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 [17, 9]. The Sprout and Sapling protocols supported by Zcash [11] (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 [2, 4]. The Mimblewimble-based construction used as the basis for Grin can leak graph information prior to a merging operation performed by miners [8]. 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 [19]. 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 [12].

Here we introduce Spark, an iteration on the Lelantus protocol enabling trustless private transactions which supports sender, receiver 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.* [10, 3] (of independent interest and used in other modified forms in Lelantus[12] and Triptych [16]) 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 [7, 13] 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 correct tag construction, enables efficient signing and multisignature operations similar to those of [14] where computationally-expensive proofs may be offloaded to more capable devices with limited trust requirements. The protocol further adds two 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.

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 $pp_{\text{com}} = (\mathbb{G}, \mathbb{F}, G, F)$ 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 $G, F \in \mathbb{G}$ are uniformly-sampled independent generators. The commitment scheme contains an algorithm $\text{Com} : \mathbb{F}^2 \to \mathbb{G}$, where $\text{Com}(v, r) = vG + rF$ that is homomorphic in the sense that

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

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

We also require a matrix version of this construction, where input values v are matrices with entries in $\mathbb{F}$, and the generator G is replaced with a corresponding matrix of independent generators. The definition and security properties extend naturally to this case.

2.2 Representation proving system

A representation proof is used to demonstrate knowledge of a discrete logarithm. 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 $(\text{RepProve}, \text{RepVerify})$ for the following relation:

$$\{pp_{\text{rep}}, G, X \in \mathbb{G}; x \in \mathbb{F} : X = xG\}$$

The well-known Schnorr proving system may be used for this purpose.

2.3 Modified Chaum-Pedersen Proving System

A Chaum-Pedersen proof is used to demonstrate discrete logarithm equality. Here we require a modification to the standard proving system that uses a second group generator. Let $pp_{\text{chaum}} = (\mathbb{G}, \mathbb{F}, G, F, H)$ 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 $G, F, H \in \mathbb{G}$ are uniformly-sampled independent generators.

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

$$\{pp_{\text{chaum}}, Y, Z \in \mathbb{G}; (x, y) \in \mathbb{F} : Y = xG + yF, H = xZ\}$$

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 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 [10] that was generalized by Bootle *et al.* [3].

Let $pp_{\text{par}} = (\mathbb{G}, \mathbb{F}, n, m, pp_{\text{com}})$ 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, and pp_{com} are the public parameters for a Pedersen commitment (and matrix commitment) construction.

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

$$\begin{aligned}\{pp_{\text{par}}, \{S_i, V_i\}_{i=0}^{N-1} \subset \mathbb{G}^2; l \in \mathbb{N}, (s, v) \in \mathbb{F} :&\\ 0 \leq l < N, S_l = \text{Com}(0, s), V_l = \text{Com}(0, v)\}&\end{aligned}$$

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

2.5 Authenticated Encryption Scheme

We require the use of an authenticated symmetric encryption scheme. In the context of the Spark protocol, this construction is used to encrypt value and arbitrary memo data for use by the sender and recipient of a transaction.

Let pp_{sym} be the public parameters for such a construction. The construction itself is a tuple of algorithms $(\text{SymKeyGen}, \text{SymEncrypt}, \text{SymDecrypt})$. Here SymKeyGen is a key derivation function that accepts as input an arbitrary string, and produces a key in the appropriate key space. The algorithm SymEncrypt accepts as input a key and arbitrary message string, and produces ciphertext in the appropriate space. The algorithm SymDecrypt accepts as input a key and ciphertext string, and produces a message in the appropriate space if authentication succeeds.

Assume that such a construction is indistinguishable against chosen-plaintext attack (IND-CPA), indistinguishable against adaptive chosen-ciphertext attack (IND-CCA2), and key-private under chosen-ciphertext attacks (IK-CCA) in this context.

2.6 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 $(\text{RangeProve}, \text{RangeVerify})$ for the following relation:

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

In practice, an efficient instantiation like Bulletproofs [5] or Bulletproofs+ [6] 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.

Addresses. Users generate addresses that enable transactions. An address consists of a tuple

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

For each address, addr_{pk} is the public address used for receiving funds, addr_{in} is an incoming view key used to identify received funds, $\text{addr}_{\text{full}}$ is a full view key used to identify outgoing funds and conduct computationally-heavy proving operations, and addr_{sk} is the spend key used to generate transactions.

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

– A (secret) serial number that uniquely defines the coin.
– A serial number commitment.
– An integer value for the coin.
– An encrypted value intended for decryption by the recipient.
– A value commitment.
– A range proof for the value commitment, or a proof that a plaintext value is represented by the value commitment.
– A memo with arbitrary recipient data.
– An encrypted memo intended for decryption by the recipient.
– A recovery key used by the recipient to identify the coin and decrypt private data.

Coins additionally bind recipient addresses in an indistinguishable way; this may be useful for out-of-band payment proofs that require such binding.

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 coin is of the expected value. A *Mint* transaction creates transaction data $\mathrm{tx}_{\mathrm{mint}}$ for recording on a ledger.
- *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. A *Spend* transaction creates transaction data $\mathrm{tx}_{\mathrm{spend}}$ for recording on a ledger.

