Lelantus Spark: Secure and Flexible Private Transactions

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

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

August 23, 2021

This preprint represents work in progress, and is unfinished. It may contain errors, and has not yet undergone external formal review. The forthcoming full version of this preprint contains protocol security proofs. The authors welcome corrections, comments, suggestions, and other feedback.

Abstract

We propose a modified Lelantus construction to provide recipient privacy and additional functionality. Compared to the original Lelantus design, our construction 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 construction is compatible with straightforward linear multisignature operations to allow mutually non-trusting parties to cooperatively receive and generate transactions associated to a multisignature address.

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 digital asset projects 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 due to the space and time scaling of its underlying signature scheme [15, 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], though the protocol in Beam adds a one-of-many ambiguity layer as a mitigation. And the Lelantus protocol used in Firo supports only simple mint-and-burn operations that interact with transparent Bitcoin-style outputs in ways that can leak information [12].

*Corresponding author: aram@firo.org

Here we introduce Spark, an iteration on the Lelantus protocol enabling trustless private transactions. 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] (and used in other modified forms in Lelantus and Triptych [14]) 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 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.

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

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

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

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		Accept if and only if
$r_A, r_B, r_C, r_D, a_{j,1}, \ldots, a_{j,n-1} \leftarrow_R \mathbb{F}$		
$\forall j \in [0, m)$		
$\quad a_{j,0} = -\sum_{i=1}^{n-1} a_{j,i}$		
$B := \text{Com}(\sigma_{l_0,0}, \ldots, \sigma_{l_{m-1},n-1}; r_B)$		
$A := \text{Com}(a_{0,0}, \ldots, a_{m-1,n-1}; r_A)$		
$C := \text{Com}(\{a_{j,i}(1 - 2\sigma_{l_j,i})\}_{j,i=0}^{m-1,n-1}; r_C)$		
$D := \text{Com}(-a_{0,0}^2, \ldots, -a_{m-1,n-1}^2; r_D)$		
$\forall k \in [0, m)$	$A, B, C, D,$ $\{G_K^S, G_K^V\}_{k=0}^{m-1}$ $\longrightarrow$	
$\quad \rho_k^S, \rho_k^V \leftarrow_R \mathbb{F}$		
$\quad G_k^S = \sum_{i=0}^{N-1} p_{i,k} S_i + \text{Com}(0, \rho_k^S)$		
$\quad\quad$ computing $p_{i,k}$ as in the orig. paper		The values
$\quad G_k^V = \sum_{i=0}^{N-1} p_{i,k} V_i + \text{Com}(0, \rho_k^V)$		$A, B, C, D, G_0^S, G_0^V \ldots, G_{m-1}^S, G_{m-1}^V \in \mathbb{G}$
	$\xleftarrow{x \leftarrow \{0,1\}^\lambda}$	$\{f_{j,i}\}_{j,i=0,1}^{m-1,n-1}, z_A, z_C, z_V, z_R \in \mathbb{F}$
$\forall j \in [0, m), i \in [1, n)$		
$\quad f_{j,i} = \sigma_{l_j i} x + a_{j,i}$		$\forall j : f_{j,0} = x - \sum_{i=1}^{n-1} f_{j,i}$
$z_A = r_B \cdot x + r_A$		$B^x A = \text{Com}(f_{0,0}, \ldots, \ldots f_{m-1,n-1}; z_A)$
$z_C = r_C \cdot x + r_D$		$C^x D = \text{Com}(\{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}; z_C)$
$z_S = s \cdot x^m - \sum_{k=0}^{m-1} \rho_k^S \cdot x^k$	$f_{0,1}, \ldots f_{m-1,n-1}$	
$z_V = v \cdot x^m - \sum_{k=0}^{m-1} \rho_k^V \cdot x^k$	z_A, z_C, z_S, z_V $\longrightarrow$	$\sum_{i=0}^{N-1} \left(\prod_{j=0}^{m-1} f_{j,i_j}\right) S_i - \sum_{k=0}^{m-1} x^k G_k^S$ $= \text{Com}(0, z_S)$
		$\sum_{i=0}^{N-1} \left(\prod_{j=0}^{m-1} f_{j,i_j}\right) V_i + \sum_{k=0}^{m-1} x^k G_k^V$ $= \text{Com}(0, z_V)$

Figure 1: Parallel one-of-many protocol

2.4 Parallel one-of-many proving system

We require the use of a parallel one-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 (ParProve, ParVerify) for the following relation, where we let $N = n^m$:

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

The protocol is shown in Figure 1.

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.

2.5 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 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.

Aside from the usual correctness definition, we require that the construction maintain indistinguishability under adaptive chosen ciphertext attack (IND-CCA2).

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

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

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, addr_{del} is a proof delegation key used to produce one-of-many proofs, 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.
- 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.

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

- *Mint* transactions. A *Mint* transaction generates new coins 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 tx_{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 tx_{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 a tuple of polynomial-time algorithms:

(**Setup, CreateAddress, CreateCoin, Mint, Identify, Recover, Spend, Verify**)

- **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, proof delegation 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 tx_{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 proof delegation 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, and a set of output coins to be generated, and outputs a spend transaction tx_{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[1] using a Zerocash-type security model [1].

