then evaluated over the entire proof transcript up until that point in the protocol. It is critcially important that the entire transcript be included, including all public information that varies with the proof. Failure to properly hash everything has been a source of serious security vulnerabilities [45].

2.5 Accumulators and Set Membership

A cryptographic accumulator [46] is an object that supports efficient witnesses of membership for some set. It must also be difficult to prove that a value is not present in a set. There are many accumulator protocols, including RSA accumulators, which can be built in arbitrary groups of unknown order [47], and Merkle trees. Accumulators over groups of unknown order support efficient zero knowledge proofs of membership in the accumulator as well, which Merkle trees do not in general.

2.6 Schwartz-Zippel Lemma

The Schwartz-Zippel lemma states that, given a multivariate polynomial F over a finite field set S of size $N = |S|$, the probability that F is zero on a uniformly randomly chosen input is inversely proportional to N and is proportional to the total degree d of F, that is the monomial with the highest sum of powers.

Formally

$$P\left(F(\mathbf{x}) = 0 \mid x_i \leftarrow S\right) = \frac{d}{N}$$

This forms the basis of essentially all the soundness arguments of the zero knowledge arguments of this paper, and many protocols more broadly. Typically, the prover will commit to the polynomial F, sometimes explicitly and other times F will be defined as a combination of the committed information and public information part of the protocol itself, this commitment will be passed to hash function. This hash is treated, for the purposes of analyzing the security of the protocol, as a random oracle. Assuming its output is randomly distributed, the polynomial F is evaluated at these values. From the structure of F being zero at its input, the statement to be proved follows as a consequence of the Schwartz-Zippel lemma.

2.6.1 Short Challenges

Given that the probability of a non-zero polynomial being zero at a random value drawn uniformly at random with probability p is exactly p, it follows that p can be set to exactly $2^{-\lambda}$ while retaining acceptably low probability of soundness failure. Elliptic curves, with or without CM, with security level λ can be selected to have prime group order $2^{2\lambda}$ and be defined over fields of the same magnitude. This is twice the length necessary to achieve soundness from the Schwartz-Zippel lemma.

In certain circumstances, the prover can draw challenges from a smaller space, of size 2^{λ} or half the "length" of the field, and still have sound proofs. If the challenge is used for a polynomial with larger degree, the prover might elect to use a slightly larger challenge space. In any case, especially when dealing with ECIP proofs, using smaller challenges has a direct, linear effect on the witness size and proving complexity.

2.6.2 Short Challenges With CM

Note also that the Schwartz-Zippel lemma places no requirement on which particular set of challenges the challenge is drawn from, only that it be a particular size and the challenge be drawn uniformly at random. For elliptic curves with complex multiplication, rather than choosing a challenge $e \in (-2^{\lambda-1}, 2^{\lambda-1}]$ the verifier can choose two challenges $a, b \in (-2^{\lambda/2-1}, 2^{\lambda/2-1}]$ and use $e = a + \omega b$.

Depending on the specific CM field, the following argument can be modified to show that all challenges of this form are distinct. Supposing that $\omega^3 = 1$ is a primitive third root of unity, given $a_0 \neq a_1$ and $b_0 \neq b_1$ then $e_0 \neq e_1$ by the following reasoning. By way of contradiction assume

$$e_0 - e_1 = (a_0 - a_1) + (b_0 - b_1)\omega = 0 \mod p$$

Then, it follows that the norm of this value, with respect to the CM field, is also zero, as it is the product of this value with another value. In the third root of unity case, this implies

$$(a_0 - a_1)^2 - (a_0 - a_1)(b_0 - b_1) + (b_0 - b_1)^2 = 0 \mod p$$

As a consequence of the size of the distribution from which a_i, b_i are sampled, each of these terms cannot exceed 2^{λ}, and the sum of them therefore cannot exceed $3 * 2^{\lambda}$. Since there are no non-negative values other than 0 less than p that are divisible by p, this norm must be identically zero. Additionally, this relation holds over the integers, and since there are no elements of $\mathbb{Q}(\omega)$ with zero norm other than zero, $e_0 - e_1 = 0$ which is a contradiction.

This works for larger distributions up until just slightly less than half the length of the prime field, at which point collisions are possible. For different number fields, the distance below which collisions are possible depends upon the size of minimal polynomial of the CM field, as measured by the coefficients. For CM fields that can be used to generate amicable curves, these values are practically bounded by the size of the associated Hilbert polynomial, so in practice it should not matter what CM the curves have so long as the challenge distribution is significantly smaller than the size of the field. This distribution will be referred to as half(F).

2.6.3 Rational Schwartz-Zippel Lemma

While the Schwartz-Zippel Lemma is defined over polynomials, it generalizes straightforwardly to rational functions in the zero knowledge proof setting. If the prover wants to show that some rational function vanishes, then evaluating at a random point e and checking that

$$\frac{f(e)}{g(e)} = 0$$

Implies that the numerator is identically zero, if neither the numerator nor the denominator is zero at the challenge point. That is, applying the Schwartz-Zippel lemma to the polynomial $f(e)g(e)$. If both polynomials are of negligible total degree, then soundness follows. The reason why both need to be checked is that, typically, one commits to the coefficients of f and the coefficients of g. Then, the verifier selects the challenge e and the prover responds with some value r. The prover will then show that

$$f(e) = rg(e)$$

If $g(e)$ is zero and $f(e)$ is zero, then r can be any value. In certain cases, one can obtain a stronger bound on the soundness of the protocol, for example if $f(e) = 1$ as a polynomial. However, it is sufficient to show that the probability of being a root of either polynomial in general is negligible.

2.7 Bulletproofs

Bulletproofs allow the prover to convince the verifier that a Pedersen to two vectors $\mathbf{x}$ and $\mathbf{y}$, and a scalars v satisfy an inner product relation $v = \langle \mathbf{x}, \mathbf{y} \rangle$. This can be modified to prove a weighted, by the powers of some scalar q, inner product relation as in Bulletproofs+ [48]. The original Bulletproof argument comes from the following polynomial relation

$$\mathbf{x} = \mathbf{x}_0 \oplus \mathbf{x}_1 \qquad \mathbf{y} = \mathbf{y}_0 \oplus \mathbf{y}_1$$

$$\left\langle e^{-1}\mathbf{x}_0 + e\mathbf{x}_1, e\mathbf{y}_0 + e^{-1}\mathbf{y}_1 \right\rangle = e^{-2} \left\langle \mathbf{x}_0, \mathbf{y}_1 \right\rangle + \left\langle \mathbf{x}, \mathbf{y} \right\rangle + e^2 \left\langle \mathbf{x}_1, \mathbf{y}_0 \right\rangle$$

Given the Pedersen commitment C to a correct inner product

$$\mathrm{C} = \langle \mathbf{x}, \mathbf{y} \rangle \mathrm{G} + \langle \mathbf{x}, \mathbf{G} \rangle + \langle \mathbf{y}, \mathbf{H} \rangle$$

There are three inner product present here, between the witness vectors and between each witness vector and its basis points. The polynomial relation is applied to all three inner products, and all the e^{-2} terms are aggregated into a commitment L and all the e^2 terms into R. Following commitment to these values, the verifier will choose a random challenge e and the prover will repeat the procedure on the commitment $\mathrm{C}' = e^{-2}\mathrm{L} + C + \mathrm{R}$ with the update basis vectors.

Verification can be performed by a single ECIP computation. Given the fully reduced commitment $\mathrm{C}' = ab\mathrm{G} + a\mathrm{G}' + b\mathrm{H}'$, as well as all the responses for each round, the prover will check that

$$C + \sum_{i=0}^{n} e_i^{-2} L_i + e_i^2 R_i = ab\mathrm{G} + \langle a\mathbf{t}(e_i : i), \mathbf{G} \rangle + \left\langle b\mathbf{t}(e_i^{-1} : i), \mathbf{H} \right\rangle$$

Where $\mathbf{t}(e)$ is generated as a tensor product of the challenge vectors. Supposing that the vectors are split in half, rather than as in Bulletproofs++ where they are split into interleaved halves, this tensor product is given by

$$\mathbf{t}(\mathbf{e}) = \bigotimes_{i=0}^{n} (e_{n-i} \oplus e_{n-i}^{-1})$$

2.7.1 Linear Argument

The Bulletproof argument can be modified to prove different relations, as will the norm relations or linear relations of Bulletproofs++. Halo similarly modifies Bulletproofs to prove a different relation. The linear relation for Bulletproofs uses the same polynomial relation, but applied to inner products. One for the witness with a public constraint vector $\mathbf{c}$ and one for the witness with the basis.

$$C = \langle \mathbf{c}, \mathbf{x} \rangle \mathrm{G} + \langle \mathbf{x}, \mathbf{G} \rangle$$

The linear relation is not sufficient by itself to prove arithmetic circuits, although it can be combined with an external multiplication of public scalars, as will be described in the following section on circuits. There are a few other benefits of the linear argument. Firstly, it is possible for the protocol to choose a stopping point of the argument to maximize efficiency. When the length of the witness vector is less than six, applying the Bulletproof reduction again replaces two or fewer scalars from the witness with two curve points. This actually increases the proof size, so the linear argument should be stopped when the remaining witness vector is of length less than six.

Verification of the linear vector is also simplified. At each point in the circuit, the intermediate commitment $\mathbf{C}$ can always be written a linear combination of the basis points, plus the scalar basis point scaled by the associated $\mathbf{c}$ coefficient. That is

$$C = \sum_i x_i(\mathrm{G}_i + c_i\mathrm{G}) = \langle \mathbf{x}, \mathbf{G} + \mathbf{c}\mathrm{G}\rangle$$

This will become particularly useful when performing the argument in zero knowledge with respect to the responses. This version will fully reduce the witness vector to a single scalar and then check that the remaining commitment has a known discrete log with respect to the full final basis point.

2.7.2 Generalized Round Sizes

Both the inner product and linear arguments can be generalized to more responses per round of prover and verifier interaction in a natural way. Partitioning the vectors into k approximately equal length subvectors, the new polynomial relation becomes

$$\mathbf{x} = \bigoplus_{j=0}^{k-1} \mathbf{x}_j \qquad \mathbf{y} = \bigoplus_{j=0}^{k-1} \mathbf{y}_j$$

$$\left\langle \sum_{j=0}^{k-1} e^{-j}\mathbf{x}_j, \sum_{j=0}^{k-1} e^{j}\mathbf{y}_j \right\rangle = \langle \mathbf{x}, \mathbf{y}\rangle + \sum_{j=1}^{k-1} e^{-j} l_j + e^j r_j$$

Where l_j and r_j are produced by grouping like terms by e power coefficient. For any $k > 2$, this produces a larger number of responses for the same size vector, as $n/\log n$ is increasing for $n \geq 2$. However, if the Bulletproof relation were proved in zero knowledge with respect to the responses, the prover could publish a single commitment to the responses. Ignoring the potential cost of proving the ECIP relation for the round, this would make proof size proportional only to the number of rounds. In this case, proof size can be substantially reduced by using larger rounds. In practice, there is a maximum ECIP that can be proved without using more commitments and without increasing the size of the proving proof, but in general it is typically feasible to substantially reduce proof size for witness lengths greater than some minimum value.

When used in the protocol, the protocol object $\mathtt{BP}(\mathbf{c}, \mathrm{C}, n)$ is defined to be the application of a reduction of size n to linear bulletproof with constraint vector $\mathbf{c}$ and $\mathbf{wit}(\mathrm{C}) = \mathbf{x}$. Then the responses $\mathbf{L}$ and $\mathbf{R}$ are defined as above and the challenge vector $\mathbf{e}$ is the e power corresponding to the response. The accessor $\mathbf{c}$ returns the new constraint vector. The new commitment is given by

$$\mathrm{C} + \langle \mathbf{e}(\mathtt{BP}), \mathbf{L}(\mathtt{BP}) \oplus \mathbf{R}(\mathtt{BP})\rangle = \mathrm{C} + \sum_{i=1}^{n-1} e(\mathtt{BP})^{-i}\mathrm{L}_i(\mathtt{BP}) + e(\mathtt{BP})^{i}\mathrm{R}_i(\mathtt{BP})$$

2.8 Weil ECIPs

There are many different methods for proving elliptic curve inner products (ECIPs) in zero knowledge. Frequently, these have been used for Pedersen hash functions, for example in JubJub [49] and Bandersnatch [50]. The methods of [6] using Weil reciprocity yield the smallest witness sizes and smallest number of multiplications for large numbers of elliptic curve operations. I will use the ECIP protocol outlined there exactly throughout these protocols, in the membership proofs as well as proving the Bulletproof argument in zero knowledge. By witness size and proving time, the vast majority of the complexity of this protocol involves proving elliptic curve operations.

These proofs rely on the fact that the zeros and poles of a rational function of an elliptic curve $f \in F(E)$ with multiplicity sum to zero in the elliptic curve group. This is a consequence of the definition of the elliptic curve group as the Picard group of the curve. These roots and poles, which are treated as roots with negative multiplicity, form the principal divisor associated with the function, and so I will refer to the function itself as the divisor witness or sometimes just the divisor. As compared to just storing a principal divisor directly, as a list curve points, using a rational function reduces the size of the witness by a factor of two, as each point adds only one term to the function. Below, the first sum should be understood as a formal sum over curve points and the latter as a sum in the curve group.

$$\mathrm{div}(f) = \sum_{\mathrm{P} \in E} v_{\mathrm{P}}(f)\mathrm{P} \qquad \sum_{\mathrm{P}} v_{\mathrm{P}}(f)\mathrm{P} = \mathrm{O}$$

To prove that the divisor witness contains a prescribed set of points, the prover will use Weil reciprocity. For the purposes of the proof, it is sufficient to consider the restricted form of Weil reciprocity for polynomials of the elliptic curve which states that a polynomial f evaluated at the roots of the polynomial g equals g evaluated at the roots of f up to a constant depending only on the degrees of the polynomials and their leading coefficients.

$$\prod_{f(\mathrm{P})=0} g(\mathrm{P})^{v_{\mathrm{P}}(f)} = C \prod_{g(\mathrm{P})=0} f(\mathrm{P})^{v_{\mathrm{P}}(g)}$$

Letting $g = y - \lambda x - \mu$ be a random line with points of intersection $\mathrm{A}_i \in E/F$ for $i = 0, 1, 2$, then both sides of the above expression are polynomials in the coefficients of the line. Treating these coefficients as formal variables, the prover can take the logarithmic derivative of both sides. This helpfully eliminates the constant and transforms both sides into sums of rational functions. However, since the points A_i are not independent of the line, differentiating their coordinates with respect to say μ yields a non-trivial value. The equation becomes

$$\sum_i \frac{g'(\mathrm{A}_i)}{g(\mathrm{A}_i)} \frac{dx(\mathrm{A}_i)}{d\mu} = -\sum_{\mathrm{P}} \frac{\frac{d\lambda}{d\mu}x(\mathrm{P}) + 1}{y(\mathrm{P}) - \lambda x(\mathrm{P}) - \mu}$$

Where the derivatives are any solution to the system of linear equations

$$\left(\frac{3x(\mathrm{A}_i)^2 - A}{2y(\mathrm{A}_i)} - \lambda\right) \frac{dx(\mathrm{A}_i)}{d\mu} - \frac{d\lambda}{d\mu}x(\mathrm{A}_i) - 1 = 0 \qquad 2\lambda \frac{d\lambda}{d\mu} = \sum_i \frac{dx(\mathrm{A}_i)}{d\mu}$$

These four equations are necessary because in the case of a double point $A_0 = A_1$, the three line equations degenerate into two, whereas the latter does not. Using the higher multiplicity challenge point reduces the number of terms on the left hand side, so this is the form that will be used in this paper.

Much like the Bulletproofs++ permutation argument, this makes the relations linear in the following sense. Given two functions over divisors with non-trivial intersection, the prover can take a linear combination of the equations and combine the line terms evaluated at the same point. For an ECIP computation, each row is structured as such a function over a common basis and two point encoding the previous sum and the result of the current row. Taking a linear combination of these rows corresponding to the base of the ECIP causes all the intermediate points to cancel and allows aggregating all the basis terms into a single term. There are risks if the scalars are too large, but these can be avoided by using CM to split the scalars or by using small challenges when possible.

2.8.1 Public Scalars, Private Points

The most common version of the ECIP protocol in this paper will be the public scalar, private point variation. In this version, the scalars are known to the verifier but the points are not. This has a variety of uses beyond membership proofs, for example in proving a Schnorr signature is valid in zero knowledge with respect to the public key. This ECIP protocol is taken directly from the Weil ECIP paper, and works for elliptic curves with zero j invariant. It uses base -3 and computes each row in reverse order. The scalars are parameterized by their length l, which is the length of the longest scalar. To show the ECIP relation where

$$\mathrm{Q} = \sum_i (a_i + \omega b_i)\mathrm{P}_i$$

$$a_i = \sum_{j=0}^{l-1} d_{ij}(-3)^j \qquad b_i = \sum_{j=0}^{l-1} e_{ij}(-3)^j \qquad d_{ij}, e_{ij} \in \{-1, 0, 1\}$$

The prover will commit to the points P_i and line B_i for each point constructing the value $(1-\omega)P_i$ as a linear combination of P_i and ωP_i. The other point not expressible as a $-\omega$ multiple of P_i is given by $y \mapsto -y$ for the same line. Then each of the ± 1 linear combination of P_i and ωP_i is treated as a basis point. Each row will construct

$$\mathrm{Q}_j + 3Q_{j+1} = \sum_i (d_{ij} + \omega e_{ij}) P_i$$

Using the row polynomial D_j with a root at $-Q_j$, a triple root at $-Q_{n+1}$ and $D_j((d_{ij} + \omega e_{ij})P_i)$. The final row for $Q_0 = Q$ outputs the negated result. Note that the negative base ensures that the result from previous can be used directly. Using a non-negative base would require that each polynomial be evaluated at alternating inverse challenge points to achieve cancellation.

The polynomials will be canonically normalized so that the constant term is 1. At least one term must be provably non-zero, and this allows more flexibility in the use of rows of secret length, although this is not necessary for the public scalar case. Following the commitment to the divisor witnesses, the verifier will select the challenge point A_0 and the prover will respond with all the rational functions for each row and for each basis point. In total, this is $2l$ for the rows and $10n$ for n points, as two of the basis points are also functions. Note that each of the numerators and denominators of these evaluations is a linear combination of the divisor witnesses and the point witnesses. The resulting arithmetic circuit just needs to check these multiplications and one weighted sum of all the evaluations. The weights from from the point derivates and the multiplicities.

2.8.2 Private Scalars, Public Points

The private scalar, public point case is also used once for the linear Bulletproof verification. It uses the same basic structure as the public scalar case but with different trade offs. The prover does not need to evaluate the point terms in zero knowledge, as the basis point is public, but must commit to the multiplicities along with the circuit and needs an addition for every digit. This allows the prover to use a different basis consisting of the $0, 1, -1$ linear combinations of the values $3^{54n}\mathrm{P}$ where $n = 0, 1, 2$. This consists of 26 points, so the total size of the witness is $26 + 54 = 80$ scalars per multiplication.

Since the structure of the proof is the same, the prover can reuse the spine of -3 multiplications and prove that a public linear combination of private points by public scalars and a private linear combination of public points yields a particular value. It is also often possible to commit to the multiplicities of the secret scalar with functions, as they are not input to the random oracle for the selection of A_0. This is the case for the final step of the Bulletproof verification.

2.8.3 Variable Witness Sizing

The witness for these proofs is dominated by the divisor witnesses, which in the case of public scalar, private point multiplications are actually of variable size. The size is proportional to the number of non-zero digits, which while typically about 8/9 of the total number of digits can vary. As the number of digits increases the variance will diminish to zero, but for certain portions of the proof unusually large witnesses may not fit in the available space, which can be problematic. The probability of such a large witness is non-negligible, so it must be addressed.

