Lelantus Spark: Secure and Flexible Private Transactions

Aram Jivanyan[1,2⋆] and Aaron Feickert[3]

[1] Firo
[2] Yerevan State University
[3] Cypher Stack

Abstract. We propose a modification to the Lelantus private transaction protocol to provide recipient privacy, improved security, and additional usability features. Our decentralized anonymous payment (DAP) construction, Spark, enables non-interactive one-time addressing to hide recipient addresses in transactions. The modified address format permits flexibility in transaction visibility. Address owners can securely provide third parties with opt-in visibility into incoming transactions or all transactions associated to the address; this functionality allows for offloading chain scanning and balance computation without delegating spend authority. It is also possible to delegate expensive proving operations without compromising spend authority when generating transactions. Further, the design is compatible with straightforward linear multisignature operations to allow mutually non-trusting parties to cooperatively receive and generate transactions associated to a multisignature address. We prove that Spark satisfies formal DAP security properties of balance, non-malleability, and ledger indistinguishability.

1 Introduction

Distributed digital asset protocols have seen a wealth of research since the introduction of the Bitcoin transaction protocol, which enables transactions generating and consuming ledger-based outputs, and provides a limited but useful scripting capability. However, Bitcoin-type protocols have numerous drawbacks relating to privacy: a transaction reveals source addresses and amounts, and subsequent spends reveal destination addresses. Further, data and metadata associated with transactions, like script contents, can provide undesired fingerprinting of transactions.

More recent research has focused on mitigating or removing these limitations, while permitting existing useful functionality like multisignature operations or opt-in third-party transaction viewing. Designs in privacy-focused cryptocurrencies like Beam, Firo, Grin, Monero, and Zcash take different approaches toward this goal, with a variety of different tradeoffs. The RingCT-based protocol currently used in Monero, for example, practically permits limited sender anonymity

⋆ Corresponding author: aram@firo.org

due to the space and time scaling of its underlying signature scheme [25,15]. The Sprout and Sapling protocols supported by Zcash [17] (and their currently-deployed related updates) require trusted parameter generation to bootstrap their circuit-based proving systems, and interact with transparent Bitcoin-style outputs in ways that can leak information [5,7]. The Mimblewimble-based construction used as the basis for Grin can leak graph information prior to a merging operation performed by miners [13]. To mitigate Mimblewimble's linkability issue, Beam has designed and implemented into its system an adaption of Lelantus for use with the Mimblewimble protocol which enables obfuscation of the transaction graph [27]. The Lelantus protocol currently used in Firo does not provide recipient privacy; it supports only mints and signer-ambiguous spends of arbitrary amounts that interact with transparent Bitcoin-style outputs, which can leak information about recipient identity [18]. Seraphis [28] is a transaction protocol framework of similar design being developed concurrently.

Here we introduce Spark, an iteration on the Lelantus protocol enabling trustless private transactions which supports sender, recipient, and transaction amount privacy. Transactions in Spark, like those in Lelantus and Monero, use specified sender anonymity sets composed of previously-generated shielded outputs. A parallel proving system adapted from a construction by Groth and Bootle *et al.* [16,6] (of independent interest and used in other modified forms in Lelantus[18] and Triptych [24]) proves that a consumed output exists in the anonymity set; amounts are encrypted and hidden algebraically in Pedersen commitments, and a tag derived from a verifiable random function [11,20] prevents consuming the same output multiple times, which in the context of a transaction protocol would constitute a double-spend attempt.

Spark transactions support efficient verification in batches, where range and spend proofs can take advantage of common proof elements and parameters to lower the marginal cost of verifying each proof in such a batch; when coupled with suitably-chosen sender anonymity sets, the verification time savings of batch verification can be significant.

Spark enables additional useful functionality. The use of a modified Chaum-Pedersen discrete logarithm proof, which asserts spend authority and correct tag construction, enables efficient signing and multisignature operations similar to those of [21,19,10,3] where computationally-expensive proofs may be offloaded to more capable devices with limited trust requirements. The protocol further adds three levels of opt-in visibility into transactions without delegating spend authority. Incoming view keys allow a designated third party to identify transactions containing outputs destined for an address, as well as the corresponding amounts and encrypted memo data. Full view keys allow a designated third party to additionally identify when received outputs are later spent (but without any recipient data), which enables balance auditing and further enhances accountability in threshold multisignature applications where this property is desired. Payment proofs allow a sender to assert the destination, value, and memo of a coin while proving (in zero knowledge) that it knows the secret data used to

produce the coin; this permits more fine-grained disclosure without revealing view keys.

All constructions used in Spark require only public parameter generation, ensuring that no trusted parties are required to bootstrap the protocol or ensure soundness.

2 Cryptographic Preliminaries

Throughout this paper, we use additive notation for group operations. Let $\mathbb{N}$ be the set $\{0, 1, 2, \ldots\}$ of non-negative integers.

2.1 Pedersen Commitment Scheme

A homomorphic commitment scheme is a construction producing one-way algebraic representations of input values. The Pedersen commitment scheme is a homomorphic commitment scheme that uses a particularly simple linear combination construction. Let $pp_{\text{com}} = (\mathbb{G}, \mathbb{F}, G, H)$ be the public parameters for a Pedersen commitment scheme, where $\mathbb{G}$ is a prime-order group where the discrete logarithm problem is hard, $\mathbb{F}$ is its scalar field, and $G, H \in \mathbb{G}$ are uniformly-sampled independent generators. The commitment scheme contains an algorithm $\mathsf{Com} : \mathbb{F}^2 \to \mathbb{G}$, where $\mathsf{Com}(v, r) = vG + rH$ that is homomorphic in the sense that

$$\mathsf{Com}(v_1, r_1) + \mathsf{Com}(v_2, r_2) = \mathsf{Com}(v_1 + v_2, r_1 + r_1)$$

for all such input values $v_1, v_2 \in \mathbb{F}$ and masks $r_1, r_2 \in \mathbb{F}$. Further, the construction is perfectly hiding and computationally binding.

This definition extends naturally to a double-masked commitment scheme. Let $pp_{\text{comm}} = (\mathbb{G}, \mathbb{F}, F, G, H)$ be the public parameters for a double-masked Pedersen commitment scheme, where $\mathbb{G}$ is a prime-order group where the discrete logarithm problem is hard, $\mathbb{F}$ is its scalar field, and $F, G, H \in \mathbb{G}$ are uniformly-sampled independent generators. The commitment scheme contains an algorithm $\mathsf{Comm} : \mathbb{F}^3 \to \mathbb{G}$, where $\mathsf{Comm}(v, r, s) = vF + rG + sH$ that is homomorphic in the sense that

$$\mathsf{Comm}(v_1, r_1, s_1) + \mathsf{Comm}(v_2, r_2, s_2) = \mathsf{Comm}(v_1 + v_2, r_1 + r_2, s_1 + s_2)$$

for all such input values $v_1, v_2 \in \mathbb{F}$ and masks $r_1, r_2, s_1, s_2 \in \mathbb{F}$. Further, the construction is perfectly hiding and computationally binding.

2.2 Representation proving system

A representation proof is used to demonstrate knowledge of a set of discrete logarithms in zero knowledge. Let $pp_{\text{rep}} = (\mathbb{G}, \mathbb{F})$ be the public parameters for such a construction, where $\mathbb{G}$ is a prime-order group where the discrete logarithm problems is hard and $\mathbb{F}$ is its scalar field.

The proving system itself is a tuple of algorithms $(\mathsf{RepProve}, \mathsf{RepVerify})$ for the following relation:

$$\left\{pp_{\text{rep}}, G, \{Y_i\}_{i=0}^{l-1} \subset \mathbb{G}; \{y_i\}_{i=0}^{l-1} \subset \mathbb{F} : \forall i \in [0, l), Y_i = y_i G\right\}$$

We require that the proving system be complete, special honest-verifier zero knowledge, and special sound; these definitions are standard [16].

An aggregated Schnorr proving system, like that in [14], may be used for this purpose.

As a matter of notational convenience, we drop the set and subscript notation from these algorithms in the case where $l = 1$; this represents the case of a standard (non-aggregated) representation proof.

2.3 Modified Chaum-Pedersen Proving System

A Chaum-Pedersen proof is used to demonstrate discrete logarithm equality in zero knowledge. Here we require a modification to the standard proving system that uses additional group generators and supports multiple assertions within a single proof. Let $pp_{\text{chaum}} = (\mathbb{G}, \mathbb{F}, F, G, H, U)$ be the public parameters for such a construction, where $\mathbb{G}$ is a prime-order group where the discrete logarithm problem is hard, $\mathbb{F}$ is its scalar field, and $F, G, H, U \in \mathbb{G}$ are uniformly-sampled independent generators.

The proving system is a tuple of algorithms $(\mathsf{ChaumProve}, \mathsf{ChaumVerify})$ for the following relation:

$$\begin{aligned}\Big\{pp_{\text{chaum}}, \{S_i, T_i\}_{i=0}^{l-1} \subset \mathbb{G}^2; (\{x_i, y_i, z_i\}_{i=0}^{l-1}) \subset \mathbb{F}^3 : \\ \forall i \in [0, l), S_i = x_i F + y_i G + z_i H, U = x_i T_i + y_i G\Big\}\end{aligned}$$

We require that the proving system be complete, special honest-verifier zero knowledge, and special sound.

We present an instantiation of such a proving system in Appendix A, along with security proofs.

2.4 Parallel One-out-of-Many Proving System

We require the use of a parallel one-out-of-many proving system that shows knowledge of openings of commitments to zero at the same index among two sets of group elements in zero knowledge. In the context of the Spark protocol, this will be used to mask consumed coin serial number and value commitments for balance, ownership, and double-spend purposes. We show how to produce such a proving system as a straightforward modification of a construction by Groth and Kohlweiss [16] that was generalized by Bootle *et al.* [6], with a further optimization from Esgin *et al.* [12].

Let $pp_{\text{par}} = (\mathbb{G}, \mathbb{F}, n, m, pp_{\text{com}}, pp_{\text{comm}})$ be the public parameters for such a construction, where $\mathbb{G}$ is a prime-order group where the discrete logarithm problem is hard, $\mathbb{F}$ is its scalar field, $n > 1$ and $m > 1$ are integer-valued

size decomposition parameters, pp_{com} are the public parameters for a Pedersen commitment construction, and pp_{comm} are the public parameters for a double-masked Pedersen commitment construction.

The proving system itself is a tuple of algorithms $(\mathsf{ParProve}, \mathsf{ParVerify})$ for the following relation, where we let $N = n^m$:

$$\{pp_{\text{par}}, \{S_k, V_k\}_{k=0}^{N-1} \subset \mathbb{G}^2, S', V' \in \mathbb{G}; l \in \mathbb{N}, (s, v) \in \mathbb{F} : \\ 0 \leq l < N, S_l - S' = \mathsf{Comm}(0, 0, s), V_l - V' = \mathsf{Com}(0, v)\}$$

We require that the proving system be complete, special honest-verifier zero knowledge, and special sound.

We present an instantiation of such a proving system in Appendix B.

2.5 Key-Committing Authenticated Encryption Scheme

We require the use of an authenticated symmetric encryption with associated data (AEAD) scheme that commits to keys. In the context of the Spark protocol, this construction is used to encrypt value, memo, and other data for use by the recipient of a transaction. Note that in general, AEAD schemes need not commit to keys; because Spark payment proofs require this property, we require it in the overall protocol. It is possible to generically extend an AEAD scheme using the technique of [1] in a flexible manner by including a commitment to the key as part of the ciphertext, and by checking this commitment during authentication; we assume this approach when describing the algorithms here.

Let pp_{aead} be the public parameters for such a construction. The construction itself is a tuple of algorithms $(\mathsf{AEADKeyGen}, \mathsf{AEADEncrypt}, \mathsf{AEADDecrypt})$. Here $\mathsf{AEADKeyGen}$ is a key derivation function that accepts as input an arbitrary string, and produces a key in the appropriate key space. The algorithm $\mathsf{AEADEncrypt}$ accepts as input a key, associated data, and arbitrary message string, and produces ciphertext in the appropriate space. The algorithm $\mathsf{AEADDecrypt}$ accepts as input a key, associated data, and ciphertext string, and produces a message in the appropriate space if authentication succeeds (and fails otherwise).

Assume that such a construction is indistinguishable against adaptive chosen-ciphertext attacks (IND-CCA2) and key-private under chosen-ciphertext attacks (IK-CCA) in this context [2].

2.6 Symmetric Encryption Scheme

We require the use of a symmetric encryption scheme. In the context of the Spark protocol, this construction is used to encrypt diversifier indices used to produce public addresses.

Let pp_{sym} be the public parameters for such a construction. The construction itself is a tuple of algorithms $(\mathsf{SymKeyGen}, \mathsf{SymEncrypt}, \mathsf{SymDecrypt})$. Here $\mathsf{SymKeyGen}$ is a key derivation function that accepts as input an arbitrary string,

and produces a key in the appropriate key space. The algorithm $\mathsf{SymEncrypt}$ accepts as input a key and arbitrary message string, and produces ciphertext in the appropriate space. The algorithm $\mathsf{SymDecrypt}$ accepts as input a key and ciphertext string, and produces a message in the appropriate space.

Assume that such a construction is indistinguishable against adaptive chosen-ciphertext attacks (IND-CCA2) in this context.

2.7 Range Proving System

We require the use of a zero-knowledge range proving system. A range proving system demonstrates that a commitment binds to a value within a specified range. In the context of the Spark protocol, it avoids overflow that would otherwise fool the balance definition by effectively binding to invalid negative values. Let $pp_{\text{rp}} = (\mathbb{G}, \mathbb{F}, v_{\max}, pp_{\text{com}})$ be the relevant public parameters for such a construction, where pp_{com} are the public parameters for a Pedersen commitment construction.

The proving system itself is a tuple of algorithms $(\mathsf{RangeProve}, \mathsf{RangeVerify})$ for the following relation:

$$\{pp_{\text{rp}}, C \in \mathbb{G}; (v, r) \in \mathbb{F} : 0 \leq v \leq v_{\max}, C = \mathsf{Com}(v, r)\}$$

We require that the proving system be complete, special honest-verifier zero knowledge, and special sound.

In practice, an efficient instantiation like Bulletproofs [8] or Bulletproofs+ [9] may be used to satisfy this requirement.

3 Concepts and Algorithms

We now define the main concepts and algorithms used in the Spark transaction protocol.