4 Algorithm Constructions

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

[1]These proofs appear in the forthcoming full version of this preprint.

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-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 $\mathcal{H}_{\text{ser}}, \mathcal{H}_{\text{val}}, \mathcal{H}_{\text{ser}'}, \mathcal{H}_{\text{val}'} : \{0,1\}^* \to \mathbb{F}$ 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-of-many proving system.
8. Compute the public parameters pp_{sym} of a 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 $(\text{addr}_{\text{pk}}, \text{addr}_{\text{in}}, \text{addr}_{\text{full}}, \text{addr}_{\text{del}}, \text{addr}_{sk})$

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 $\text{addr}_{\text{full}} = (s_1, s_2)$.
6. Set $\text{addr}_{\text{del}} = D$.
7. Set $\text{addr}_{\text{sk}} = (s_1, s_2, r)$.
8. Output the tuple $(\text{addr}_{\text{pk}}, \text{addr}_{\text{in}}, \text{addr}_{\text{full}}, \text{addr}_{\text{del}}, \text{addr}_{sk})$.

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 $v \in [0, v_{max})$, memo m

Outputs: Coin public key S, recovery key K, value commitment C, value commitment range proof Π_{rp}, encrypted value $\overline{v}$, 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 $S = \text{Com}(\mathcal{H}_{\text{ser}}(kQ_1), 0) + Q_2$.
4. Compute the recovery key $K = \text{Com}(k, 0)$.
5. Generate the value commitment $C = \text{Com}(v, \mathcal{H}_{\text{val}}(kQ_1))$ and range proof
$$\Pi_{\text{rp}} = \text{RangeProve}(pp_{\text{rp}}, C; (v, \mathcal{H}_{\text{val}}(kQ_1))).$$
6. Generate a symmetric encryption key $k_{\text{sym}} = \text{SymKeyGen}(kQ_1)$; encrypt the value $\overline{v} = \text{SymEnc}(k_{\text{sym}}, v)$ and memo $\overline{m} = \text{SymEnc}(k_{\text{sym}}, m)$.
7. Output the tuple $(S, K, C, \Pi_{\text{rp}}, \overline{v}, \overline{m})$.

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 $\text{CreateCoin}(\text{addr}_{\text{pk}}, v, m) \rightarrow \text{Coin} = (S, K, C, \Pi_{\text{rp}}, \overline{v}, \overline{m})$.
2. Generate a value representation proof on the value commitment:
$$\Pi_{\text{bal}} = \text{RepProve}(pp_{\text{rep}}, F, C - \text{Com}(v, 0); \mathcal{H}_{\text{val}}(kQ_1))$$
3. Output the tuple $\text{tx}_{\text{mint}} = (\text{Coin}, v, \Pi_{\text{bal}})$.

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 $\overline{v}$, and encrypted memo $\overline{m}$ from Coin.
3. If $\text{Com}(\mathcal{H}_{\text{ser}}(s_1 K), 0) + Q_2 \neq S$, return failure.
4. Generate a symmetric encryption key $k_{\text{sym}} = \text{SymKeyGen}(s_1 K)$; decrypt the value $v = \text{SymDec}(k_{\text{sym}}, \overline{v})$ 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 full view key $\text{addr}_{\text{full}} = (s_1, s_2)$ 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 $\overline{v}$, and encrypted memo $\overline{m}$ from Coin.
3. If $\text{Com}(\mathcal{H}_{\text{ser}}(s_1 K), 0) + Q_2 \neq S$, return failure.
4. Generate a symmetric encryption key $k_{\text{sym}} = \text{SymKeyGen}(s_1 K)$; decrypt the value $v = \text{SymDec}(k_{\text{sym}}, \overline{v})$ 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. Compute the serial number $s = \mathcal{H}(s_1 K) + s_2$ 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 proof delegation key is required to generate the parallel one-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, and that it has run CreateCoin to produce all coins that it wishes to generate in the transaction.

Inputs:

- Security parameter λ and public parameters pp
- A proof delegation key addr_{del}
- A spend key addr_{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: $(l_u, s_u, T_u, v_u, K_u)$
- A set of t output coins OutCoins