In the final phase of the anonymous transaction protocol, which is the only place this problem arises for this protocol, the proof need not be perfectly hiding. This means it is acceptable for the prover to "grind" a small divisor witness by sequentially trying different transcripts. This removes less than one bit of security from the challenge space, subject to the assumption that digits are distributed independently at random, and is therefore sound. In cases where the commitments are to secret data, the prover could alternatively sample multiple challenges in parallel. The number of challenges can be tuned so the probability of all of them having large divisor witnesses is negligible. This is not necessary for this protocol.

2.8.4 Protocol

The ECIP protocol will take place over two rounds, one for the divisor witness and one for the evaluations, plus some number of rounds for the points. The exact number of point commitments will vary depending of the context, but will typically include at least one. During these rounds, the linear combination will also typically be constructed by the rest of the protocol in context. However, for the purposes of the protocol object, both the points and the linear combination can be treated as fixed. With this in mind, an ECIP object is defined by a vector of public scalars $\mathbf{v}$, a vector of secret points $\mathbf{P}$, a vector of secret scalars $\mathbf{w}$, a vector of public points $\mathbf{Q}$, and a secret result points S. The object will show that

$$\mathtt{ECIP}(\mathrm{S}, \mathbf{v}, \mathbf{P}, \mathbf{w}, \mathbf{Q}) \qquad \langle \mathbf{v}, \mathbf{P} \rangle + \langle \mathbf{w}, \mathbf{Q} \rangle = \mathrm{S}$$

In the case that either of the vectors in a product is zero, it may be dropped from the invocation. From this, the first witness vector is the vector of basis points. This includes the private points as well as the basis divisors such that

$$\mathbf{basis}(\mathrm{P}_i) = (x(\mathrm{P}_i), y(\mathrm{P}_i), \lambda_i, \mu_i) \text{ where } \mathrm{div}(y - \lambda_i x - \mu_i) = (\mathrm{P}_i) + (-\omega \mathrm{P}_i) + ((\omega - 1)\mathrm{P}_i) - 3(\mathrm{O})$$

$$\mathbf{pnts}(\texttt{ECIP}(\mathrm{S}, \mathbf{v}, \mathbf{P}, \mathbf{w}, \mathbf{Q})) = \bigoplus_i \mathbf{basis}(\mathrm{P}_i)$$

Note that this does not, by default, include the points S, which will typically be considered separately. One could add $-\mathrm{S}$ to the vector of points and treat the result as O instead. The second witness vector consists of the divisor witnesses, as well as a separate witness vector for the secret multiplicities when present.

$$\mathbf{divs}(\texttt{ECIP}(\mathrm{S}, \mathbf{v}, \mathbf{P}, \mathbf{w}, \mathbf{Q})) = \mathbf{mults} \oplus \bigoplus_{i=0}^{l} D_i \qquad \mathbf{mults}(\texttt{ECIP}(\mathrm{S}, \mathbf{v}, \mathbf{P}, \mathbf{w}, \mathbf{Q})) = \bigoplus_i \mathbf{mults}(w_i)$$

The divisor witnesses here are constructed in accordance with the Weil ECIP procedure, with each row represented as a list of point additions. Each is canonically normalized so that $D_i(0,0) = 1$, and therefore for n points the divisor will have n coefficients. The multiplicities are defined as before. The number of divisor rows depends on the parameter l which is function of the length of the challenges. If the challenges are drawn from the short distribution $l = 40$ and if they are drawn from the whole field $l = 80$.

Following commitment to all of the aforementioned witnesses, the verifier will select the challenge point A_0. The prover will then commit to the evaluations of all the rational functions. As functions of points and functions respectively the evaluations are

$$\mathbf{dv}(D) = \left(\frac{g'(\mathrm{A}_i)}{g(\mathrm{A}_i)} \frac{dx(\mathrm{A}_i)}{d\mu} : i = 0, 1, 2\right) \qquad pt(\mathrm{P}) = -\frac{\frac{d\lambda}{d\mu} x(\mathrm{P}) + 1}{y(\mathrm{P}) - \lambda x(\mathrm{P}) - \mu}$$

The point evaluations **pntevals** include $pt((-\omega)^i \mathrm{P})$ for all the basis points as well as $\mathbf{dv}(B)$ and $\mathbf{dv}(B(x, -y))$ for the basis points. The divisor evaluations include $\mathbf{dv}(D_i)$ evaluated at each row divisor.

$$\mathbf{pntevals} = \bigoplus_j \mathbf{dv}(B_j) \oplus \mathbf{dv}(B_j(x, -y)) \oplus (pt((-\omega)^i \mathrm{P}) : j) \qquad \mathbf{divevals} = \bigoplus_i \mathbf{dv}(D_i)$$

$$\mathbf{evals}(\texttt{ECIP}(\mathrm{S}, \mathbf{v}, \mathbf{P}, \mathbf{w}, \mathbf{Q})) = \mathbf{pntevals} \oplus \mathbf{divevals}$$

The protocol is now complete, up to the arithmetic circuit. This circuit verifies that the evaluations are of the correct form, as well as that the sum of all the evaluations appropriately scaled, as well as the multiplicities if present, sum to zero. All the numerators and denominators of the evaluations are expressible as linear combinations of the divisors or the points, so the circuit is actually quite simple. For simplicity, the function **errs** is also defined on `ECIP` objects and returns the PLI error terms, to be defined in the following section.

3 Circuits

Many zero knowledge proof systems, including Bulletproofs, are powerful enough to prove that an arithmetic circuit is satisfied in zero knowledge. An arithmetic circuit is analogous to a digital circuit, where the operations of exclusive or and binary conjunction are replaced with field addition and multiplication respectively. Arithmetic circuits are convenient for many zero knowledge proof systems, especially those based on the discrete log problem, as the space of witnesses for these proof systems is naturally sequences of field elements and field operations are the "primitive" operations of the proof systems. Other operations that might be efficient to perform directly, like binary operations, must be re-encoded into the field of the proof system.

While the anonymous transaction protocol of this paper is not a general purpose zero knowledge proof system, it does use many of the techniques necessary for such general proof systems. In particular, the anonymous transaction protocol consists of a large sequence of rounds, and in order to minimize the size of the witness and complexity of the verifier the prover will incrementally construct the proof. This allows portions of the proof to be proved, blinded, and subsequently shared with other untrusted provers. This is not especially different from the multiparty proving protocols of Bulletproof type protocols that have already been published, with the provers organized sequentially rather than in parallel.

Several of the protocols of this section are used only once, but for clarity of both the circuit representation protocol and the invoking protocol they will be described abstractly in this section. Unlike some other Bulletproof based protocols, the Bulletproof argument will be used to prove a linear relation, along with a single external multiplication, which requires carefully managing the number of error terms generated by the protocol. By proving different parts of the proof sequentially, the number of error terms can be minimized. To avoid excessive complexity, much of the circuit data like constraint matrices is passed implicitly in the protocol and the protocols instead focus on data layout and round structure.

3.1 Circuit Representations

There are many different, equivalent representations of arithmetic circuits, including straightline programs and systems of polynomial equations. The latter is, by the standard NP reduction, polynomially equivalent to a system of quadratic equations. The way many proof systems have elected to represent such proofs in practice is via Rank 1 Constraint Systems (R1CS) or some variation thereof. An R1CS consists of a witness vector $\mathbf{x}$, with the first element canonically set to the constant 1 publicly, and three constraint matrices encoding the circuit (A, B, C). The circuit is said to be satisfied if the componentwise products the matrices times the witness satisfy the relation

$$(A\mathbf{x})_i(B\mathbf{x})_i = (C\mathbf{x})_i$$

Bulletproofs use a slightly different, but equivalent representation. Given a triple of witness vectors $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and a triple of constraint matrices (A, B, C) the circuit is satisfied if the componentwise products of $\mathbf{x}$ and $\mathbf{y}$ equal $\mathbf{z}$, and the linear constraints of the matrices and the vectors sum to zero, or to a prescribed linear combination of the "inputs" to the circuit. In the latter case, the vectors $\mathbf{v}$ is also part of the witness and the additional constraint matrix D must also have a left inverse, in order to ensure witness extractibility.

$$x_i y_i = z_i \qquad A\mathbf{x} + B\mathbf{y} + C\mathbf{z} = D\mathbf{v}$$

In both cases, the arithmetic circuit encoded by the matrices is treated as fixed, although this need not be the case in context. Since the matrices are generated in public by the verifier, they can involve arbitrary dependencies on challenge values or other public information. Apart from the matrix D, this has little bearing on the zero knowledge and witness extraction properties of the underlying proof system, and it is merely necessary to show that the circuit encoded by the matrices is sound with respect to the statement to be proved. One simple example of this is the two phase witness and circuit construction procedure used by Bulletproofs++ for the reciprocal argument.

3.2 Error Terms

When proving that multiple multiplications are satisfied, as is necessary in the R1CS and Bulletproof circuit representations, proof systems typically attempt to reduce multiple multiplications into a single multiplication. In doing so, the prover introduces "error terms" that are necessary to prove that the multiplications are satisfied, but that do not actually encode any useful information about the multiplications directly.

There are two methods for reducing the number of multiplications that will be used in this paper: polar lagrange interpolation and convolution. The first is a slight variation on the more common Lagrange interpolation, which is used by many SNARK protocols. To show that a number of multiplicative constraints are satisfied, the prover constructs polynomials such that evaluating the polynomials at certain, prescribed points yields the values. Then, the prover shows that the relation between the polynomials holds modulo a polynomial with roots at the prescribed points. This method produces one error value per multiplication.

The second method encodes the values as the coefficients of polynomials. If the coefficients proceed in opposite directions, of equivalently if the polynomials are evaluated at reciprocal points, then the central coefficient encodes the inner product of the coefficients of the two polynomials. This inner product can be efficiently augmented to scale each product by a random value, so that each multiplicative constraint can be considered seperately as a consequence of the Schwartz-Zippel lemma. This method produces two error terms for each additional multiplication, but can be augmented by a generalization of the Bulletproof technique to introduce more rounds, and reduce the number of error terms exponentially in the number of rounds.

3.2.1 Polar Lagrange Interpolation

The first method of reducing multiple products encodes the vectors as the residues of rational functions. I will use scalar multiplications to demonstrate for simplicity, although the techniques are applicable to any bilinear operation. Given the vectors **a**, **b**, and **c** the prover wants to show

$$a_i b_i = c_i$$

To do so, consider the rational functions for each each

$$v(e) = \sum_{i=1}^{n} \frac{v_{i-1}}{e+i} \qquad v = a, b, c$$

The inner product equalities hold if and only if there exists a rational function $q(e)$ with poles of order one satisfying

$$a(e)b(e) - c(e)\frac{m'(e)}{m(e)} = q(e) = \sum_{i=1}^{n} \frac{q_i}{e+i}$$

Where $m(-i) = 0$ is the polynomial with roots at the poles. Equivalently

$$\frac{m'(e)}{m(e)} = \sum_{i=1} \frac{1}{e+i}$$

Soundness follows from the expansion of the multiplications and the simplification of the product of two poles at different values via partial fractions. That is by repeated application when $i \neq j$ of

$$\frac{1}{(e+i)(e+j)} = \frac{1}{j-i}\left(\frac{1}{e+i} - \frac{1}{e+j}\right)$$

This moves all the cross terms to the poles of order one, leaving the desired product constraints on the poles of order two. If these poles of order two do not vanish, then it is not possible to construct a $q(e)$ satisfying the equality with poles of order one.

Perhaps unsurprisingly, this shares certain similarlities with PLONK as well as KZG [51] commitments and other protocols that make use of Lagrange bases. While, I don't believe anyone has used the partial fractions trick directly, it is in a certain sense implicit in many uses of Lagrange polynomials. One advantage of simply using reciprocals is that one can do so without ever specifying directly what the roots are. Often the roots are chosen to be roots of unity for efficiency's sake, but in the case of a protocol like Bulletproofs++ a similar construction occurs where the roots are secret. It may be possible to combine the products of poles technique with the permutation argument of Bulletproofs++ to, for example, encode arithmetic circuits.

3.2.2 Equivalence to Lagrange

To see the equivalence to Lagrange interpolation, notice that the Lagrange basis polynomials are

$$L_i(e) = \prod_{j \neq i} \frac{e+j}{-i+j} = \frac{m(e)}{(e+i)m'(-i)}$$

These polynomials satisfy

$$L_i(e)^2 = L_i(e) \mod m(e) \qquad L_i(e)L_j(e) = 0 \mod m(e)$$

Then, the multiplications $a_i b_i = c_i$ can first be transformed into the multiplications $a'_i b'_i = c'_i m'(1-i)$ where each value is multiplied by $m'(1-i)$, so that for example $a'_i = a_{i-1}m'(1-i)$. These multiplications can be encoded as

$$\left(\sum_{i=1}^{n} a'_i L_i(e)\right)\left(\sum_{i=1}^{n} b'_i L_i(e)\right) - \left(\sum_{i=1}^{n} c'_i L_i(e)\right) m'(e) = m(e)q(e)$$

Dividing the entire equation through by $m(e)^2$ yields the polar interpolation, since the $m'(i)$ factors cancel from the Lagrange polynomials, leaving only the reciprocal monomials.

$$\left(\sum_{i=1}^{n} \frac{a_{i-1}}{e+i}\right)\left(\sum_{i=1}^{n} \frac{b_{i-1}}{e+i}\right) - \left(\sum_{i=1}^{n} \frac{c_{i-1}}{e+i}\right)\frac{m'(e)}{m(e)} = \frac{q(e)}{m(e)}$$

Clearly, the right hand side cannot have any poles of order 2, and it can be expanded by partial fractions to write as a sum of reciprocals as well. Notice that, since the left hand side has degree at most $2n$, the polynomial $q(e)$ cannot have degree greater than n.

3.2.3 Convolution

To reduce the products using convolution, each vector will be encoded as the coefficient of a polynomial like

$$v(x) = \sum_{i=1}^{n} v_{i-1}x^i \qquad v = a, b, c$$

As well as for the other vectors. The prover will choose a random value q and let the scalar valued polynomial

$$p(x) = a(qx)b(q/x) - c(q^2)$$

The central coefficient of this polynomial encodes a random linear combination, by the powers of q^2, of the multiplicative constraints

$$\left[x^0\right] p(x) = \sum_{i=0}^{n-1} q^{2i+2}(a_i b_i - c_i)$$

To show that this equality is satisfied, after selecting q, the prover will commit to all the other terms p and the verifier will then select a challenge value for x. This requires two rounds and committing to $2(n-1)$ error terms. However, the q round can be dropped in situations where the prover wants to show an inner product relation like

$$\langle \mathbf{a}, \mathbf{b} \rangle = v$$

3.2.4 Generalized Bulletproof Reduction

Using a single x challenge, the convolution technique has more error terms than PLI. Using a technique similar to Bulletproofs, the prover can reduce the number of terms by a factor of k at a cost of $2(k-1)$ additional terms and one more round. Applied multiple times, the number of error terms can be made logarithmic in a logarithmic number of rounds. To see how, replace each monomial x^{ki+j} in $a(x), b(x), c(x)$ with $y^i z^j$ where $0 \leq j < k$. The resulting polynomial $p(y, z)$ has as its y constant term

$$\left[y^0\right] p(y, z) = v + l(1/z) + r(z)$$

Where v is the constant terms of $p(x)$ and the polynomials l and r do not depend on y. If the prover commits to the coefficients of l and r, then the verifier can choose z_0, and the prover can show that for $p'(y) = p(y, z_0)$

$$\left[y^0\right] p'(y) = v + l(1/z_0) + r(z_0)$$

The polynomial p' has degree n/k and the polynomials l and r have degrees $(k-1)$. If this relation holds, it must also be the case that the $p(x)$ condition holds with overwhelming probability. The procedure can be repeated on $p'(y)$ with a new challenge. Selecting $k = 2$ yields the argument structure of a Bulletproof, but inside a proof using larger k and fewer rounds is often preferable.

3.3 Extending Bulletproof Circuits

The proofs of this paper will use the linear argument for Bulletproofs, so it is necessary to represent the outer most arithmetic circuit as a R1CS, instead of as a large hadamard operation typical of other Bulletproof protocols. This will use three matrices (A, B, C) to encode linear combinations of the witness into multiplicative constraints. The prover will show

$$(A\mathbf{x})_i(B\mathbf{x})_i = (C\mathbf{x})_i$$

This requires committing to error terms associated with the transformation of this relation into a single product of inner products. The error term technique will vary according to the circumstances in which the protocol is instantiated. However, for the purposes of this general discussion, it is assumed that the reduction has already occurred and the prover has a collection of witnesses $\mathbf{x}_i = \mathbf{wit}\left(\mathrm{C}_i\right)$, and a collection of constraint vectors $\mathbf{a}_i$, $\mathbf{b}_i$, $\mathbf{c}_i$, and $\mathbf{q}_i$. These must be shown to satisfy

$$\left(\sum_i \langle \mathbf{a}_i, \mathbf{x}_i \rangle\right)\left(\sum_i \langle \mathbf{b}_i, \mathbf{x}_i \rangle\right) - \sum_i \langle \mathbf{c}_i + \mathbf{q}_i, \mathbf{x}_i \rangle = 0$$

Note that $\mathbf{c}_i$ and $\mathbf{q}_i$ for the same witness can be combined, but I will write them separately for clarity, as $\mathbf{q}_i$ is the linear combination of error terms. To show this, assume the constraint vectors are fixed, the prover will compute the quantities a and b as the linear combinations of the above inner products. For the purposes of this section, these values are assumed to be independent of any secret information. However, rather than publishing them, the prover will commit to the values $\alpha = a^2$, $\beta = b^2$ such that

$$(a + bt)^2 = \alpha + (c + q)t + \beta t^2$$

Along with these, the prover will use PLI to prove all the inner products. Each inner product will be assigned a pole, and the prover will commit to all the individual products and error terms. Following this commitment, the verifier will select a random value for t and the prover will publish

$$v = a + bt$$

The PLI values and error terms must be committed on their own basis elements, as this saves a round. These values will be randomly scaled by a value chosen after all the $\mathbf{x}_i$ commitments to prevent interference. Then, the verifier will choose the PLI challenge, which combines all the commitments and all the constraints. The final remaining witness has three constraints, one for the PLI, one for v as the linear combination of all the products making up a and b appropriately scaled, and one for v^2 in terms of α and β. The verifier will choose a random linear combination of these constraints and the prover will then engage the linear argument. Verification consists of finding the appropriate constraint, using v and v^2, and then verifying the linear argument.