Keys and addresses. Users generate keys and addresses that enable transactions. A set of keys consists of a tuple

$$(\mathsf{addr}_{\text{in}}, \mathsf{addr}_{\text{full}}, \mathsf{addr}_{\text{sk}}).$$

In this notation, $\mathsf{addr}_{\text{in}}$ is an incoming view key used to identify received funds, $\mathsf{addr}_{\text{full}}$ is a full view key used to identify outgoing funds and conduct certain computationally-heavy proving operations, and $\mathsf{addr}_{\text{sk}}$ is the spend key used to generate transactions. Spark addresses are constructed in such a way that a single set of keys can be used to construct any number of *diversified* public addresses that appear indistinguishable from each other or from public addresses produced from a different set of keys. Diversified addressing allows a recipient to provide distinct public addresses to different senders, but scan transactions on chain only once for identification and recovery of incoming coins destined for any of its diversified public addresses.

Coins. A coin encodes the abstract value which is transferred through the private transactions. Each coin is associated with:

– A secret nonce
– A recipient address
– An integer value
– A memo containing arbitrary recipient data

The recipient address and value are hidden using commitments. The nonce, a part of the recipient address, the value, and the memo are encrypted to the recipient (unless the value is made public as part of a mint operation).

Private Transactions. There are two types of private transactions in Spark:

– Mint transactions. A mint transaction generates new coins of public value destined for a recipient public address in a confidential way, either through a consensus-enforced mining process, or by consuming transparent outputs from a non-Spark base layer. In this transaction type, a representation proof is included to show that the minted coins are of the expected values.
– Spend transactions. A spend transaction consumes existing coins and generates new coins destined for one or more recipient public addresses in a confidential way. In this transaction type, a representation proof is included to show that the hidden input and output values are equal.

Tags. Tags are used to prevent coins from being consumed in multiple transactions. When generating a spend transaction, the sender produces the tag for each consumed coin and includes it on the ledger. When verifying transactions are valid, it suffices to ensure that tags do not appear on the ledger in any previous transactions. Tags are uniquely bound to validly-recoverable coins, but cannot be associated to specific coins without the corresponding full view key.

Algorithms. Spark is a decentralized anonymous payment (DAP) system defined as the following polynomial-time algorithms:

– Setup: This algorithm produces all public parameters used by the protocol and its underlying components. The setup process does not require any trusted parameter generation.
– CreateKeys: This algorithm produces keys that are used when constructing addresses, processing coins, and spending coins.
– CreateAddress: This algorithm produces diversified public addresses used for receiving coins.
– CreateCoin: This algorithm produces a coin of a given value that is destined for a recipient public address.
– Mint: This algorithm produces a transaction transferring public value to recipient public addresses.
– Identify: This algorithm processes a coin to determine if it is destined for a diversified address controlled by a recipient.
– Recover: This algorithm processes a coin to determine if it is destined for a diversified address controlled by a recipient, and produces additional data used for spending the coin or determining if it is already spent.
– Spend: This algorithm produces a transaction consuming existing coins and generating new coins of hidden value to recipient public addresses.

– Verify: This algorithm determines if a given transaction is valid.

We provide detailed descriptions below, and show security of the resulting protocol in Appendix C.

4 Algorithm Constructions

In this section we provide detailed description of the DAP scheme algorithms.

4.1 Setup

This algorithm produces public parameters required for the protocol. The security parameter and resulting public parameters are assumed to be available to all other algorithms, even where not specifically noted.

Inputs: Security parameter λ, size decomposition parameters $n > 1$ and $m > 1$, maximum value parameter $v_{\max}$

Outputs: Public parameters pp

1. Sample a prime-order group $\mathbb{G}$ in which the discrete logarithm, decisional Diffie-Hellman, and computational Diffie-Hellman problems are hard. Let $\mathbb{F}$ be the scalar field of $\mathbb{G}$.
2. Sample $F, G, H, U \in \mathbb{G}$ uniformly at random. In practice, these generators may be chosen using a suitable cryptographic hash function on public input.
3. Sample cryptographic hash functions

$$\mathcal{H}_k, \mathcal{H}_{Q_2}, \mathcal{H}_{\text{ser}}, \mathcal{H}_{\text{val}}, \mathcal{H}_{\text{ser}'}, \mathcal{H}_{\text{val}'}, \mathcal{H}_{\text{bind}} : \{0,1\}^* \to \mathbb{F}$$

   and

$$\mathcal{H}_{\text{div}} : \{0,1\}^* \to \mathbb{G}$$

   uniformly at random. In practice, these hash functions may be chosen using domain separation of a single suitable cryptographic hash function on public input.
4. Compute the public parameters $pp_{\text{com}} = (\mathbb{G}, \mathbb{F}, G, H)$ of a Pedersen commitment scheme.
5. Compute the public parameters $pp_{\text{comm}} = (\mathbb{G}, \mathbb{F}, F, G, H)$ of a double-masked Pedersen commitment scheme.
6. Compute the public parameters $pp_{\text{rep}} = (\mathbb{G}, \mathbb{F})$ of a representation proving system.
7. Compute the public parameters $pp_{\text{chaum}} = (\mathbb{G}, \mathbb{F}, F, G, H, U)$ of the modified Chaum-Pedersen proving system.
8. Compute the public parameters $pp_{\text{par}} = (\mathbb{G}, \mathbb{F}, n, m, pp_{\text{com}}, pp_{\text{comm}})$ of the parallel one-out-of-many proving system.
9. Compute the public parameters pp_{aead} of an authenticated symmetric encryption scheme.
10. Compute the public parameters pp_{sym} of a symmetric encryption scheme.
11. Compute the public parameters $pp_{\text{rp}} = (\mathbb{G}, \mathbb{F}, v_{\max}, pp_{\text{com}})$ of a range proving system.
12. Output all generated public parameters and hash functions as pp.

4.2 CreateKeys

We describe the construction of key types used in the protocol.

Inputs: Security parameter λ, public parameters pp

Outputs: Key tuple $(\mathsf{addr}_{\text{in}}, \mathsf{addr}_{\text{full}}, \mathsf{addr}_{sk})$

1. Sample $s_1, s_2, r \in \mathbb{F}$ uniformly at random, and let $D = \mathsf{Comm}(0, r, 0)$ and $P_2 = \mathsf{Comm}(s_2, r, 0)$.
2. Set $\mathsf{addr}_{\text{in}} = (s_1, P_2)$.
3. Set $\mathsf{addr}_{\text{full}} = (s_1, s_2, D, P_2)$.
4. Set $\mathsf{addr}_{\text{sk}} = (s_1, s_2, r)$.
5. Output the tuple $(\mathsf{addr}_{\text{in}}, \mathsf{addr}_{\text{full}}, \mathsf{addr}_{\text{sk}})$.

4.3 CreateAddress

This algorithm generates a *diversified* public address from an incoming view key. A given public address is privately and deterministically tied to an index called the *diversifier*. Diversified public addresses share the same set of keys for efficiency purposes, but are not linkable without non-public information.

Inputs: Security parameter λ, public parameters pp, incoming view key $\mathsf{addr}_{\text{in}}$, diversifier $i \in \mathbb{N}$

Outputs: Diversified address $\mathsf{addr}_{\text{pk}}$

1. Parse the incoming view key $\mathsf{addr}_{\text{in}} = (s_1, P_2)$.
2. Compute the diversified address components:

$$\begin{aligned} d &= \mathsf{SymEncrypt}(\mathsf{SymKeyGen}(s_1), i) \\ Q_{1,i} &= s_1 \mathcal{H}_{\text{div}}(d) \\ Q_{2,i} &= \mathsf{Comm}(\mathcal{H}_{Q_2}(s_1, i), 0, 0) + P_2 \end{aligned}$$

3. Set $\mathsf{addr}_{\text{pk}} = (d, Q_{1,i}, Q_{2,i})$ and output this tuple.

Note that we drop the diversifier index i from subsequent notation when referring to addresses in operations performed by entities other than the incoming view key holder, since such users are not provided this index and cannot compute it.

4.4 CreateCoin

This algorithm generates a new coin destined for a given public address. It uses a type bit to determine if the value is intended to be publicly visible.

Inputs: Security parameter λ, public parameters pp, destination public address $\mathsf{addr}_{\text{pk}}$, value $v \in [0, v_{\max})$, memo m, type bit b

Outputs: Coin Coin, nonce k

1. Parse the recipient address $\mathsf{addr}_{\text{pk}} = (d, Q_1, Q_2)$.
2. Sample a nonce $k \in \mathbb{F}$.
3. Compute the recovery key $K = \mathcal{H}_k(k)\mathcal{H}_{\text{div}}(d)$.

4. Compute the serial number commitment

$$S = \mathsf{Comm}(\mathcal{H}_{\text{ser}}(k), 0, 0) + Q_2.$$

5. Generate the value commitment $C = \mathsf{Com}(v, \mathcal{H}_{\text{val}}(k))$.
6. If $b = 0$, generate a range proof

$$\Pi_{\text{rp}} = \mathsf{RangeProve}(pp_{\text{rp}}, C; (v, \mathcal{H}_{\text{val}}(k)).$$

7. If $b = 0$, set the recipient data $r = (v, d, k, m)$; otherwise, set $r = (d, k, m)$.
8. Generate an AEAD encryption key $k_{\text{aead}} = \mathsf{AEADKeyGen}(\mathcal{H}_k(k)Q_1)$; encrypt the recipient data

$$\overline{r} = \mathsf{AEADEncrypt}(k_{\text{aead}}, \mathtt{r}, r).$$

9. If $b = 0$, output the coin $\mathsf{Coin} = (S, K, C, \Pi_{\text{rp}}, \overline{r})$ and nonce k; otherwise, output the coin $\mathsf{Coin} = (S, K, C, v, \overline{r})$ and nonce k.

The case $b = 0$ represents a coin with hidden value being generated in a spend transaction, while the case $b = 1$ represents a coin with plaintext value being generated in a mint transaction.

The nonce k is returned for use by other algorithms, but is not public.

4.5 Mint

This algorithm generates new coins from either a consensus-determined mining process, or by consuming non-Spark outputs from a base layer with public value. Note that while such implementation-specific auxiliary data may be necessary for generating such a transaction and included, we do not specifically list this here. Notably, the coin value used in this algorithm is assumed to be the sum of all public input values as specified by the implementation.

Inputs:

- Security parameter λ and public parameters pp
- A set of t output coin public addresses, values, and memos:

$$\{\mathsf{addr}_{\text{pk},j}, v_j, m_j\}_{j=0}^{t-1}$$

Outputs: Mint transaction tx_{mint}

1. Generate a set $\mathsf{OutCoins} = \{\mathsf{CreateCoin}(\mathsf{addr}_{\text{pk},j}, v_j, m_j, 1)\}_{j=0}^{t-1}$ of output coins.
2. Parse the output coin value commitments $\{\overline{C}_j\}_{j=0}^{t-1}$ from $\mathsf{OutCoins}$, where each $\overline{C}_j$ contains nonce k_j.
3. Generate a representation proof for value assertion:

$$\Pi_{\text{val}} = \mathsf{RepProve}\left(pp_{\text{rep}}, H, \{\overline{C}_j - \mathsf{Com}(v_j, 0)\}_{j=0}^{t-1}; \{\mathcal{H}_{\text{val}}(k_j)\}_{j=0}^{t-1}\right)$$

4. Output the mint transaction $\mathsf{tx}_{\text{mint}} = (\mathsf{OutCoins}, \Pi_{\text{val}})$.

4.6 Identify

This algorithm allows a recipient (or designated entity) to determine if it controls a coin; if so, it computes the value, memo, and diversifier from the coin (in addition to the coin nonce). It requires the incoming view key used to produce diversified addresses to do so. If the coin is not destined for any diversified address, the algorithm returns failure.

It is assumed that the recipient has run the Verify algorithm on the transaction generating the coin being identified.

Inputs: Security parameter λ, public parameters pp, incoming view key $\mathsf{addr}_{\text{in}}$, coin Coin

Outputs: Value v, memo m, diversifier i, nonce k

1. Parse the incoming view key $\mathsf{addr}_{\text{in}} = (s_1, P_2)$.
2. If Coin was generated in a mint transaction, parse $\mathsf{Coin} = (S, K, C, v, \overline{r})$; otherwise, parse $\mathsf{Coin} = (S, K, C, \mathit{\Pi}_{\text{rp}}, \overline{r})$.
3. Generate an AEAD encryption key $k_{\text{aead}} = \mathsf{AEADKeyGen}(s_1 K)$ and decrypt
$$r = \mathsf{AEADDecrypt}(k_{\text{aead}}, \mathtt{r}, \overline{r});$$
if decryption fails, return failure.
4. If Coin was generated in a mint transaction, parse the recipient data $r = (d, k, m)$; otherwise, parse $r = (v, d, k, m)$.
5. Check that $K = \mathcal{H}_k(k)\mathcal{H}_{\text{div}}(d)$, and return failure otherwise.
6. Check that $C = \mathsf{Com}(v, \mathcal{H}_{\text{val}}(k))$, and return failure otherwise.
7. Decrypt the diversifier $i = \mathsf{SymDecrypt}(\mathsf{SymKeyGen}(s_1), d)$.
8. Check that
$$S = \mathsf{Comm}(\mathcal{H}_{\text{ser}}(k), 0, 0) + \mathsf{Comm}(\mathcal{H}_{Q_2}(s_1, i), 0, 0) + P_2,$$
and return failure otherwise.
9. Output (v, m, i, k).

4.7 Recover

This algorithm allows a recipient (or designated entity) to determine if it controls a coin; if so, it computes the serial number, tag, value, memo, and diversifier from the coin (in addition to the coin nonce). It requires the full view key used to produce diversified addresses to do so. If the coin is not destined for any diversified address, the algorithm returns failure.

It is assumed that the recipient has run the Verify algorithm on the transaction generating the coin being recovered.

