Spats: user-defined confidential assets for the Spark transaction protocol

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

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

March 18, 2022

In privacy-preserving transaction protocols, confidential asset designs permit transfer of quantities of distinct asset types in a way that obscures their types and values. Spark is a protocol that provides flexible privacy properties relating to addressing, transaction sources and recipients, and value transfer; however, it does not natively support the use of multiple confidential asset types. Here we describe Spats, a new design for confidential assets compatible with Spark that focuses on efficient and modular implementation. It does so by extending coin value commitments to bind and mask an asset type, and asserting in zero knowledge that this type is maintained throughout transactions. We describe the cryptographic components and changes to the Spark protocol necessary for the design of Spats.

1 Introduction

Privacy-preserving transaction protocols have seen a wealth of research. Protocols like RingCT [13] in Monero, Sprout and Sapling [1, 3, 10] in Zcash, Mimblewimble [7] in Grin and Beam, and Lelantus [11] in Firo each take different approaches toward privacy. These protocols may provide useful functionality relating to transaction graph ambiguity or obfuscation, addressing, amount confidentiality, and scaling.

Spark [12] is a recent protocol, based on techniques from Lelantus, that provides trustless and flexible confidential transactions. In Spark, coins of hidden and arbitrary value can be transferred in a manner that uses zero-knowledge proofs to hide amounts and provide ambiguity as to the coins consumed in a transaction, while using a non-interactive ephemeral approach to dissociate recipient addresses from coins directed to them.

*Corresponding author: aram@firo.org

However, Spark, like many other privacy-focused protocols, is limited to the transfer of a single asset type. The concept of confidential assets refers to protocol designs that permit the maintenance of distinct pools of value that can be transferred in a manner hiding the amounts and types of assets involved in transactions.

Early work in this area was conducted by Poelstra and collaborators [14], with the Bitcoin and Mimblewimble designs in mind. This approach to confidential assets uses Pedersen commitments with asset-specific generators to produce transactions where the types, amounts, and distributions of transferred assets are hidden; however, the transaction graph itself is only obscured up to the practical limits imposed by the overlying protocols. Further, this relative lack of information leakage comes at a cost: commitment range proofs cannot be aggregated to save space, and each generated coin must come equipped with a proof that it corresponds to a valid asset type.

Later work by Beam [5] extends the Lelantus protocol by applying Poelstra *et al.*'s construction to take advantage of transaction graph obfuscation. In this design, the identity of consumed coins is hidden by re-randomized commitments, and one-of-many asset validity proofs use a design by Groth and Kohlweiss [9] (later optimized by Bootle and collaborators [2]) with improved scaling.

In this paper, we propose Spats, a new design for confidential assets that offers different tradeoffs in efficiency and information leakage. Unlike earlier work that relies on asset-specific commitment generators to separate assets for balance integrity, Spats coins bind to asset types using an extended commitment with fixed global generators. This design choice permits range proof aggregation; when used with range proof constructions like Bulletproofs [4] or Bulletproofs+ [6] that scale logarithmically with the number of commitment assertions, the space savings can be impressive. Further, by restricting each transaction to a single hidden asset type in addition to a base asset used for fee handling, we replace multiple asset validity proofs with a single fixed-size asset type equality proof that applies to all consumed and generated coins of the hidden type.

We note carefully that a modification of the full protocol security model of [12] accounting for our changes, while important, is deferred to future work.

2 Primitives

We describe the cryptographic constructions required for the Spats protocol. Throughout this paper, let $\mathbb{G}$ be a prime-order group where the discrete logarithm, computational Diffie-Hellman, and decisional Diffie-Hellman problems are hard, and let $\mathbb{F}$ be its scalar field.

2.1 Extended commitments

The constructions used in Spats, like some of those used in the original Spark protocol, use homomorphic commitments with two independent value components; for clarity, we refer to these as *extended commitments*. Such a commitment scheme is common, and is often used more generally to bind to and hide a vector of values with a single mask. The public parameters are $pp_{\text{com}} = (\mathbb{G}, \mathbb{F}, F, G, H)$, where $F, G, H \in \mathbb{G}$ are independent public generators; that is, they have no efficiently-computable discrete logarithm relationship. The commitment scheme consists of a function $\mathsf{Comm} : \mathbb{F}^3 \to \mathbb{G}$ that is additively homomorphic. For our purposes, we let Comm be a Pedersen commitment, where

$$\mathsf{Comm}(v, v', m) = vF + v'G + mH$$

for all values $v, v' \in \mathbb{F}$ and masks $m \in \mathbb{F}$.