This general encoding of the circuits saves a round by committing to the final error terms on their own basis elements and only requires the publication of the scalar v. It is important that there not be an excessively large number of commitments combined in the final stage, as each commitment adds two reserved terms, which reduces the available witness space in earlier commitments. This behavior will be encapsulated into `Circ` with properties encoding the the basis points, the error terms, and the published scalar. The constraint vectors for the input commitments should be inferred from context.

$$\mathbf{Points}(\mathtt{Circ}(\mathbf{P})) = \mathbf{P} \qquad \mathbf{errs}(\mathtt{Circ}) \qquad v(\mathtt{Circ})$$

3.3.1 Witness Extraction

In this procedure, all the witnesses are combined in one step using PLI, and the error terms are also combined on a distinct basis. It is sufficient to run the blinding step twice in the emulator, which will return the openings of the PLI linear combination and the error terms with overwhelming probability. Then, querying the oracle once per commitment will yield a matrix of rational functions evaluated at distinct challenge points with overwhelming probability. This determinant over the rational field in all the challenges is not degenerate so long as the functions in the same row are not linearly dependent over the base field. As a consequence of the Schwartz-Zippel lemma over rational functions, the determinant does not vanish or diverge with overwhelming probability. Therefore, the emulator can invert the matrix and extract the witness with overwhelming probability.

$$M_{ij} = f_j(e_i) \qquad \nexists \mathbf{c} \in F^k. \quad \langle \mathbf{c}, \mathbf{f}(e) \rangle = 0$$

$$\det M \in F(e_0, ..e_k) \qquad \det M \neq 0$$

3.4 Incremental Construction

The same Bulletproof reduction technique that was originally applied to witness vectors can also be applied to constraint matrices in the R1CS setting in the following way. Suppose that the verifier has already selected q and the prover effectively wants to show for q scaled constraint matrices (A, B, C) that

$$\langle A\mathbf{x}, B\mathbf{x} \rangle - \langle C\mathbf{x}, \mathbf{1} \rangle = 0$$

The polynomial version of this is obtained by instead left multiplying each matrix, vector product by an appropriate power vector

$$b(t)_i = t^i \qquad p(t) = \left\langle \mathbf{b}(t)^\top A_n \mathbf{x}, \mathbf{b}(1/t)^\top B_n \mathbf{x} \right\rangle - \langle C_n \mathbf{x}, \mathbf{1} \rangle$$

Applying the Bulletproof reduction to this polynomial has the effect of collapsing every k adjacent rows of the matrices and adding some new error polynomials l and r that account for the error terms. Repeatedly applying this transformation produces for challenges $e_0, ..., e_n$ the relation

$$p_n(t) = \left\langle \mathbf{v}(t)^\top A_n \mathbf{x}, \mathbf{v}(1/t)^\top B_n \mathbf{x} \right\rangle - \langle C_n \mathbf{x}, \mathbf{1} \rangle + \sum_{i=0}^{n} l_i(1/e_i) + r_i(e_i)$$

Where the $l_i(0) = r_i(0) = 0$ as before. Note that the witness vectors are left completely unchanged. In the setting where prover wants to prove a series of circuits are satisfied, this procedure can be augmented to append the new witnesses and new circuit matrices onto the existing witness and existing matrices. Assuming that the new matrices have already been transformed into the inner product form and that their central coefficient is a linear combination by non-zero q powers, it is safe to simply append the new circuit matrices and the new witness vector row wise onto the existing matrices.

If the prover applies the Bulletproof reduction technique once again to the rows of the new matrix, it will reduce the old matrix by a factor of k and the new matrix by a factor of k. This means that as new circuits are added, and subsequently reduced, the number of constraints is always constant and given by a geometric sum of inverse k powers multiplied by the size of the circuits. If the circuit size and k values change per round, then the size can vary, but the asymptotic behavior remains the same assuming all the circuits are of bounded size and $k > 1$ for each round. Note that the l_i and r_i coefficients can be included in the witness for the next circuit and encoded using a matrix Q.

Future circuits can operate over historical witness vectors, but if the "look back" is bounded then at some point the matrices for a circuit will be fully reduced to single vectors. At that point, the prover can commit to the value of these constraint vectors applied to their witness in the next circuit, add these values to the next $a(t)$ and $b(t)$ matrices, and add linear constraints that a random linear combination of the residual constraint vectors equals the same random linear combination of the new committed values. This has the effect of reducing the number of constraints for a "closed" witness to one, which will reduce the number of error and value terms in the final proof.

This procedure is used as part of the nested Bulletproof phase. Since the Bulletproof argument is structure as a sequence of challenge-response interactions, and each one produces an ECIP relation to prove, the zero knowledge proof naturally forms as a sequence of mostly independent circuits. Each ECIP is itself spread over multiple rounds, so each commitment will commit to a different round of a different

ECIP protocol. After the ECIP protocol is complete, and the verifier has selected the q to randomize the resulting circuit, this protocol will be used to combine these circuits and manage the total number of error terms. This helps proving time and allows using a smaller commitment witness size than would be possible if all the error terms for each circuit were directly committed.

The protocol object `IncErrs` encapsulates this behavior, accepting optionally a previous incremental object and a collection of circuits to add the error terms for. This object requires a challenge q and e as well as a reduction factor. If the circuit is computing an inner product, the q is not necessary. The function **errs** returns $2(n-1)$ error terms. If no new circuits are added, sequential invocations of $\mathbf{errs}_i$ can also be applied and multiple rounds of multiplicities can be passed at once.

$$\texttt{IncErrs}(\texttt{Prev}, \texttt{C}, n) \qquad \mathbf{errs}(\texttt{IncErrs})$$

3.4.1 Witness Extraction

This portion of the proof does not need to be zero knowledge. To ensure witness extraction, it is sufficient to observe that all of the commitments' witnesses from this portion of the proof are open going into the next phase. In that phase the prover will use the R1CS procedure described before, which has already been shown to a have a witness extractor.

3.5 Multiple Parallel Provers

There is another circuit representation used in the membership phase that is worth considering explicitly. Here, a collection of mutually distrusting provers have a set of circuits applied to disjoint witnesses and want to show that all the circuits are satisfied without revealing their witnesses. A complicating factor is that the witnesses are spread across multiple commitments in some cases, and these commitments may include witnesses from multiple provers. This is done to save space. Finally, the provers would like to complete this as quickly, measured in verifier interactions, as possible to minimize proof size.

Firstly, each circuit includes an ECIP, so the PLI error terms will be committed along with the divisor evaluations. These error terms will be calculated on a per circuit basis, as distinct provers cannot combine their multiplicative constraints. Each witness chunk is defined to be a contiguous piece of a circuit witness in a single commitment, and each constraint fragment per chunk will have a blinding scalar. This blinding scalar will be incorporated into the blinding constraint for that matrix. This causes the application of each constraint to each fragment to be distributed uniformly at random.

Note that the provers also want to be able to prover linear constraints across their witnesses privately, so these constraints will share a pole across circuits. This requires that the circuits have distinct poles for their multiplicative constraints. The verifier will select the PLI challenge, and this will make all the constraint matrices into constraint vectors per fragment. The prover will commit to two blinding vectors supported over the entire witness space for the commitments.

Then, the provers will conduct another round of PLI, but this time between the constraint vectors and the witnesses. Each witness fragment constraint multiplication will be assigned a pole, and the blinding vectors will also be assigned poles with no associated constraints. The resulting values from this are computable without revealing the witnesses. Each pole will produce two terms, the blinded fragment product and an error term. The error term will be blinded by the constraint applied to the blinding vectors.

In order for the blinding to behave correctly, i.e. to actually blind the witness, each fragment constraint vector needs to be linearly independent over the vector space of commitment witness. If the fragment constraints are not linearly independent, then the given the blinded error terms it is possible to compute a linear combination of the terms such that the blinding cancels, revealing a linear combination of the unblinded witnesses.

Assuming the constraints are linearly independent, this procedure is blinding by the following reasoning. Following the PLI constraint procedure, each component of the opening is given as a linear combination of

$$\frac{b_{0,i}}{e} + \frac{b_{1,i}}{e+1} + \sum_j w_{i,j} f_j(e)$$

Where $f_j(e)$ is the sum of reciprocals for the constraints applied to witness vector j. The error terms are then given by the application of the constraint vector k applied to all the witnesses and the blinding vectors as

$$\epsilon_k = \frac{\langle \mathbf{b}_0, \mathbf{c}_k \rangle}{-k} + \frac{\langle \mathbf{b}_1, \mathbf{c}_k \rangle}{1-k} + \sum_j \left[\frac{1}{e+k} \right] \frac{f_j(e)}{e+k} \langle \mathbf{w}_j, \mathbf{c}_k \rangle$$

These quantities are given by a certain primitive witness independent of the blinding values plus a linear function of the blinding values. This linear function is given by a matrix that has a left null space of dimension at least one, given by the verification equation, and exactly one if all the constraints $\mathbf{c}_k$ are linearly independent. This follows because the matrix itself is formed by two concatenated diagonal matrices with entries $1/e$ and $1/(e+1)$ and the rows given by vectors $-1/k\mathbf{c}_k \oplus 1/(1-k)\mathbf{c}_k$. Any element of the right null space taken as a funtion of e gives

$$e\sum_k \frac{-b_k(e)}{k}\mathbf{c}_k = \mathbf{v}(e) = (e+1)\sum_k \frac{b_k(e)}{1-k}\mathbf{c}_l$$

Equivalently

$$\sum_k \left(\frac{-e}{k} - \frac{e+1}{k-1}\right) b_k(e)\mathbf{c}_k = 0$$

Apart from the actual constraints, any other solution yields a linear relation between the $\mathbf{c}_k$. From this point, the simulator follows straightforwardly. The adversary will choose a uniformly random set of valid scalars and a uniformly random set of commitments except for one of the blinding commitments. The final blinding commitment is given uniquely in terms of the others according to the verification equation and so the protocol is zero knowledge.

3.5.1 Ensuring Linear Independence

To ensure that this is the case, first consider the fragment constraint vectors as vectors over the vector space of rational functions in the PLI challenge. If these vectors are linearly independent over the field of rational functions, then it follows that their evaluation at a random PLI challenge yields a set of linearly independent vectors with overwhelming probability. To see why, take the transpose of the matrix and multiply along the larger dimension and take the determinant. If the determinant of this matrix is not identically zero, then the rows of the matrix formed by the constraints vectors are linearly independent over the field of rational functions. Since the determinant is a rational function, the probability that it vanishes at a random challenge point negligible by the Schwartz-Zippel lemma.

While the prover and verifier could actually check that the constraints are linearly independent, this would be unacceptably computationally expensive. Instead, the prover will arrange each constraint so that the fragment blinding components lie on the diagonal. It is sufficient to ensure that they lie on distinct components for each fragment. This is always possible since the organization of witnesses within a single commitment can be freely permuted without affecting the rest of the protocol, so long as the fragment constraints are also inversely permuted. Then, treat the blinding constraint monomial as a formal variable t and take the determinant as before.

The result is a polynomial in the PLI challenge and t. This polynomial cannot vanish since the vectors are always linearly independent over $F(e,t)$. This polynomial has a finite number of roots where $t \in F(e)$ causes the determinant to vanish. Many of these roots will not actually lie in the rational field, but among those that do there cannot be more $2c$ where c is then number of constraints, as this is the degree in the polynomial in t. Choosing the monomial where $t = 1/(e+r)$ at random with respect to r will select one of these candidate roots with negligible probability. Therefore, choosing the blinding monomial at random will produce a non-degenerate determinant with overwhelming probability.

The protocol will also call for "accessor" fragments that return an unblinded component of the witness among the fragments. To show that these constraints do not cause the matrix to degenerate, the prover will organize the accessors to also be along the diagonal, which ensures that the diagonal blinding components are disjoint from the accessors. These accessors for an orthogonal basis for the components that they lie over. It is sufficient therefore to show that the span of other rows, truncated to not include these components where the accessors are supported, does not include the zero vector in order to show that the rest of the constraints are linearly independent of the accessors. This immediately follows from the same argument applied to the truncated constraints vectors, which notably contains the blinding components along the diagonal. If the accessor components do not "fit" on the diagonal, since the number of constraints is too large, then this argument fails and the constraints are not linearly independent. This argument applies equally well to any linearly independent set of accessors over a subset of components and could probably be generalized to small dimensional linear subspaces.

3.5.2 Protocol Object

This will all be encoded in the `ParErrs` protocol object, which will accept as arguments the two blinding commitments and all the commitments to the input circuits. The vector **poles** will be all the poles from the second PLI round to be applied to each input point, the vector of which will be denoted **Points**. The vector of fragments and error terms will be denoted **frags**.

$$\mathbf{Points}(\texttt{ParErrs}(\mathbf{P})) = \mathbf{P} \qquad \mathbf{poles}(\texttt{ParErrs}) \qquad \mathbf{frags}(\texttt{ParErrs})$$

3.6 Blinding

The Bulletproofs++ blinding procedure will be used in the first phase of the proof for the confidential transaction protocol. Recall that in that case, the prover has a collection of commitments and constraints, as before, but also want to blind the final linear combination. When the prover commits to the error terms, they will also commit to the blinding for the proof and arrange to have this commitment by of order zero. Since each other commitment is scaled by a power monomial and each error term component is scaled by an error term monomial, the prover can set the error term components of earlier commitments, when they do not map to the value term, and subtract the result from the actual error term in the blinding commitment.

This turns out for more than 2 commitments to be sufficient to blind the proof. In order to prevent interference in the value term, the error term components will be randomly scaled by a value selected before the blinding commitment. Unfortunately, this only works for convolution of the constraints and commitments and does not obviously generalize to the PLI case. In the PLI case, the blinding commitment would need to be able to set the products directly, and it is not obvious how one would prevent the earlier terms from interfering while also blinding the values. If the values do not need to be blinded, then there it is sufficient to simply commit to the values as in the above protocol.

3.6.1 Preventing Blinding

In certain portions of the proof, specifically in conjunction with the Nullifier protocol, it will be necessary for the prover to provably not be able to blind certain commitments. This is awkward, since blinding is typically intentionally designed to be invisible to the proof, and checking that it satisfies particular properties is difficult. Especially so if the same mechanism for blinding will be used in the rest of the proof.

This will be accomplished with respect to the Nullifiers in the following manner. There will be three blinding components H_i for $i = 0, 1, 2$ in the proof dealing with the openings of the nullifiers. The first blinding component is used for all the commitments before and when the nullifiers are committed. Following commitment to the nullifiers, it will become legible to the proof and the proof will check that it is zero as part of the membership proof where the nullifier equation is verified.

Following commitment to the nullifiers, the proof will choose a random linear combination H'_1 and H'_2 of the other two blinding components. The first of these is the blinding component for the rest of the proof and the second is a "check" component that must be identically zero in all commitments. Any commitment from before the random linear combination was chosen will, with overwhelming probability, have a non-zero H'_2 scalar. Conveniently, the proof that all H'_2 terms vanish is quite simple: given that it does not appear in the final Bulletproof opening and given a witness extractor for the proof, it follows as a direct corollary that the term is zero in every commitment. An alternative heuristic way to think about this is that every commitment is "randomly" linearly combined with the rest of the commitments so it is not possible to compensate for earlier commitments, specifically the nullifiers, that are not zero.

4 Transaction Body

The first phase of the anonymous transaction protocol deals with the confidential portion of the transaction. This includes the range proofs for transaction outputs as well as the typed conservation of money proof for the amounts. It also includes the nullifier protocol, which also acts as a signature protocol, and includes timelocks. Along with adaptor signatures on the nullifiers, the timelocks are sufficient to support atomic swaps and other scriptless scripts. In order to for this to work, it must be possible to complete the transaction body protocol, in the sense that knowledge of the confidential information is not necessary to produce a valid transaction from a complete execution of the transaction body protocol, without knowledge of the membership witnesses.

The primary transaction body protocol consists of modifications to the confidential transaction protocol of Bulletproofs++. Specifically the base 16, 64 bit range proofs and typed conservation of money. The timelocks also use these range proofs. The primary difference form Bulletproofs++ confidential transactions is how the commitments for transaction inputs, as the commitment values themselves must be kept secret to preserve the mutual privacy of participants in the same transaction. The protocol is also extended to support multiple commitments worth of range proof data, as the commitment length cannot be freely extended as in the original Bulletproofs++ protocol without significantly affecting other parts of the proof.

4.1 Nullifier Protocol

A nullifier is a value published along with the transaction that consumes a particular input commitment. This value must be deterministically generated from the input commitment, and allows the verifier to check that the input has not yet been spent by a different transaction. The nullifier must also be unlinkable with the original input, even by the creator of the input, as it would otherwise be feasible for the sender of a payment to detect when the recipient ultimately spent it later. Most importantly, it must be efficiently possible to prove that the nullifier was correctly constructed in zero knowledge, and this should be possible for multiple provers to do together for multiparty TXOs.

The simplest protocol that supports all of these requirements is essentially a generalization of the protocol used by Monero. Given a nullifier base B and a nullifier key $\mathrm{K} = x\mathrm{G}$, the prover will generate the nullifier value $\mathrm{N} = x\mathrm{K}$. In monero the value N is called a key image, but it serves the same purpose as a nullifier. Another important difference between this nullifier protocol and Monero is how the base is generated. In Monero, it is publicly computed using a random oracle hash function, whereas in this protocol the structure of B is completely unconstrained and transaction participants are free to use any protocol they like to generate this value. This makes the nullifier protocol must simpler to use, especially for anonymous transaction, but requires care to avoid certain kinds of attacks.

The properties of this nullifier protocol follow naturally from standard discrete log assumptions. Given K and B, computing N reduces to the Computational Diffie-Hellman problem (CDH). Deciding for a particular transaction and nullifier, or equivalently the triple $(\mathrm{K}, \mathrm{B}, \mathrm{N})$ whether the nullifier come from that transaction reduces directly to the Decision Diffie-Hellman problem (DDH). Finally, proving that the nullifier is well formed via a proof of discrete log equality reduces to the Discrete Log problem itself in the random oracle model. This sort of proof, which is essentially just a Schnorr signature over two bases simultaneously is amenable to multiparty proving using a variety of multiparty Schnorr signature protocols [52].

4.1.1 Adversarial Collisions

Unfortunately, the multiparty case does have a problem. If one of the owners of an output with nullifier N learns this value, as they inevitably will when spending the output, they can create a new TXO will nullifier key $\mathrm{K}' = (1/y)\mathrm{G}$ and nullifier base $y\mathrm{B}$, which will yield the same nullifier value. If the adversary is able to spend this output first, the original output will become unspendable.

To avoid this problem, every nullifier will be tweaked by a randomizer derived from the minting transaction. After the prover has committed to all the transaction inputs, transaction outputs, and witnesses for the Bulletproofs++ confidential transaction protocol, the verifier will be give a randomizer $\mathrm{R} = r\mathrm{G}$ by randomly selecting r. This randomizer will be stored in the transaction output tree, and any transaction spending an output will need to add the randomizer to the transaction base before producing the nullifier $\mathrm{N} = x(\mathrm{B} + \mathrm{R})$.

Since r is selected after committing to the transaction body, but before engaging the rest of the transaction protocol, it is possible to compute the correct r value for a transaction output that has not been published. It also has the effect of requiring that a transaction spending an output commit to spending an output from a particular transaction, as the same commitment appearing as the output of a different transaction will have a different randomizer with overwhelming probability. This allows the construction of arbitrarily complex graphs of transactions spending outputs from other transactions without ever needing to publish any of them, which is useful for scriptless scripts.

4.1.2 Security Argument

An adversary attempting to construct an output with a prescribed nullifier in a transaction other than the one which produces the original nullifier would need to anticipate the randomly selected r value. In the random oracle model, this is impossible by assumption, so the adversary cannot do better than randomly trying different transactions, which has negligible probability of success.

For two outputs of the same transaction, the situation is somewhat subtler but still secure even using the same randomizer. Assuming the adversary does not know the secret key x, as such knowledge would allow them to steal the output directly, the adversary must choose a value of y and nullifier base such that

$$x(\mathrm{B} + r\mathrm{G}) = y(\mathrm{B}' + r\mathrm{G})$$

$$x\mathrm{B} - y\mathrm{B}' = (y - x)r\mathrm{G}$$

Either $x = y$ and $\mathrm{B} = \mathrm{B}'$ in which case and r produces the same nullifier, but which is impossible by assumption, or the verifier must choose exactly the discrete log with respect to H of $(y-x)^{-1}(x\mathrm{B} - y\mathrm{B}')$, which is fixed prior to the choice of r. Assuming the r is chosen honestly at random, the probability of this is negligible. Even in the general case with multiple adversarial outputs and multiple target outputs, the probability is only proportional to the product of the number of adversarial and number target outputs, which is negligible.

4.1.3 Protocol

As a consequence of the structure of the membership proofs, the verification protocol for the nullifier is a little complex. The provers will commit to the randomized nullifier bases B'_i and nullifier keys K_i, as well as a collection of nonces $\mathrm{Bl}(\mathrm{N}_i) = s_i\mathrm{B}'_i$ and $\mathrm{Bl}(\mathrm{K}_i) = t_i\mathrm{G} + u_i\mathrm{H}$. Assuming that the prover has shown the nullifier base and key are the same as those in the input, the prover must now show that the key and nullifier have the same discrete log.