Inputs: Security parameter λ, public parameters pp, full view key $\mathsf{addr}_{\text{full}}$, coin Coin

Outputs: Serial number s, tag T, value v, memo m, diversifier i, nonce k

1. Parse the full view key $\mathsf{addr}_{\text{full}} = (s_1, s_2, D, P_2)$.

2. Arrange the corresponding incoming view key $\mathsf{addr}_{\text{in}} = (s_1, P_2)$.
3. Run $\mathsf{Identify}(\mathsf{addr}_{\text{in}}, \mathsf{Coin})$ to obtain (v, m, i, k), and return failure if this operation fails.
4. Compute the serial number

$$s = \mathcal{H}_{\text{ser}}(k) + \mathcal{H}_{Q_2}(s_1, i) + s_2$$

and tag

$$T = (1/s)(U - D).$$

5. If T has been constructed in any other valid recovery, return failure.
6. Output (s, T, v, m, i, k).

4.8 Spend

This algorithm allows a recipient to generate a transaction that consumes coins it controls, and generates new coins destined for arbitrary public addresses. The process is designed to be modular; in particular, only the full view key is required to generate the parallel one-out-of-many proof, which may be computationally expensive. The use of spend keys is only required for the final Chaum-Pedersen proof step, which is of lower complexity.

It is assumed that the recipient has run the $\mathsf{Recover}$ algorithm on all coins that it wishes to consume in such a transaction.

Inputs:

- Security parameter λ and public parameters pp
- A full view key $\mathsf{addr}_{\text{full}}$
- A spend key $\mathsf{addr}_{\text{sk}}$
- For each $u \in [0, w)$ coin to spend, a set of N input coins $\mathsf{InCoins}_u$ as part of a cover set[4] selected from coins generated in previous valid transactions
- For each $u \in [0, w)$ coin to spend, the index in $\mathsf{InCoins}_u$, serial number, tag, value, and nonce: $(l_u, s_u, T_u, v_u, k_u)$
- An integer fee value $f \in [0, v_{\max})$
- A set of t output coin public addresses, values, and memos:

$$\{\mathsf{addr}_{\text{pk},j}, v_j, m_j\}_{j=0}^{t-1}$$

Outputs: Spend transaction tx_{spend}

1. Parse the required full view key component D from $\mathsf{addr}_{\text{full}}$.
2. Parse the spend key $\mathsf{addr}_{\text{sk}} = (s_1, s_2, r)$.
3. For each $u \in [0, w)$:
   (a) Parse the cover set serial number commitments and value commitments $\{(S_{i,u}, C_{i,u})\}_{i=0}^{N-1}$ from $\mathsf{InCoins}_u$.

[4] It is straightforward to support the more general case where cover sets do not share a common size; for simplicity, we assume a shared common size here.

(b) Compute the serial number commitment offset:

$$S'_u = \mathsf{Comm}(s_u, 0, -\mathcal{H}_{\mathrm{ser}'}(s_u, D)) + D$$

(c) Compute the value commitment offset:

$$C'_u = \mathsf{Com}(v_u, \mathcal{H}_{\mathrm{val}'}(s_u, D))$$

(d) Generate a parallel one-out-of-many proof:

$$(\Pi_{\mathrm{par}})_u = \mathsf{ParProve}(pp_{\mathrm{par}}, \{S_{i,u}, C_{i,u}\}_{i=0}^{N-1}, S'_u, C'_u; \\ (l_u, \mathcal{H}_{\mathrm{ser}'}(s_u, D), \mathcal{H}_{\mathrm{val}}(k) - \mathcal{H}_{\mathrm{val}'}(s_u, D)))$$

4. Generate a set $\mathsf{OutCoins} = \{\mathsf{CreateCoin}(\mathsf{addr}_{\mathrm{pk},j}, v_j, m_j, 0)\}_{j=0}^{t-1}$ of output coins.
5. Parse the output coin value commitments $\{\overline{C}_j\}_{j=0}^{t-1}$ from $\mathsf{OutCoins}$, where each $\overline{C}_j$ contains nonce k_j.
6. Generate a representation proof for balance assertion:

$$\Pi_{\mathrm{bal}} = \mathsf{RepProve}\left(pp_{\mathrm{rep}}, H, \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \mathsf{Com}(f, 0); \right. \\ \left. \sum_{u=0}^{w-1} \mathcal{H}_{\mathrm{val}'}(s_u, D) - \sum_{j=0}^{t-1} \mathcal{H}_{\mathrm{val}}(k_j)\right)$$

7. Let $\mu = \mathcal{H}_{\mathrm{bind}}(\mathsf{OutCoins}, f, \{\mathsf{InCoins}_u, S'_u, C'_u, T_u, (\Pi_{\mathrm{par}})_u, \}_{u=0}^{w-1}, \Pi_{\mathrm{bal}})$.
8. Generate a modified Chaum-Pedersen proof, where we additionally bind μ to the initial transcript:

$$\Pi_{\mathrm{chaum}} = \mathsf{ChaumProve}((pp_{\mathrm{chaum}}, \mu), \{S'_u, T_u\}_{u=0}^{w-1}; \\ (\{s_u, r, -\mathcal{H}_{\mathrm{ser}'}(s_u, D)\}_{u=0}^{w-1}))$$

9. Output the tuple:

$$\mathrm{tx}_{\mathrm{spend}} = (\mathsf{OutCoins}, f, \\ \{\mathsf{InCoins}_u, S'_u, C'_u, T_u, (\Pi_{\mathrm{par}})_u, \Pi_{\mathrm{chaum}}\}_{u=0}^{w-1}, \Pi_{\mathrm{bal}})$$

Note that it is possible to modify the balance proof to account for other input or output values not represented by coin value commitments, similarly to the handling of fees. This observation can allow for the transfer of value into new coins without the use of a mint transaction, or a transfer of value to a transparent base layer. Such transfer functionality is likely to introduce practical risk that is not captured by the protocol security model, and warrants thorough analysis.

4.9 Verify

This algorithm assesses the validity of a transaction.

Inputs: either a mint transaction $\mathrm{tx}_{\mathrm{mint}}$ or a spend transaction $\mathrm{tx}_{\mathrm{spend}}$
Outputs: a bit that represents the validity of the transaction
If the input transaction is a mint transaction:

1. Parse the transaction $\mathrm{tx}_{\mathrm{mint}} = (\mathsf{OutCoins}, \Pi_{\mathrm{val}})$.
2. Parse the output coin values serial number commitments, and value commitments $\{(v_j, \overline{S}_j, \overline{C}_j)\}_{j=0}^{t-1}$ from $\mathsf{OutCoins}$.
3. For each $j \in [0, t)$:
   (a) If $\overline{S}_j$ appears in an output coin in this transaction or in any previously-verified transaction, output 0.
   (b) Check that $v_j \in [0, v_{\max})$, and output 0 if this fails.
4. Check that $\mathsf{RepVerify}\left(pp_{\mathrm{rep}}, \Pi_{\mathrm{val}}, H, \{\overline{C}_j - \mathsf{Com}(v_j, 0)\}_{j=0}^{t-1}\right)$, and output 0 if this fails.
5. Output 1.

If the input transaction is a spend transaction:

1. Parse the transaction:

$$\mathrm{tx}_{\mathrm{spend}} = (\mathsf{OutCoins}, f, \{\mathsf{InCoins}_u, S'_u, C'_u, T_u, (\Pi_{\mathrm{par}})_u, \Pi_{\mathrm{chaum}}\}_{u=0}^{w-1}, \Pi_{\mathrm{bal}})$$

2. Parse the cover set serial number commitments and value commitments $\{(S_i, C_i)\}_{i=0}^{N-1}$ from $\mathsf{InCoins}$.
3. For each $u \in [0, w)$:
   (a) Parse the cover set serial number commitments and value commitments $\{(S_{i,u}, C_{i,u})\}_{i=0}^{N-1}$ from $\mathsf{InCoins}_u$, and check that each cover set member corresponds to a valid coin generated in a previous valid transaction.
   (b) Check that T_u does not appear again in this transaction or in any previously-verified transaction, and output 0 if it does.
   (c) Check that $\mathsf{ParVerify}(pp_{\mathrm{par}}, (\Pi_{\mathrm{par}})_u, \{S_{i,u}, C_{i,u}\}_{i=0}^{N-1}, S'_u, C'_u)$, and output 0 if this fails.
4. Compute the binding hash μ as before, check that

$$\mathsf{ChaumVerify}((pp_{\mathrm{chaum}}, \mu), \Pi_{\mathrm{chaum}}, \{S'_u, T_u\}_{u=0}^{w-1}),$$

   and output 0 if this fails.
5. For each $j \in [0, t)$:
   (a) If $\overline{S}_j$ appears in an output coin in this transaction or in any previously-verified transaction, output 0.
   (b) Check that $\mathsf{RangeVerify}(pp_{\mathrm{rp}}, (\Pi_{\mathrm{rp}})_j, C)$, and output 0 if this fails.
6. Check that $f \in [0, v_{\max})$, and output 0 if this fails.
7. Check that

$$\mathsf{RepVerify}\left(pp_{\mathrm{rep}}, \Pi_{\mathrm{bal}}, H, \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \mathsf{Com}(f, 0)\right)$$

   and output 0 if this fails.
8. Output 1.

5 Multisignature Operations

It is often useful to permit transactions requiring multiple parties to authorize; the parties may be mututally untrusting, and it may not be sufficient to rely on a separate trusted third party. In this case, we require processes for distributed key and spend transaction generation that require either a set of specified parties or a threshold subset of a given size to complete. Specifically, we describe a method for such signing groups to perform the CreateKeys and Spend algorithms to produce keys and spend transactions indistinguishable from others.

The method we use is inspired by techniques from previous work. Because view keys are assumed to be fully known by all signers in a group, we use a method related to that of MuSig [21], for which we do not require any threshold computations. For spend key generation and threshold signing, FROST [19] introduces a one-round process with a precomputation phase that can be performed either during key generation or at the time of signing. Later work [10] examines FROST security and proposes a more efficient signing protocol for it; however, this introduces subtle security implications that are examined and addressed in [3]. We use this approach for our signing process. Note that we defer a complete security analysis of our construction to future work.

Throughout this section, suppose we have a group of ν players who wish to collaboratively produce keys, and such that a specified threshold $1 \leq t \leq \nu$ of the players is required to produce an authorizing proof spending coins directed to any address associated to the keys.

For the modified algorithms we present here, sample cryptographic hash functions

$$\mathcal{H}_{\text{pok}}, \mathcal{H}_{s_1}, \mathcal{H}_{s_2}, \mathcal{H}_{\rho}, \mathcal{H}_{F}, \mathcal{H}_{H} : \{0,1\}^* \to \mathbb{F}$$

uniformly at random.

5.1 CreateKeys

To produce key components, each player $1 \leq \alpha \leq \nu$ engages in the following two-round key generation process:

1. Selects a set of coefficients $\{a_{\alpha,j}\}_{j=0}^{t-1} \subset \mathbb{F}$ uniformly at random, and uses them to define the polynomial $f_\alpha(x) = \sum_{j=0}^{t-1} a_{\alpha,j} x^j$.
2. Selects view key shares $s_{1,\alpha}, s_{2,\alpha} \in \mathbb{F} \setminus \{0\}$ uniformly at random.
3. Produces a proof of knowledge of $a_{\alpha,0}$:
   (a) Chooses $k_\alpha \in \mathbb{F}$ uniformly at random.
   (b) Sets $R_\alpha = k_\alpha G$.
   (c) Sets $c_\alpha = \mathcal{H}_{\text{pok}}(\alpha, a_{\alpha,0}G, R_\alpha)$.
   (d) Sets $\mu_\alpha = k_\alpha + a_{\alpha,0} c_\alpha$.
4. Produces a vector of commitments $C_\alpha = \{C_{\alpha,j}\}_{j=0}^{t-1} = \{a_{\alpha,j}G\}_{j=0}^{t-1}$ to its coefficients.
5. Sends the tuple $(R_\alpha, \mu_\alpha, C_\alpha, s_{1,\alpha}, s_{2,\alpha})$ to all other players $1 \leq \beta \neq \alpha \leq \nu$.
6. On receipt of a tuple $(R_\beta, \mu_\beta, C_\beta, s_{1,\beta}, s_{2,\beta})$ from another player β:

(a) Checks that $s_{1,\beta} \neq 0$ and $s_{2,\beta} \neq 0$ and aborts otherwise.
(b) Verifies the proof of knowledge by checking that

$$\mu_\beta G - \mathcal{H}_{\text{pok}}(\beta, C_{\beta,0}, R_\beta) C_{\beta,0} = R_\beta$$

and aborting otherwise.

7. For each $1 \leq \beta \leq \nu$, computes a player share $\widehat{r}_{\alpha,\beta} = f_\alpha(\beta)$ and sends it to player β.
8. On receipt of a player share $\widehat{r}_{\beta,\alpha}$ from another player β, verifies the share by checking that

$$\sum_{j=0}^{t-1} \alpha^j C_{\beta,j} = \widehat{r}_{\beta,\alpha} G$$

and aborting otherwise.

9. Computes its private spend key share $r_\alpha = \sum_{\beta=1}^{\nu} \widehat{r}_{\beta,\alpha}$ and full view key component $D = \sum_{\beta=1}^{\nu} C_{\beta,0}$.
10. Computes the group view keys:

$$s_1 = \sum_{\beta=1}^{\nu} \mathcal{H}_{s_1}(\{s_{1,\gamma}\}_{\gamma=1}^{\nu}, \beta) s_{1,\beta}$$

$$s_2 = \sum_{\beta=1}^{\nu} \mathcal{H}_{s_2}(\{s_{2,\gamma}\}_{\gamma=1}^{\nu}, \beta) s_{2,\beta}$$

Using the tuple (s_1, s_2, D), any player can additionally compute the full view key component $P_2 = \mathsf{Comm}(s_2, 0, 0) + D$. Since each player holds the aggregate incoming view key, it can compute public addresses using CreateAddress as needed.

Note that each player should confirm that all other players have completed the key generation process before making addresses available to receive coins; otherwise, a malicious player might selectively fail to send its shares to all other players, meaning some players may be unable to properly compute their private spend key shares.

5.2 Precompute

The signing group can reduce the communication complexity of proof generation by precomputing and sharing sets of nonce data. Each future signing operation uses one such nonce set for each signing player, which cannot be reused. The signing group can precompute as many nonce sets as needed for expected signing operations, and can perform this operation whenever additional nonce sets are required. In particular, the group may wish to do so during the key generation process, where the added communication round may be less impactful than during later proof generation.