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 bound to coins uniquely via the serial number, 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 outputs all public parameters used by the protocol and its underlying components. The setup process does not require any trusted parameter generation.
- **CreateAddress**: This algorithm outputs a public address, incoming view key, full view key, and spend key.
- **CreateCoin**: This algorithm takes as input a public address, coin value, and memo, and outputs a coin destined for the public address.
- **Mint:** This algorithm takes as input a public address, value, and (optionally) implementation-specific data relating to base-layer outputs, and outputs a mint transaction $\mathrm{tx}_{\mathrm{mint}}$.
- **Identify:** This algorithm takes as input a coin and an incoming view key, and outputs the coin value and memo.
- **Recover:** This algorithm takes as input a coin and a full view key, and outputs the coin value, memo, serial number, and tag.
- **Spend:** This algorithm takes as input a full view key, a spend key, a set of input coins (including coins used as a larger ambiguity set), the indexes of coins to be spent, the corresponding serial numbers and values, a fee value, and a set of output coins to be generated, and outputs a spend transaction $\mathrm{tx}_{\mathrm{spend}}$.
- **Verify:** This algorithm accepts either a mint transaction or a spend transaction, and outputs a bit to assess validity.

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

4 Algorithm Constructions

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

4.1 Setup

In our setup the public parameters pp are comprised of the corresponding public parameters of a Pedersen commitment (and matrix commitment) scheme, representation proving system, modified Chaum-Pedersen proving system, parallel one-out-of-many proving system, symmetric encryption scheme, and range proving system.

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 $F, G, H \in \mathbb{G}$ uniformly at random. In practice, these generators may be chosen using a suitable cryptographic hash function on public input.
3. Sample cryptographic hash functions
$$\mathcal{H}_{\text{ser}}, \mathcal{H}_{\text{val}}, \mathcal{H}_{\text{ser}'}, \mathcal{H}_{\text{val}'}, \mathcal{H}_{\text{bind}} : \{0,1\}^* \to \mathbb{F}$$
uniformly at random. In practice, these hash functions may be chosen using domain separation of a single suitable cryptographic hash function.
4. Compute the public parameters $pp_{\text{com}} = (\mathbb{G}, \mathbb{F}, G, F)$ of a Pedersen commitment scheme.
5. Compute the public parameters $pp_{\text{rep}} = (\mathbb{G}, \mathbb{F})$ of a representation proving system.
6. Compute the public parameters $pp_{\text{chaum}} = (\mathbb{G}, \mathbb{F}, G, F, H)$ of the modified Chaum-Pedersen proving system.
7. Compute the public parameters $pp_{\text{par}} = (\mathbb{G}, \mathbb{F}, n, m, pp_{\text{com}})$ of the parallel one-out-of-many proving system.
8. Compute the public parameters pp_{sym} of an authenticated symmetric encryption scheme.
9. Compute the public parameters $pp_{\text{rp}} = (\mathbb{G}, \mathbb{F}, v_{\max}, pp_{\text{com}})$ of a range proving system.
10. Output all generated public parameters and hash functions as pp.

4.2 CreateAddress

We describe the construction of all addresses and underlying key types used in the protocol.

Inputs: Security parameter λ, public parameters pp

Outputs: Address key tuple $(\text{addr}_{\text{pk}}, \text{addr}_{\text{in}}, \text{addr}_{\text{full}}, \text{addr}_{sk})$

1. Sample $s_1, s_2, r \in \mathbb{F}$ uniformly at random, and let $D = \text{Com}(0, r)$.
2. Compute $Q_1 = \text{Com}(s_1, 0)$ and $Q_2 = \text{Com}(s_2, r)$.
3. Set $\text{addr}_{\text{pk}} = (Q_1, Q_2)$.
4. Set $\text{addr}_{\text{in}} = s_1$.
5. Set $\text{addr}_{\text{full}} = (s_1, s_2, D)$.
6. Set $\text{addr}_{\text{sk}} = (s_1, s_2, r)$.
7. Output the tuple $(\text{addr}_{\text{pk}}, \text{addr}_{\text{in}}, \text{addr}_{\text{full}}, \text{addr}_{sk})$.

4.3 CreateCoin

This algorithm generates a new coin destined for a given public address. Note that while this algorithm generates a serial number commitment, it cannot compute the underlying serial number.

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

Outputs: Coin public key S, recovery key K, value commitment C, value commitment range proof $\mathit{\Pi}_{\text{rp}}$ (if $b = 0$), encrypted value $\overline{v}$ (if $b = 0$) or value v (if $b = 1$), encrypted memo $\overline{m}$

1. Parse the recipient address $\text{addr}_{\text{pk}} = (Q_1, Q_2)$.
2. Sample $k \in \mathbb{F}$.
3. Compute the serial number commitment $S = \text{Com}(\mathcal{H}_{\text{ser}}(kQ_1, Q_1, Q_2), 0) + Q_2$.
4. Compute the recovery key $K = \text{Com}(k, 0)$.
5. Generate the value commitment $C = \text{Com}(v, \mathcal{H}_{\text{val}}(kQ_1))$.
6. If $b = 0$, generate a range proof
$$\mathit{\Pi}_{\text{rp}} = \text{RangeProve}(pp_{\text{rp}}, C; (v, \mathcal{H}_{\text{val}}(kQ_1))).$$
7. Generate a symmetric encryption key $k_{\text{sym}} = \text{SymKeyGen}(kQ_1)$; encrypt the value $\overline{v} = \text{SymEnc}(k_{\text{sym}}, v)$ (if $b = 0$) and memo $\overline{m} = \text{SymEnc}(k_{\text{sym}}, m)$.
8. If $b = 0$, output the tuple $(S, K, C, \mathit{\Pi}_{\text{rp}}, \overline{v}, \overline{m})$. Otherwise, output the tuple $(S, K, C, v, \overline{m})$.

The case $b = 0$ represents a coin with hidden value being generated in a Spend transaction, while the case $b = 0$ represents a coin with plaintext value being generated in a Mint transaction. Note that it is possible to securely aggregate range proofs within a transaction; this does not affect protocol security.