2.2 Parallel one-of-many proving system

The parallel one-of-many proving system definition used in Spark is modified to support extended commitments. The public parameters are $pp_{\text{par}} = (n, m, pp_{\text{comm}})$, where pp_{com} are the public parameters for an extended commitment construction. The algorithm tuple $(\mathsf{ParallelProve}, \mathsf{ParallelVerify})$ is modified to support the following relation:

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

The existing parallel one-of-many proving system in the Spark protocol is trivially modified to support this definition, since it also reduces to an assertion of commitments to zero.

2.3 Range proving system

The range proving construction is modified to support extended commitments. The public parameters are $pp_{\text{rp}} = (v_{\max}, pp_{\text{comm}})$, where pp_{comm} are the public parameters for an extended commitment construction. The algorithm tuple $(\mathsf{RangeProve}, \mathsf{RangeVerify})$ is modified to support the following relation:

$$\{pp_{\text{rp}}, \{C_j\}_{j=1}^{m} \in \mathbb{G}; \{(a_j, v_j, r_j)\}_{j=1}^{m} \in \mathbb{F} : \\ \forall j \in [1, m], 0 \leq v_j < v_{\max}, C_j = \mathsf{Comm}(a_j, v_j, r_j)\}$$

We describe here a modification to the Bulletproofs+ range proving system to support this construction. Zarcanum [15] provides a security proof for such a proving system. Note that we use additive notation to describe these changes, in order to match this convention in the Spark protocol, and we describe here only those aspects of the prover and verifier algorithms that are modified from those in [6].

The prover selects a value $\underline{\alpha} \in \mathbb{F}$ uniformly at random. It constructs

$$A = \vec{a}_L \vec{G} + \vec{a}_R \vec{H} + \mathsf{Comm}(\underline{\alpha}, 0, \alpha)$$

and defines

$$\underline{\widehat{\alpha}} = \underline{\alpha} + \sum_{j=1}^{m} z^{2j} a_j y^{mn+1}$$

as an additional witness to the weighted inner-product argument protocol we describe now.

The zero-knowledge weighted inner-product protocol is modified to support the following relation:

$$\{pp_{\text{erp}}, \vec{G}, \vec{H} \in \mathbb{G}^n, P \in \mathbb{G}; (\vec{a}, \vec{b}) \in \mathbb{F}^n, (\alpha, \underline{\alpha}) \in \mathbb{F} : \\ P = \vec{a}\vec{G} + \vec{b}\vec{H} + \mathsf{Comm}(\underline{\alpha}, \vec{a} \odot_y \vec{b}, \alpha)\}$$

In the case $n > 1$, the prover selects $\underline{d}_L, \underline{d}_R \in \mathbb{F}$ uniformly at random. It computes

$$\begin{aligned} L &= (y^{-\widehat{n}}\vec{a}_1)\vec{G}_2 + \vec{b}_2\vec{H}_1 + \mathsf{Comm}(\underline{d}_L, c_L, d_L) \\ R &= (y^{\widehat{n}}\vec{a}_2)\vec{G}_1 + \vec{b}_1\vec{H}_2 + \mathsf{Comm}(\underline{d}_R, c_R, d_R) \end{aligned}$$

and $\underline{\widehat{\alpha}} = \underline{d}_L e^2 + \underline{\alpha} + \underline{d}_R e^{-2}$.

In the case $n = 1$, the prover selects $\underline{\delta}, \underline{\eta} \in \mathbb{F}$ uniformly at random. It computes

$$\begin{aligned} A &= r\vec{G} + s\vec{H} + \mathsf{Comm}(\underline{\delta}, r \odot_y \vec{b} + s \odot_y \vec{a}, \delta) \\ B &= \mathsf{Comm}(\underline{\eta}, r \odot_y s, \eta) \end{aligned}$$

and $\underline{\delta}' = \underline{\eta} + \underline{\delta}e + \underline{\alpha}e^2$. It sends $\underline{\delta}'$ to the verifier.