To precompute π sets of nonce data, each player $1 \leq \alpha \leq \nu$ engages in the following one-round process:

1. For $0 \leq k < \pi$, it selects $d_{\alpha,k}, e_{\alpha,k} \in \mathbb{F} \setminus \{0\}$ uniformly at random and defines $D_{\alpha,k} = d_{\alpha,k}G$ and $E_{\alpha,k} = e_{\alpha,k}G$.
2. Generates a vector L_α such that for $0 \leq k < \pi$, we have $L_{\alpha,k} = (D_{\alpha,k}, E_{\alpha,k})$; that is, L_α contains π nonce pairs.
3. Sends L_α to all other players.
4. On receipt of a vector L_β from another player β, checks that $D_{\beta,k} \neq 0$ and $E_{\beta,k} \neq 0$ for all $0 \leq k < \pi$, and aborts otherwise.

5.3 Spend

Because all players possess the aggregate full view key corresponding to public addresses, any player can use it to construct all transaction components except the modified Chaum-Pedersen proof. We describe now how a threshold of t signers collaboratively produce such a proof to authorize the spending of coins, with the following proof inputs (using our previous notation):

$$\{pp_{\text{chaum}}, \{S'_u, T_u\}_{u=0}^{w-1}; (\{s_u, r, -\mathcal{H}_{\text{ser}'}(s_u, D)\}_{u=0}^{w-1})\}$$

For the sake of notation convenience, we assume that the signing players are indexed $1 \leq \alpha \leq t$. Further, assume that each nonce list L_α contains $k + 1 \geq w$ unused nonces. We also assume the context binding value μ has been defined. Each such player α engages in the following one-round process:

1. Parses the next available set of nonces $\{(D_{\beta,k-u}, E_{\beta,k-u})\}_{u=0}^{w-1}$ in each L_β for $1 \leq \beta \leq t$ (and removes them from each list after use) to compute, for $0 \leq u < w$ and $1 \leq \gamma \leq t$, the following:

$$\rho_{u,\gamma} = \mathcal{H}_\rho(\{\beta, D_{\beta,k-u}, E_{\beta,k-u}\}_{\beta=1}^{t}, \gamma, \mu, S'_u, T_u)$$

2. Sets the initial proof commitments

$$A_1 = \sum_{u=0}^{w-1} \left(\mathcal{H}_F\left(\{\rho_{u,\beta}\}_{\beta=1}^{t}\right) F + \mathcal{H}_H\left(\{\rho_{u,\beta}\}_{\beta=1}^{t}\right) H + \sum_{\beta=1}^{t} \left(D_{\beta,k-u} + \rho_{u,\beta} E_{\beta,k-u}\right) \right)$$

and

$$\{A_{2,u}\}_{u=0}^{w-1} = \left\{ \mathcal{H}_F\left(\{\rho_{u,\beta}\}_{\beta=1}^{t}\right) T_u + \sum_{\beta=1}^{t} \left(D_{\beta,k-u} + \rho_{u,\beta} E_{\beta,k-u}\right) \right\}.$$

3. Computes the challenge c using the proof transcript as in the original Spend description.

4. Computes its Lagrange coefficient

$$\lambda_\alpha = \prod_{\beta=1,\beta\neq\alpha}^{t} \left(\frac{\beta}{\beta-\alpha}\right)$$

and the response share

$$t_{2,\alpha} = \sum_{u=0}^{w-1}(d_{\alpha,k-u} + \rho_{u,\alpha}e_{\alpha,k-u} + \lambda_\alpha r_\alpha c^u),$$

and sends $t_{2,\alpha}$ to the other signing players.

5. On receipt of $t_{2,\beta}$ from another player, checks that

$$t_{2,\beta}G = \sum_{u=0}^{w-1}\left(D_{\beta,k-u} + \rho_{u,\beta}E_{\beta,k-u} + c^u\lambda_\beta \sum_{\gamma=1}^{\nu}\sum_{j=0}^{t-1}\beta^j C_{\gamma,j}\right)$$

and aborts otherwise.

6. On receipt of all $t_{2,\beta}$ values, computes the proof responses:

$$\begin{aligned}\{t_{1,u}\}_{u=0}^{w-1} &= \left\{\mathcal{H}_F\left(\{\rho_{u,\beta}\}_{\beta=1}^{t}\right) + c^u s_u\right\}\\ t_2 &= \sum_{\beta=1}^{t} t_{2,\beta}\\ t_3 &= \sum_{u=0}^{w-1}\left(\mathcal{H}_H\left(\{\rho_{u,\beta}\}_{\beta=1}^{t}\right) - c^u\mathcal{H}_{\text{ser}'}(s_u, D)\right)\end{aligned}$$

6 View Keys and Payment Proofs

The key and proof structures in Spark enable flexible and useful functionality relating to transaction scanning, generation, and disclosure.

The incoming view key is used in Identify operations to determine when a coin is directed to an associated public address, and to determine the coin's value and associated memo data. This permits two use cases of note. In one case, blockchain scanning can be delegated to a device or service without delegating spend authority for identified coins. In another case, wallet software in possession of a spend key can keep this key encrypted or otherwise securely stored during scanning operations, reducing key exposure risks.

The full view key is used in Recover operations to additionally compute the serial number and tag for coins directed to an associated public address. These tags can be used to identify a transaction spending the coin. Providing this key to a third party permits identification of incoming transactions and detection of outgoing transactions, which additionally provides balance computation, without delegating spend authority. Users like public charities may wish to permit public oversight of funds with this functionality. Other users may wish to provide this

functionality to an auditor or accountant for bookkeeping purposes. In the case where an address is used in threshold multisignature operations, a cosigner may wish to know if or when another cohort of cosigners has produced a transaction spending funds.

Further, the full view key is used in Spend to generate one-out-of-many proofs. Since the parallel one-out-of-many proof used in Spark can be computationally expensive, it may be unsuitable for generation by a computationally-limited device like a hardware wallet. Providing this key to a more powerful device enables easy generation of this proof (and other transaction components like range proofs), while ensuring that only the device holding the spend key can complete the transaction by generating the simple modified Chaum-Pedersen proof.

Payment proofs, which we introduce in Appendix D, allow for disclosure of data for individual coins. Specifically, a payment proof asserts in zero knowledge that the prover knows the spend key used to authorize the transaction generating a given coin that is destined for a given public address. The proof convinces a verifier that the holder of the incoming view key for the public address can successfully identify the coin, as well as provides the verifier with the value and memo for the coin. Unlike view keys, which provide broad visibility into transactions associated to a public address, a payment proof is limited to a single coin and can be bound to an arbitrary proof context to prevent replay.

Payment proofs may be useful in a number of circumstances. For example, a customer may issue a payment to a retailer, but fail to use the correct diversified address or memo required by the retailer to associate the payment to the customer's order. By providing the retailer with a payment proof, the customer can assert that it produced a coin destined for the retailer's address. In another use case, a business may wish to make public details of a donation to a charity without publicly disclosing its full view key. By providing a payment proof, anyone can verify that the specified coin was destined for the charity's address and confirm the value and memo associated to the coin.

7 Efficiency

It is instructive to examine the efficiency of spend transactions in size, generation complexity, and verification complexity. In addition to our previous notation for parameters, let $v_{\max} = 2^{64}$, so coin values and fees can be represented by 8-byte unsigned integers. Further, suppose coin memos are fixed at M bytes in length, diversifiers are restricted to I bytes in length, with a 16-byte authentication tag and 32-byte key commitment; this is the case for the ChaCha20-Poly1305 authenticated symmetric encryption construction (modified using the technique in [1]), for example [23]. Additionally, the arguments in [22] imply that Schnorr representation proofs can use truncated hash outputs for reduced proof size. Transaction size data for specific component instantiations is given in Table 1, where we consider the size in terms of group elements, field elements, and other

data. Note that we do not include input ambiguity set references in this data, as this depends on implementation-specific selection and representation criteria.

Table 1. Spend transaction size by component

Component	**Instantiation**	**Size ($\mathbb{G}$)**	**Size ($\mathbb{F}$)**	**Size (bytes)**
f				8
Π_{rp}	Bulletproofs+	$2\lceil \lg(64t) \rceil + 3$	3	
Π_{bal}	Schnorr (short)		1.5	
Π_{chaum}	this paper	$w + 1$	$w + 2$	
Input data (w coins)				
(S', C')		$2w$		
Π_{par}	this paper	$(2m + 2)w$	$[m(n-1) + 3]w$	
Output data (t coins)				
(S, K, C)		$3t$		
$\overline{r}$	ChaCha20-Poly1305			$(8 + M + I + 48)t$

To evaluate the verification complexity of spend transactions using these components, we observe that verification in constructions like the parallel one-out-of-many proving system in this paper, Bulletproof+ range proving system, Schnorr representation proving system, and modified Chaum-Pedersen proving system in this paper all reduce to single linear combination evaluations in $\mathbb{G}$. Because of this, proofs can be evaluated in batches if the verifier first weights each proof by a random value in $\mathbb{F}$, such that distinct group elements need only appear once in the resulting weighted linear combination. Notably, techniques like that of [26] can be used to reduce the complexity of such evaluations by up to a logarithmic factor. Suppose we wish to verify a batch of B transactions, each of which spends w coins and generates t coins. Table 2 shows the verification batch complexity in terms of total distinct elements of $\mathbb{G}$ that must be included in a linear combination evaluation.

Table 2. Spend transaction batch verification complexity for B transactions with w spent coins and t generated coins

Component	**Complexity**
Parallel one-out-of-many	$B[w(2m + 2) + 2n^m] + 2mn + 1$
Bulletproofs+	$B(t + 2\lg(64t) + 3) + 128T + 2$
Modified Chaum-Pedersen	$B(3w + 1) + 4$
Schnorr	$B(w + t + 1) + 2$

We further comment that the parallel one-out-of-many proving system presented in this paper may be further optimized in verification. Because corresponding elements of the $\{S_i\}$ and $\{V_i\}$ input sets are weighted identically in the protocol verification equations, it may be more efficient (depending on imple-

mentation) to combine these elements with a sufficient weight prior to applying the proof-specific weighting identified above for batch verification. Initial tests using a variable-time curve library suggest significant reductions in verification time with this technique.

Acknowledgments

The authors thank pseudonymous collaborator `koe` for ongoing discussions during the development of this work. The authors gratefully acknowledge Nikolas Krätzschmar for identifying an earlier protocol flaw relating to tag generation. The authors further thank independent researcher Luke Parker (`kayabaNerve`) for pointing out the failure of the security model to capture the case of an adversary producing a duplicate serial commitment prior to an honest transaction being added to the ledger.

References

1. Albertini, A., Duong, T., Gueron, S., Kölbl, S., Luykx, A., Schmieg, S.: How to abuse and fix authenticated encryption without key commitment. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 3291–3308. USENIX Association, Boston, MA (Aug 2022), https://www.usenix.org/conference/usenixsecurity22/presentation/albertini
2. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) Advances in Cryptology — ASIACRYPT 2001. pp. 566–582. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
3. Bellare, M., Tessaro, S., Zhu, C.: Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Paper 2022/833 (2022), https://eprint.iacr.org/2022/833
4. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
5. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: Proceedings of the 23rd USENIX Conference on Security Symposium. p. 781–796. SEC'14, USENIX Association, USA (2014)
6. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C.: Short accountable ring signatures based on DDH. In: Pernul, G., Y A Ryan, P., Weippl, E. (eds.) Computer Security – ESORICS 2015. pp. 243–265. Springer International Publishing, Cham (2015)
7. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050 (2017), https://ia.cr/2017/1050
8. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334 (2018). https://doi.org/10.1109/SP.2018.00020

9. Chung, H., Han, K., Ju, C., Kim, M., Seo, J.H.: Bulletproofs+: Shorter proofs for privacy-enhanced distributed ledger. Cryptology ePrint Archive, Report 2020/735 (2020), https://ia.cr/2020/735
10. Crites, E., Komlo, C., Maller, M.: How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375 (2021), https://ia.cr/2021/1375
11. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) Public Key Cryptography - PKC 2005. pp. 416–431. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
12. Esgin, M.F., Zhao, R.K., Steinfeld, R., Liu, J.K., Liu, D.: MatRiCT: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 567–584. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3354200
13. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: A cryptographic investigation of Mimblewimble. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 657–689. Springer International Publishing, Cham (2019)
14. Gennaro, R., Leigh, D., Sundaram, R., Yerazunis, W.: Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In: Lee, P.J. (ed.) Advances in Cryptology - ASIACRYPT 2004. pp. 276–292. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
15. Goodell, B., Noether, S., RandomRun: Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Report 2019/654 (2019), https://ia.cr/2019/654
16. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. pp. 253–280. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
17. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification (2021), https://github.com/zcash/zips/blob/master/protocol/protocol.pdf
18. Jivanyan, A.: Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Report 2019/373 (2019), https://ia.cr/2019/373
19. Komlo, C., Goldberg, I.: FROST: Flexible round-optimized Schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852 (2020), https://ia.cr/2020/852
20. Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: Scaling private payments without trusted setup. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 31–48. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3345655
21. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to Bitcoin. Designs, Codes and Cryptography **87**(9), 2139–2164 (2019)
22. Neven, G., Smart, N.P., Warinschi, B.: Hash function requirements for Schnorr signatures. Journal of Mathematical Cryptology **3**(1), 69–87 (2009). https://doi.org/10.1515/JMC.2009.004
23. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF protocols. RFC 7539, RFC Editor (May 2015), http://www.rfc-editor.org/rfc/rfc7539.txt

24. Noether, S., Goodell, B.: Triptych: Logarithmic-sized linkable ring signatures with applications. In: Garcia-Alfaro, J., Navarro-Arribas, G., Herrera-Joancomarti, J. (eds.) Data Privacy Management, Cryptocurrencies and Blockchain Technology. pp. 337–354. Springer International Publishing, Cham (2020)
25. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
26. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
27. Pyrros Chaidos, V.G.: Lelantus-CLA. Cryptology ePrint Archive, Report 2021/1036 (2021), `https://ia.cr/2021/1036`
28. `koe`: Seraphis: A privacy-preserving transaction protocol abstraction. GitHub repository release `DRAFT-v0.0.11` (2021), `https://github.com/UkoeHB/Seraphis/releases/tag/DRAFT-v0.0.11`

A Modified Chaum-Pedersen Proving System

The proving system is a tuple of algorithms (ChaumProve, ChaumVerify) for the following relation:

$$\{pp_{\text{chaum}}, \{S_i, T_i\}_{i=0}^{l-1} \subset \mathbb{G}^2; (\{x_i, y_i, z_i\}_{i=0}^{l-1}) \subset \mathbb{F}^3 : \\ \forall i \in [0, l), S_i = x_i F + y_i G + z_i H, U = x_i T_i + y_i G\}$$

Our protocol uses a power-of-challenge technique inspired by a method used for aggregating Schnorr signatures [14]. The protocol proceeds as follows:

1. The prover selects random $\{r_i, s_i\}_{i=0}^{l-1}, t \in \mathbb{F}$. It computes the values
$$A_1 = \sum_{i=0}^{l-1} r_i F + \sum_{i=0}^{l-1} s_i G + tH$$
$$\{A_{2,i}\}_{i=0}^{l-1} = \{r_i T_i + s_i G\}_{i=0}^{l-1}$$
and sends these values to the verifier.
2. The verifier selects a random challenge $c \in \mathbb{F}$ and sends it to the prover.
3. The prover computes responses
$$\{t_{1,i}\}_{i=0}^{l-1} = \{r_i + c^{i+1} x_i\}_{i=0}^{l-1}$$
$$t_2 = \sum_{i=0}^{l-1} (s_i + c^{i+1} y_i)$$
$$t_3 = t + \sum_{i=0}^{l-1} c^{i+1} z_i$$
and sends them to the verifier.
4. The verifier accepts the proof if and only if
$$A_1 + \sum_{i=0}^{l-1} c^{i+1} S_i = \sum_{i=0}^{l-1} t_{1,i} F + t_2 G + t_3 H$$

and

$$\sum_{i=0}^{l-1}(A_{2,i} + c^{i+1}U) = \sum_{i=0}^{l-1} t_{1,i}T_i + t_2 G.$$

This interactive protocol can be made non-interactive using the Fiat-Shamir technique, which replaces the verifier challenge c with the output of a cryptographic hash function on transcript inputs. We now prove that the protocol is complete, special sound, and special honest-verifier zero knowledge.