4.4 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, public parameters pp, destination public address addr_{pk}, coin value $v \in [0, v_{\max})$, memo m

Outputs: Mint transaction tx_{mint}

1. Generate the new coin $\text{CreateCoin}(\text{addr}_{\text{pk}}, v, m, 1) \to \text{Coin} = (S, K, C, v, \overline{m})$.
2. Generate a value representation proof on the value commitment:

$$\varPi_{\text{bal}} = \text{RepProve}(pp_{\text{rep}}, F, C - \text{Com}(v, 0); \mathcal{H}_{\text{val}}(kQ_1))$$

3. Output the tuple $\text{tx}_{\text{mint}} = (\text{Coin}, \varPi_{\text{bal}})$.

4.5 Identify

This algorithm allows a recipient (or designated entity) to compute the value and memo from a coin destined for its public address. It requires the incoming view key corresponding to the public address to do so. If the coin is not destined for the public address, the algorithm returns failure.

Inputs: Security parameter λ, public parameters pp, incoming view key addr_{in}, public address addr_{pk}, coin Coin.

Outputs: Value v, memo m

1. Parse the incoming view key $\text{addr}_{\text{in}} = s_1$ and public address $\text{addr}_{\text{pk}} = (Q_1, Q_2)$.
2. Parse the serial number commitment S, value commitment C, recovery key K, encrypted value $\overline{v}$ (if the coin is of type $b = 0$) or value v (if the coin is of type $b = 1$), and encrypted memo $\overline{m}$ from Coin.
3. If $\text{Com}(\mathcal{H}_{\text{ser}}(s_1 K, Q_1, Q_2), 0) + Q_2 \neq S$, return failure.
4. Generate a symmetric encryption key $k_{\text{sym}} = \text{SymKeyGen}(s_1 K)$; decrypt the value $v = \text{SymDec}(k_{\text{sym}}, \overline{v})$ (if $b = 0$) and memo $m = \text{SymDec}(k_{\text{sym}}, \overline{m})$.
5. If $\text{Com}(v, \mathcal{H}_{\text{val}}(s_1 K)) \neq C$, return failure.
6. Output the tuple (v, m).

4.6 Recover

This algorithm allows a recipient (or designated entity) to compute the serial number, tag, value, and memo from a coin destined for its public address. It requires the full view key corresponding to the public address to do so. If the coin is not destined for the public address, the algorithm returns failure.

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

Outputs: Coin serial number s, tag T, value v, memo m

1. Parse the required full view key components as $\text{addr}_{\text{full}} = (s_1, s_2)$ and public address $\text{addr}_{\text{pk}} = (Q_1, Q_2)$.
2. Parse the serial number commitment S, value commitment C, recovery key K, encrypted value $\overline{v}$ (if the coin is of type $b = 0$) or value v (if the coin is of type $b = 1$), and encrypted memo $\overline{m}$ from Coin.

3. If $\mathrm{Com}(\mathcal{H}_{\mathrm{ser}}(s_1 K, Q_1, Q_2), 0) + Q_2 \neq S$, return failure.
4. Generate a symmetric encryption key $k_{\mathrm{sym}} = \mathrm{SymKeyGen}(s_1 K)$; decrypt the value $v = \mathrm{SymDec}(k_{\mathrm{sym}}, \overline{v})$ (if $b = 1$) and memo $m = \mathrm{SymDec}(k_{\mathrm{sym}}, \overline{m})$.
5. If $\mathrm{Com}(v, \mathcal{H}_{\mathrm{val}}(s_1 K)) \neq C$, return failure.
6. Compute the serial number $s = \mathcal{H}_{\mathrm{ser}}(s_1 K, Q_1, Q_2) + s_2$ and tag $T = (1/s)H$.
7. Output the tuple (s, T, v, m).

4.7 Spend

This algorithm allows a recipient to generate a transaction that consumes coins destined to its public address, 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 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 $\mathrm{addr}_{\mathrm{full}}$
- A spend key $\mathrm{addr}_{\mathrm{sk}}$
- A set of N input coins InCoins as part of a cover set
- For each $u \in [0, w)$ coin to spend, the index in InCoins, serial number, tag, value, and recovery key: $(l_u, s_u, T_u, v_u, K_u)$
- An integer fee value $f \in [0, v_{\max})$
- A set of t output coin public addresses, values, and memos:

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

Outputs: Spend transaction $\mathrm{tx}_{\mathrm{spend}}$

1. Parse the required full view key component as $\mathrm{addr}_{\mathrm{full}} = D$.
2. Parse the spend key $\mathrm{addr}_{\mathrm{sk}} = (s_1, s_2, r)$.
3. Parse the cover set serial number commitments and value commitments as $\mathrm{InCoins} = \{(S_i, C_i)\}_{i=0}^{N-1}$.
4. For each $u \in [0, w)$:
   (a) Compute the serial number commitment offset:
   $$S'_u = \mathrm{Com}(s_u, -\mathcal{H}_{\mathrm{ser'}}(s_u, D)) + D$$
   (b) Compute the value commitment offset:
   $$C'_u = \mathrm{Com}(v_u, \mathcal{H}_{\mathrm{val'}}(s_u, D))$$
   (c) Generate a parallel one-out-of-many proof:
   $$(\Pi_{\mathrm{par}})_u = \mathrm{ParProve}(pp_{\mathrm{par}}, \{S_i - S'_u, C_i - C'_u\}_{i=0}^{N-1}; (l_u, \mathcal{H}_{\mathrm{ser'}}(s_u, D), \mathcal{H}_{\mathrm{val}}(s_1 K_u) - \mathcal{H}_{\mathrm{val'}}(s_u, D)))$$

5. Generate a set $\text{OutCoins} = \{\text{CreateCoin}(\text{addr}_{\text{pk},j}, v_j, m_j, 0)\}_{j=0}^{t-1}$ of output coins.
6. Parse the output coin value commitments as $\text{OutCoins} = \{\overline{C}_j\}_{j=0}^{t-1}$, where each $\overline{C}_j$ contains a recovery key preimage k_j and destination address component $(Q_1)_j$.
7. Generate a representation proof for balance assertion:

$$\Pi_{\text{bal}} = \text{RepProve}\left(pp_{\text{rep}}, F, \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \text{Com}(f, 0); \right.$$
$$\left. \sum_{u=0}^{w-1} \mathcal{H}_{\text{val}'}(s_u, D) - \sum_{j=0}^{t-1} \mathcal{H}_{\text{val}}(k_j(Q_1)_j)\right)$$

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

$$(\Pi_{\text{chaum}})_u = \text{ChaumProve}(pp_{\text{chaum}}, S'_u, T_u; (s_u, r - \mathcal{H}_{\text{ser}'}(s_u, D)))$$

10. Output the tuple:

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

Remark 1. We 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.