As a consequence of the structure of the membership proofs, it is more efficient to perform this discrete log equality proof indirectly. Notice that the nonce secrets are different for the key and base blinding. The provers will commit to these nonce secrets separately, along with the nullifier secret. Then, the prover will construct two proofs of knowledge for the nullifier and the nullifier key with different challenges. Finally, the prover will show that the same PoK equations hold for the seperately committed nonce secrets and the nullifier private key. Given that all of this holds, the nullifier and nullifier key have the same discrete log with respect to their bases.

Along with the nonces, the prover will commit to the nonce secrets and the nullifier private key again separately in a form legible to a Bulletproof. Then, the verifier will choose the nullifier challenge value $e(\mathrm{N}_i)$ and the provers will respond publicly with the scalars $r(\mathrm{N}_i)$ such that

$$\mathrm{Bl}(\mathrm{N}_i) + e(\mathrm{N}_i)\mathrm{B}_i = r(\mathrm{N}_i)\mathrm{N}_i$$

Then, the verifier will choose the nullifier key challenge $e(\mathrm{K}_i)$ and the prover will commit, but not directly publish, the responses $r(\mathrm{K}_i)$ such that

$$\mathrm{Bl}(\mathrm{K}_i) + e(\mathrm{K}_i)\mathrm{K}_i = r(\mathrm{K}_i)\mathrm{G} + u_i\mathrm{H}$$

This commitment can be done in parallel with other parts of the protocol, as will be elaborated later. The prover must finally show that the linear equations over the committed nonce secrets are satisfied for these response. By structuring the challenges and responses in sequence like this, it becomes possible for the membership provers later to simply check that

$$\sum_i e(\mathrm{K}_i)\left(\mathrm{Bl}(\mathrm{N}_i) + e(\mathrm{N}_i)\mathrm{B}_i - r(\mathrm{N}_i)\mathrm{N}_i\right) = \mathrm{O}$$

Notice also that the challenge values can be drawn from the short distribution and that, without knowledge of u_i, the value $\mathrm{Bl}(\mathrm{K}_i)$ can be published to the other provers without revealing the nullifier public key.

4.2 Timelocks

Time locks will be provided for every input, so as to make atomic swaps and payment channels completely indistinguishable from ordinary transactions. Each output will commit to a time t_1 and a flag f_t in the output. The flag will regulate whether the time lock is relative or absolute. The miner will, along with the randomizer for the nullifiers of the transaction, set the time that the block was mined to be t_0. Note that these times could be measured in clock time or block time, although the precision of clock time will be limited be the speed of the consensus algorithm.

For the inputs to the transaction, the prover will agree on the current time that the transaction was constructed t. Each input must show for it's timelock tuple (t_0, t_1, f_t) that

$$t > t_1 + f_t t_0$$

If the flag f_t is set to 1, then this behaves like a relative timelock with t_1 being the offset. If the flag is set to 0, then this behaves like an absolute timelock and t_1 an offset from zero. In either case, the provers can show that this inequality holds using a range proof. In this way, every input has a range proof, for its timelock, and each output has a range proof, for its amount. For outputs that do not need a timelock, setting $t_1 = f_t = 0$ will render the output always spendable.

4.3 Adaptor Signatures

Adaptor signatures are a protocol that can be integrated, without any special modifications, into Schnorr signatures that allow one party to exchange a discrete log relation for a signature with another. This works by having Alice construct a Schnorr signature with a nonce offset T. That is, a signature

$$\mathrm{R} + e\mathrm{P} = r\mathrm{G} \qquad e = h(R + T)$$

This is not a valid signature, but given a valid signature on the nonce $R + T$, another party Bob can easily learn the discrete log of T with respect to G. If this signature is necessary to, for example, complete a transaction on a blockchain it is possible to structure a higher order protocol to pay one party and publish a secret to another party. This is the essential structure of an atomic swap.

Since the communication of the adaptor signature occurs privately, the publication of the regular signature does not reveal any information about the existence of the adaptor signature or higher order protocol. Interestingly, adaptor signatures also possess a kind of deniability, since anyone can construct an adaptor signature from a valid signature for an offset whose discrete log they already know.

The nullifier protocol already has the structure of a Schnorr signature, and involves the publication of the value r. This allows the use of adaptor signatures. There is a slight difference, in that each offset must be defined with respect to a different base. Certain protocols may require the ability perform addition of different offsets, in which case the adaptor signature can be accompanied by a discrete log equality proof.

4.3.1 Atomic Swaps

For the purposes of explanation, I will describe how to adapt a basic atomic swap protocol to these anonymous transactions. To do this, the participants must agree on a Schnorr multisignature protocol. Technically, this does not matter for the structure of the adaptor signatures, so to avoid any issues I will use a conservative version where the nonces are fixed in advance. Suppose Alice controls currency of type a and Bob currency of type b. Both Alice and Bob will set agree on

$$(k_0, k_1) = h(\mathrm{P}_a, \mathrm{R}_a, \mathrm{P}_b, \mathrm{R}_b, \mathrm{T}_a)$$

With subscript points indicating the owner of that point. This allows them to agree on the joint public key

$$\mathrm{P} = \mathrm{T}_a + k_0 \mathrm{P}_b \qquad \mathrm{Q} = \mathrm{P}_a + k_1 \mathrm{P}_b$$

Then, Alice and Bob will construct a transaction paying her currency to a multisig output with nullifier key P and a separate transaction paying Bob's currency to a different multisig output with nullifier key Q. They will not sign the nullifier inputs for these transactions, and will stop as soon as the randomizers for the multisig outputs have been selected.

Next, Alice and Bob will construct a transaction paying multisig output of currency of type a back to Alice with an absolute time lock, as well as the same for Bob. These transactions will be completed up to the membership proof phase and will be used by both parties if one decides to abort the swap. Once these transactions are completed up to the membership proofs, both parties can sign the transactions creating the multisig outputs.

At this point, the nullifier base for the transaction with key P be B_a and the nullifier for the other transaction be B_b. Alice will share with Bob $\mathrm{T}' = t_a\mathrm{B}_b$ where $\mathrm{T} = t_a\mathrm{G}$ as well as a discrete log equality proof. Then, Alice and Bob will do the same for the nonces on their nullifier bases respectively. Since all the information was hashed prior to the selection of the k_i values, the signatures can be safely constructed as a linear combination of the independent signatures using the same challenges.

Alice will send Bob her half of the transaction paying herself with the offset nonce. Then Bob will send Alice his half of the signature paying her. When Alice receives this, she can complete the transaction by publishing the real multisig from which Bob will learn the nonce offset. From this, he will be able to complete the signature for P and claim his currency from Alice. So long as he is able to perform this before the timelock is released for Alice to claim her money back, the swap is atomic.

4.4 Bulletproofs++ Confidential Transactions

Bulletproofs++ confidential transactions offer several compelling advantages as the basis for the transaction protocol. Specifically, they support larger bases for range proofs, which reduces witness size, and they support multi-party, multi-asset confidential transactions efficiently. The range proofs used in the transactions, both for amounts and for time locks, will use base 16 digits to represent a 64 bit range. Both of these properties follow from the reciprocal permutation argument.

The protocol is lifted mostly unchanged from the Bulletproofs++ paper to a slightly more general form. It is split over multiple witness commitments to support larger transactions and needs to be modified to protect prover privacy and integrate with the membership prover. However, I have elected to still use the norm structure for the confidential transactions. It is relatively simple, but since the underlying argument is linear, and not the Bulletproofs++ norm argument, the norm must be proved in a subsequent phase.

4.4.1 Reciprocal Argument

The Bulletproofs++ range proofs and typed conservation of money proofs of rely on the reciprocal permutation argument, which I will review here. Given two sets of values in F and multiplicities also in F, called A and B, the prover wants to show that each value occurs with the same total multiplicitity in A and B. To do this, after committing to A and B, the prover will show that, for random challenge e

$$\sum_{(a,m)\in A} \frac{m}{e+a} = \sum_{(b,m)\in B} \frac{m}{e+b}$$

To do this, following the commitment to the sets, the verifier will select the challenge value e uniformly at random, and the prover will commit to the evaluations of each reciprocal. The prover will then show that each reciprocal value is properly constructed, and that the net sum of reciprocals equals zero.

The security of this protocol follows straightforwardly from the Schwartz-Zippel lemma applied to the numerator and denominator of the rational function formed by the total sum of all the reciprocals. Since the probability of the challenge being a root of the denominator or the numerator is negligible, the equation holds at a random point only if the equation holds as an equation of rational functions. This, in turn, follows from the linear independence of reciprocal factors with distinct roots over F.

4.4.2 Confidential Transactions

Applying the reciprocal argument to range proofs, the right hand side will be the digits of the value in a prescribed base, which in this protocol will always be 16, with multiplicity 1, and the right hand side will be the valid digit symbols of the base, with multiplicity equal to the number of times each symbol occurs in the value. For confidential transactions, the roots will be the type of each input or output and the multiplicities will be the amounts of the each input or output, with the left hand side running over the inputs and the right hand side running over the outputs.

Since both the range proof and conservation of money argument have the same structure, Bulletproofs++ is able to perform both efficiently. The only substantial addition to the Bulletproofs++ protocol that is necessary for the transaction body is to commit to the entire input and output openings in the transaction body commitments and to incorporate a check for the relative timelocks. To review the hexadecimal range proof for 64 bit range proofs, the provers will commit to the digits and shared multiplicities for each range

$$n_i = \sum_{k=0}^{15} d_{ik} 16^i \qquad m_{ik} = \#\left\{d_{ij} = k+1 : j = 0..15\right\} \qquad \mathbf{m} = \sum_i \mathbf{m}_i$$

$$\mathbf{d}(\texttt{Range}(\mathbf{n})) = \mathbf{m} \oplus \bigoplus_i \mathbf{d}_i$$

The verifier will choose a challenge value e and the prover will commit to evaluations of the reciprocals

$$r_{ij} = \frac{1}{e + d_{ij}} \qquad \mathbf{r}(\texttt{range}(\mathbf{n})) = \mathbf{0} \oplus \bigoplus_i \mathbf{r}_i$$

And the proof will show

$$(e + d_{ij}) r_{ij} = 1 \qquad \sum_{k=0}^{14} m_k \left(\frac{1}{e+k+1} - \frac{1}{e}\right) = \sum_{i,j} r_{ij} - \frac{1}{e}$$

As explained in the Bulletproofs++ paper, typed conservation of money can be checked by appending the values $\mathbf{n}$ onto each digit vector and the reciprocal type term onto the vector of reciprocals. This adds a single term per input and per output.

$$\mathbf{d}(\texttt{Range}(\mathbf{n})) = \mathbf{m} \oplus \left(\bigoplus_i \mathbf{d}_i \oplus v(\mathrm{CI}_i)\right) \oplus \left(\bigoplus_j v(\mathrm{CO}_j)\right)$$

$$s(\mathrm{C}) = \frac{1}{e + t(\mathrm{C})} \qquad \mathbf{r}(\texttt{range}(\mathbf{n})) = \mathbf{0} \oplus \left(\bigoplus_i \mathbf{r}_i \oplus s(\mathrm{CI}_i)\right) \oplus \left(\bigoplus s(\mathrm{CO}_j)\right)$$

The $\mathbf{d}$ and $\mathbf{r}$ will be organized on terms in a polynomial so that its norm has their componentwise product is encoded on the central coefficient. The linear constraints for each witness will be organized on the opposite term and scaled do as not to interfere with the products or any other linear constraint. The constant term of the polynomial will be the blinding terms for the confidential transaction and also commit to the error terms. This is elaborated in Bulletproofs++.

$$\mathbf{p}(t) = \mathbf{bl} + (\mathbf{d} + Q^{-1}\mathbf{v})t + (\mathbf{r} + Q^{-1}\mathbf{u})t^2$$

$$|\mathbf{p}(t)|_q = \left(\langle \mathbf{d}, \mathbf{r}\rangle_q + \langle \mathbf{u}, \mathbf{d}\rangle + \langle \mathbf{v}, \mathbf{r}\rangle\right) t^3 + e_0 + e_1 t + e_2 t^2 + e_4 t^4$$

4.4.3 Multiple Commitments

The transaction body is not typically on the critical path for the anonymous transactions, except in the case of extremely large numbers of outputs, so it is necessary to support splitting the commitment to $\mathbf{d}$ and $\mathbf{r}$ over a larger number of actual commitments to extract the maximum proof capacity. This works in a very straightforward manner where the $\mathbf{d}_i$ and $\mathbf{r}_i$ are organized on terms with degrees that sum to the central coefficient. The constraint vectors are organized on complementary terms as well. The Bulletproofs++ blinding protocol generalizes as well in the way described in the original Bulletproofs++ paper. Note this does consume an amount of witness space quadratic in the number of commitments, but the amount of witness space will also increase quadratically. At a certain point it will become more economical to organize the error terms into their own commitment and add a round to the final protocol.

$$\mathrm{D}_i = \mathrm{Com}_0\left(\mathbf{d}_i\right) \qquad \mathrm{R}_i = \mathrm{Com}_0\left(\mathbf{r}_i\right)$$

$$\mathbf{p}(t) = \mathbf{bl} + \sum_{i=0}^{k-1} (\mathbf{d}_i + Q^{-1}\mathbf{v}_i)t^{i+1} + (\mathbf{r} + Q^{-1}\mathbf{u}_i)t^{2k-i}$$

$$|\mathbf{p}(t)|_q = \left(\langle \mathbf{d}, \mathbf{r}\rangle_q + \langle \mathbf{u}, \mathbf{d}\rangle + \langle \mathbf{v}, \mathbf{r}\rangle\right) t^{2k+1} + \sum_{i \neq 2k+1} e_i t^i$$

4.5 Transaction Outputs

Every transaction output will commit to 8 scalar values: the type of currency, the amount of currency, the timelock offset t_1, the timelock flag f_t, and the coordinates of both the nullifier base B and the nullifier key K. The transaction outputs will be canonically in E_0, making these scalars in F_1, and the nullifier base and key in E_1. Since the relationship between E_0 and E_1 is symmetric, in practice there is degree of freedom in which instantiated curve is which. It may be convenient for compatibility with existing cryptocurrencies to set E_1 to be the curve in which signatures are previously conducted over, as this allows existing public keys to be used directly as nullifier keys.

4.5.1 TXO Tree

All the outputs of a transaction will be organized into a small Curve Tree. This allows transactions to have very large numbers of outputs at an essentially negligible increase in proof size. In order to do this, the transaction will use a modified membership proof for this tree of outputs. After the transaction is included in a block, all these output trees can be included in the tree for the block, which can then be inserted into the global transaction tree. This technique is somewhat similar to the Bitcoin Taproot construction. In the future it may be possible to augment the output tree to include other scripting functionality.

The TXO tree for the transaction also serves as the mechanism for setting the tag information for the transaction outputs. The transaction will show that the tags for all the nodes inside the TXO tree are zero, and then the prover will add to the root of the tree a commitment to the tag information, consisting of the nullifier randomizer R and the current block time t_0. Since the randomizer must be in the same curve as the nullifier, this means the tag information must lie over F_1 and so the root of the TXO tree must be in E_0.

Since the insertion of this tree into the block and global transaction tree is performed transparently, and since the interior nodes of the tree are proven to have no tags, it follows that for each leaf there is exactly one interior node with a non-zero tag. The spending transaction for the output needs to be able to prove that the tag data for its inputs is correct in zero knowledge with respect to which node along the path has the tag, as this would leak information about the minting transaction. Since all the other tags are provably zero, it is sufficient to simple sum all the tags.

4.5.2 Proof of Knowledge

In order to preserve the privacy of multiple provers in the same transaction, the mechanism used to open input commitments must be designed such that the value of the commitments is kept private. In Bulletproofs++ and other confidential transactions protocols, this is not the case and the commitments can be shared among all provers. However, since the party that created the input might be a participant in the transaction that is spending the input, this would break the privacy of the recipient in the anonymous transaction protocol.

To handle this, the provers will perform a Schnorr proof of knowledge over the input and output commitments. However, they will not share the commitments themselves, but rather commitments to the commitments. In context, this is involves a single multiparty commitment to all the points and their nonces. This is also when the provers will commit to the input commitments as well as publish the TXO tree for the transaction.

$$\mathrm{TXO} = \mathrm{MrklCom}(\mathrm{CO}_j : j)$$

Then, the verifier will select a challenge for each input and each output commitment, from the short distribution, and they will share the openings associated with their independent signatures with each other. This allows them to collectively compute the opening for the entire proof of knowledge without revealing the values of their commitments, since both the commitments and nonces are committed. Later, in the membership phase, the membership provers will show the other side of this equation by performing zero knowledge proofs of elliptic curve operations to multiply the commitments while still keeping them private.

$$\mathbf{schnorr}(\texttt{TXBody}) = \mathbf{wit}\left(\mathrm{Bl}(\mathrm{TXO}) + \left(\sum_{i=0}^{n-1} \mathrm{Bl}(\mathrm{CI}_i) + e(\mathrm{CI}_i)\mathrm{CI}_i\right) + \left(\sum_{j=0}^{m-1} e(\mathrm{CO}_j)\mathrm{CO}_j\right)\right)$$

At the same time, the transaction body proof will commit to the openings of the input and output commitments inside the transaction body. Once the Schnorr challenges have been selected, the provers will also show that the associated linear combination of values from inside the transaction body commitment satisfy the same linear relations. This is essentially the same procedure that the provers will use for the nullifier keys, although since the scalars in that case are in the opposite field the procedures must happen in different Bulletproofs. It is not necessary to additionally commit to this value, as all curve points committing to the opening have already been committed.

4.6 Protocol

With the modifications for the proof of knowledge, as well as for the timelocks on each input, every input and output has an additional 9 scalars. Two for the value and the type, four for the nullifier base and nullifier key, two for the t_1 and f_t, and one for the t_0. Conveniently, the t_0 only needs to be committed in the reciprocal commitment, so the total number of witness elements per input and per output is 24 in $\mathbf{d}$ and $\mathbf{r}$. Since each input and output also has a typed conservation of money check, this becomes 25 witness elements per input and output. All the blinding will be handled according to Bulletproofs++ protocol, and the value t is public with respect to this protocol.

$$t_{\Delta,i} = t - t_1 - f_t t_0 \qquad \texttt{Ranges} = \texttt{Range}(\mathbf{t}_\Delta, \mathbf{v})$$

$$\mathbf{d}(\texttt{TXBody}) = \left(\bigoplus_i \mathbf{wit}\left(\mathrm{CI}_i\right)\right) \oplus \left(\bigoplus_j \mathbf{wit}\left(\mathrm{CO}_j\right)\right) \oplus \mathbf{d}\left(\texttt{Ranges}\right)$$

$$\mathbf{r}(\texttt{TXBody}) = \left(\bigoplus_i \mathbf{0} \oplus -t_0\right) \oplus \mathbf{0} \oplus \mathbf{r}\left(\texttt{Ranges}\right)$$

In addition to the range proof constraints, this will also include the proof of knowledge constraint and will modify the range proofs for the $\mathbf{t}_\Delta$ values to include the $-t_0 f_t$ values. This requires aligning the t_0 and f_t values in the $\mathbf{r}$ and $\mathbf{d}$ constraints respectively.