Proof. Completeness of the protocol follows by inspection.

We now show it is (l+1)-special sound by building a polynomial-time extractor as follows. Given a statement and initial proof transcript $(A_1, \{A_{2_i}\}_{i=0}^{l-1})$, the verifier sends $l+1$ distinct challenge values $c_0, c_1, \ldots, c_l$ and receives the corresponding transcript values $(\{t_{1,i}^0\}_{i=0}^{l-1}, t_2^0, t_3^0), \ldots, (\{t_{1,i}^l\}_{i=0}^{l-1}, t_2^l, t_3^l)$ from the prover. From the first verification equation, we build the following linear system:

$$\begin{aligned}
A_1 + \sum_{i=0}^{l-1} c_0^{i+1} S_i &= \sum_{i=0}^{l-1} t_{1,i}^0 F + t_2^0 G + t_3^0 H \\
A_1 + \sum_{i=0}^{l-1} c_1^{i+1} S_i &= \sum_{i=0}^{l-1} t_{1,i}^1 F + t_2^1 G + t_3^1 H \\
&\vdots \\
A_1 + \sum_{i=0}^{l-1} c_l^{i+1} S_i &= \sum_{i=0}^{l-1} t_{1,i}^l F + t_2^l G + t_3^l H
\end{aligned}$$

Subtracting the first equation from the rest, we obtain another linear system:

$$\begin{aligned}
\sum_{i=0}^{l-1}(c_1^{i+1} - c_0^{i+1}) S_i &= \sum_{i=0}^{l-1}(t_{1,i}^1 - t_{1,i}^0) F + (t_2^1 - t_2^0) G + (t_3^1 - t_3^0) H \\
\sum_{i=0}^{l-1}(c_2^{i+1} - c_0^{i+1}) S_i &= \sum_{i=0}^{l-1}(t_{1,i}^2 - t_{1,i}^0) F + (t_2^2 - t_2^0) G + (t_3^2 - t_3^0) H \\
&\vdots \\
\sum_{i=0}^{l-1}(c_l^{i+1} - c_0^{i+1}) S_i &= \sum_{i=0}^{l-1}(t_{1,i}^l - t_{1,i}^0) F + (t_2^l - t_2^0) G + (t_3^l - t_3^0) H
\end{aligned} \tag{1}$$

Finally, we let the set $\{x_i\}_{i=0}^{l-1} \subset \mathbb{F}$ be defined through the following linear system:

$$\begin{aligned}
\sum_{i=0}^{l-1}(c_1^{i+1} - c_0^{i+1})x_i &= \sum_{i=0}^{l-1}(t_{1,i}^1 - t_{1_i}^0) \\
\sum_{i=0}^{l-1}(c_2^{i+1} - c_0^{i+1})x_i &= \sum_{i=0}^{l-1}(t_{1,i}^2 - t_{1_i}^0) \\
&\vdots \\
\sum_{i=0}^{l-1}(c_l^{i+1} - c_0^{i+1})x_i &= \sum_{i=0}^{l-1}(t_{1,i}^l - t_{1_i}^0)
\end{aligned}$$

Since each challenge is uniformly distributed at random, the square coefficient matrix corresponding to the system has nonzero determinant except with negligible probability, and hence the system is solvable for all $\{x_i\}_{i=0}^{l-1}$. Further, we can form similar linear systems to define corresponding $\{y_i\}_{i=0}^{l-1}$ and $\{z_i\}_{i=0}^{l-1}$ such that we let $S_i = x_i F + y_i G + z_i H$ and the equations in system 1 hold.

It remains to show that these solutions are unique; that is, that no S_i has a different representation with coefficients x_i', y_i', z_i' consistent with successful verification. If this were the case, then we must have the polynomial equation $\sum_{i=0}^{l-1} c^{i+1}(x_i - x_i') = 0$ in c; however, since c is selected randomly by the verifier, all coefficients of the polynomial must (with overwhelming probability) be zero by the Schwartz-Zippel lemma. Hence each $x_i = x_i'$ (and by the same reasoning, $y_i = y_i'$ and $z_i = z_i'$), and the extracted witness set is unique.

To show the protocol is special honest-verifier zero knowledge, we construct a valid simulator producing transcripts identically distributed to those of valid proofs. The simulator chooses a random challenge $c \in \mathbb{F}$ and random values $\{t_{1,i}\}_{i=0}^{l-1}, t_2, t_3 \in \mathbb{F}$. It also randomly selects $\{A_{2,i}\}_{i=1}^{l-1} \in \mathbb{G}$, and sets

$$A_1 = \sum_{i=0}^{l-1} t_{1,i} F + t_2 G + t_3 H - \sum_{i=0}^{l-1} c^{i+1} S_i$$

and

$$A_{2,0} = \sum_{i=0}^{l-1} t_{1,i} T_i + t_2 G - \sum_{i=1}^{l-1} A_{2,i} - \sum_{i=0}^{l-1} c^{i+1} U.$$

The forms of A_1 and $A_{2,0}$ are defined such that the verification equations hold, and therefore such a transcript will be accepted by an honest verifier. Observe that all transcript elements in a valid proof are independently distributed uniformly at random if the generators F, G, H, U are independent, as are transcript elements produced by the simulator.

This completes the proof.

B Parallel One-out-of-Many Proving System

The proving system itself is a tuple of algorithms $(\mathsf{ParProve}, \mathsf{ParVerify})$ for the following relation, where we let $N = n^m$:

$$\{pp_{\text{par}}, \{S_k, V_k\}_{i=0}^{N-1} \subset \mathbb{G}^2, S', V' \in \mathbb{G}; l \in \mathbb{N}, (s, v) \in \mathbb{F} : \\ 0 \leq l < N, S_l - S' = \mathsf{Comm}(0, 0, s), V_l - V' = \mathsf{Com}(0, v)\}$$

Let $\delta(i, j) : \mathbb{N}^2 \to \mathbb{F}$ be the Kronecker delta function. For any integers k and j such that $0 \leq k < N$ and $0 \leq j < m$, let k_j denote the j digit of the n-ary decomposition of k. Let $\mathsf{MatrixCom} : \mathbb{F}^{mn} \times \mathbb{F}^{mn} \times \mathbb{F} \to \mathbb{G}$ be an additively-homomorphic matrix commitment construction that commits to the entries of two matrices, and is perfectly hiding and computationally binding.

The protocol proceeds as follows, where we use some of the notation of [18,24]:

1. The prover selects
$$r_A, r_B, \{a_{j,i}\}_{j=0,i=1}^{m-1,n-1} \in \mathbb{F}$$
uniformly at random, and, for each $j \in [0, m)$, sets
$$a_{j,0} = -\sum_{i=1}^{n-1} a_{j,i}.$$

2. The prover computes the following:
$$A \equiv \mathsf{MatrixCom}\left(\{a_{j,i}\}_{j,i=0}^{m-1,n-1}, \{-a_{j,i}^2\}_{j,i=0}^{m-1,n-1}, r_A\right)$$
$$B \equiv \mathsf{MatrixCom}\left(\{\delta(l_j, i)\}_{j,i=0}^{m-1,n-1}, \{a_{j,i}(1 - 2\delta(l_j, i))\}_{j,i=0}^{m-1,n-1}, r_B\right)$$

3. For each $j \in [0, m)$, the prover selects $\rho_j, \rho'_j \in \mathbb{F}$ uniformly at random, and computes the following:
$$X_j \equiv \sum_{k=0}^{N-1} p_{k,j}(S_k - S') + \mathsf{Comm}(0, 0, \rho_j)$$
$$X'_j \equiv \sum_{k=0}^{N-1} p_{k,j}(V_k - V') + \mathsf{Com}(0, \rho'_j)$$
Here each $p_{k,j}$ is defined such that for all $k \in [0, N)$ we have
$$\prod_{j=0}^{m-1} \left(\delta(l_j, k_j)x + a_{j,k_j}\right) = \delta(l, k)x^m + \sum_{j=0}^{m-1} p_{k,j}x^j$$
for indeterminate x.
4. The prover sends $A, B, \{X_j, X'_j\}_{j=0}^{m-1}$ to the verifier.
5. The verifier selects $x \in \mathbb{F}$ uniformly at random and sends it to the prover.

6. For each $j \in [0, m)$ and $i \in [1, n)$, the prover computes $f_{j,i} \equiv \delta(l_j, i)x + a_{j,i}$ and the following values:

$$z \equiv r_A + xr_B$$
$$z_S \equiv sx^m - \sum_{j=0}^{m-1} \rho_j x^j$$
$$z_V \equiv vx^m - \sum_{j=0}^{m-1} \rho'_j x^j$$

7. The prover sends $\{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}, z, z_S, z_V$ to the verifier.
8. For each $j \in [0, m)$, the verifier sets $f_{j,0} \equiv x - \sum_{i=1}^{n-1} f_{j,i}$ and accepts the proof if and only if

$$A + xB = \mathsf{MatrixCom}\left(\{f_{j,i}\}_{j,i=0}^{m-1,n-1}, \{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}, z\right)$$

and

$$\sum_{k=0}^{N-1}\left(\prod_{j=0}^{m-1} f_{j,k_j}\right)(S_k - S') - \sum_{j=0}^{m-1} x^j X_j = \mathsf{Comm}(0, 0, z_S)$$
$$\sum_{k=0}^{N-1}\left(\prod_{j=0}^{m-1} f_{j,k_j}\right)(V_k - V') - \sum_{j=0}^{m-1} x^j X'_j = \mathsf{Com}(0, z_V)$$

are true.

This interactive protocol can be made non-interactive using the Fiat-Shamir technique, which replaces the verifier challenge x with the output of a cryptographic hash function on transcript inputs.

We now prove that the above protocol is complete, special sound, and honest-verifier zero knowledge. The proofs proceed similarly to those of [6,18,24].

Proof. Completeness of the protocol follows by straightforward algebra.

To show that the protocol is special honest-verifier zero knowledge, we construct a simulator that, when provided a valid statement and random verifier challenge x, produces a proof transcript identically distributed to that of a real proof.

To produce our simulated transcript on random x, the simulator samples

$$B, \{X_j, X'_j\}_{j=1}^{m-1} \in \mathbb{G}$$

and

$$z, z_S, z_V, \{f_{j,i}\}_{j=0,i=1}^{m-1,n-1} \in \mathbb{F}$$

uniformly at random. It defines

$$f_{j,0} = x - \sum_{i=1}^{n-1} f_{j,i}$$

for each $j \in [0, m)$, and sets

$$A = \mathsf{MatrixCom}\left(\{f_{j,i}\}_{j,i=0}^{m-1,n-1}, \{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}, z\right) - xB$$

as well. It uses the final two verification equations to compute X_0 and X_0':

$$X_0 = \sum_{k=0}^{N-1}\left(\prod_{j=0}^{m-1} f_{j,k_j}\right)(S_k - S') - \sum_{j=1}^{m-1} x^j X_j - \mathsf{Comm}(0, 0, z_S)$$

$$X_0' = \sum_{k=0}^{N-1}\left(\prod_{j=0}^{m-1} f_{j,k_j}\right)(V_k - V') - \sum_{j=1}^{m-1} x^j X_j' - \mathsf{Comm}(0, z_V)$$

Since the challenge x is sampled uniformly at random by construction, the commitment constructions are perfectly hiding, $\{\rho_j, \rho_j'\}_{j=0}^{m-1}$ are sampled uniformly at random in a real proof, and the decisional Diffie-Hellman problem is hard in $\mathbb{G}$, all proof elements in both the simulation and real proofs are either independently uniformly distributed at random or uniquely determined by other transcript elements. Hence the protocol is special honest-verifier zero knowledge.