4.8 Verify

This algorithm assesses the validity of a transaction.

Inputs: either a mint transaction tx_{mint} or a spend transaction tx_{spend}
Outputs: a bit that represents the validity of the transaction
If the input transaction is a mint transaction:

1. Parse the transaction $\text{tx}_{\text{mint}} = (\text{Coin}, \Pi_{\text{bal}})$.
2. Parse the coin value and value commitment as $\text{Coin} = (v, C)$.
3. Check that $v \in [0, v_{\text{max}})$, and output 0 if this fails.
4. Check that $\text{RepVerify}(pp_{\text{rep}}, \Pi_{\text{bal}}, F, C - \text{Com}(v, 0))$, and output 0 if this fails.
5. Output 1.

If the input transaction is a spend transaction:

1. Parse the transaction:
$$\mathrm{tx}_{\mathrm{spend}} = (\mathrm{InCoins}, \mathrm{OutCoins}, f, \{S'_u, C'_u, T_u, (\Pi_{\mathrm{par}})_u, (\Pi_{\mathrm{chaum}})_u\}_{u=0}^{w-1}, \Pi_{\mathrm{bal}})$$
2. Parse the cover set serial number commitments and value commitments as $\mathrm{InCoins} = \{(S_i, C_i)\}_{i=0}^{N-1}$.
3. Parse the output coin value commitments and range proofs as $\mathrm{OutCoins} = \{\overline{C}_j, (\Pi_{\mathrm{rp}})_j\}_{j=0}^{t-1}$.
4. For each $u \in [0, w)$:
   (a) Check that T_u does not appear in any previously-verified transaction, and output 0 if it does.
   (b) Check that $\mathrm{ParVerify}(pp_{\mathrm{par}}, (\Pi_{\mathrm{par}})_u, \{S_i - S'_u, C_i - C'_u\}_{i=0}^{N-1})$, and output 0 if this fails.
   (c) Check that $\mathrm{ChaumVerify}(pp_{\mathrm{chaum}}, (\Pi_{\mathrm{chaum}})_u, S'_u, T_u)$, and output 0 if this fails.
5. For each $j \in [0, t)$:
   (a) Check that $\mathrm{RangeVerify}(pp_{\mathrm{rp}}, (\Pi_{\mathrm{rp}})_j, C)$, and output 0 if this fails.
6. Check that $f \in [0, v_{\max})$, and output 0 if this fails.
7. Check that
$$\mathrm{RepVerify}\left(pp_{\mathrm{rep}}, \Pi_{\mathrm{bal}}, F, \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \mathrm{Com}(f, 0)\right)$$
   and output 0 if this fails.
8. Output 1.

5 Multisignature Operations

Spark addresses and transactions support efficient and secure multisignature operations, where a group of signers are required to authorize transactions. We describe a method for signing groups to perform the CreateAddress and Spend algorithms to produce multisignature addresses and spend transactions indistinguishable from others.

Throughout this section, suppose we have a group of ν signers who wish to collaboratively produce an address or transaction. Further, sample a cryptographic hash function $\mathcal{H}_{\mathrm{agg}} : \{0,1\}^* \to \mathbb{F}$ uniformly at random.

5.1 CreateAddress

1. Each player $\alpha \in [0, \nu)$ chooses $s_{1,\alpha}, s_{2,\alpha}, r_\alpha \in \mathbb{F}$ uniformly at random, and sets $D_\alpha = \mathrm{Com}(0, r_\alpha)$. It sends the the values $s_{1,\alpha}, s_{2,\alpha}, D_\alpha$ to all players.

2. All players compute the aggregate incoming view key and full view key components:

$$s_1 = \sum_{\alpha=0}^{\nu-1} \mathcal{H}_{\text{agg}} \left(\{s_{1,\beta}\}_{\beta=0}^{\nu-1}, \alpha \right) s_{1,\alpha}$$
$$s_2 = \sum_{\alpha=0}^{\nu-1} \mathcal{H}_{\text{agg}} \left(\{s_{2,\beta}\}_{\beta=0}^{\nu-1}, \alpha \right) s_{2,\alpha}$$
$$D = \sum_{\alpha=0}^{\nu-1} \mathcal{H}_{\text{agg}} \left(\{D_{\beta}\}_{\beta=0}^{\nu-1}, \alpha \right) D_{\alpha}$$

3. All players compute the aggregate public address components:

$$Q_1 = \text{Com}(s_1, 0)$$
$$Q_2 = \text{Com}(s_2, 0) + D$$

Note that each player α keeps its spend key share r_α private.