The verifier accepts if and only if the following holds:

$$e^2P + eA + B = (r'e)\vec{G} + (s'e)\vec{H} + \mathsf{Comm}(\underline{\delta}', r' \odot_y s', \delta')$$

Observe that this construction adds only $\underline{\delta}'$ to the overall proof structure.

2.4 Aggregated representation proof

We require a proving system to assert in zero knowledge that the prover knows discrete logarithm representations of a set of extended commitments with a zero component. In the context of Spats, this is used to show that certain extended commitments reduce to standard commitments by setting one component's discrete logarithm to zero, and is useful by allowing optimal aggregation of commitment range proofs for space efficiency. Let $pp_{\text{agg}} = (pp_{\text{comm}})$ be the public parameters of such a proving system, where pp_{comm} are the public parameters for an extended commitment construction.

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

$$\{pp_{\text{agg}}, \{C_i\}_{i=0}^{n-1}; (\{y_i, z_i\}_{i=0}^{n-1}) : \forall i \in [0, n), C_i = \mathsf{Comm}(0, y_i, z_i)\}$$

The batch Schnorr proving system described in [8] may be easily generalized for this purpose, since an extended commitment with a zero component is algebraically equivalent to a standard Pedersen commitment.

2.5 Asset type equality proof

We introduce a new proving system to prove in zero knowledge that a set of extended commitments share a common single discrete logarithm in one component. In the context of Spats, this proving system will be used to assert in zero knowledge that coins consumed and generated in a transaction share a common asset type. Let

$$pp_{\text{type}} = (pp_{\text{comm}})$$

be the public parameters of such a proving system, where pp_{comm} are the public parameters for an extended commitment construction.

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

$$\{pp_{\text{type}}, \{C_i\}_{i=0}^{n-1} \in \mathbb{G}; (x, \{y_i, z_i\}_{i=0}^{n-1}) \in \mathbb{F} : \forall i \in [0, n), C_i = \mathsf{Comm}(x, y_i, z_i)\}$$

The protocol proceeds as follows:

1. The prover selects $r_x, r_y, r_z, s_y, s_z \in \mathbb{F}$ uniformly at random.
2. The prover computes $A = \mathsf{Comm}(r_x, r_y, r_z)$ and $B = \mathsf{Comm}(0, s_y, s_z)$, and sends these values to the verifier.
3. The verifier selects $c \in \mathbb{F}$ uniformly at random, and sends this challenge value to the prover.
4. The prover computes the values

$$\begin{aligned} t_x &= r_x + cx \\ t_y &= r_y + cy_0 \\ t_z &= r_z + cz_0 \\ u_y &= s_y + \sum_{i=1}^{n-1} c^i (y_i - y_0) \\ u_z &= s_z + \sum_{i=1}^{n-1} c^i (z_i - z_0) \end{aligned}$$

   and sends them to the verifier.

5. The verifier accepts the proof if and only if the following hold:

$$\mathsf{Comm}(t_x, t_y, t_z) = A + cC_0 \tag{1}$$

$$\mathsf{Comm}(0, u_y, u_z) = B + \sum_{i=1}^{n-1} c^i (C_i - C_0) \tag{2}$$

We now prove that this construction is a sigma protocol for the given relation.

Proof. To show the construction is a sigma protocol, we must show that it is complete, special sound, and honest-verifier zero knowledge.

Completeness follows trivially by inspection.

We now show that the protocol is n-special sound by constructing an extractor that produces a witness set on n accepting transcripts with distinct challenges. Consider n distinct challenges $\{c_j\}_{j=0}^{n-1}$ corresponding to response transcripts $\{(t_x^j, t_y^j, t_z^j, u_y^j, u_z^j)\}_{j=0}^{n-1}$, respectively, where we use superscript indexing. By applying Equation 1 to the $j = 0$ and $j = 1$ transcripts and subtracting them, we obtain the following:

$$\mathsf{Comm}(t_x^1 - t_x^0, t_y^1 - t_y^0, t_z^1 - t_z^0) = (c_1 - c_0)C_0$$

Since $c_1 \neq c_0$ by assumption, we have the following extracted witnesses:

$$\begin{aligned} x &= \frac{t_x^1 - t_x^0}{c_1 - c_0} \\ y_0 &= \frac{t_y^1 - t_y^0}{c_1 - c_0} \\ z_0 &= \frac{t_z^1 - t_z^0}{c_1 - c_0} \end{aligned}$$