We now show that the protocol is $(m+1)$-special sound for $m > 1$. That is, we construct an extractor that, when presented with a set of $m+1$ distinct challenges and corresponding responses to the same initial statement, produces a set of extracted witness elements consistent with the statement. Consider a collection of $m+1$ distinct challenges $\{x_\iota\}_{\iota=0}^{m}$, and corresponding valid responses:

$$\left\{\{f_{j,i}^{(\iota)}\}_{j=0,i=1}^{m-1,n-1}, z^{(\iota)}, z_S^{(\iota)}, z_V^{(\iota)}\right\}_{\iota=0}^{m}$$

Successful verification on indices $\iota \in \{0, 1\}$ gives the following:

$$\begin{aligned}(x^{(0)} - x^{(1)})B = \mathsf{MatrixCom}\Big(&\{f_{j,i}^{(0)} - f_{j,i}^{(1)}\}_{j,i=0}^{m-1,n-1},\\ &\{f_{j,i}^{(0)}(x^{(0)} - f_{j,i}^{(0)}) - f_{j,i}^{(1)}(x^{(1)} - f_{j,i}^{(1)})\}_{j,i=0}^{m-1,n-1}, z^{(0)} - z^{(1)}\Big)\end{aligned}$$

For all $j \in [0, m)$ and $i \in [0, n)$, if we let

$$b_{j,i} = \frac{f_{j,i}^{(0)} - f_{j,i}^{(1)}}{x^{(0)} - x^{(1)}}$$

and

$$c_{j,i} = \frac{f_{j,i}^{(0)}(x^{(0)} - f_{j,i}^{(0)}) - f_{j,i}^{(1)}(x^{(1)} - f_{j,i}^{(1)})}{x^{(0)} - x^{(1)}}$$

and

$$r_B = \frac{z^{(0)} - z^{(1)}}{x^{(0)} - x^{(1)}},$$

then we can express

$$B = \mathsf{MatrixCom}\left(\{b_{j,i}\}_{j,i=0}^{m-1,n-1}, \{c_{j,i}\}_{j,i=0}^{m-1,n-1}, r_B\right).$$

If for $j \in [0, m)$ and $i \in [0, n)$ we further define

$$a_{j,i} = f_{j,i}^{(0)} - x^{(0)} b_{i,j}$$

and

$$d_{i,j} = f_{j,i}^{(0)}(x^{(0)} - f_{j,i}^{(0)}) - x^{(0)} c_{j,i}$$

and $r_A = z^{(0)} - x^{(0)} r_B$, then we can express

$$A = \mathsf{MatrixCom}\left(\{a_{j,i}\}_{j,i=0}^{m-1,n-1}, \{d_{j,i}\}_{j,i=0}^{m-1,n-1}, r_A\right)$$

as well. Observe that since the commitment construction is computationally binding, for all $\iota \in [0, m]$ we must have $b_{j,i} x^{(\iota)} + a_{j,i} = f_{j,i}^{(\iota)}$ and $c_{j,i} x^{(\iota)} + d_{j,i} = f_{j,i}^{(\iota)}(x^{(\iota)} - f_{j,i}^{(\iota)})$ for $j \in [0, m)$ and $i \in [0, n)$. This implies in particular that for $\iota \in \{0, 1, 2\}, j \in [0, m), i \in [0, n)$ we have

$$c_{j,i} x^{(\iota)} + d_{j,i} = b_{j,i}(1 - b_{j,i}) x^{(\iota)2} + (1 - 2b_{j,i}) a_{j,i} x^{(\iota)} - a_{j,i}^2$$

and hence $b_{j,i}(1 - b_{j,i}) = 0$, so each $b_{j,i} \in \{0, 1\}$.

We also have, by construction, that

$$x^{(\iota)} = \sum_{i=0}^{n-1} f_{j,i}^{(\iota)} = x^{(\iota)} \sum_{i=0}^{n-1} b_{j,i} + \sum_{i=0}^{n-1} a_{j,i}$$

for $\iota \in [0, m], j \in [0, m)$, so $\sum_{i=0}^{n-1} b_{j,i} = 1$. This means we can extract $l \in [0, N)$ such that each $b_{j,i} = \delta(l_j, i)$.

Now if we define for each $k \in [0, N)$ the polynomial

$$p_k(x) = \prod_{j=0}^{m-1} \left[\delta(l_j, k_j) x + a_{j,k_j}\right]$$

in x, we have $\deg(p_k) = m$ if and only if $k = l$. Verification can therefore be expressed as

$$x^{(\iota)m}(S_l - S') - \sum_{j=0}^{m-1} x^{(\iota)j} \overline{X}_j = \mathsf{Comm}(0, 0, z_S^{(\iota)})$$

$$x^{(\iota)m}(V_l - V') - \sum_{j=0}^{m-1} x^{(\iota)j} \overline{X}'_j = \mathsf{Com}(0, 0, z_V^{(\iota)})$$

for $\iota \in [0, m]$, where the sets $\{\overline{X}_j\}_{j=0}^{m-1}$ and $\{\overline{X}'_j\}_{j=0}^{m-1}$ can be uniquely derived. Consider a Vandermonde matrix V such that the ι row is the vector $(1, x^{(\iota)}, \ldots, x^{(\iota)m})$, and note since each challenge is distinct, we have $\det(V) \neq 0$ with high probability, so the rows of V span $\mathbb{F}^{m+1}$. This means we can find $\{\theta_\iota\}_{\iota=0}^{m}$ such that the equation

$$\sum_{\iota=0}^{m} \theta_\iota x^{(\iota)j} = \delta(j, m)$$

holds for $j \in [0, m]$.

We can therefore build a linear combination of each of the two above verification equations, taking advantage of the Vandermonde-derived weights:

$$S_l - S' = \sum_{\iota=0}^{m} \theta_\iota x^{(\iota)m}(S_l - S') + \sum_{\iota=0}^{m} \theta_\iota \left(x^{(\iota)j} \overline{X}_j \right) = \mathsf{Comm}\left(0, 0, \sum_{\iota=0}^{m} \theta_\iota z_S^{(\iota)}\right)$$
$$V_l - V' = \sum_{\iota=0}^{m} \theta_\iota x^{(\iota)m}(S_l - S') + \sum_{\iota=0}^{m} \theta_\iota \left(x^{(\iota)j} \overline{X}'_j \right) = \mathsf{Com}\left(0, \sum_{\iota=0}^{m} \theta_\iota z_V^{(\iota)}\right)$$

These equations provide the remaining extractions

$$s = \sum_{\iota=0}^{m} \theta_\iota z_S^{(\iota)}$$

and

$$v = \sum_{\iota=0}^{m} \theta_\iota z_V^{(\iota)}$$

such that $S_l - S' = \mathsf{Comm}(0, 0, s)$ and $V_l - V' = \mathsf{Com}(0, v)$, which completes the proof.

C Payment System Security

Zerocash [4] established a robust security framework for decentralized anonymous payment (DAP) scheme security that captures a realistic threat model with powerful adversaries who are permitted to add malicious coins into transactions' input ambiguity sets, control the choice of transaction inputs, and produce arbitrary transactions to add to a ledger. Here we formally prove Spark's security within a related (but modified) security model; proofs follow somewhat similarly to that of [4].

The DAP construction is a tuple of algorithms

$$(\mathsf{Setup}, \mathsf{CreateKeys}, \mathsf{CreateAddress}, \mathsf{CreateCoin}, \mathsf{Mint}, \mathsf{Identify}, \mathsf{Recover}, \mathsf{Spend}, \mathsf{Verify})$$

that is secure if it satisfies properties of completeness, balance, non-malleability, and ledger indistinguishability.

Each security property is formalized as a game between a polynomial-time adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where in each game the behavior of honest parties is simulated via an oracle $\mathcal{O}^{\mathsf{DAP}}$. The oracle $\mathcal{O}^{\mathsf{DAP}}$ maintains a ledger L of transactions and provides an interface for executing the CreateAddress, Mint, and Spend algorithms. To simulate behavior from honest parties, $\mathcal{A}$ passes a query to $\mathcal{C}$, which makes sanity checks and then proxies the queries to $\mathcal{O}^{\mathsf{DAP}}$, returning the responses to $\mathcal{A}$ as needed. For CreateAddress queries, the oracle first runs the CreateKeys protocol algorithm, then calls CreateAddress using the resulting

incoming view key and a randomly-selected diversifier index, and finally returns the public address $\mathsf{addr}_{\mathrm{pk}}$. For Mint queries, the adversary specifies the value, memo, and destination public address for the transaction, and the resulting transaction is produced and returned if the inputs are semantically valid. For Spend queries, the adversary specifies the input coins to be consumed, as well as the values, memos, and destination public addresses for the transaction, and the resulting transaction is produced after coin recovery if the inputs are semantically valid, all consumed coins are validly controlled by an address produced by the oracle, and all consumed coins are unspent according to the ledger state. The oracle $\mathcal{O}^{\mathsf{DAP}}$ also provides an Insert query that allows the adversary to insert arbitrary and potentially malicious $\mathrm{tx}_{\mathrm{mint}}$ or $\mathrm{tx}_{\mathrm{spend}}$ transactions into the ledger L, provided they are semantically valid and pass verification by the oracle.

For each security property, we say the DAP satisfies the property if the adversary can win the corresponding game with only negligible probability.

We now state a lemma that will be useful when examining the security of our construction.

Lemma 1. *Given a ledger, two otherwise valid spend transactions reveal the same tag only if there exist coins with serial commitments S_1, S_2 produced in previous valid transactions and an extractor that produces representations of the following form:*

$$S_1 = \mathsf{Comm}(x, y, \beta_1)$$
$$S_2 = \mathsf{Comm}(x, y, \beta_2)$$

Proof. Let T be the tag common to the two spend transactions. Each transaction has a valid modified Chaum-Pedersen proof. One transaction's valid proof yields statement values $T, S_1' \in \mathbb{G}$ and witness values $x_1, y_1, z_1 \in \mathbb{F}$ such that $U = x_1 T + y_1 G$ and $S_1' = x_1 F + y_1 G + z_1 H$. Similarly, the other transaction's valid proof yields statement values $T, S_2' \in \mathbb{G}$ and witness values $x_2, y_2, z_2 \in \mathbb{F}$ such that $U = x_2 T + y_2 G$ and $S_2' = x_2 F + y_2 G + z_2 H$. Since U and G are independent and Pedersen commitments are computationally binding, we must have (except with negligible probability) that $x_1 = x_2 = x$ and $y_1 = y_2 = y$. Hence $S_1' = xF + yG + z_1 H$ and $S_2' = xF + yG + z_2 H$.

Each transaction further has a valid parallel one-of-many proof. From the first transaction's proof we have (by index extraction referencing an element of its input cover set) a group element $S_1 \in \mathbb{G}$ and scalar $\alpha_1 \in \mathbb{F}$ such that $S_1 - S_1' = \alpha_1 H$. For the second proof, we similarly have $S_2 \in \mathbb{G}$ and $\alpha_2 \in \mathbb{F}$ such that $S_2 - S_2' = \alpha_2 H$.

This means in particular that

$$S_1 = xF + yG + (z_1 + \alpha_1)H$$

and

$$S_2 = xF + yG + (z_2 + \alpha_2)H$$

by combining these results. Since transaction validity requires all input cover set elements to exist as outputs of previous valid transactions, we have extracted representations of the desired form by setting $\beta_1 = z_1 + \alpha_1$ and $\beta_2 = z_2 + \alpha_2$.

Observe that the result also holds for duplicate tags revealed in the same (otherwise valid) transaction, with almost identical reasoning.

C.1 Completeness

Completeness requires that no bounded adversary can prevent an honest user from spending a coin. [5] Specifically, this means that if the user is able to identify a coin using its incoming view key, then it can recover the coin using its full view key and generate a valid spend transaction consuming the coin using its spend key.

To see why this property holds, note that by construction, if an honest user is unable to produce a spend transaction for a coin with serial commitment S that it has recovered, the corresponding tag T must appear in a previous valid transaction. The identified coin S must be a commitment of the form $S = \mathsf{Comm}(s, r, 0)$ for serial number s and spend key component r according to the $\mathsf{Identify}$ definition. By Lemma 1, any previous transaction revealing T must consume a coin with serial commitment $\overline{S} = \mathsf{Comm}(s, r, z)$ for $z \neq 0$ (since coins must have unique serial commitments). Since the user cannot have identified $\overline{S}$ because z is nonzero, it did not generate the transaction consuming $\overline{S}$, a contradiction since that transaction implies knowledge of the spend key r by extraction.

C.2 Balance

Balance requires that no bounded adversary $\mathcal{A}$ can control more coins than are minted or spent to it. It is formalized by a BAL game. The adversary $\mathcal{A}$ adaptively interacts with $\mathcal{C}$ and the oracle with queries, and at the end of the interaction outputs a set of coins $\mathsf{AdvCoins}$. Letting ADDR be set of all addresses of honest users generated by $\mathsf{CreateAddress}$ queries, $\mathcal{A}$ wins the game if

$$v_{\text{unspent}} + v_{\mathcal{A}\to\mathsf{ADDR}} > v_{\text{mint}} + v_{\mathsf{ADDR}\to\mathcal{A}},$$

which implies that the total value the adversary can spend or has spent already is greater than the value it has minted or received. Here:

- v_{unspent} is the total value of unspent coins in $\mathsf{AdvCoins}$;
- v_{mint} is the total value minted by $\mathcal{A}$ to itself through Mint or Insert queries;
- $v_{\mathsf{ADDR}\to\mathcal{A}}$ is the total value of coins received by $\mathcal{A}$ from addresses in ADDR; and

[5] We note that the security model does not capture the case where an adversary observes a valid transaction prior to its addition to the ledger and generates its own transaction producing a coin with the same serial commitment, possibly rendering the honest transaction invalid. It is possible to mitigate this by allowing duplicate serial commitments on the ledger and requiring either coin identification to account for this by only accepting one such coin, or by binding unique data like linking tags or other additional context into serial commitments to be checked during coin identification.

– $v_{\mathcal{A}\to\mathsf{ADDR}}$ is the total value of coins sent by the adversary to the addresses in ADDR.

We say a DAP scheme Π is BAL-secure if the adversary $\mathcal{A}$ wins the game BAL only with negligible probability:

$$\Pr[\mathsf{BAL}(\Pi, \mathcal{A}, \lambda) = 1] \leq \mathrm{negl}(\lambda)$$

Assume the challenger maintains an extra augmented ledger $(L, \vec{a})$ where each a_i contains secret data from transaction tx_i in L. In that case where tx_i was produced by a query from $\mathcal{A}$ to the challenger $\mathcal{C}$, a_i contains all secret data used by $\mathcal{C}$ to produce the transaction. If instead tx_i was produced by a direct Insert query from $\mathcal{A}$, a_i consists of all extracted witness data from proofs contained in the transaction. The resulting augmented ledger $(L, \vec{a})$ is balanced if the following conditions are true:

1. Each valid spend transaction $\mathrm{tx}_{\mathrm{spend},k}$ in $(L, \vec{a})$ consumes distinct coins, and each consumed coin is the output of a valid $\mathrm{tx}_{\mathrm{mint},i}$ or $\mathrm{tx}_{\mathrm{spend},j}$ transaction for some $i < k$ or $j < k$. This requirement implies that all transactions spend only valid coins, and that no coin is spent more than once within the same valid transaction.
2. No two valid spend transactions in $(L, \vec{a})$ consume the same coin. This implies no coin is spent through two different transactions. Together with the first requirement, this implies that each coin is spent at most once.
3. For each $(\mathrm{tx}_{\mathrm{spend}}, a)$ in $(L, \vec{a})$ consuming input coins with value commitments $\{C_u\}_{u=0}^{w-1}$, for each $u \in [0, w)$:
   – If C_u is the output of a valid mint transaction with augmented ledger witness a', then the value of C_u contained in a' is the same as the corresponding value contained in a for the value commitment offset C'_u.
   – If C_u is the output of a valid spend transaction with augmented ledger witness a', then the value of C_u contained in a' is the same as the corresponding value contained in a for the value commitment offset C'_u.