5.2 Spend

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

$$pp_{\text{chaum}}, S'_u, T_u; (s_u, r - \mathcal{H}_{\text{ser}'}(s_u, D))$$

1. Each player $\alpha \in [0, \nu)$ chooses $\overline{r}_\alpha, \overline{s}_\alpha \in \mathbb{F}$ uniformly at random. It generates a commitment to the tuple $(\overline{r}_\alpha, \overline{s}_\alpha F)$ and sends it to all players.
2. Each player reveals its commitment opening to all players, verifies all players' openings, and aborts if any are invalid.
3. All players compute the initial proof terms:

$$A_1 = \left(\sum_{\beta=0}^{\nu-1} \overline{r}_\beta \right) G$$
$$A_2 = \left(\sum_{\beta=0}^{\nu-1} \overline{r}_\beta \right) T_u$$
$$A_3 = \sum_{\beta=0}^{\nu-1} (\overline{s}_\beta F)$$

They compute the challenge c from the initial proof transcript.

4. Each player $\alpha \in [0, \nu)$ computes the following:

$$t_1 = \sum_{\beta=0}^{\nu-1} \overline{r}_\beta + cs_u$$
$$t_{2,\alpha} = \overline{s}_\alpha + c\mathcal{H}_{\text{agg}}\left(\{D_\beta\}_{\beta=0}^{\nu-1}, \alpha\right) r_\alpha$$

   It sends $t_{2,\alpha}$ to all players.
5. All players compute the final proof term:

$$t_2 = \sum_{\beta=0}^{\nu-1} t_{2,\beta} - c\mathcal{H}_{\text{ser}'}(s_u, D)$$

6 Applications of Key Structures

The key structure in Spark permits flexible and useful functionality relating to transaction scanning and generation.

The incoming view key is used in Identify operations to determine when a coin is directed to the 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 the 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 a public 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 from its address.

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 proofs.

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, with a 16-byte authentication tag; this is the case for the ChaCha20-Poly1305 authenticated symmetric encryption construction, for example [15]. 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		2	
Input data (w coins)				
(S', C')		$2w$		
Π_{par}	this paper	$(2m+4)w$	$[m(n-1)+4]w$	
Π_{chaum}	this paper	$3w$	$2w$	
Output data (t coins)				
(S, K, C)		$3t$		
$(\overline{v}, \overline{m})$	ChaCha20-Poly1305			$(8+M+16)t$

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 [18] 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.

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 implementation) to combine these elements with a sufficient weight prior to applying

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	$B[w(2m+6)+2m^m]+m^n+1$
Bulletproofs+	$B(t+2\lg(64t)+3)+128T+2$
Schnorr	$B(5w)+3$
Modified Chaum-Pedersen	$B(w+t+1)+2$

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.

References

1. 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
2. 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)
3. 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)
4. 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
5. 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
6. 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
7. 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)

8. 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)
9. 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
10. 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)
11. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification (2021), https://github.com/zcash/zips/blob/master/protocol/protocol.pdf
12. Jivanyan, A.: Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Report 2019/373 (2019), https://ia.cr/2019/373
13. 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, https://doi.org/10.1145/3319535.3345655
14. 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)
15. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF protocols. RFC 7539, RFC Editor (May 2015), http://www.rfc-editor.org/rfc/rfc7539.txt, http://www.rfc-editor.org/rfc/rfc7539.txt
16. 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)
17. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
18. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
19. Pyrros Chaidos, V.G.: Lelantus-CLA. Cryptology ePrint Archive, Report 2021/1036 (2021), https://ia.cr/2021/1036

A Modified Chaum-Pedersen Proving System

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

$$\{pp_{\text{chaum}}, Y, Z \in \mathbb{G}; (x, y) \in \mathbb{F} : Y = xG + yF, H = xZ\}$$

The protocol proceeds as follows:

1. The prover selects random $r, s \in \mathbb{F}$. It computes

$$(A_1, A_2, A_3) := (rG, rZ, sF)$$

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 $t_1 := r + cx$ and $t_2 := s + cy$, and sends these values to the verifier.
4. The verifier accepts the proof if and only if

$$A_1 + A_3 + cY = t_1 G + t_2 F$$

and

$$A_2 + cH = t_1 Z.$$

We now prove that the protocol is complete, special sound if G and F are independent, and special honest-verifier zero knowledge if G and H are independent.

Proof. Completeness of this protocol follows trivially by inspection.

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 $c \in \mathbb{F}$, random values $t_1, t_2 \in \mathbb{F}$, and a random value $A_1 \in \mathbb{G}$. It sets $A_2 := t_1 Z - cH$ and $A_3 := t_1 G + t_2 F - cY - A_1$. 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 are independent, as are transcript elements produced by the simulator.

To show the protocol is special sound, consider two accepting transcripts with distinct challenge values $c \neq c' \in \mathbb{F}$:

$$(A_1, A_2, A_3, c, t_1, t_2)$$

and

$$(A_1, A_2, A_3, c', t_1', t_2')$$

The first verification equation applied to the two transcripts implies that $(c - c')Y = (t_1 - t_1')G + (t_2 - t_2')F$, so we extract the witness values $x := (t_1 - t_1')/(c - c')$ and $y := (t_2 - t_2')/(c - c')$, or a nontrivial discrete logarithm relation between G and F (a contradiction if these generators are independent). Similarly, the second verification equation implies that $(c - c')H = (t_1 - t_1')Z$, yielding the same value for x as required.

This completes the proof.

Remark 2. Note that extraction of a valid witness to show special soundness (as opposed to a nontrivial discrete logarithm relation between generators) requires that G and F be independent.

Remark 3. Note that transcripts produced by the simulator are indistinguishable from those of real proofs only if G and H are independent.

B Parallel One-out-of-Many Proving System

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

$$\begin{aligned}\{pp_{\text{par}}, \{S_i, V_i\}_{i=0}^{N-1} \subset \mathbb{G}^2; l \in \mathbb{N}, (s, v) \in \mathbb{F} :& \\ 0 \leq l < N, S_l = \text{Com}(0, s), V_l = \text{Com}(0, v)\}&\end{aligned}$$

The protocol is shown in Figure 1, where we use the notation of [12].