Part of the nullifier signature is also handled by this protocol, as it is necessary to keep this information secret to preserve privacy, but the way this will work in context is difficult to separate from the structure of the Bulletproof phase. For this reason, the portions of this protocol dealing with this should be understood as placeholder protocols, although they should be correct. As written, the Sig commitment would require its own Bulletproof with the appropriate linear constraints to show that all the proofs of knowledge are satisfied with respect to the published $r(\mathrm{K}_i)$.

$$\mathbf{sig}(\mathrm{K}_i) = (x(\mathrm{K}_i), s(\mathrm{K}_i), t(\mathrm{N}_i)) \qquad \mathrm{Sig} = \mathrm{Com}_1\left(\mathbf{sig}(\mathrm{K}_i) : i\right)$$

$$\mathrm{Null} = \mathrm{Com}_0\left(\mathrm{B}_i, \mathrm{B}'_i, \mathrm{Bl}(\mathrm{B}'_i), \mathrm{K}_i, \mathrm{Bl}(\mathrm{K}_i) : i\right)$$

The `TXBody` protocol object will be defined to support the accessor functions **coeffs** and **Points** to access the coefficients and points for the $\mathbf{p}(t)$ polynomial, as well **open** such that

$$\mathbf{p}(t) = \mathbf{open}(\texttt{TXBody}) = \mathbf{wit}\left(\langle \mathbf{coeffs}(\texttt{TXBody}), \mathbf{Points}(\texttt{TXBody})\rangle\right)$$

Algorithm 1 Transaction Body

```
commit $\mathrm{TXO}, \mathrm{CI}_i, \mathrm{Bl}(\mathrm{CO}), \mathrm{Bl}(\mathrm{CI}_i)$    ▷ Commit to the input and output commitments
commit $\mathrm{Null}, \mathrm{Sig}, \mathrm{N}_i$    ▷ and the nullifier bases, keys, and blinding for each
commit $\mathbf{D}(\texttt{TXBody})$
query $e(\mathrm{K}_i) \leftarrow \mathrm{half}(F_1)$    ▷ Receive nullifier challenge
commit $r(\mathrm{N}_i)$    ▷ Publish nullifier responses
query $e(\mathrm{CO}_j), e(\mathrm{CI}_i), e(\mathrm{K}_i) \leftarrow \mathrm{half}(F_1)$    ▷ Receive IO PoK and nullifier key challenges
query $e, r \leftarrow F_1$    ▷ also challenges for range proof, error terms
commit $\mathbf{R}(\texttt{TXBody})$
commit $r(\mathrm{K}_i), \mathrm{Bl}(\mathrm{Sig})$    ▷ Commit to IO PoK opening, nullifiers and their openings
query $x, q, e(\mathrm{Sig}), M \leftarrow F_1$    ▷ Receive range proof challenges and blinding matrix
commit $\mathrm{Bl}(\texttt{TXBody})$
query $t \leftarrow F_1$
publish $\mathbf{open}(\texttt{TXBody}), \mathbf{wit}\left(\mathrm{Bl}(\mathrm{Sig}) + e(\mathrm{Sig})\mathrm{Sig}\right)$    ▷ Typically send to membership provers
```

5 Membership Proofs

The second section of the anonymous transaction protocol deals with the set membership proofs of the inputs. It can be performed given only the output of the transaction body proof and given that the membership provers collectively know the input commitments and their Merkle witnesses. These membership provers might the same as the transaction body provers, they might be a subset of the transaction body provers as in the atomic swap case, they might be an entirely unrelated set of provers, or they might be any combination of these.

The membership proof phase of the transaction protocol is typically, especially for larger transactions, the most computationally intensive part of the transaction. It consists of many ECIP proofs, one for each input and one for the outputs. For each input, the prover will take a random linear combination of the Pedersen hashes from the odd levels, which will all be curve points in E_1, and the even levels, which will all be curve points in E_0. Then, the resulting linear combinations will be exchanged between the proofs and treated as Pedersen commitments. This allows opening the results very efficiently.

Taken in isolation, this construction is fairly simple and pleasantly symmetric. It also allows integrating the opening of the input and output commitments, via the Schnorr proof of knowledge from the previous section, and the nullifier protocol very efficiently. Each input already has an ECIP circuit in each proof, so adding more scalar multiplications is relatively cheap. Through out the section, it is important to keep in mind that there are proofs in both curves, and that, for example, the input commitment openings and the nullifier protocol will happen in the different Bulletproofs.

In the final section of the paper, the symmetry of the membership proofs will be broken, with one Bulletproof being partially inside the other. For clarity, this section will not discuss this symmetry breaking at all and instead will simply refer to two simultaneously occurring, independent Bulletproofs. For the purposes of the Fiat-Shamir transformation, simultaneous commitments in different proofs will both be provided to the random oracle together, and challenges in different fields will be produced together. It is not secure for the proof to use seperate oracles.

5.1 Inter-Transaction Protocol

Each transaction will produce a tree of output, as described in the previous section, and will produce zkSMPs for all of its inputs in the global TXO tree. The inter-transaction protocol deals with how these per-transaction trees are combined into the global tree. While this paper considers the transaction protocol in the context of a public blockchain, it is sufficiently general to be adaptable many other circumstances. For example, an e-cash system, such as that of Chaum [35], relies on both the central party to process transactions and also to not secretly create new money. This anonymous transaction protocol could be used to make the latter attack impossible, given that the transaction record were made public.

In any case, the protocol can be organized so that concurrent transactions are batched into "blocks." In a public blockchain protocol, this procedure will be carried out by miners, as in most existing cryptocurrencies, and the output trees will be organized into a block tree. These trees must be arranged so that all the outputs are at the same depth. For simplicity, the protocol can impose a standard block depth as well. All of the information necessary to construct the block tree must be published so that the owners of each transaction output can construct Merkle witnesses on their own. Using block trees makes it easier for this process to occur, as users can be expected to record the information of the block in which their transaction appears.

5.1.1 Merkle Mountain Range

The blocks themselves will also be organized into a Curve Tree. However, unlike the tree inside the block, this tree must be regularly updated. To do this, the provers can use the Merkle Mountain Range (MMR) protocol [53], which allows for efficient append-only insertion of elements into a Merkle tree. In an MMR, the $N < 2^n$ leaves of the tree are organized in the sequence they were inserted. The oldest 2^{n-1} are organized into a complete Merkle tree and the process is then repeated on the remaining $N - 2^{n-1}$ leaves. This produces a complete Merkle tree for each of set bit of N written in binary. When a leaf is inserted, if there is a singleton tree, the new leaf and the singleton are combined into a tree with two leaves, and this tree is inserted in the higher order trees. If there is already a tree with two elements, these are also combined, and so on, until there is not a tree of the same size present. In that case the new tree is inserted as the complete Merkle tree of that size. This follows the same pattern of carries when adding $N + 1$ in binary.

In the worst case when $N = 2^n - 1$, this requires $n = \lceil \log_2 N \rceil$ hashes. The MMR construction also has the very convenient property that paths to the leaves "converge" in the sense that the elements closest to the leaf do not change, and the segment of the path which doesn't change grows over time. Additionally, verifier only need to keep the MMR tips in order to update the tree in real time. Since each transaction provides its own membership proof for each input, verifiers do not necessarily need to keep the structure of each block tree.

5.1.2 Transaction Age

The inter-transaction protocol should also require that transaction have a minimum and maximum age. This prevents transactions from immediately spending outputs, and also prevents leaking information about transactions that were prepared far in advance, for example as part of a long lasting payment channel. The separation of the transaction body from the other phases of the proof makes this feasible.

By setting a maximum transaction age, it is also possible to limit the amount of data that verifiers must keep around. Each transaction will specify a Merkle root with respect to which the membership proofs are defined. Either, the transaction must provide a Merkle membership witness of this root in the block tree, or the verifier must keep a record of all the Merkle witnesses in order to check the validity of each transaction. It may be possible to avoid this problem by having the miner provide a zkSMP for each Merkle root for each transaction, in which case the verifiers would not need to be keep all the roots but the miners would. This would still leak information about old transactions.

5.2 Curve Tree Accumulator

Given an amicable cycle of elliptic curves and a pair of Pedersen hash functions

$$h_0 : F_1^{n_1} \to E_0 \qquad h_1 : F_0^{n_0} \to E_1$$

$$h_b(\mathbf{x}) = \langle \mathbf{x}, \mathbf{G}_b \rangle$$

A Curve Tree over leaves with representation over $\mathbb{F}_0$ is defined to be a Merkle tree where alternating levels of the tree use alternating hash functions. This ensures that the field over which the hash is applied agrees with its inputs. Canonically, the leaves will be the 0 layer of the tree, the next layer of the tree the 1 layer of the tree, and so on. Each layer of a Curve Tree may also, optionally, contain some tag information in this protocol.

When applying the hashes to their children, the protocol must fix an encoding of the elliptic curve. Without loss of generality, the protocol can use affine Weierstrass coordinates as every elliptic curve can be put into Weierstrass form. However, this does make it impossible to hash the point at infinity. For empty subtrees, the protocol will use the coordinates $(0, 0)$, although it is possible to use any coordinates for empty subtress that do not equal a cuve point. Alternatively, it might be possible to use an different curve representation like (twisted) Hessian curves [54], which represent the point at infinity in affine coordinates as $(0, 0)$ and therefore have a convenient closure property for empty subtrees, as a commitment to two empty subtrees is the point at infinity in the other curve. I am not sure if this introduces any problems with the protocol, as it becomes possible to open leaves in the empty subtree to commitments to zero.

This tree is sound as a Merkle tree if both hash functions have second pre-image resistance. Pedersen hash functions satisfy this property if the discrete log problem is hard in the output curve. Therefore, the CT is sound if the discrete log problem is hard in both E_0 and E_1. As this is true by assumption, it is sound. Equivalently, any method for constructing a path witness for an element not present in the tree can be used to construct a pair of $\mathbf{w}$ and $\mathbf{w}'$ such that

$$h_b(\mathbf{w}) = h_b(\mathbf{w}')$$

By the linearity of Pedersen hashes, this immediately yields an element of the nullspace of h as a linear transformation. This method therefore constitutes a solution to the discrete log problem in E_b.

$$h_b(\mathbf{w} - \mathbf{w}') = \mathcal{O}$$

5.2.1 Path Witness

A set membership witness for a leaf in such a tree, as with an ordinary Merkle tree, consists of a sequence of openings, beginning at the parent of the leaf and continuing to the root of the tree. Canonically, the root will also be in E_0, and the hashes of along the path may be treated as Pedersen commitments $\mathrm{C}_1..\mathrm{C}_{2n}$. All the even index commitments lie in E_0 and the odd index commitments in E_1. In many cases, the leaves will also be commitments, in which case they can be denoted as C_0. Letting the openings of these hashes be

$$\mathrm{C}_i = h_{i \bmod 2}(\mathrm{A}_{i-1}, \mathrm{B}_{i-1}, \mathbf{t}_{i-1})$$

Where A_{i-1} is the left child, B_{i-1} is the right child, and $\mathbf{t}_{i-1}$ is the tag information.

This path is valid if and only if there exists direction values $d_{i-1} \in 0, 1$ such that

$$\mathrm{C}_i = (1 - d_i)\mathrm{A}_i + d_i\mathrm{B}_i$$

For the leaf value C_0, the path witness consists of two vectors of values over both fields each consisting of these values. Note that all the points are committed separately. While it might be technically feasible to avoid committing to C in addition to the other information, the structure of the ECIP proofs requires that each basis point have 5 additional conjugate point terms, so the result would increase proof size.

$$\mathbf{path}(\mathrm{C}_0) = ((\mathrm{A}_{2i+b}, \mathrm{B}_{2i+b}, \mathrm{C}_{2i+b}, d_{2i+b}, \mathbf{t}_{2i+b}) : i = 0..n-1)$$

This does not include the path root, as that information will be public with respect to the transaction. It must be included in the proof transcript, for the purposes of generating challenges, but it also needs to be verifiably a valid TXO set root. The protocol could be augmented to behave differently. The path witness also does not include the opening of the leaf commitment, both for generality's sake as trees could have non-commitment roots, as well as to preserve the separation between the transaction body and membership proof phases.

5.3 Zero Knowledge Set Membership

The zkSMP begins with the prover committing to both halves of the path witness in the opposite elliptic curve, so that the scalar field it is represented over agrees with that of the witness. These commitments will also include blinding commitments Bl_b supported over the entire image space of the corresponding hash. In transactions, this as well as all the other commitments of this section, may require multiple Bulletproof commitments, which are of fixed length, to represent the entire witness. Once the path witness has been committed, the verifier will choose random, short challenge values a_i, which $a_{2i+b} \in F_b$.

Then, the prover will publish the linear combinations S_0 and S_1 of the hashes by theses challenges. They will also show using a public scalar, private point ECIP protocol that the linear combinations are satisfied

$$\mathrm{S}_0 = \mathrm{a}'_1\mathrm{Bl}_0 + \sum_{i=1}^{n-1} a_{2i-1}\mathrm{C}_{2i} \qquad \mathrm{S}_1 = \mathrm{a}'_0\mathrm{Bl}_1 + \sum_{i=0}^{n-1} a_{2i}\mathrm{C}_{2i+1}$$

This involves committing to a large amount, as measured by scalars, of divisor witnesses. The precise size of the witnesses will depend on the depth of the Merkle tree, each proof having as many scalar multiplications as the depth of the tree. The proofs will then take the published combinations S_b and treat them as Pedersen commitments. This allows "exchanging" them between the proofs and opening them. The proofs will show that, upon opening, they have the form

$$\mathrm{S}_0 = h_0\left(a'_1\mathbf{bl}_1 + \sum_{i=1}^{n-1} a_{2i-1}\mathbf{wit}\left(\mathrm{C}_{2i}\right)\right) \qquad \mathrm{S}_1 = h_1\left(a'_0\mathbf{bl}_0 + \sum_{i=0}^{n-1} a_{2i}\mathbf{wit}\left(\mathrm{C}_{2i+1}\right)\right)$$

5.3.1 Direction Selection

Additionally, the proofs will show that $\mathrm{C}_i = \mathrm{A}_i$ or $\mathrm{C}_i = \mathrm{B}_i$. Since the ECIP proof already involves the computation of the quantity, for random λ, μ, P

$$f(\mathrm{C}_i) = \frac{x(\mathrm{P}) - x(\mathrm{C}_i)}{\mu + \lambda x(\mathrm{C}_i) - y(\mathrm{C}_i)}$$

The proof can reuse this value and just show that

$$d_i(f(\mathrm{A}_i) - f(\mathrm{C}_i)) + (1 - d_i)(f(\mathrm{B}_i) - f(\mathrm{C}_i)) = 0$$

It is easy to show that the only witnesses for which this relation holds for random challenges, as a consequence of the Schwartz-Zippel lemma, are $d_i = 0, \mathrm{A}_i = \mathrm{C}_i$, $d_i = 1, \mathrm{B}_i = \mathrm{C}_i$, or $\mathrm{A}_i = \mathrm{B}_i = \mathrm{C}_i$. Technically, d_i is unconstrained in the final case, but regardless of its value the membership is valid if the equation is satisfied. Proofs that require d_i be binary for other reasons could additionally check that $d_i^2 = d_i$.

5.3.2 Tag Sum

Since the output tree of every transaction is of variable depth, and since the tag for a new transaction is inserted at the root of the output tree, each transaction must open this tag in zero knowledge with respect to its position along the path. Each tag consists of three scalar values representing the time and the randomizer point

$$\mathbf{t}_i = (t_{pub,i}, \mathrm{R}_i)$$

Every transaction will show that the tags internal to its output tree are zero, and all the public tags from the TXO tree are verifiably zero. This ensures that for every leaf there is exactly one non-zero tag in the path from the root to the leaf. The prover can therefore sum all the tags along the path and the resulting value will be the non-zero tag. This is zero knowledge with respect to which level of the tree the tag comes from.

5.3.3 Membership Proofs without a Cycle

While this protocol focuses on Curve Trees defined over a cycle of curves, I think it is worth mentioning that there is a reasonably efficient zkSMP protocol for a Merkle tree constructed out of a single Pedersen hash function. There are practical reasons why the amicable version is preferable, although these largely have to do with the nested Bulletproof construction, not the membership proofs themselves.

Consider an elliptic curve E_0 used for proving and an elliptic curve E_1 used for the Merkle tree, with the characteristic of E_1 equal to the order of a subgroup of E_0, but not necessarily in the other direction. The Merkle tree will use a Pedersen hash function over E_1 and, when hashing F_1 elements will split them into chunks less than half the field length. For the purposes of this explanation, assume the inputs are split into chunks one third the length of the field by powers of 2. While this makes the tree more expensive to construct in zero knowledge, it is straightforward to do this using range proofs.

Then, to show membership in the tree the prover will commit to the hashes and the witnesses, consisting of the chunks above. These chunks are assumed to be correctly constructed, which requires the tree either be constructed publicly or the minting transaction use range proofs. Then the prover will select challenges of half length. This ensures that the length of the chunks plus the length of the challenges does not exceed the length of the scalar field of E_1. The prover will also commit to the linear combination of witnesses and open the sum of hashes in zero knowledge using private scalar, public point multiplications.

The prover can show that the same linear combination of the committed chunks by the challenges equals the opening of the linear combination of hashes. While it is possible to commit to chunks that would cause the linear combination to exceed the characteristic of the scalar field of E_0, they cannot do this and obtain the same result as the linear combination of hashes, as that quantity cannot exceed the characteristic by assumption. Therefore, the Schwartz-Zippel lemma can be applied in the scalar field of E_0.

The same basic technique can be applied to opening leaf commitments, although the data in the leaf commitments must also be split into chunks. I am not sure if the nullifier protocol requires special modifications to work in the non-amicable case, although even if it cannot be integrated into the input ECIP it can be checked separately. Blinding for commitments does not need to be split into chunks. Since the proof system does not require an amicable cycle, the curve E_0 is completely unconstrained, all that is necessary is to find a curve with suitable structure over its scalar field.

Note that this is not the same as a Verkle tree, since each level is hashed directly using the Pedersen hash. It is however quite similar to the algebraic hashes used by Zcash Sapling and in other curves that do not use a cycle of curves.

5.4 zkSMPs for all Leaves

The output tree of the transaction requires membership proofs to open the outputs inside the proof. While it would be possible to simply use the membership proof above, this would introduce redundant scalar multiplications of the same hash in multiple proofs. Instead, the provers can take the linear combination over all internal nodes at alternating levels. The resulting proof no longer needs direction selection proofs and does not redundantly perform scalar multiplications.

Amortized across all the outputs, this proof requires only 1 scalar multiplication per leaf, although this does not include opening the leaves which require an additional multiplication. These multiplications are not evenly split between the two proofs, as the number of multiplication per level doubles as one progresses from the root to the leaves. This proof also needs to, implicitly, show that the tag information for all the internal nodes is set to zero.

5.5 Integrated Leaf Openings

Each input zkSMP has an extra "free" term in the linear combination of points given by coefficient 1 portion of the linear combination. This is the constant term of the polynomials over the challenge space. The prover can use this free term to perform the portion of the proof of knowledge for the transaction body as part of the ECIP. Since the challenge for this proof of knowledge is also short, and since the signature is not additionally scaled as part of the ECIP by virtue of being the constant term, all of the scalars are still short.

Similarly, all the scalar multiplications for the output commitments can be performed as part of the output ECIP, as it also have a free constant term. Then, since each input ECIP result is blinded separately by its prover, the input linear combinations can simply be added to the output ECIP unscaled. If all the challenges are selected at the same time, for all the inputs' and outputs' ECIPs, then this can be opened to the full linear combination of all the nodes from all the proofs plus the final opening of the Schnorr proof of knowledge from the transaction body. That is, all the constant terms sum together to evaluate the PoK.

$$\mathrm{S}_1(\mathrm{CI}_i) = \mathrm{Bl}(\mathrm{CI}_i) + e(\mathrm{CI}_i)\mathrm{CI}_i + \text{path terms}$$

$$\mathrm{S}_1(\mathrm{TXO}) = \mathrm{Bl}(\mathrm{TXO}) + \left(\sum_i \mathrm{S}_1(\mathrm{CI}_i)\right) + \text{path terms}$$

This indirect structure of computing the ECIP has no bearing on the soundness of the argument, but does have the effect of preserving the secrecy of the value of the leaf commitments. This is necessary both because the creator of an output always knows its value and because the outputs are shared among all the participants in a transaction. This reorganization is essentially free, in the sense that even if the PoK were to be proved more directly it would require the same number of scalar multiplications. The only additional cost is one point addition, i.e. one scalar, per input.