Outputs: Spend transaction tx_{spend}

1. Parse the proof delegation key $\text{addr}_{\text{del}} = D$.
2. Parse the spend key $\text{addr}_{\text{sk}} = (s_1, s_2, r)$.
3. Parse the cover set serial number commitments and value commitments as $\text{InCoins} = \{(S_i, C_i)\}_{i=0}^{N-1}$.
4. For each $u \in [0, w)$:
   (a) Compute the serial number commitment offset $S'_u = \text{Com}(s_u, -\mathcal{H}_{\text{ser}'}(s_u, D)) + D$.
   (b) Compute the value commitment offset $C'_u = \text{Com}(v_u, \mathcal{H}_{\text{val}'}(s_u, D))$.
   (c) Generate a parallel one-of-many proof:
   $$(\Pi_{\text{par}})_u = \text{ParProve}(pp_{\text{par}}, \{S_i - S'_u, C_i - C'_u\}_{i=0}^{N-1}; (l_u, \mathcal{H}_{\text{ser}'}(s_u, D), \mathcal{H}_{\text{val}}(s_1 K_u) - \mathcal{H}_{\text{val}'}(s_u, D)))$$
   (d) Generate a modified Chaum-Pedersen proof:
   $$(\Pi_{\text{chaum}})_u = \text{ChaumProve}(pp_{\text{chaum}}, S'_u, T_u; (s_u, r - \mathcal{H}_{\text{ser}'}(s_u, D)))$$
5. 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 $(Q_1)_j$.
6. Generate a representation proof for balance assertion:
   $$\Pi_{\text{bal}} = \text{RepProve}\left(pp_{\text{rep}}, F, \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j; \sum_{u=0}^{w-1} \mathcal{H}_{\text{val}'}(s_u, D) - \sum_{j=0}^{t-1} \mathcal{H}_{\text{val}}(k_j (Q_1)_j)\right)$$
7. Output the tuple $\text{tx}_{\text{spend}} = \left(\text{InCoins}, \text{OutCoins}, \{S'_u, C'_u, T_u, (\Pi_{\text{par}})_u, (\Pi_{\text{chaum}})_u\}_{u=0}^{w-1}, \Pi_{\text{bal}}\right)$.

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

If the input transaction is a spend transaction:

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

2. Parse the cover set serial number commitments and value commitments as $\text{InCoins} = \{(S_i, C_i)\}_{i=0}^{N-1}$.
3. Parse the output coin value commitments and range proofs as $\text{OutCoins} = \{\overline{C}_j, (\Pi_{\text{rp}})_j\}_{j=0}^{t-1}$.
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 $\text{ParVerify}(pp_{\text{par}}, (\Pi_{\text{par}})_u, \{S_i - S'_u, C_i - C'_u\}_{i=0}^{N-1})$, and output 0 if this fails.
   (c) Check that $\text{ChaumVerify}(pp_{\text{chaum}}, (\Pi_{\text{chaum}})_u, S'_u, T_u)$, and output 0 if this fails.
5. For each $j \in [0, t)$:
   (a) Check that $\text{RangeVerify}(pp_{\text{rp}}, (\Pi_{\text{rp}})_j, C)$, and output 0 if this fails.
6. Check that $\text{RepVerify}\left(pp_{\text{rep}}, \Pi_{\text{bal}}, F, \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j\right)$, and output 0 if this fails.
7. 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 $\mathcal{H}_{\text{agg}} : \{0,1\}^* \to \mathbb{F}$ 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 $D_\alpha = \text{Com}(0, r_\alpha)$. 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, full view key, and proof delegation 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 and proof delegation key corresponding to the aggregate public address, any player can use these values 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:

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

   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

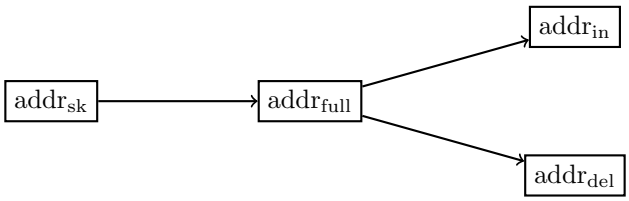

Figure 2: Key derivation structure

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.

The proof delegation key is used in Spend to generate one-of-many proofs. Since the parallel one-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 the delegation 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.

The derivation structure of these keys is shown in Figure 2, where an arrow $A \to B$ indicates that knowledge of A can be used to derive B.

Acknowledgments

The authors thank pseudonymous researcher `koe` for helpful collaboration and discussion during the initial design process for Spark.

References

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

[2] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, page 781–796, USA, 2014. USENIX Association.

[3] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on DDH. In Günther Pernul, Peter Y A Ryan, and Edgar Weippl, editors, *Computer Security – ESORICS 2015*, pages 243–265, Cham, 2015. Springer International Publishing.

[4] Sean Bowe, Ariel Gabizon, and Ian Miers. 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] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[6] Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter proofs for privacy-enhanced distributed ledger. Cryptology ePrint Archive, Report 2020/735, 2020. https://ia.cr/2020/735.

[7] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005*, pages 416–431, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[8] Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 657–689, Cham, 2019. Springer International Publishing.

[9] Brandon Goodell, Sarang Noether, and RandomRun. Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Report 2019/654, 2019. https://ia.cr/2019/654.

[10] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 253–280, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[11] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, 2021. https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

[12] Aram Jivanyan. Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Report 2019/373, 2019. https://ia.cr/2019/373.

[13] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 31–48, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Sarang Noether and Brandon Goodell. Triptych: Logarithmic-sized linkable ring signatures with applications. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomarti, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 337–354, Cham, 2020. Springer International Publishing.

[15] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.