This protocol is complete, special sound, and special honest-verifier zero knowledge; the proof is essentially the same as in the original construction, with only minor straightforward modifications.

C Payment System Security

Zerocash [1] 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 [1].

We recall the security definition for a DAP scheme

$$\Pi = (\text{Setup}, \text{CreateAddress}, \text{Mint}, \text{Spend}, \text{Recover}, \text{Verify}),$$

which is secure if it satisfies definitions for ledger indistinguishability, transaction non-malleability, and balance security properties, which we define below.

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 $\mathcal{O}^{DAP}$. The oracle $\mathcal{O}^{DAP}$ maintains a ledger L of transactions and provides an interface for executing CreateAddress, Mint, and Spend algorithm operations for honest parties. To simulate behavior from honest parties, $\mathcal{A}$ passes a query to $\mathcal{C}$, which makes sanity checks and then proxies the queries to $\mathcal{O}^{DAP}$, returning the responses to $\mathcal{A}$ as needed. For CreateAddress queries, $\mathcal{C}$ runs the CreateAddress protocol algorithm and returns the public address addr_{pk} to $\mathcal{A}$. For Mint queries, the adversary specifies the value and destination public address for the transaction, and the resulting transaction is produced and returned by $\mathcal{C}$ if valid. For Spend queries, the adversary specifies the input coins to be consumed, as well as the values and destination public addresses for the transaction, and the resulting transaction is produced (after $\mathcal{C}$ recovers the consumed coins) and returned by $\mathcal{C}$ if valid. The oracle $\mathcal{O}^{DAP}$ also provides an Insert query that allows the adversary to insert arbitrary and potentially malicious tx_{mint} or tx_{spend} transactions to the ledger L, provided they are valid.

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

ParProve $\left(pp_{\text{par}}, \{S_i, V_i\}_{i=0}^{N-1}; (l, s, v)\right)$

ParVerify $\left(pp_{\text{par}}, \{S_i, V_i\}_{i=0}^{N-1}\right)$

Compute:

$r_A, r_B, r_C, r_D, \{a_{j,i}\}_{j=0,i=1}^{m-1,n-1} \leftarrow_R \mathbb{F}$

$\forall j \in [0, m)$

$\quad a_{j,0} = -\sum_{i=1}^{n-1} a_{j,i}$

$A \equiv \text{Com}(\{a_{j,i}\}_{j,i=0}^{m-1,n-1}, r_A)$

$B \equiv \text{Com}(\{\sigma_{l_j,i}\}_{j,i=0}^{m-1,n-1}, r_B)$

$C \equiv$

$\quad \text{Com}(\{a_{j,i}(1 - 2\sigma_{l_j,i})\}_{j,i=0}^{m-1,n-1}, r_C)$

$D \equiv \text{Com}(\{-a_{j,i}^2\}_{j,i=0}^{m-1,n-1}, r_D)$

$\forall j \in [0, m)$

$\quad \rho_j^S, \rho_j^V \leftarrow_R \mathbb{F}$

$\quad G_j^V \equiv \sum_{i=0}^{N-1} p_{i,j} V_i + \text{Com}(0, \rho_j^V)$

$\quad G_j^S \equiv \sum_{i=0}^{N-1} p_{i,j} S_i + \text{Com}(0, \rho_j^S)$

$\quad\quad$ (computing $p_{i,j}$

$\quad\quad$ as in the orig. paper)

$A, B, C, D,$

$\{G_j^S, G_j^V\}_{j=0}^{m-1}$

$$\xrightarrow{\hspace{4em}} A, B, C, D, \{G_j^S, G_j^V\}_{j=0}^{m-1} \in \mathbb{G}$$

$$\xleftarrow{x \leftarrow \{0,1\}^\lambda}$$

Accept if and only if:

$\forall j \in [0, m), i \in [1, n)$

$\quad f_{j,i} \equiv \sigma_{l_j i} x + a_{j,i}$

$z_A = r_B x + r_A$

$z_C = r_C x + r_D$

$z_S = sx^m - \sum_{j=0}^{m-1} \rho_j^S x^j$

$z_V = vx^m - \sum_{j=0}^{m-1} \rho_h^V x^j$

$\{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}$

z_A, z_C, z_S, z_V

$$\xrightarrow{\hspace{4em}}$$

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

$z_A, z_C, z_V, z_R \in \mathbb{F}$

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

$D + xC =$

$\text{Com}(\{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}; z_C)$

$A + xB =$

$\text{Com}(\{f_{j,i}\}_{j,i=0}^{m-1,n-1}; z_A)$

$\sum_{i=0}^{N-1} \overline{f}_i S_i - \sum_{j=0}^{m-1} x^j G_j^S$

$\quad = \text{Com}(0, z_S)$

$\sum_{i=0}^{N-1} \overline{f}_i V_i - \sum_{j=0}^{m-1} x^j G_j^V$

$\quad = \text{Com}(0, z_V)$

where $\overline{f}_i \equiv \prod_{j=0}^{m-1} f_{j,i_j}$

Fig. 1. Parallel one-out-of-many protocol

Remark 4. We also require the DAP scheme to be complete, which implies that any unspent coin on the ledger can be spent. This property means that if the coin appears on the ledger L as an output of a transaction, a user in possession of the corresponding secret data can generate a valid Spend transaction consuming it. This property immediately follows from the completeness properties of the underlying cryptographic constructions.