5.6 Integrated Nullifier Proofs

The nullifier proofs can be incorporated into the membership ECIPs in a similar manner. Since the leaves are in E_0, the constant terms of the odd ECIPs, the curve the nullifier and nullifier keys are defined over, are still available. However, unlike the leaves, the odd ECIPs must perform two Schnorr verifications and they cannot both be present in the constant term. Instead, the prover will scale one of the verification equations by a random scalar and split it over both the input and output ECIPs in the following way.

The constant term of each input ECIP will contain the nullifier verification equation, that is the nonce and nullifier base, in the constant term and the nullifier key scaled by a new challenge, chosen along with the membership challenges. Then, instead of adding the input ECIP linear combination to the output ECIP, the provers will scale the linear combinations by the original nullifier key challenges. This has the effect of scaling the secret part of the nullifier verification equation in the constant of each input ECIP, yielding exactly the verification equation shown to be sound for nullifiers from the transaction body.

Then, the nullifier key nonce is scaled by the new nullifier key challenge, which yields the nullifier key verification equation. Since this nonce is itself a blinded Pedersen commitment, knowledge of its value yields no information about the nullifier key. Each input prover will demonstrate that the this linear combination opens to the correct scalars previously committed.

$$\mathrm{S}_1(\mathrm{CI}_i) = \mathrm{Bl}(\mathrm{N}_i) + e(\mathrm{N}_i)\mathrm{B}'_i + a(\mathrm{K}_i)\mathrm{K}_i + \text{path terms}$$

$$\mathrm{S}_1(\mathrm{TXO}) = \left(\sum_i e(\mathrm{K}_i)\mathrm{S}_1(\mathrm{CI}_i) + a(\mathrm{K}_i)\mathrm{Bl}(\mathrm{K}_i)\right) + \text{path terms}$$

Where the rest of the sums range over the input paths or the output nodes respectively as before. In order to check that the nullifiers are exactly the correct value, when the receiving Bulletproof opens the odd linear combination it must check that it exactly matches the committed openings. That is, it cannot contain any blinding not legible to the proof. This is where the technique to prevent error term blinding is necessary to prevent the use of blinding before a certain point the proof. Since the nullifiers are committed before it is possible to blind using the error term blinding, it is sufficient to check the single blinding element that the initial commitments use.

5.7 Witness Sizes

The witness for a membership proof, not counting blinding, is split over three rounds, each of which has witnesses in two proofs. For the purposes of estimating the maximum transaction capacity it is sufficient to simply treat witnesses between rounds as approximately fungible and divide the total witness size by the witness size per commitment. For the purposes of computing the sizes, I will use $d = d_0 + d_1$ where d_b is the number of levels of that parity. The root is not included in this number, so if the root is fixed to be in the even curve both will equal half the total number of levels.

The membership protocol objects are defined essentially as a concatenation of `ECIP` protocol objects over the S_b points as defined, with the additional witness information laid out in the **path** values. The coefficients and points of these `ECIP` objects should be inferred from their definitions above.

$$\texttt{Mem}_b = \texttt{ECIP}(\mathrm{TXO}) \oplus \bigoplus_i \texttt{ECIP}(\mathrm{CI}_i) \qquad \mathbf{path}(\texttt{Mem}_b) = \mathbf{points}(\texttt{Mem}_b) \oplus \mathbf{path}(\texttt{TXO}) \oplus \bigoplus_i \mathbf{path}(\mathrm{CI}_i)$$

The even path witness for the entire membership proof includes the ECIP point witnesses, 4 scalars per, for $d_0 + 1$ points where the additional one is the blinding. Note that as defined the d_0 includes the leaves and technically the leaves have already been committed in the transaction body phase along with their blinding. Note that each level commits to the four coordinates of leaves of the parent of this level as well as the direction for $5d_0$ more values. Each output tree point in the even curve also has a scalar multiplication.

$$\mathrm{len}\left(\mathbf{path}(\texttt{Mem}_0)\right) = (4 + 9d_0)n + (4/3)m$$

The odd path witness has a fairly similar structure. The odd input membership ECIPs perform 2 additional scalar multiplications for the nullifier base and the nullifier key. The path must include the witness for the nullifier randomization as well, 2 scalars. Each level stores the same information as before, the children and direction, plus three values for the tag at that level. The output ECIP additionally has two scalar multiplications per input and half as many for the output tree. This witness also includes the overall time lock range proof, for 21 total scalars.

$$\mathrm{len}\left(\mathbf{path}(\texttt{Mem}_1)\right) = (22 + 12d_1)n + (8/3)m + 21$$

The divisors are fairly each to calculate from the path witnesses. Since all the scalars are short, there are only 123 values in each ECIP spine. Each input has a spine and all the outputs share a spine. Each scalar multiplication costs 36 scalars on average, and each unweighted multiplication costs one additional scalar.

$$\mathrm{len}\left(\mathbf{divs}(\texttt{Mem}_0)\right) = (161 + 36d_0)n + 48m + 124$$

$$\mathrm{len}\left(\mathbf{divs}(\texttt{Mem}_1)\right) = (303 + 36d_1)n + 24m + 123$$

Similarly, the reciprocal witnesses follow. Each doubles the number of multiplications performed to account for both the evaluation and error term. Each spine has 82 total scalars for evaluations and each scalar multiplication 10. Each direction selection involves two multiplications, and each unweighted point addition one. In the odd witness, the range proof costs 16.

$$\mathrm{len}\left(\mathbf{evals}(\texttt{Mem}_0)\right) = 2((94 + 12d_0)n + 40/3m + 82)$$

$$\mathrm{len}\left(\mathbf{evals}(\texttt{Mem}_1)\right) = 2((143 + 12d_1)n + (20/3)m + 98)$$

This yields the total witness sizes of

$$\mathrm{len}\left(\mathbf{memwit}_0\right) = (353 + 69d_0)n + (188/3)m + 288$$

$$\mathrm{len}\left(\mathbf{memwit}_1\right) = (611 + 72d_1)n + (120/3)m + 340$$

From which it is easy to extract the approximate relative costs between outputs and inputs as the ratio of the m and n coefficients. The cost per input is clearly much higher as d_b ranges between 16 and 24. At the high end, the number of scalars per input is 2009 in the even case and 2339 in the odd case, which both are about 30% larger than the witness lengths for their respective commitments, 1536 and 1744. This means every 4 commitments converts to approximately 3 inputs. Alternatively, each input converts to approximately 31 outputs in the even witness and 58 outputs in the odd witness. This asymmetry is actually reversed by the fact that the odd witness also needs to commit to range proofs and other information in the transaction body. This adds 50 total scalars to each odd output, which brings the ratio closer to one.

5.8 Error Terms and Blinding

The membership phase will use exactly the protocol detailed in parallel prover subsection of the Circuits section. Following the commitments to the divisor witness evaluations along with the PLI error terms from the local circuits, the verifer will select the PLI challenge and the prover will commit to two blinding commitments as well as all the fragments and error terms. The constraints must be arranged here so as to be linearly independent. The verifier will choose the second PLI challenge to combine all the membership witness commitments, and the prover will send to the fragments and error terms to the Bulletproof prover for the final round.

This uses only one additional round to blind the entire membership proof. Note also that, apart from the commitments to the fragments and error terms, this combines all the commitments from the entire proof thus far, as the output of the transaction body phase is handled by the membership provers. Any information that needs to be communicated to the Bulletproof phase can be incorporated into the fragments as a separate accessor constraint and output as another value among the fragments. As a result, the Bulletproof phase can treat the entire first two phases opaquely and does not need to be able to open any earlier commitments, besides the blinded membership witnesses and the commitment to the fragments and error terms.

5.9 Protocol

Assume the constraints and witnesses are laid out according to the blinding procedure for this protocol as described. The error term object needs to be instatiated over all the commitments involved in each membership proof. In the Bulletproof phase some of these commitments will be reorganized, but this will not effect the correctness of the protocol. Note additionally that the linear combination of transaction body commitments is treated as a single opaque commitment for the purposes of this protocol.

$$\texttt{MemErrs}_0 = \texttt{ParErrs}(\text{Null}, \mathbf{D}(\texttt{TXBody}), \mathbf{R}(\texttt{TXBody}), \text{Sum}_0, \mathbf{Path}_0, \mathbf{Divs}_0, \mathbf{Evals}_0)$$

$$\texttt{MemErrs}_1 = \texttt{ParErrs}(\text{Sum}_1, \mathbf{Path}_1, \mathbf{Divs}_1, \mathbf{Evals}_1)$$

Where the various commitment vectors are defined according to

$$\mathbf{Path}_b = \mathbf{Com}_b\left(\mathbf{path}(\texttt{Mem}_{1-b})\right) \qquad \mathbf{Divs}_b = \mathbf{Com}_b\left(\mathbf{divs}(\texttt{Mem}_{1-b})\right)$$

$$\mathbf{Evals}_b = \mathbf{Com}_b\left(\mathbf{evals}(\texttt{Mem}_{1-b}), \mathbf{errs}(\texttt{Mem}_{1-b})\right)$$

$$\texttt{TXBodyErrs} = \texttt{IncErrs}(\texttt{TXBody}, \sqrt[3]{17n}, \sqrt[3]{17n}, \sqrt[3]{17n})$$

Algorithm 2 Membership Proofs

```
Require: execute $\texttt{TXBody}$                                              ▷ Run body protocol first
  commit $\mathbf{Path}_0, \mathbf{Path}_1$                                     ▷ Using as many commitments as necessary
  commit $\text{Com}_0\left(\mathbf{schnorr}(\texttt{TXBody}), \mathbf{errs}_0(\texttt{TXBodyErrs})\right)$   ▷ Split error terms for TX Body
  query $\mathbf{a}_0 \leftarrow \text{half}(F_0), \mathbf{a}_1 \leftarrow \text{half}(F_1)$   ▷ Challenges for path sum
  query $e_0(\texttt{TXBodyErrs}) \leftarrow F_0$
  commit $\mathbf{Divs}_0, \mathbf{Divs}_1$
  commit $\text{Sum}_0, \text{Sum}_1$                                           ▷ Both sums used in both proofs
  commit $\text{Com}_0\left(\mathbf{errs}_1(\texttt{TXBodyErrs})\right)$
  query $\text{A}_0(\texttt{Mem}_0) \leftarrow E_0, \text{A}_0(\texttt{Mem}_1) \leftarrow E_1$   ▷ Challenges for ECIPs
  query $e_1(\texttt{TXBodyErrs}) \leftarrow F_0$
  commit $\mathbf{Evals}_0, \mathbf{Evals}_1$
  commit $\text{Bl}_{0\oplus 1}(\texttt{Mem}_0), \text{Bl}_{0\oplus 1}(\texttt{Mem}_1)$
  commit $\text{Com}_0\left(\mathbf{errs}_2(\texttt{TXBodyErrs})\right)$
  query $e_0(\texttt{Mem}_0) \leftarrow F_0, e_0(\texttt{Mem}_1) \leftarrow F_1$   ▷ Challenges for individual circuits
  query $e_2(\texttt{TXBodyErrs}) \leftarrow F_0$
  commit $\text{Com}_1\left(\mathbf{frags}(\texttt{MemErrs}_0)\right), \text{Com}_0\left(\mathbf{frags}(\texttt{MemErrs}_1)\right)$
  query $e_1(\texttt{Mem}_0) \leftarrow F_0, e_1(\texttt{Mem}_1) \leftarrow F_1$   ▷ Challenges for combined circuit
  publish $\mathbf{open}(\texttt{MemErrs}_0), \mathbf{open}(\texttt{MemErrs}_1)$   ▷ Opening used by Bulletproof provers
```

6 Nested Bulletproofs

The final phase of the protocol does not involve any private information and exists entirely to reduce the proof size by partial verification of the Bulletproofs in zero knowledge. This phase of the protocol can be performed by anyone, without giving up the privacy of the transaction participants. This phase of the proof is also where the bottlenecks to increasing transaction size without increasing commitment witness length occur.

While the membership proof is largely symmetric with respect to the curves and proofs over those curves, this phase of the protocol will break that symmetry by designating the proof over E_1 as the "outer" Bulletproof and the proof over E_0 as the "inner" Bulletproof. While this is mostly nominally accurate, at least in the Bulletproof phase, the inner proof will also prove part of the outer proof. This construction naturally generalizes to even deeper levels of nesting, in which case both proofs are partially inside and partially outside of the other. All commitments will be organized as a tree, with the outermost one over E_1, which will commit to the data for the outer proof as well as to commitments over E_0 for the inner proof.

This construction is necessary for proving the inner Bulletproof in zero knowledge with respect to the values of the response commitments in the argument. It also allows publishing a single outer commitment per round, which can commit to multiple inner commitments, which in some cases can even commit to multiple outer commitments once again. In this way, the inner Bulletproof will prove part of the outer Bulletproof taking the linear combination of membership witnesses. To reconcile this portion of the outer Bulletproof inside the inner Bulletproof, the prover will use a single commitment to facilitate communication. This is sufficient to implement the communication for the membership proofs as well, as the other direction is free as a consequence of the nested proof structure.

6.1 Nested Protocols

The transaction body and membership proof phases need to be encoded into the hierarchical tree commitment structure to be compatible with the nested Bulletproofs. One advantage to this, besides compatibility with nested Bulletproof verification, is that each round of interaction between the prover and verifier, for the most part, corresponds to a single outer commitment. Even in rounds where the provers need to send a large number of commitments, for example in the membership proof where many commitments may be necessary to fit all the ECIPs, the entire round can be made to fit in a single outer commitment. The only exceptions to this rule are the nullifiers, nullifier responses, as they must be public, and an additional commitment to facilitate communication between the proofs.

6.1.1 Transaction Body

The transaction body has four rounds, one of which only involves publication of nullifier information, so there are only three commitments. These three commitments commit to the three phases of the confidential transaction protocol over F_0 in the inner commitment as well as the nullifier key information in the outer commitment. The first commitment, therefore, commits to everything from the first round: the first transaction body commitment, nullifier bases, randomizers, keys, private keys, nonce secrets, and the nullifiers themselves separately:

$$\mathrm{P}_0 = \mathrm{Com}_1\left(\mathbf{D}(\texttt{TXBody}), \mathrm{Null}, \mathrm{CI}_i, \mathrm{Bl}(\mathrm{CI}_i), \mathbf{sig}(\mathrm{K}_i) : i\right)$$

Then, the verifier will select the challenges for the nullifier signatures, and the prover will publish the nullifier responses. These are published in the clear in order to facilitate the adaptor signature protocols described earlier. The verifier will then choose the nullifier key challenges and the reciprocal challenge for the confidential transaction. The prover will commit to the blinding for the nullifier secret information and the responses for the nullifier secret key verification.

$$\mathrm{P}_1 = \mathrm{Com}_1\left(\mathbf{R}(\texttt{TXBody}), \mathbf{bl}(\mathbf{sig}(\mathrm{K}_i)), r(\mathrm{K}_i) : i\right)$$

Then, the verifier will select the multiplication challenge for the confidential transaction and a linear challenge to combine all the verification equations for the nullifier secrets into a single linear constraint. The prover will commit again to the two inner commitments along with the blinding commitment for the confidential transaction protocol. The prover will also commit to a linear combination of all the data from P_0 and P_1 that will be used later, i.e. everything except the nullifier key secrets.

$$\mathrm{P}_2 = \mathrm{Com}_1\left(\mathrm{Bl}(\texttt{TXBody}), c_0, c_1\right)$$

Finally, the verifier will choose the challenge to blind the confidential transaction and the challenge to blind the key responses. The provers will open the confidential transaction, as well as the outer commitment up to the responses, which will be shared with the membership prover for that input. The first two commitments will be combined according to the challenge, and the membership provers will be responsible for showing that they satisfy the remaining constraints. The membership prover will be given, along with knowledge of their opening distributed among the appropriate provers

$$\mathrm{P}_0 + e(\mathrm{Sig})\mathrm{P}_1, \mathrm{P}_2 \in E_1 \qquad \mathrm{Null}, \langle \mathbf{coeffs}(\texttt{TXBody}), \mathbf{Points}(\texttt{TXBody})\rangle \in E_0$$

6.1.2 Membership Proofs

The nesting of the membership proof phase occurs at one level deeper than that of the transaction body phase. That is, rather than committing to the F_0 witness directly in the outer commitment, the provers will commit to commitments to this witness inside the inner commitment. This is necessary both to fit all the commitments to the membership witness in a single outer commitment and to facilitate the nested Bulletproofs later. Thus, the fourth commitment becomes

$$\mathbf{Path}_1 = \mathbf{Com}_1\left(\mathbf{path}\left(\texttt{Mem}_0\right)\right)$$

$$\mathbf{Path}_0 = \mathbf{Com}_0\left(\mathbf{schnorr}(\texttt{TXBody}), \mathbf{errs}_0(\texttt{TXBody}), \mathbf{basis}(\mathbf{Path}_1), \mathbf{path}\left(\texttt{Mem}_1\right)\right)$$

$$\mathrm{P}_3 = \mathrm{Com}_1\left(\mathbf{basis}(\mathbf{Path}_0)\right)$$

Some data, specifically the result of the result of the linear combinations will be committed in the outer commitment. This allows the inner proof to open it and verify it commits to the correct values. For the purposes of the membership proofs, it is sufficient to treat both the outer commitments and the inner commitments as part of the same proof.

$$\mathbf{Divs}_1 = \mathbf{Com}_1\left(\mathbf{divs}\left(\texttt{Mem}_0\right)\right)$$

$$\mathbf{Divs}_0 = \mathbf{Com}_0\left(\mathbf{errs}_1(\texttt{TXBody}), \mathbf{basis}\left(\mathrm{Sum}_1, \mathbf{Divs}_1\right), \mathbf{divs}\left(\texttt{Mem}_1\right)\right)$$

$$\mathrm{P}_4 = \mathrm{Com}_1\left(\mathbf{basis}\left(\mathrm{Sum}_0, \mathbf{Divs}_0\right)\right)$$

Then, following the commitment to the divisors, the provers will commit to the evaluations for the membership proofs along with the error terms for the circuits and both blinding commitments for the final linear combination of the membership proofs.

$$\mathbf{Evals}_1 = \mathbf{Com}_1\left(\mathbf{evals}\left(\texttt{Mem}_0\right), \mathbf{errs}\left(\texttt{Mem}_0\right)\right)$$

$$\begin{aligned}\mathbf{Evals}_0 = \mathbf{Com}_0\big(&\mathbf{errs}_2(\texttt{TXBody}), \mathbf{basis}\left(\mathbf{Evals}_1, \mathrm{Bl}_0(\texttt{Mem}_0), \mathrm{Bl}_1(\texttt{Mem}_0)\right),\\ &\mathbf{evals}\left(\texttt{Mem}_1\right), \mathbf{errs}\left(\texttt{Mem}_1\right)\big)\end{aligned}$$

$$\mathrm{P}_5 = \mathrm{Com}_1\left(\mathbf{Evals}_0, \mathrm{Bl}_0(\texttt{Mem}_1), \mathrm{Bl}_1(\texttt{Mem}_1)\right)$$

The final round of the membership proofs involves committing to the error terms for the combination of the membership proofs. Note that here error terms for the even membership proof are committed in the outer commitment. These values can be freely shared with the Bulletproof prover without compromising the privacy of the transaction. Note that $\texttt{MemErrs}_b$ must be modified to use the nested commitments structure and to provide accessor constraints for commitments that are necessary for performing the Bulletproof phase. When written, the commitments following this point in the protocol actually will refer to components of the fragment commitment. All preceeding commitments are included in these error term computations

$$\texttt{MemErrs}_0 = \texttt{ParErrs}(\mathrm{Null}, \mathbf{D}(\texttt{TXBody}), \mathbf{R}(\texttt{TXBody}), \mathrm{Bl}(\texttt{TXBody}), \mathrm{Sum}_0, \mathbf{Path}_0, \mathbf{Divs}_0, \mathbf{Evals}_0)$$

$$\texttt{MemErrs}_1 = \texttt{ParErrs}(\mathrm{P}_0, ..., \mathrm{P}_5, \mathrm{Sum}_1, \mathbf{Path}_1, \mathbf{Divs}_1, \mathbf{Evals}_1)$$

$$\mathrm{Frag}_1 = \mathrm{Com}_0\left(\mathbf{frags}(\texttt{MemErrs}_0)\right)$$

$$\mathrm{P}_6 = \mathrm{Com}_1\left(\mathbf{frags}(\texttt{MemErrs}_1), \mathrm{Frag}_1\right)$$

Following the selection of the challenge value, the prover will publish the portion of the linear combination of commitments from the outer proof that are inside the inner proof. This quantity will be referred to as X and will then be included in the outer linear combination. Then, in the next phase, the prover will show that this value was properly constructed from these inner commitments using an ECIP proof. In a certain sense, this is where the inner to outer half of the membership phase communication occurs.