To extract the remaining witness elements, apply Equation 2 to obtain the following system of equations:

$$\begin{aligned} B + \sum_{i=1}^{n-1} c_0^i (C_i - C_0) &= \mathsf{Comm}(0, u_y^0, u_z^0) \\ B + \sum_{i=1}^{n-1} c_1^i (C_i - C_0) &= \mathsf{Comm}(0, u_y^1, u_z^1) \\ &\vdots \\ B + \sum_{i=1}^{n-1} c_{n-1}^i (C_i - C_0) &= \mathsf{Comm}(0, u_y^{n-1}, u_z^{n-1}) \end{aligned} \tag{3}$$

Subtract each equation from the first:

$$\begin{aligned} \sum_{i=1}^{n-1} (c_1^i - c_0^i)(C_i - C_0) &= \mathsf{Comm}(0, u_y^1 - u_y^0, u_z^1 - u_z^0) \\ \sum_{i=1}^{n-1} (c_2^i - c_0^i)(C_i - C_0) &= \mathsf{Comm}(0, u_y^2 - u_y^0, u_z^2 - u_z^0) \\ &\vdots \\ \sum_{i=1}^{n-1} (c_{n-1}^i - c_0^i)(C_i - C_0) &= \mathsf{Comm}(0, u_y^{n-1} - u_y^0, u_z^{n-1} - u_z^0) \end{aligned}$$

Then we define a set $\{y_i\}_{i=1}^{n-1}$ via the following linear system:

$$\sum_{i=1}^{n-1}(c_1^i - c_0^i)(y_i - y_0) = u_y^1 - u_y^0$$
$$\sum_{i=1}^{n-1}(c_2^i - c_0^i)(y_i - y_0) = u_y^2 - u_y^0$$
$$\vdots$$
$$\sum_{i=1}^{n-1}(c_{n-1}^i - c_0^i)(y_i - y_0) = u_y^{n-1} - u_y^0$$

Since the challenges are uniformly distributed, the system has a unique solution in $\{y_i\}_{i=1}^{n-1}$ with high probability. We can form a similar linear system to obtain unique $\{z_i\}_{i=1}^{n-1}$.

This implies that the terms $C_i - C_0 = \mathsf{Comm}(0, y_i - y_0, z_i - z_0)$ satisfy the original linear system in Equation 3 for all $i \in [1, n)$. From our earlier extraction we have $C_0 = \mathsf{Comm}(x, y_i, z_i)$, so for $i \in [1, n)$ we have $C_i = \mathsf{Comm}(x, y_i, z_i)$ as required.

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

It remains to prove that the protocol is honest-verifier zero knowledge. To show this, we construct a simulator that produces transcripts distributed identically to those of valid proofs. Given a valid statement and random challenge $c \in \mathbb{F}$, the simulator chooses $t_x, t_y, t_z \in \mathbb{F}$ uniformly at random, and sets $A = \mathsf{Comm}(t_x, t_y, t_z) - cC_0$ using these values. It selects $u_y, u_z \in \mathbb{F}$ uniformly at random, and sets

$$B = \mathsf{Comm}(0, u_y, u_z) - \sum_{i=1}^{n-1} c^i(C_i - C_0)$$

using these values. By construction, this transcript passes the verification Equations 1 and 2. Since the extended commitment generators are independent, all proof elements in both the simulation and real proofs are identically uniformly distributed.

This completes the proof. □

2.6 Balance proof

We require a representation proof for use in balance assertion. It proves that a extended commitment is bound to a zero value. In the context of Spats, this proving system will be used to assert that value is maintained between consumed and generated coins in a transaction. Let

$$pp_{\text{bal}} = (pp_{\text{comm}})$$

be the public parameters of such a proving system, where pp_{comm} are the public parameters for an extended commitment construction.