C.1 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 AdvCoins. Letting ADDR be set of all addresses of honest users generated by CreateAddress queries, $\mathcal{A}$ wins the game if

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

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

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

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 $\text{tx}_{\text{spend},k}$ in $(L, \vec{a})$ consumes distinct coin serial/value commitment pairs, and each consumed coin is the output of a valid $\text{tx}_{\text{mint},i}$ or $\text{tx}_{\text{spend},j}$ 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 $(\text{tx}_{\text{spend}}, a)$ 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 $(\text{tx}_{\text{spend}}, a)$ 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 $(\text{tx}_{\text{mint}}, a)$ in $(L, \vec{a})$ with public value v that generates a coin with value commitment C, a contains a value v' corresponding to the commitment such that $v = v'$. This implies that values cannot be created arbitrarily.
5. For each tx_{spend} in $(L, \vec{a})$ inserted by $\mathcal{A}$ through an Insert query, each consumed coin in tx_{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
$$v_{\text{mint}} + v_{\text{ADDR} \to \mathcal{A}} \leq v_{\text{unspent}} + v_{\mathcal{A} \to \text{ADDR}}$$
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 number commitment S. Validity of the corresponding parallel one-out-of-many proofs $(\Pi_{\text{par}})_{u_1}$ and $(\Pi_{\text{par}})_{u_2}$ yields extractions x_{u_1} and x_{u_2} such that $S - S'_{u_1} = x_{u_1} F$ and $S - S'_{u_2} = x_{u_2} F$ for the serial number commitment offsets S'_{u_1} and S'_{u_2}, respectively. Validity of the modified Chaum-Pedersen proofs $(\Pi_{\text{chaum}})_{u_1}$ and $(\Pi_{\text{chaum}})_{u_2}$ give extracted openings $S'_{u_1} = s_{u_1} G + r_{u_1} F$ and

$S'_{u_2} = s_{u_2}G + r_{u_2}F$ and tag representations $T_{u_1} = (1/s_{u_1})H$ and $T_{u_2} = (1/s_{u_2})H$. This implies

$$S = s_{u_1}G + (r_{u_1} + x_{u_1})F = s_{u_2}G + (r_{u_2} + x_{u_2})F$$

represents two openings of S. However, transaction validity means $T_{u_1} \neq T_{u_2}$, and injectivity of the tag construction asserts $s_{u_1} \neq s_{u_2}$. Hence we have distinct openings of S, which contradicts the binding property of the underlying commitment scheme.

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. Validity of the modified Chaum-Pedersen proof for such an input gives a tag representation $T = (1/s)H$ and serial number commitment offset $S' = sG + rF$. Validity of the parallel one-out-of-many proof for the input gives an index l such that $S_l - S' = xF$, meaning $S_l = sG + (r+x)F$ is an opening of this commitment. Because transaction validity requires all input ambiguity set elements to be produced in previous valid transactions as valid commitments, the adversary knows an opening of such a commitment, which is a contradiction.

$\mathcal{A}$ 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 $C = vG + aF$ from either the balance 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 $C - C' = xF$, where C' is the input's value commitment offset. But this immediately gives $C' = vG + (a-x)F$, 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 balance proof Π_{bal} 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 can be recovered by a public address $(Q_1, Q_2) \in \text{ADDR}$. Let (s_1, s_2, r) be the secret key corresponding to this address. Since tx_{spend} can be recovered by this honest address, the serial number commitment for the consumed coin is of the form

$$\begin{aligned}S =&\mathcal{H}_{\text{ser}}(s_1K, Q_1, Q_2)G + Q_2\\ =&(\mathcal{H}_{\text{ser}}(s_1K, Q_1, Q_2) + s_2)G + rF\end{aligned}$$

for recovery key K. Validity of the parallel one-out-of-many proof corresponding to S gives an extraction x such that $S - S' = xF$, where S' is the serial number commitment offset. Hence

$$S' = (\mathcal{H}_{\text{ser}}(s_1K, Q_1, Q_2) + s_2)G + (r - x)F$$

and the corresponding tag (via validity of the modified Chaum-Pedersen proof) is

$$T = 1/(\mathcal{H}_{\text{ser}}(s_1K, Q_1, Q_2) + s_2)H.$$

Since we can model $\mathcal{H}_{\text{ser}}$ as a random oracle, $\mathcal{A}$ has produced an opening to S' and a discrete logarithm to T, but did not produce the secret key (s_1, s_2, r) corresponding to the public address (Q_1, Q_2), a contradiction.

This completes the proof.

C.2 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 **TR-NM**, in which a bounded adversary $\mathcal{A}$ adaptively interacts with the oracle $\mathcal{O}^{\text{DAP}}$, and then outputs a spend transaction tx'. If we let T denote the set of all transactions produced by Spend queries to $\mathcal{O}^{\text{DAP}}$, and L denote the final ledger, $\mathcal{A}$ wins the game if there exists $\text{tx} \in T$ such that:

- $\text{tx}' \neq \text{tx}$;
- 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 **TR-NM**-secure if the adversary $\mathcal{A}$ wins the game **TR-NM** only with negligible probability:

$$\Pr[\textbf{TR-NM}(\Pi, \mathcal{A}, \lambda) = 1] \leq \text{negl}(\lambda)$$

Let $\mathcal{T}$ be the set of all tx_{spend} transactions generated by the $\mathcal{O}^{DAP}$ 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 modified Chaum-Pedersen proofs into each such proof via $\mathcal{H}_{\text{bind}}$ and the proof transcripts. Therefore, in order to produce valid $\text{tx}' \neq \text{tx}$, 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 $\mathcal{H}_{\text{bind}}$ 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, validity of the modified Chaum-Pedersen proof Π'_{chaum} contained in tx' leads to an extraction of an opening of the serial number commitment offset $S' = sG + rF$. Further, validity of the parallel one-out-of-many proof Π'_{par} contained in the transaction yields an index l and discrete logarithm extraction such that $S_l = sG + (r + x)F$, where S_l is contained in InCoins. However, S_l was generated in a transaction in L' preceding tx, and contains either a proof of representation (if generated in a Mint transaction) or range proof (if generated in a Spend transaction) that asserts its representation with respect to the generators G, F is unique up to the binding property of the commitment scheme.