6.2 Bulletproof Bodies

Immediately following P_6 and the selection of the challenge, and concurrently with X, the Bulletproof prover will begin an ECIP protocol for both the even and odd membership proof commitments. These protocols accomplish the same objective, but will have different structure. The outer Bulletproof body, that is the even membership proof, will have a single round of divisor commitments, but will allow using multiple commitments. The inner Bulletproof body, on the other hand, can only commit to one commitments worth of divisor witness information per round, but will be able to do this over three rounds. This definition is a slight abuse of notation, as the $\mathtt{MemErrs}_0$ object includes the linear combination of transaction body commitments, but for the Bulletproof phase the individual scalar multiplications must be performed. So, the poles multiplied against this commitment should be distributed to the **coeff** terms and those added to the ECIP. It also requires distinguishing the P_i scalar multiplication, which are not included in the ECIP and those that are used to compute X.

$$\langle \mathbf{poles}(\mathtt{MemErrs}_1), \mathbf{Points}(\mathtt{MemErrs}_1)\rangle = \langle \mathbf{poles}_0, \mathrm{P}_0, ..., \mathrm{P}_5\rangle + \mathrm{X}$$

$$\mathtt{BPBody}_0 = \mathtt{ECIP}(\mathrm{BodySum}_0, \mathbf{poles}(\mathtt{MemErrs}_0), \mathbf{Points}(\mathtt{MemErrs}_0))$$

$$\mathtt{BPBody}_1 = \mathtt{ECIP}(\mathrm{X})$$

$$\mathbf{BodyDivs}_0 = \mathbf{Com}_0\left(\mathrm{BodySum}_1, \mathbf{divs}\left(\mathtt{BPBody}_1\right)\right)$$

$$\mathrm{P}_7 = \mathrm{Com}_1\left(\mathrm{BodySum}_0, \mathbf{divs}_0(\mathtt{BP\text{-}Body}_0), \mathbf{basis}(\mathbf{BodyDivs}_0)\right)$$

For the outer Bulletproof body, the next round will commit to the evaluations for the ECIP protocol. These will include the error terms for the ECIP circuit as well as for the circuit verifying the output of the odd membership proof. The ECIP for the inner Bulletproof will commit to more divisor witness information. The subscripts on divisor accessor functions should be understood to denote fragments of the larger vector of divisor information.

$$\mathbf{BodyEvals}_0 = \mathbf{Com}_0\left(\mathbf{evals}\left(\mathtt{BPBody}_1\right)\right)$$

$$\mathrm{P}_8 = \mathrm{Com}_1\left(\mathbf{divs}_1(\mathtt{BPBody}_0), \mathbf{basis}(\mathbf{BodyEvals}_0)\right)$$

In the next round, the prover will commit to the error terms for the inner Bulletproof. This will use the inital circuit representation from the Circuits section, where each of the remaining commitment-constraint multiplications is assigned a pole. There are a variable number of such commitments depending on the size of the divisors and evaluations. In practice, there will not be more than 6 of these commitments, so with the additional commitment to fragments there will be at most 7 commitments. Each of these has three constraint vectors, which yields a total of 42 error terms.

$$\mathtt{BodyErrs}_0 = \mathtt{Circ}\left(\mathrm{Frag}_1, \mathbf{BodyDivs}_0, \mathbf{BodyEvals}_0\right)$$

$$\mathrm{P}_9 = \mathrm{Com}_1\left(\mathbf{divs}_2(\mathtt{BPBody}_0), \mathrm{Com}_0\left(\mathbf{errs}(\mathtt{BodyErrs}_0)\right)\right)$$

Following the commitment to the error terms for the inner proof, the inner proof using the circuit argument described earlier will publish the scalar v_1, and at the same time the prover will begin an ECIP proof for the remaining commitments of the inner Bulletproof. The prover will also publish the evaluations for the initial ECIP proof for the inner Bulletproof and begin the incremental error term construction, and the first set of responses for the inner Bulletproof argument. The remaining points can include points from the poles inner product that did not fit in the original body proof as well as the commitments to the outer body ECIP. As written, the second ECIP only includes the outer body commitments, but it can be modified without any additional changes to include the scaled commitments from the membership proof. The points of round 0 are the first round of Bulletproof responses, as will be elaborated in the next section.

$$\mathtt{OuterBody} = \mathtt{ECIP}\left(\mathrm{Q}_0, \mathbf{coeffs}(\mathtt{BodyErrs}_0), \mathbf{Points}(\mathtt{BodyErrs}_0)\right)$$

$$\mathrm{P}_{10} = \mathrm{Com}_1\left(\mathbf{divs}\left(\mathtt{OuterBody}\right), \mathbf{evals}_0(\mathtt{BPBody}_0), \mathbf{points}\left(\mathtt{Round}_0\right)\right)$$

6.3 Inner Bulletproof Argument

The rest of the protocol, apart from the final round which commits to error terms for the outer Bulletproof, performs the inner Bulletproof argument in zero knowledge. Each round of the Bulletproof involves an independent ECIP circuit, each of which takes three rounds in isolation: one for the points, one for the divisor witnesses, and one for the evaluations. To reduce the number of actual rounds, the prover will use a pipelined structure, where each commitment will commit to points for the current round, divisors for the previous round, and evaluations for the round before that. This pipelined structure is also well suited to the incremental error term construction.

The final round of the protocol will include a private scalar multiplication for the linear verification step, and so will behave slightly differently. It will not commit to any new responses, as there are none, and will also not commit to the evaluations of the second to last round. It will perform a round of error term reduction. The final round will commit to the evaluations for the previous two rounds as well as all the remaining error terms. Since there are no divisor witnesses in the this commitment, there is more space available for this.

Unlike the rest of the protocol, which will work with any witness lengths, this portion of the proof has a complex relationship with witness lengths and the number of rounds. For a generalized Bulletproof, the size of the witnesses grows with the sum of the round sizes, whereas the capacity of the Bulletproof argument grows with the product of the round sizes. Assuming that w_0 and w_1 are approximately the same size, as they grow the number of rounds necessary to prove the inner argument will decline asymptotically to two. As they shrink, the number of rounds will decline to the binary logarithm of the witness lengths until the ECIP proofs will not fit in a single round.

Since the outer Bulletproof argument increases with the log of the witness size, increasing witness lengths past a certain point, while it will reduce the number of rounds necessary to perform the inner proof, will still increase the proof size. The witness length of $w_0 = 1536$ seems to result in the minimal proof size using this technique, as increasing the witness enough to eliminate a round will add at least one more scalar or curve point to the outer proof. The maximum inner witness length that fits with w_0 is $w_1 = 1764$, which corresponds to round sizes of $\mathbf{l} = (4, 7, 7, 9)$.

6.3.1 Four Round Instantiation

Each of these ECIPs forms a distinct circuit defined over all the previous circuits' witnesses and the incremental circuit construction will be used to reduce the number of error terms. The first circuit this will be applied to is the inner Bulletproof body ECIPs, and each subsequent error term object will be instantiated with the subsequent circuit. The number of error terms per round can be dynamically tuned to fill all the available space, if for example the divisor witness in a particular round is smaller than average, but the error term sizes $\mathbf{n} = (10, 50, 50, 25)$ will fit on average and will reduce the error terms fast enough.

$$\mathtt{Errs}_0 = \mathtt{IncErrs}(\mathtt{BPBody}_0, \mathtt{OuterBody}, n_0) \qquad \mathtt{Errs}_{i+1} = \mathtt{IncErrs}(\mathtt{Round}_i, \mathtt{Errs}_i, n_i)$$

Letting the commitments Q_0 be the linear combination of inner membership body commitments as defined before, each of the first three rounds commits to a different round of a Bulletproof ECIP in a pipelined fashion.

$$\mathtt{Arg}_i = \mathtt{BP}(\mathrm{Q}_i, l_i) \qquad \mathtt{Round}_i = \mathtt{ECIP}(\mathrm{Q}_i, \mathbf{e}(\mathtt{Arg}_i), \mathbf{L}(\mathtt{Arg}_i) \oplus \mathbf{R}(\mathtt{Arg}_i))$$

$$\mathrm{Q}_{i+1} = \langle \mathbf{e}(\mathtt{Arg}_i), \mathbf{L}(\mathtt{Arg}_i) \oplus \mathbf{R}(\mathtt{Arg}_i) \rangle$$

$$\mathrm{P}_{11} = \mathrm{Com}_1\left(\mathbf{evals}(\mathtt{BPBody}_0), \mathbf{evals}\left(\mathtt{OuterBody}\right), \mathbf{divs}\left(\mathtt{Round}_0\right), \mathbf{points}\left(\mathtt{Round}_1\right), \mathbf{errs}\left(\mathtt{Errs}_0\right)\right)$$

$$\mathrm{P}_{12} = \mathrm{Com}_1\left(\mathbf{evals}\left(\mathtt{Round}_0\right), \mathbf{divs}\left(\mathtt{Round}_1\right), \mathbf{points}\left(\mathtt{Round}_2\right), \mathbf{errs}\left(\mathtt{Errs}_1\right)\right)$$

$$\mathrm{P}_{13} = \mathrm{Com}_1\left(\mathbf{evals}\left(\mathtt{Round}_1\right), \mathbf{divs}\left(\mathtt{Round}_2\right), \mathbf{points}\left(\mathtt{Round}_3\right), \mathbf{errs}\left(\mathtt{Errs}_3\right)\right)$$

For the fourth round of divisors, the prover will also include the final scalar opening of the fully reduced vector as a secret scalar multiplication, which costs 80 scalars. In order to accommodate this value and the larger final round size the evaluations from round 2 will be deferred a round and committed along with the evaluations from the divisors of round 3.

$$\mathrm{Q}_3 = s_3(c\mathrm{G} + \mathrm{G}') \qquad \mathtt{Round}_3 = \mathtt{ECIP}(\mathrm{Q}_3, \mathbf{e}(\mathtt{Arg}_3), \mathbf{L}(\mathtt{Arg}_3) \oplus \mathbf{R}(\mathtt{Arg}_3), s_3, c\mathrm{G} + \mathrm{G}')$$

$$\mathrm{P}_{14} = \mathrm{Com}_1\left(\mathbf{divs}\left(\mathtt{Round}_3\right), \mathbf{errs}\left(\mathtt{Errs}_4\right)\right)$$

The final round then commits to all the remaining evaluations from the final two rounds and all the remaining error terms.

$$\mathrm{P}_{15} = \mathrm{Com}_1 \left(\mathbf{evals}\left(\mathtt{Round}_2\right), \mathbf{evals}\left(\mathtt{Round}_3\right), \mathbf{errs}\left(\mathtt{Errs}_5\right)\right)$$

6.3.2 Batch Verification

Bulletproofs support efficient batch verification. Since the verification step consists of a single ECIP computation it is possible for the verifier to take a random linear combination of Bulletproofs over the same basis and take the same random linear combination of the openings of the proofs to verify. Since elliptic curve operations are significantly more computationally expensive to perform than field operations, this makes the marginal cost of verifying an additional Bulletproof very small as it eliminates the vast majority of the elliptic curve scalar multiplications.

It is important for the nested Bulletproof to retain this property. The outer proof can use exactly the same batch verification as the other Bulletproof protocols. Since the responses that make up the Bulletproof body of the inner proof are not published, the original Bulletproof batch verification cannot be applied directly to the inner proof. Instead, the prover will include the final, public basis point for the private scalar multiplication along with the proof transcript. While it is possible for the verifier to compute this point, that requires a large number of scalar multiplications and cannot be efficiently batched. By including the point, the verifier can simply check that it is of the correct form, which can be efficiently batched.

While it might be possible to save a round of the inner Bulletproof by reducing the proof to larger final vector, this would either break the batchable verification, in the case that the public points are not included with the proof, or would increase the proof size by one point per scalar in the final vector. This makes it most efficient, from the perspective of proof size, to reduce the inner Bulletproof all the way.

6.4 Outer Bulletproof Argument

Following the final commitment of the inner Bulletproof argument, the prover will commit to one more round of error terms for the outer bulletproof before engaging the argument. This will use the PLI technique for the final circuit. Note that there are a total of 11 commitments, but that through the incremental error term reduction technique only the last 4 will have all three constraint vectors. The remaining 7 commitments will have a single constraint for a total of 21 constraints. These each require 2 scalars for 44 total reserved terms. Note that commitments P_{14} and P_{15} only have 2 of the possible 3 constraints due to the circuit structure, which saves 4 terms for a total of 40 reserved terms.

$$\mathtt{BodyErrs}_1 = \mathtt{Circ}(\mathrm{X}, \mathrm{P}_6, \mathrm{P}_7, ..., \mathrm{P}_{15})$$

$$\mathrm{P}_{16} = \mathrm{Com}_1 \left(\mathbf{errs}(\mathtt{BP\text{-}Body}_1)\right)$$

Following P_{16} and the subsequent commitment to v_0, the prover will perform the linear Bulletproof argument. For a vector of length 1536, this involves 9 rounds resulting in a final vector length of 3 and 18 total response commitments. Since all the information in the Bulletproof phase is public, it is not necessary to blind the proof before performing the argument.

6.5 Maximum Transaction Size

From a certain perspective, the maximum transaction size with minimal witness lengths does not actually matter. The transaction can be treated as fixed from the perspective of the protocol and, if it does not fit, the witness lengths can be increased until it does. However, it is useful for the purpose of comparison to other protocols to determine exactly how many inputs and outputs can fit in a single transaction without increasing the witness length. The answer is approximately 1000 outputs or 50 to 36 inputs for $d = 32$ to $d = 48$. This transaction size is substantially exceeds the typical Bitcoin transaction size [55].

To calculate the maximum transaction size, it is easiest to work backwards given the witness lengths $w_0 = 1536$ and $w_1 = 1744$. The final round will reserve 30 group elements for the final error terms, but otherwise the Bulletproof argument rounds do not impose significant restrictions on the proof capacity. The commitment P_{11} must commit to the following data

$$\mathrm{len}\left(\mathbf{points}(\mathtt{Round}_1)\right) = 4(2(7-1)) = 48$$

$$\mathrm{len}\left(\mathbf{divs}(\mathtt{Round}_0)\right) = 243 + 72(2(4-1) - 1/2) = 639$$

$$\mathrm{len}\left(\mathbf{errs}(\texttt{Errs}_0)\right) < 20$$

For a total occupied space of 736 scalars, leaving 760 available for the body evaluations. The spine requires 162 scalars, and each multiplication in the body requires 10 scalars for the evaluations, which places an upper bound on the total number of multiplications of $860 - 162 > 10(59)$. The previous round also commits to evaluations as well as a round of divisors. The only reserved space in P_{10} is for the error terms and the points, for a total of 64 scalars since

$$\mathrm{len}\left(\mathbf{points}(\texttt{Round}_0)\right) = 4(2(4-1)) = 24$$

This leaves 1472 scalars available for the divisors and the evaluations. For simplicity, all scalar multiplications will be assumed to have 72 non-zero digits although the total number can vary slightly. The divisor commitment requires 243 values for the spine, as does the evaluation commitment require 162, leaving 1067 scalars. Counting evaluations and divisors together, for a total of 82 per point, this fits 13 points, one of which is the commitment to $\mathbf{errs}(\texttt{BP-Body}^{(0)})$ and a variable number of which commit to the outer Bulletproof body proof.

The amount of space available in P_7, P_8, and P_9 for scalar multiplications is 3 times the witness per commitment minus the space for the spine, or 4245 scalars. As each multiplication takes 72 scalars on average, this leaves 58.9 scalar multiplications on average, which is less than the 59 maximum. Along with the 13 scalar multiplications in P_{10}, this leaves a total of 71 scalar multiplications. These multiplications must be distributed among the outer Bulletproof body, the odd membership proof, and the outer transaction body, as well as a fixed number of commitments required by each of these phases. The optimal distribution will vary according to the structure of the transaction.

There are seven required multiplications for, in reverse order: the inner BP body error terms, the membership fragments, two for the membership blinding, one for the even path sum, one for the transaction body blinding, and one for the nullifier commitment Null from P_0. This leaves 64 total commitments for the variable sized portions of the transaction, which include the $\mathbf{D}$ and $\mathbf{R}$ commitments, the membership witness commitments, and the outer Bulletproof body commitments. The optimal layout of the transaction is complicated by the fact that input membership proofs have approximately symmetric cost in each membership proof, while output membership proofs have approximately twice the cost in the even membership proof. Additionally, the output membership proofs are efficient enough that the cost of range proofs in the transaction body is a non-trivial component of the maximum transaction capacity.

6.5.1 Membership Witness Size

Recall the formulae for the total membership witness size were calculated in the membership witness section to be

$$\mathrm{len}\left(\mathbf{memwit}_0\right) = (353 + 69d_0)n + (188/3)m + 288$$

$$\mathrm{len}\left(\mathbf{memwit}_1\right) = (611 + 72d_1)n + (120/3)m + 340$$

The values of d_0 and d_1 are approximately half of the depth of the tree, which for the purposes of a cryptocurrency has been assumed to be less than 48. At this level, the per input witness sizes were calculated to be 2009 and 2339 scalars respectively. These are each about 1.3 times larger than the witness length, and so it is reasonable to estimate that three inputs require about four commitments. Since the tranasction body and the output proof require at least six commitments, this leaves 58 commitments for the inputs. The outer body will require at least 3 commitments as well, leaving 55 commitments with room for about 41 inputs. At depths of 32, which still exceeds the total number of extant Bitcoin outputs, the per input witness length is approximately equal to the per commitment witness length, 1457 and 1759, which brings the estimated maximum number of inputs closer to 59.

To calculate the maximum number of outputs, it is necessary to also include the cost per output in the transaction body. This brings the total per output cost to $188/3 \approx 63$ and $90 = 40 + 2 * 25$, which is about 20 outputs per commitment. While this counts the transaction body commitments, it is necessary to deduct at least three commitments for the single input, leaving the same number of commitments available for outputs, 58, in the inner body available for 964 total estimated outputs. The maximum number of outputs does not depend on the d parameter, so the estimated ratio of maximum number of inputs to outputs depends on d. It should additionally be noted that this is estimate is intended to be a lower bound.

6.5.2 Asymptotic Proof Capacity

When increasing the witness sizes w_0 and w_1, suppose that the prover maintains approximately fixed ratio between the proof sizes. Then, a linear increase in the outer witness length yields an asymptotically linear increase in the number of scalar multiplications that can be performed in a fixed number of rounds as part of the inner Bulletproof body. Since the inner witness length is also proportionally increased, this increases the amount of witness space available as part of the inner Bulletproof body for a fixed number of outer commitments *quadratically* in factor of increase. In fact, if the number of commitments to the outer Bulletproof body is allowed to proportionally increase as well, this produces a cubic increase in the available proof space for the outer Bulletproof body. For simplicity, I do not attempt to exploit this potential third layer of nesting, and instead opt to maintain a fixed ratio of outer and inner Bulletproof body space that is quadratic in the factor of increase.