   This implies that values are maintained between transactions.
4. For each $(\mathrm{tx}_{\mathrm{spend}}, a)$ in $(L, \vec{a})$ with fee f that consumes input coins with value commitment offsets $\{C'_u\}_{u=0}^{w-1}$ and generates coins with value commitments $\{\overline{C}_j\}_{j=0}^{t-1}$, a contains values $\{v_u\}_{u=0}^{w-1}$ and $\{\overline{v}_j\}_{j=0}^{t-1}$ corresponding to the commitments such that the balance equation

$$\sum_{u=0}^{w-1} v_u = \sum_{j=0}^{t-1} \overline{v}_j + f$$

   holds. For each $(\mathrm{tx}_{\mathrm{mint}}, a)$ in $(L, \vec{a})$ generating coins with value commitments $\{\overline{C}_j\}_{j=0}^{t-1}$ and public values $\{\overline{v}_j\}_{j=0}^{t-1}$, a contains values $\{\overline{v}'_j\}_{j=0}^{t-1}$ corresponding to the commitments such that $\overline{v}_j = \overline{v}'_j$ for all $j \in [0, t)$. This implies that values cannot be created arbitrarily.
5. For each $\mathrm{tx}_{\mathrm{spend}}$ in $(L, \vec{a})$ inserted by $\mathcal{A}$ through an Insert query, each consumed coin in $\mathrm{tx}_{\mathrm{spend}}$ is not recoverable by any address in ADDR. This implies that the adversary cannot generate a transaction consuming coins it does not control.

If these five conditions hold, then $\mathcal{A}$ did not spend or control more money than was previously minted or spent to it, and the inequality

$$v_{\text{mint}} + v_{\mathsf{ADDR}\to\mathcal{A}} \leq v_{\text{unspent}} + v_{\mathcal{A}\to\mathsf{ADDR}}$$

holds. We now prove that Spark is BAL-secure under this definition.

Proof. By way of contradiction, assume the adversary $\mathcal{A}$ interacts with $\mathcal{C}$ leading to a non-balanced augmented ledger $(L, \vec{a})$ with non-negligible probability; then at least one of the five conditions described above is violated with non-negligible probability:

$\mathcal{A}$ violates Condition 1: Suppose that the probability $\mathcal{A}$ wins the game violating Condition 1 is non-negligible. Each tx_{spend} generated by a non-Insert oracle query satisfies this condition already, so there must exist a transaction $(\text{tx}_{\text{spend}}, a)$ in $(L, \vec{a})$ inserted by $\mathcal{A}$.

Suppose there exist inputs $u_1, u_2 \in [0, w)$ of tx_{spend} that consume the same coin with serial commitment S. Validity of the modified Chaum-Pedersen proof Π_{chaum} gives extracted openings $S'_{u_1} = s_{u_1}F + r_{u_1}G + y_{u_1}H$ and $S'_{u_2} = s_{u_2}F + r_{u_2}G + y_{u_2}H$ and tag representations such that $U = s_{u_1}T_{u_1} + r_{u_1}G$ and $U = s_{u_2}T_{u_2} + r_{u_2}G$. Because transaction validity implies $T_{u_1} \neq T_{u_2}$, we must have $(s_{u_1}, r_{u_1}) \neq (s_{u_2}, r_{u_2})$. Validity of the corresponding parallel one-out-of-many proofs $(\Pi_{\text{par}})_{u_1}$ and $(\Pi_{\text{par}})_{u_2}$ yields indices (corresponding to the same input set group element S) and discrete logarithm extractions such that $S - S'_{u_1} = x_{u_1}H$ and $S - S'_{u_2} = x_{u_2}H$. This means

$$S = \mathsf{Comm}(s_{u_1}, r_{u_1}, x_{u_1} + y_{u_1}) = \mathsf{Comm}(s_{u_2}, r_{u_2}, x_{u_2} + y_{u_2}),$$

a contradiction since the commitment scheme is computationally binding.

The second possibility for violation of the condition is that the transaction tx_{spend} consumes a coin that is not generated in any previous valid transaction. This follows immediately using similar reasoning as above, since transaction validity asserts knowledge of an opening to a commitment contained in the input set, all of which must have been previously generated in valid transactions by definition.

$\mathcal{A}$ violates Condition 2: Suppose that the probability $\mathcal{A}$ wins the game violating Condition 2 is non-negligible. This means the augmented ledger $(L, \vec{a})$ contains two valid spend transactions consuming the same coin but producing distinct tags. Similarly to the previous argument, this implies distinct openings of the coin serial number commitment, which is a contradiction.

$\mathcal{A}$ violates Condition 3: Suppose that the probability $\mathcal{A}$ wins the game violating Condition 3 is non-negligible. Let C be the value commitment of the coin consumed by an input of tx_{spend} and generated in a previous transaction (of either type) in $(L, \vec{a})$. Since the generating transaction is valid, we have an extracted opening $C = vG + aH$ from either the value proof (in a mint transasction) or the range proof (in a spend transaction). Validity of the corresponding parallel one-out-of-many proof in tx_{spend} gives an extracted discrete logarithm

$C - C' = xH$, where C' is the input's value commitment offset. But this immediately gives $C' = vG + (a - x)H$, a contradiction since the commitment scheme is binding.

$\mathcal{A}$ violates Condition 4: Suppose that the probability $\mathcal{A}$ wins the game violating Condition 4 is non-negligible. If the augmented ledger $(L, \vec{a})$ contains a spend transaction that violates the balance equation, this immediately implies a break in the commitment binding property since the corresponding balance proof Π_{bal} is valid, which is a contradiction. If instead the augmented ledger $(L, \vec{a})$ contains a mint transaction that violates the balance requirement, this immediately implies a break in the commitment binding property since the corresponding value proof Π_{val} is valid, again a contradiction.

$\mathcal{A}$ violates Condition 5: Suppose that the probability $\mathcal{A}$ wins the game violating Condition 5 is non-negligible. That is, $\mathcal{A}$ produces a spend transaction tx_{spend} by an Insert question that is valid on the augmented ledger $(L, \vec{a})$ and consumes a coin corresponding to a coin serial number commitment S that can be recovered by a public address $(d, Q_1, Q_2) \in \mathsf{ADDR}$.

Validity of the Chaum-Pedersen proof corresponding to tx_{spend} yields an extracted representation $S' = s'F + r'G + yH$. Validity of the corresponding parallel one-of-many proof gives a serial number commitment S and extraction such that $S - S' = xH$, so $S = s'F + r'G + (x + y)H$.

Now let (s_1, s_2, r) be the spend key corresponding to the address (d, Q_1, Q_2). Since tx_{spend} consumes a coin recoverable by this address, a serial number commitment for the recovered coin is

$$\begin{aligned}\overline{S} &= \mathcal{H}_{\text{ser}}(k)F + Q_2 \\ &= (\mathcal{H}_{\text{ser}}(k) + \mathcal{H}_{Q_2}(s_1, i) + s_2)F + rG\end{aligned}$$

for nonce k and some diversifier index i.

Since the commitment scheme is binding, we must therefore have $r' = r$, which is a contradiction since $\mathcal{A}$ cannot extract this discrete logarithm from the public address.

This completes the proof.

C.3 Transaction Non-Malleability

This property requires that no bounded adversary can substantively alter a valid transaction. In particular, non-malleability prevents malicious adversaries from modifying honest users' transactions by altering data or redirecting the outputs of a valid transaction before the transaction is added to the ledger. Since non-malleability of mint transactions is offloaded to authorizations relating to consensus rules or base-layer operations, we need only consider the case of spend transactions.

This property is formalized by an experiment TRNM, in which a bounded adversary $\mathcal{A}$ adaptively interacts with the oracle $\mathcal{O}^{\mathsf{DAP}}$, and then outputs a spend transaction tx'. If we let T denote the set of all transactions produced by Spend queries to $\mathcal{O}^{\mathsf{DAP}}$, and L denote the final ledger, $\mathcal{A}$ wins the game if there exists $\text{tx} \in T$ such that:

– $\mathrm{tx}' \neq \mathrm{tx}$;
– tx' reveals a tag also revealed by tx; and
– both tx' and tx are valid transactions with respect to the ledger L' containing all transactions preceding tx on L.

We say a DAP scheme Π is TRNM-secure if the adversary $\mathcal{A}$ wins the game TRNM only with negligible probability:

$$\Pr[\mathsf{TRNM}(\Pi, \mathcal{A}, \lambda) = 1] \leq \mathrm{negl}(\lambda)$$

Let $\mathcal{T}$ be the set of all $\mathrm{tx}_{\mathrm{spend}}$ transactions generated by the $\mathcal{O}^{\mathsf{DAP}}$ in response to Spend queries. Since these transactions are generated by these oracle queries, $\mathcal{A}$ does not learn any secret data used to produce these transactions.

Proof. Assume that the adversary $\mathcal{A}$ wins the game with non-negligible probability. That is, $\mathcal{A}$ produces a transaction tx' revealing a tag T also revealed in a transaction tx. Without loss of generality, assume each transaction consumes a single coin.

Observe that a valid spend binds all transaction elements except for the modified Chaum-Pedersen proof into each such proof via $\mathcal{H}_{\mathrm{bind}}$ and the proof transcripts. Therefore, in order to produce valid $\mathrm{tx}' \neq \mathrm{tx}$, we consider two cases:

– the modified Chaum-Pedersen proofs are identical, but tx' and tx differ in another element of the transaction structures; or
– the modified Chaum-Pedersen proof in tx' is distinct from the proof in tx.

In the first case, at least one input to the binding hash $\mathcal{H}_{\mathrm{bind}}$ used to initialize the modified Chaum-Pedersen transcripts must differ between the proofs. Because we model this hash function as a random oracle, the outputs differ except with negligible probability, a contradiction since the resulting proof structures must be identical.

In the second case, because the tag revealed in both tx' and tx is identical, Lemma 1 gives extractions of the form (s, r, y) and $(s, r, 0)$ respectively. Further, the coin $S = \mathsf{Comm}(s, r, 0)$ consumed in tx was generated such that r is a spend key component for an address (d, Q_1, Q_2) not controlled by $\mathcal{A}$. Since $\mathcal{A}$ does not control this address, it cannot produce r without extracting from S or from any set of corresponding diversified address components $\{Q_{2,i}\}_i$ produced from the same spend key. However, each $Q_{2,i}$ is produced linearly against Q_2 and querying $\mathcal{H}_{Q_2}$ with unique (s_1, i) input, a contradiction.

C.4 Ledger Indistinguishability

This property implies that no bounded adversary $\mathcal{A}$ received any information from the ledger except what is already publicly revealed, even if it can influence valid ledger operations by honest users.

Ledger indistinguishability is formalized through an experiment LIND between a bounded adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, which terminates with a binary

output b' by $\mathcal{A}$. At the beginning of the experiment, $\mathcal{C}$ samples $\mathsf{Setup}(1^\lambda) \to pp$ and sends the parameters to $\mathcal{A}$; next it samples a random bit $b \in \{0,1\}$ and initializes two separate DAP oracles $\mathcal{O}_0^{\mathsf{DAP}}$ and $\mathcal{O}_1^{\mathsf{DAP}}$, each with its own separate ledger and internal state. At each consecutive step of the experiment:

1. $\mathcal{C}$ provides $\mathcal{A}$ two ledgers $(L_{\text{left}} = L_b, L_{\text{right}} = L_{1-b})$ where L_b and L_{1-b} are the current ledgers of the oracles $\mathcal{O}_b^{\mathsf{DAP}}$ and $\mathcal{O}_{1-b}^{\mathsf{DAP}}$ respectively.
2. $\mathcal{A}$ sends to $\mathcal{C}$ two queries Q, Q' of the same type (one of CreateAddress, Mint, Spend, or Insert).
   - If the query type is Insert or Mint, $\mathcal{C}$ forwards Q to L_b and Q' to L_{1-b}, permitting $\mathcal{A}$ to insert its own transactions or mint new coins to L_{left} and L_{right}.
   - For all queries of type CreateAddress or Spend, $\mathcal{C}$ first checks if the two queries Q and Q' are publicly consistent, and then forwards Q to $\mathcal{O}_0^{\mathsf{DAP}}$ and Q' to $\mathcal{O}_1^{\mathsf{DAP}}$. It receives the two oracle answers (a_0, a_1), but returns (a_b, a_{1-b}) to $\mathcal{A}$.

As the adversary does not know the bit b and the mapping between $(L_{\text{left}}, L_{\text{right}})$ and (L_0, L_1), it cannot learn whether it affects the behavior of honest parties on (L_0, L_1) or on (L_1, L_0). At the end of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a bit $b' \in \{0,1\}$. The challenger outputs 1 if $b = b'$, and 0 otherwise.

We require the queries Q and Q' be publicly consistent as follows If the query type of Q and Q' is CreateAddress, both oracles generate the same address. If the query type of Q and Q' is Mint, then the number of generated coins and the public value of each coin must be equal in both queries. If the query type of Q and Q' is Spend, then:

- Both Q and Q' must be well-formed and valid, so the referenced input coins must have been generated in a previous transaction on the ledger and be unspent. Further, the transaction must balance.
- The number of spent coins and output coins must be the same in Q and Q'.
- If a consumed coin in Q references a coin in L_0 posted by $\mathcal{A}$ through an Insert query, then the corresponding index in Q' must also reference a coin in L_1 posted by $\mathcal{A}$ through an Insert query and the values of these two coins must be equal as well (and vice versa for Q').
- If an output coin referenced by Q does not reference a recipient address in the oracle ADDR list (and therefore is controlled by $\mathcal{A}$), then the corresponding value must equal that of the corresponding coin referenced by Q at the same index (and vice versa for Q').