However, the adversary does not know the opening of S_l (since it was generated by the oracle), and the modified Chaum-Pedersen proof Π_{chaum} from tx authorizing the spend of S_l is special honest-verifier zero knowledge, the adversary can only produce its valid proof Π_{chaum} with negligible probability, a contradiction.

C.3 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 **L-IND** 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 $\text{Setup}(1^\lambda) \to pp$ and sends the parameters to $\mathcal{A}$; next it samples a random bit $b \in \{0, 1\}$ and initializes two separate DAP oracles $\mathcal{O}_0^{DAP}$ and $\mathcal{O}_1^{DAP}$, 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 $\mathcal{O}_b^{DAP}$ and $\mathcal{O}_{1-b}^{DAP}$ respectively.

2. $\mathcal{A}$ sends to $\mathcal{C}$ two queries Q, Q' of the same type (one of CreateAddress, Mint, Spend, Recover, 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, Spend, or Recover, $\mathcal{C}$ first checks if the two queries Q and Q' are publicly consistent, and then forwards Q to $\mathcal{O}_0^{DAP}$ and Q' to $\mathcal{O}_1^{DAP}$. 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 weather 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 $\mathcal{C}$ 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 Recover, they are publicly consistent by construction. 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, the minted values of both queries must be equal. 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 **L-IND**-secure if $\mathcal{A}$ wins the game **L-IND** only probability at most negligibly better than chance:

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

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

The simulation experiment $\mathcal{D}^{\text{sim}}$: 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 $\text{Setup}(1^\lambda) \to pp$ and a random bit b, and initializes DAP oracles $\mathcal{O}_0^{\text{DAP}}$ and $\mathcal{O}_1^{\text{DAP}}$. 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 Recover and Insert queries: The challenger proceeds as in the original **L-IND** experiment.
- Answering CreateAddress queries: In this case the challenger replaces the public address components Q_1 and Q_2 with random strings of the appropriate lengths, producing addr_{pk} that is returned to $\mathcal{A}$.
- Answering Mint queries: The challenger does the following to answer Q and Q' separately:
  1. If $\mathcal{A}$ provided a public address addr_{pk} not generated by the challenger, it produces a coin using CreateCoin as usual.
  2. Otherwise, it simulates coin generation:
     (a) Samples a recovery key K uniformly at random.
     (b) Samples a serial number commitment S uniformly at random.
     (c) Samples a value commitment C uniformly at random.
     (d) Samples a random input used to produce a symmetric encryption key $\text{SymKeyGen} \to k_{\text{enc}}$.
     (e) Simulates the memo encryption by selecting random $\widetilde{m}$ of the proper length, and encrypting it to produce $\text{SymEnc}(k_{\text{enc}}, \widetilde{m}) \to \overline{m}$.
  3. Simulates the balance proof Π_{bal} on the statement $(C - \text{Com}(v, 0)$.
  4. 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. Parse the input cover set serial number commitments and value commitments as $\text{InCoins} = \{(S_i, C_i)\}_{i=0}^{N-1}$.
  2. For each $u \in [0, w)$, where l_u represents the index of the consumed coin in InCoins:
     (a) Samples a tag T_u uniformly at random.
     (b) Samples a serial number commitment offset S'_u and value commitment offset C'_u uniformly at random.
     (c) Simulates a parallel one-out-of-many proof $(\Pi_{\text{par}})_u$ on the statement $(\{S_i - S'_u, C_i - C'_u\}_{i=0}^{N-1})$.
  3. For each $j \in [0, t)$:
     (a) If $\mathcal{A}$ provided a public address addr_{pk} not generated by the challenger, it produces a coin using 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 a symmetric encryption key SymKeyGen $\rightarrow k_{\text{enc}}$.
v. Simulates the value encryption by selecting random $\widetilde{v}$ of the proper length, and encrypting it to produce $\text{SymEnc}(k_{\text{enc}}, \widetilde{v}) \rightarrow \overline{v}_j$.
vi. Simulates the memo encryption by selecting random $\widetilde{m}$ of the proper length, and encrypting it to produce $\text{SymEnc}(k_{\text{enc}}, \widetilde{m}) \rightarrow \overline{m}_j$.
vii. Simulates a range proof $(\mathit{\Pi}_{\text{rp}})_j$ on the statement $(\overline{C}_j)$.

4. Simulates the balance proof $\mathit{\Pi}_{\text{bal}}$ on the statement

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

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

For experiments defined below, we define $\text{Adv}^{\mathcal{D}}$ as the advantage of $\mathcal{A}$ in some experiment $\mathcal{D}$ over the original **L-IND** game. By definition, all answers sent to $\mathcal{A}$ in $\mathcal{D}^{\text{sim}}$ are computed independently of the bit b, so $\text{Adv}^{\mathcal{D}^{\text{sim}}} = 0$. We will prove that $\mathcal{A}$'s advantage in the real L-IND 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 proofs. 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 $\text{Adv}^{\mathcal{D}_1} = 0$.

Experiment $\mathcal{D}_2$: This experiment modifies $\mathcal{D}_1$ by replacing all encrypted values and memos 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 $|\text{Adv}^{\mathcal{D}_2} - \text{Adv}^{\mathcal{D}_1}|$ is negligible.

Experiment $\mathcal{D}^{\textbf{sim}}$: The $\mathcal{D}^{\text{sim}}$ 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 $\mathcal{D}^{\text{real}}$), tags are generated using a pseudorandom function [7], 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 $\mathcal{D}^{\text{sim}}$ is negligibly different from its advantage in the $\mathcal{D}_2$ experiment. Hence $|\text{Adv}^{\mathcal{D}^{\text{sim}}} - \text{Adv}^{\mathcal{D}_2}|$ is negligible.

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