This quadratic increase is the reason for the 32 factor in the proof size calculation rather than 64, as each doubling of the Bulletproof witness lengths actually quadruples the proof capacity. It is also possible to define the proof capacity here as the witness length where the total witness size fits in the transaction divided by the minimal witness lengths.

$$c(d,n,m) = \max_b \left\lceil \frac{\text{len}\left(\mathbf{memwit}_b(n,m,d)\right)}{w_b} \right\rceil$$

The square root of this quantity is the amount both witness lengths need to be scaled to fit the given (d,n,m) values. When the witness lengths are increased linearly like this, the number of error terms for the transaction body and for the membership proofs increase only linearly, so increasing the witness sizes will not cause them to exceed the space allocated to handle them. Note also that the Bulletproof argument commitment increase linearly, but the maximum witness size they can reduce increases exponentially in the number of rounds. As a result, for larger witness sizes it will eventually become feasible to reduce the inner Bulletproof witness in fewer rounds, although for correctness it is sufficient to observe that it will always be possible to reduce the witness in the same number of rounds as the described protocol.

6.5.3 Increasing Nesting

It is alternatively possible to increase the proof capacity by increasing the depth of nesting. Each additional layer of nesting requires an ECIP protocol, so at least two rounds. However, the increase in proof capacity is exponential in the depth of nesting, while it is merely polynomial in the factor of increase of the witness length. Adapting the protocol to support more nesting, while not trivial, should be possible using more or less the same techniques as the described protocol.

Changing the protocol in this way would not result in a succinct verifier, although I suspect it is probably possible to adapt an alternative method of arithmetization like PLoNK to make the verifier work linear in the depth of nesting, rather than the overall proof size. This would require somewhat more work, but is fairly similar to how Halo 2 plans to eventually support recursive proof verification, and I cannot see any fundamental obstacle to eventually making something like this work. However, unlike Halo and other accumulation schemes, this would ultimately yield a proof system with a fully succinct verifier.

6.5.4 Improved ECIPs

In the membership phase, specifically commitments P_3, P_4 and P_5 the prover commits to a large number of points that will ultimately be using in an ECIP proof, but the proof is not actually constructed until commitment P_7. These additional rounds present an opportunity for the prover to commit to additional basis points, which can reduce the size of the witnesses in the Bulletproof phase. This will likely net a larger witness size overall, but in the membership phase commitments there is plenty of available unused witness and the savings would occur on the critical path. Savings in this portion of the proof directly translate into increased proof capacity. More generally, there may be more efficient ECIP techniques based on other kinds of addition chains that could reduce the witness size of the Bulletproof phase.

6.5.5 Alternative Tree Layouts

This protocol is defined using a binary tree, but it can easily be generalized to an arbitrary tree base or even a mixed base tree. Increasing the base size increases the amount and rate of Merkle witness information that must be recorded to spend an output, but it also reduces the witness size for the proof. For example using a base 4 tree would reduce the number of scalar multiplications in each input membership proof by a factor of 2. Using a mixed base tree might be better able to take advantage of asymmetric proofs.

6.6 Protocol

The first two protocols are essentially restatements of the transaction body and membership phase protocols using the tree commitments of the Bulletproof phase. The third protocol is the Bulletproof phase. Note that while these are given as separate protocols, the proof transacript of later protocols should be inherited from earlier protocols. That is, when instantiating using the Fiat Shamir heuristic, each has function should be evaluated over all the preceeding commitments, not just the commitments from the particular phase where the hash evaluated. The final proof size comes from the 18 commitments $\mathrm{P}_0, ..., \mathrm{P}_16, \mathrm{X}$ plus one for $c\mathrm{G} + \mathrm{G}'$, the two scalars v_0, v_1, and the 18 commitments and 3 scalars from the outer Bulletproof argument. This yields a total size of $37g + 5s = 1344$ where g and s are the size of the group elements and scalars respectively, with the final value occurring at $g = s = 32$ bytes. Signs for the curve points may require an additional 5 bytes.

Algorithm 3 Transaction Body Phase

```
  commit $\mathrm{P}_0$
  query $e(\mathrm{K}_i) \leftarrow \mathrm{half}(F_1)$                         ▷ Receive nullifier challenge
  commit $r(\mathrm{N}_i)$                                                     ▷ Publish nullifier responses
  query $e(\mathrm{CO}_j), e(\mathrm{CI}_i), e(\mathrm{K}_i) \leftarrow \mathrm{half}(F_1)$   ▷ Receive IO PoK and nullifier key challenges
  query $e, r \leftarrow F_1$                                                  ▷ also challenges for range proof, error terms
  commit $\mathrm{P}_1$                                                        ▷ Commit to IO PoK opening, nullifiers and their openings
  query $x, q, e(\mathrm{Sig}), M \leftarrow F_1$                              ▷ Receive range proof challenges and blinding matrix
  commit $\mathrm{P}_3$
  query $t \leftarrow F_1$
  publish open(TXBody), wit $(\mathrm{Bl}(\mathrm{Sig}) + e(\mathrm{Sig})\mathrm{Sig})$
```

Algorithm 4 Membership Proof Phase

```
Require: execute TXBody
  commit $\mathrm{P}_4$
  query $\mathbf{a}_0 \leftarrow \mathrm{half}(F_0), \mathbf{a}_1 \leftarrow \mathrm{half}(F_1)$   ▷ Challenges for path sum
  query $e_0(\mathtt{TXBodyErrs}) \leftarrow F_0$
  commit $\mathrm{P}_5$
  query $\mathrm{A}_0(\mathtt{Mem}_0) \leftarrow E_0, \mathrm{A}_0(\mathtt{Mem}_1) \leftarrow E_1$   ▷ Challenges for ECIPs
  query $e_1(\mathtt{TXBodyErrs}) \leftarrow F_0$
  commit $\mathrm{P}_6$
  query $e_0(\mathtt{Mem}_0) \leftarrow F_0, e_0(\mathtt{Mem}_1) \leftarrow F_1$   ▷ Challenges for individual circuits
  query $e_2(\mathtt{TXBodyErrs}) \leftarrow F_0$
  commit $\mathrm{P}_7$
  query $e_1(\mathtt{Mem}_0) \leftarrow F_0, e_1(\mathtt{Mem}_1) \leftarrow F_1$   ▷ Challenges for combined circuit
  publish open($\mathtt{MemErrs}_0$), open($\mathtt{MemErrs}_1$)             ▷ Opening used by Bulletproof provers
```

Algorithm 5 Bulletproof Proof Phase

```
Require: execute $\texttt{TXBody}, \texttt{Mem}$
  commit $\mathrm{P}_7, \mathrm{X}$
  query $\mathrm{A}_0(\texttt{BPBody}_1) \leftarrow E_1$    ▷ Challenge point only for outer body proof
  commit $\mathrm{P}_8$
  query $\mathrm{e}(\texttt{BPBody}_1) \leftarrow F_1$
  commit $\mathrm{P}_9$
  query $\mathrm{e}(\texttt{BodyErrs}_0) \leftarrow F_1, \mathrm{A}_0(\texttt{BPBody}_0) \leftarrow E_0$
  commit $v_1$
  query $t(\texttt{BodyErrs}_0) \leftarrow F_0$    ▷ Final challenge to combine constraints
  commit $\mathrm{P}_{10}$
  query $q(\texttt{Errs}_0) \leftarrow F_0, \mathrm{A}_0(\texttt{OuterBody}_0) \leftarrow E_0, e(\texttt{Arg}_0) \leftarrow F_0$
  commit $\mathrm{P}_{11}$
  query $e(\texttt{Errs}_0) \leftarrow F_0, q(\texttt{Errs}_1) \leftarrow F_0, \mathrm{A}_0(\texttt{Round}_0) \leftarrow E_0, e(\texttt{Arg}_1) \leftarrow F_0$
  commit $\mathrm{P}_{12}$
  query $e(\texttt{Errs}_1) \leftarrow F_0, q(\texttt{Errs}_2) \leftarrow F_0, \mathrm{A}_0(\texttt{Round}_1) \leftarrow E_0, e(\texttt{Arg}_2) \leftarrow F_0$
  commit $\mathrm{P}_{13}$
  query $e(\texttt{Errs}_2) \leftarrow F_0, q(\texttt{Errs}_3) \leftarrow F_0, \mathrm{A}_0(\texttt{Round}_2) \leftarrow E_0, e(\texttt{Arg}_3) \leftarrow F_0$
  commit $\mathrm{P}_{14}$
  query $e(\texttt{Errs}_3) \leftarrow F_0, \mathrm{A}_0(\texttt{Round}_3) \leftarrow E_0$
  commit $\mathrm{P}_{15}$
  query $e(\texttt{Round}_2), e(\texttt{Round}_3)$    ▷ Use PLI for remaining challenges
  commit $\mathrm{P}_{16}$
  query $e(\texttt{BodyErrs}_1)$
  commit $v_0$
  query $t(\texttt{BodyErrs}_1) \leftarrow F_0$    ▷ Final challenge to combine constraints
  execute Bulletproof Argument on $\langle \mathbf{coeffs}(\texttt{BodyErrs}_1), \mathbf{Points}(\texttt{BodyErrs}_1) \rangle$
```

References

[1] Daira Hopwood et al. *Zcash Protocol Specification Version 2022.3.4.* 2022. URL: `https://zips.z.cash/protocol/protocol.pdf`.

[2] Ralph C Merkle. “Protocols for public key cryptosystems”. In: *Secure communications and asymmetric cryptosystems.* Routledge, 2019, pp. 73–104.

[3] Torben Pryds Pedersen. “Non-interactive and information-theoretic secure verifiable secret sharing”. In: *Annual international cryptology conference.* Springer. 1991, pp. 129–140.

[4] Joseph H. Silverman and Katherine E. Stange. “Amicable Pairs and Aliquot Cycles for Elliptic Curves”. In: *Experimental Mathematics* 20.3 (2011), pp. 329–357. DOI: `10.1080/10586458.2011.565253`. URL: `https://doi.org/10.1080%2F10586458.2011.565253`.

[5] Matteo Campanelli and Mathias Hall-Andersen. *Curve Trees: Practical and Transparent Zero-Knowledge Accumulators.* Cryptology ePrint Archive, Paper 2022/756. `https://eprint.iacr.org/2022/756`. 2022. URL: `https://eprint.iacr.org/2022/756`.

[6] Liam Eagen. *Zero Knowledge Proofs of Elliptic Curve Inner Products from Principal Divisors and Weil Reciprocity.* Cryptology ePrint Archive, Paper 2022/596. `https://eprint.iacr.org/2022/596`. 2022. URL: `https://eprint.iacr.org/2022/596`.

[7] Benedikt Bünz et al. *Bulletproofs: Short Proofs for Confidential Transactions and More.* Cryptology ePrint Archive, Report 2017/1066. `https://ia.cr/2017/1066`. 2017.

[8] Liam Eagen. *Bulletproofs++.* Cryptology ePrint Archive, Report 2022/510. `https://ia.cr/2022/510`. 2022.

[9] Andrew Poelstra. *Scriptless Scripts.* 2017. URL: `https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-05-milan-meetup/slides.pdf`.

[10] Andrew Poelstra. *Lightning in Scriptless Scripts.* 2017. URL: `https://lists.launchpad.net/mimblewimble/msg00086.html`.

[11] Sean Bowe, Jack Grigg, and Daira Hopwood. *Recursive Proof Composition without a Trusted Setup.* Cryptology ePrint Archive, Report 2019/1021. `https://ia.cr/2019/1021`. 2019.

[12] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* 2009. URL: `https://bitcoin.org/bitcoin.pdf`.

[13] Jonas David Nick. “Data-driven de-anonymization in bitcoin”. MA thesis. ETH-Zürich, 2015.

[14] Hao Hua Sun Yin et al. “Regulating cryptocurrencies: a supervised machine learning approach to de-anonymizing the bitcoin blockchain”. In: *Journal of Management Information Systems* 36.1 (2019), pp. 37–73.

[15] URL: `https://bitcointalk.org/index.php?topic=770.0`.

[16] Li Peng et al. “Privacy preservation in permissionless blockchain: A survey”. In: *Digital Communications and Networks* 7.3 (2021), pp. 295–307.

[17] Greg Maxwell. *CoinJoin: Bitcoin privacy for the real world.* 2013. URL: `https://bitcointalk.org/?topic=279249`.

[18] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. “Threshold ring signatures and applications to ad-hoc groups”. In: *Annual International Cryptology Conference.* Springer. 2002, pp. 465–480.

[19] Koe, Kurt M. Alonzo, and Sarang Noether. *Zero to Monero: Second Edition.* `https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf`. 2020.

[20] Jens Groth. *On the Size of Pairing-based Non-interactive Arguments.* Cryptology ePrint Archive, Report 2016/260. `https://ia.cr/2016/260`. 2016.

[21] US Department of Treasury. *U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash.* 2022. URL: `https://home.treasury.gov/news/press-releases/jy0916`.

[22] Adam Back. URL: `https://bitcointalk.org/index.php?topic=305791.0`.

[23] Dimaz Ankaa Wijaya et al. “Monero Ring Attack: Recreating Zero Mixin Transaction Effect”. In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE).* 2018, pp. 1196–1201. DOI: `10.1109/TrustCom/BigDataSE.2018.00165`.

[24] Ian Miers et al. “Zerocoin: Anonymous distributed e-cash from bitcoin”. In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 397–411.

[25] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Paper 2019/953. `https://eprint.iacr.org/2019/953`. 2019. URL: `https://eprint.iacr.org/2019/953`.

[26] *halo2*. 2022. URL: `https://zcash.github.io/halo2/index.html`.

[27] Benedikt Bünz et al. *Proof-Carrying Data without Succinct Arguments*. Cryptology ePrint Archive, Paper 2020/1618. `https://eprint.iacr.org/2020/1618`. 2020. URL: `https://eprint.iacr.org/2020/1618`.

[28] Aram Jivanyan. *Lelantus: A New Design for Anonymous and Confidential Cryptocurrencies*. Cryptology ePrint Archive, Paper 2019/373. `https://eprint.iacr.org/2019/373`. 2019. URL: `https://eprint.iacr.org/2019/373`.

[29] Russell W. F. Lai et al. *Omniring: Scaling Up Private Payments Without Trusted Setup - Formal Foundations and Constructions of Ring Confidential Transactions with Log-size Proofs*. Cryptology ePrint Archive, Paper 2019/580. `https://eprint.iacr.org/2019/580`. 2019. DOI: `10.1145/3319535.3345655`. URL: `https://eprint.iacr.org/2019/580`.

[30] Riad S. Wahby et al. “Doubly-Efficient zkSNARKs Without Trusted Setup”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 926–943. DOI: `10.1109/SP.2018.00060`.

[31] Jonathan Lee. *Dory: Efficient, Transparent arguments for Generalised Inner Products and Polynomial Commitments*. Cryptology ePrint Archive, Paper 2020/1274. `https://eprint.iacr.org/2020/1274`. 2020. URL: `https://eprint.iacr.org/2020/1274`.

[32] Matteo Campanelli and Mathias Hall-Andersen. *Veksel: Simple, Efficient, Anonymous Payments with Large Anonymity Sets from Well-Studied Assumptions*. Cryptology ePrint Archive, Paper 2021/327. `https://eprint.iacr.org/2021/327`. 2021. URL: `https://eprint.iacr.org/2021/327`.

[33] John Kuszmaul. *Verkle Trees*. 2019. URL: `https://klein.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf`.

[34] Dankrad Feist. 2021. URL: `https://dankradfeist.de/ethereum/2021/06/18/verkle-trie-for-eth1.html`.

[35] David Chaum. “Blind signatures for untraceable payments”. In: *Advances in cryptology*. Springer. 1983, pp. 199–203.

[36] Pieter Wuille, Jonas Nick, and Anthony Towns. *BIP 0341*. URL: `https://en.bitcoin.it/wiki/BIP_0341`.

[37] Andrew Poelstra et al. *Adaptor Signatures and Atomic Swaps from Scriptless Scripts*. URL: `https://github.com/ElementsProject/scriptless-scripts/blob/master/md/atomic-swap.md`.

[38] Jonathan Bootle et al. “Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting”. In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 327–357. ISBN: 978-3-662-49896-5.

[39] Eli Ben-Sasson et al. “Scalable zero knowledge via cycles of elliptic curves”. In: *Algorithmica* 79.4 (2017), pp. 1102–1160.

[40] Reinier Bröker. *Constructing elliptic curves of prescribed order*. Leiden University, 2006.

[41] René Schoof. “Counting points on elliptic curves over finite fields”. en. In: *Journal de Théorie des Nombres de Bordeaux* 7.1 (1995), pp. 219–254. URL: `http://www.numdam.org/item/JTNB_1995__7_1_219_0/`.

[42] Daira Hopwood. 2020. URL: `https://github.com/zcash/pasta`.

[43] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM Review* 41.2 (1999), pp. 303–332. DOI: `10.1137/S0036144598347011`. eprint: `https://doi.org/10.1137/S0036144598347011`. URL: `https://doi.org/10.1137/S0036144598347011`.

[44] Mihir Bellare and Phillip Rogaway. “Random oracles are practical: A paradigm for designing efficient protocols”. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993, pp. 62–73.

[45] Jim Miller. 2022. URL: `https://blog.trailofbits.com/2022/04/13/part-1-coordinated-disclosure-of-vulnerabilities-affecting-girault-bulletproofs-and-plonk/`.

[46] Josh Benaloh and Michael de Mare. “One-way accumulators: A decentralized alternative to digital signatures”. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1993, pp. 274–285.

[47] Dan Boneh, Benedikt Bünz, and Ben Fisch. “Batching techniques for accumulators with applications to IOPs and stateless blockchains”. In: *Annual International Cryptology Conference*. Springer. 2019, pp. 561–586.

[48] Heewon Chung et al. *Bulletproofs+: Shorter Proofs for Privacy-Enhanced Distributed Ledger*. Cryptology ePrint Archive, Report 2020/735. `https://ia.cr/2020/735`. 2020.

[49] Daira Hopwood et al. “Zcash protocol specification”. In: *GitHub: San Francisco, CA, USA* (2016), p. 68.

[50] Simon Masson, Antonio Sanso, and Zhenfei Zhang. *Bandersnatch: a fast elliptic curve built over the BLS12-381 scalar field*. Cryptology ePrint Archive, Paper 2021/1152. `https://eprint.iacr.org/2021/1152`. 2021. URL: `https://eprint.iacr.org/2021/1152`.

[51] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. “Constant-size commitments to polynomials and their applications”. In: *International conference on the theory and application of cryptology and information security*. Springer. 2010, pp. 177–194.

[52] Gregory Maxwell et al. *Simple Schnorr Multi-Signatures with Applications to Bitcoin*. Cryptology ePrint Archive, Paper 2018/068. `https://eprint.iacr.org/2018/068`. 2018. URL: `https://eprint.iacr.org/2018/068`.

[53] Peter Todd. *Merkle Mountain Ranges*. 2013. URL: `https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md`.

[54] Reza R Farashahi and Marc Joye. “Efficient arithmetic on Hessian curves”. In: *International Workshop on Public Key Cryptography*. Springer. 2010, pp. 243–260.

[55] Befekadu G. Gebraselase, Bjarne E. Helvik, and Yuming Jiang. “Transaction Characteristics of Bitcoin”. In: *CoRR* abs/2010.10858 (2020). arXiv: `2010.10858`. URL: `https://arxiv.org/abs/2010.10858`.