We say a DAP scheme Π is LIND-secure if $\mathcal{A}$ wins the game LIND only probability at most negligibly better than chance:

$$\Pr[\mathsf{LIND}(\Pi, \mathcal{A}, \lambda) = 1] - \frac{1}{2} \leq \text{negl}(\lambda)$$

Proof. In order to prove that $\mathcal{A}$'s advantage in the LIND experiment is negligible, we first consider a simulation experiment $\mathcal{D}^{\text{sim}}$, in which $\mathcal{A}$ interacts with $\mathcal{C}$ as in the LIND experiment, but with modifications.

The simulation experiment $\mathcal{D}^{\mathbf{sim}}$: Since the parallel one-out-of-many, modified Chaum-Pedersen, representation, and range proving systems are all special honest-verifier zero knowledge, we can take advantage of the simulator for each. Given input statements and verifier challenges, each proving system's simulator produces transcripts indistinguishable from honest proofs. Additionally, we now define the behavior of the full simulator.

The simulation. The simulation $\mathcal{D}^{\text{sim}}$ works as follows. As in the original experiment, $\mathcal{C}$ samples the system parameters $\mathsf{Setup}(1^\lambda) \to pp$ and a random bit b, and initializes DAP oracles $\mathcal{O}_0^{\mathsf{DAP}}$ and $\mathcal{O}_1^{\mathsf{DAP}}$. Then $\mathcal{D}^{\text{sim}}$ proceeds in steps. At each step, it provides $\mathcal{A}$ with ledgers $L_{\text{left}} = L_b$ and $L_{\text{right}} = L_{1-b}$, after which $\mathcal{A}$ sends two publicly-consistent queries (Q, Q') of the same type. Recall that the queries Q and Q' are consistent with respect to public data and information related to the addresses controlled by $\mathcal{A}$. Depending on the query type, the challenger acts as follows:

- Answering Insert queries: The challenger proceeds as in the original LIND experiment.
- Answering $\mathsf{CreateAddress}$ queries: In this case the challenger replaces the public address components (d, Q_1, Q_2) with random strings of the appropriate lengths, producing $\mathsf{addr}_{\text{pk}}$ that is returned to $\mathcal{A}$.
- Answering Mint queries: The challenger does the following to answer Q and Q' separately, where t is the number of generated coins specified by $\mathcal{A}$ as part of its queries:
  1. For each $j \in [0, t)]$:
     (a) If $\mathcal{A}$ provided a public address $\mathsf{addr}_{\text{pk}}$ not generated by the challenger, it produces a coin using $\mathsf{CreateCoin}$ as usual.
     (b) Otherwise, it simulates coin generation:
        i. Samples a recovery key K_j uniformly at random.
        ii. Samples a serial number commitment S_j uniformly at random.
        iii. Samples a value commitment $\overline{C}_j$ uniformly at random.
        iv. Samples a random input used to produce an AEAD encryption key $\mathsf{AEADKeyGen} \to k_{\text{enc}}$.
        v. Simulates the recipient data encryption by selecting random r of the proper length, and encrypting it to produce
        $$\mathsf{AEADEncrypt}(k_{\text{enc}}, \mathtt{r}, r) \to \overline{r}.$$
  2. Simulates the value proof Π_{val} on the statement $\{\overline{C}_j - \mathsf{Com}(v_j, 0)\}_{j=0}^{t-1}$.
  3. Assembles the transaction and adds it to the ledger as appropriate.
- Answering Spend queries: The challenger does the following to answer Q and Q' separately, where w is the number of consumed coins and t the number of generated coins specified by $\mathcal{A}$ as part of its queries:
  1. For each $u \in [0, w)$, where l_u represents the index of the consumed coin in $\mathsf{InCoins}_u$:
     (a) Parses the input cover set serial number commitments and value commitments as $\mathsf{InCoins}_u = \{(S_{i,u}, C_{i,u})\}_{i=0}^{N-1}$.
     (b) Samples a tag T_u uniformly at random.

(c) Samples a serial number commitment offset S'_u and value commitment offset C'_u uniformly at random.
(d) Simulates a parallel one-out-of-many proof $(\Pi_{\text{par}})_u$ on the statement $(\{S_{i,u}, C_{i,u}\}_{i=0}^{N-1}, S'_u, C'_u)$.

2. For each $j \in [0, t)$:
   (a) If $\mathcal{A}$ provided a public address $\mathsf{addr}_{\text{pk}}$ not generated by the challenger, it produces a coin using $\mathsf{CreateCoin}$ as usual.
   (b) Otherwise, it simulates coin generation:
      i. Samples a recovery key K_j uniformly at random.
      ii. Samples a serial number commitment S_j uniformly at random.
      iii. Samples a value commitment $\overline{C}_j$ uniformly at random.
      iv. Samples a random input used to produce an AEAD encryption key $\mathsf{AEADKeyGen} \to k_{\text{enc}}$.
      v. Simulates the recipient data encryption by selecting random r of the proper length, and encrypting it to produce

$$\mathsf{AEADEncrypt}(k_{\text{enc}}, \mathtt{r}, r) \to \overline{r}.$$

      vi. Simulates a range proof $(\Pi_{\text{rp}})_j$ on the statement $(\overline{C}_j)$.
3. Simulates the balance proof Π_{bal} on the statement

$$\left(\sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \mathsf{Com}(f, 0) \right).$$

4. For each $u \in [0, w)$, computes the binding hash μ as defined and simulates the modified Chaum-Pedersen proof Π_{chaum} on the statement $(\{S'_u, T_u\}_{u=0}^{w-1})$.
5. Assembles the transaction and adds it to the ledger as appropriate.

For experiments defined below, we define $\mathsf{Adv}^{\mathcal{D}}$ as the advantage of $\mathcal{A}$ in some experiment $\mathcal{D}$ over the original LIND game. By definition, all answers sent to $\mathcal{A}$ in $\mathcal{D}^{\text{sim}}$ are computed independently of the bit b, so $\mathsf{Adv}^{\mathcal{D}^{\text{sim}}} = 0$. We will prove that $\mathcal{A}$'s advantage in the real experiment $\mathcal{D}^{\text{real}}$ is at most negligibly different than $\mathcal{A}$'s advantage in $\mathcal{D}^{\text{sim}}$. To show this, we construct intermediate experiments in which $\mathcal{C}$ performs a specific modification of $\mathcal{D}^{\text{real}}$ against $\mathcal{A}$.

Experiment $\mathcal{D}_1$: This experiment modifies $\mathcal{D}^{\text{real}}$ by simulating all one-out-of-many proofs, range proofs, representation proofs, and modified Chaum-Pedersen proof. As all these protocols are special honest-verifier zero knowledge, the simulated proofs are indistinguishable from the real proofs generated in $\mathcal{D}^{\text{real}}$. Hence $\mathsf{Adv}^{\mathcal{D}_1} = 0$.

Experiment $\mathcal{D}_2$: This experiment modifies $\mathcal{D}_1$ by replacing all encrypted recipient data in transactions with challenger-generated recipient public addresses with encryptions of random values of appropriate lengths under keys chosen uniformly at random, and by replacing recovery keys with uniformly random values. Since the underlying authenticated symmetric encryption scheme is IND-CCA and IK-CCA secure and we assume the decisional Diffie-Hellman problem is

hard, the adversarial advantage in distinguishing ledger output in the $\mathcal{D}_2$ experiment is negligibly different from its advantage in the $\mathcal{D}_1$ experiment. Hence $|\mathsf{Adv}^{\mathcal{D}_2} - \mathsf{Adv}^{\mathcal{D}_1}|$ is negligible.

Experiment $\mathcal{D}^{\mathbf{sim}}$: The $\mathcal{D}^{\mathrm{sim}}$ experiment is formally defined above. In particular, it differs from $\mathcal{D}_2$ by replacing consumed coin tags, serial number commitment offset, and value commitment offsets with uniformly random values; and by replacing output coin serial number and value commitments with random values. In previous experiments (including $\mathcal{D}^{\mathrm{real}}$), tags are generated using a pseudorandom function [11], and the other given values are generated as commitments with masks derived from hash functions modeled as independent random oracles, so the adversarial advantage in distinguishing ledger output in $\mathcal{D}^{\mathrm{sim}}$ is negligibly different from its advantage in the $\mathcal{D}_2$ experiment. Hence $|\mathsf{Adv}^{\mathcal{D}^{\mathrm{sim}}} - \mathsf{Adv}^{\mathcal{D}_2}|$ is negligible.

This shows that the adversary has only negligible advantage in the real LIND game over the simulation, where it can do no better than chance, which completes the proof.

D Payment Proofs

We describe now the informal security properties required for a payment proving system, describe such a construction, and (informally) prove that our construction meets the requirements.

The security requirements of a payment proving system are as follows:

1. The proof cannot be replayed in a different context.
2. The prover asserts that it knows secret data sufficient to authorize the transaction originally generating a specified coin.
3. The verifier can obtain and confirm the value and memo associated to the coin.
4. The holder of an incoming view key corresponding to a specified public address can successfully identify the coin.
5. A computationally-bound adversary cannot produce valid proofs for the same coin claiming distinct public addresses.

We note that coin identification relies on the assumption that the claimed recipient address was generated using the protocol-specified method from an incoming view key.

D.1 Protocol

A prover wishes to produce a payment proof on a given coin Coin with nonce k to a claimed destination public address (d, Q_1, Q_2). Suppose that tx is the spend transaction on a ledger that produced Coin. The prover does the following:

1. Parses the serial number commitment, value commitment, and recovery key from Coin: (S, C, K)

2. Generates a modified Chaum-Pedersen proof Π_{auth} using the same inputs and proving system as the proof Π_{chaum} from tx, but also binding the tuple $(\mathsf{Coin}, k, d, Q_1, Q_2)$, any context relevant to the payment proof instance, and a globally-fixed payment proof domain separator to the proof context.
3. Assembles the payment proof: $\Pi_{\text{pay}} = (\mathsf{Coin}, k, d, Q_1, Q_2, \Pi_{\text{auth}})$

To verify a payment proof on a coin, the verifier does the following:

1. Parses the payment proof: $\Pi_{\text{pay}} = (\mathsf{Coin}, k, d, Q_1, Q_2, \Pi_{\text{auth}})$
2. Parses public data from Coin: $(S, C, K, \overline{r})$
3. Verifies that tx is a valid transacton on its own ledger that originally generated Coin.
4. Verifies the proof Π_{auth} using the data from tx and the additional binding tuple $(\mathsf{Coin}, k, d, Q_1, Q_2)$, and aborts if verification fails.
5. Generates an AEAD key $k_{\text{aead}} = \mathsf{AEADKeyGen}(\mathcal{H}_k(k)Q_1)$ and decrypts the recipient data:

$$(v, d', k', m) = \mathsf{AEADDecrypt}(k_{\text{aead}}, \mathtt{r}, \overline{r})$$

   If decryption fails, or if $k' \neq k$, or if $d' \neq d$, aborts.
6. Checks that $K = \mathcal{H}_k(k)\mathcal{H}_{\text{div}}(d)$, and aborts otherwise.
7. Checks that $S = \mathsf{Comm}(\mathcal{H}_{\text{ser}}(k), 0, 0) + Q_2$, and aborts otherwise.
8. Checks that $C = \mathsf{Com}(v, \mathcal{H}_{\text{val}}(k))$, and aborts otherwise.

D.2 Security

We now describe why this construction meets our security requirements.

Requirement 1. To show that a payment proof cannot be replayed in another context, note that since proof context is bound to the transcript of Π_{auth} along with the statement and coin data, the overall payment proof Π_{pay} cannot be successfully replayed against any other context.

Requirement 2. To show that successful verification of a payment proof asserts the prover knows secret data sufficient to authorize the transaction that generated the coin, we simply note that the modified Chaum-Pedersen proof Π_{chaum} from tx uses the same statement input as Π_{auth} (albeit with different proof context).

Requirement 3. To show that the verifier can obtain the correct value and memo for the coin on successful verification of a payment proof, we simply note that successful AEAD decryption provides the unique values for the value and memo originally used to produce the coin, and that the decrypted value uniquely corresponds to the coin's value commitment since the commitment scheme is binding and successful verification implies an opening to this commitment.

Requirement 4. We now show that if the given address (d, Q_1, Q_2) was generated from an incoming view key (s_1, P_2), this key can identify Coin if a payment proof verifies. Successful verification of a payment proof implies in particular that $K = \mathcal{H}_k(k)\mathcal{H}_{\text{div}}(d)$ and that AEAD decryption succeeds on a

key generated using $\mathcal{H}_k(k)Q_1$. This implies that the incoming view holder uses the AEAD key

$$\begin{aligned} s_1 K &= s_1 \mathcal{H}_k(k) \mathcal{H}_{\text{div}}(d) \\ &= \mathcal{H}_k(k) Q_1 \end{aligned}$$

and hence decryption succeeds. The remaining steps for identification follow from corresponding steps taken during payment proof verification.

Requirement 5. We now show that a computationally-bound adversary cannot produce valid proofs against the same coin for distinct destination addresses. Suppose such an adversary produces for the same coin valid payment proofs

$$\Pi_{\text{pay}} = (\mathsf{Coin}, k, d, Q_1, Q_2, \Pi_{\text{auth}})$$

and

$$\Pi'_{\text{pay}} = (\mathsf{Coin}, k', d', Q'_1, Q'_2, \Pi'_{\text{auth}})$$

on addresses $(d, Q_1, Q_2) \neq (d', Q'_1, Q'_2)$. Note that $\mathsf{Coin} = (S, C, K, \overline{r})$ must be identical in both proofs by definition.

Successful AEAD decryption of $\overline{r}$ on both proofs implies in particular that $d = d'$ and $k = k'$ except with negligible probability. Further, the AEAD keys derived in both proofs must be equal (again except with negligible probability), so $\mathcal{H}_k(k)Q_1 = \mathcal{H}_k(k')Q'_1$ requires $Q_1 = Q'_1$. Finally, since

$$\begin{aligned} S &= \mathsf{Comm}(\mathcal{H}_{\text{ser}}(k), 0, 0) + Q_2 \\ &= \mathsf{Comm}(\mathcal{H}_{\text{ser}}(k'), 0, 0) + Q'_2 \end{aligned}$$

it also follows that $Q_2 = Q'_2$, a contradiction.