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

$$\{pp_{\text{bal}}, C \in \mathbb{G}; (x, z) \in \mathbb{F} : C = \mathsf{Comm}(x, 0, z)\}$$

The protocol proceeds as follows:

1. The prover selects $r_x, r_z \in \mathbb{F}$ uniformly at random.
2. The prover computes $A = \mathsf{Comm}(r_x, 0, r_z)$ and sends this value to the verifier.
3. The verifier selects $c \in \mathbb{F}$ uniformly at random, and sends this challenge value to the prover.
4. The prover computes the values
$$\begin{aligned} t_x &= r_x + cx \\ t_z &= r_z + cz \end{aligned}$$
and sends them to the verifier.
5. The verifier accepts the proof if and only if $\mathsf{Comm}(t_x, 0, t_z) = A + cC$ holds.

This construction is a sigma protocol for the given relation; the proof is standard, and we omit it here.

3 Protocol

We now describe the changes to the Spark protocol algorithms required for Spats. We assume the use of notation from [12]. Note that key creation, address creation, and coin recovery are unchanged, so we do not repeat those algorithms here.

In the Spats construction, an asset type is defined as a scalar $a \in \mathbb{F}$; all coins associated to a given asset type must share a common a. The intent is that asset types, like coin values, are bound to coins in a hidden manner and made public only when introducing new value into the system. Consensus rules must determine the conditions under which new value for any asset type may be added.

As a matter of terminology and notational convenience, if a coin has asset type $a = 0$, we call it a *base asset* coin; otherwise, we call it a *generic asset* coin.

3.1 CreateCoin

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

Inputs: Destination public address $\mathsf{addr}_{\text{pk}}$, value $v \in [0, v_{\text{max}})$, memo m, asset type $a \in \mathbb{F}$, type bit b

Outputs: Coin Coin, nonce k

1. Parse the recipient address $\mathsf{addr}_{\text{pk}} = (d, Q_1, Q_2)$.
2. Sample a nonce $k \in \mathbb{F}$.
3. Compute the recovery key $K = \mathcal{H}_k(k)\mathcal{H}_{\text{div}}(d)$.
4. Compute the serial number commitment
$$S = \mathsf{Comm}(\mathcal{H}_{\text{ser}}(k), 0, 0) + Q_2.$$
5. Generate the value commitment $C = \mathsf{Comm}(a, v, \mathcal{H}_{\text{val}}(k))$.
6. If $b = 0$, set the recipient data $r = (v, a, d, k, m)$; otherwise, set $r = (d, k, m)$.
7. Generate an AEAD encryption key $k_{\text{aead}} = \mathsf{AEADKeyGen}(\mathcal{H}_k(k)Q_1)$; encrypt the recipient data
$$\overline{r} = \mathsf{AEADEncrypt}(k_{\text{aead}}, \mathtt{r}, r).$$
8. If $b = 0$, output the coin $\mathsf{Coin} = (S, K, C, \overline{r})$ and nonce k; otherwise, output the coin $\mathsf{Coin} = (S, K, C, v, a, \overline{r})$ and nonce k.

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

3.2 Mint

This algorithm generates new coins of arbitrary asset type from either a mining-type process defined by consensus rules, 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.

Inputs: Set of t output coin public addresses, values, memos, and asset types:
$$\{\mathsf{addr}_{\text{pk},j}, v_j, m_j, a_j\}_{j=0}^{t-1}$$

Outputs: Mint transaction tx_{mint}

1. Generate a set $\mathsf{OutCoins} = \{\mathsf{CreateCoin}(\mathsf{addr}_{\mathrm{pk},j}, v_j, m_j, a_j, 1)\}_{j=0}^{t-1}$ of output coins.
2. Parse the output coin value commitments $\{\overline{C}_j\}_{j=0}^{t-1}$ from $\mathsf{OutCoins}$, where each $\overline{C}_j$ contains nonce k_j.
3. Generate a representation proof for value assertion:
$$\Pi_{\mathrm{val}} = \mathsf{RepProve}\left(pp_{\mathrm{rep}}, H, \{\overline{C}_j - \mathsf{Comm}(a_j, v_j, 0)\}_{j=0}^{t-1}; \{\mathcal{H}_{\mathrm{val}}(k_j)\}_{j=0}^{t-1}\right)$$
4. Output the mint transaction $\mathsf{tx}_{\mathrm{mint}} = (\mathsf{OutCoins}, \Pi_{\mathrm{val}})$.

3.3 Identify

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

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

Inputs: Incoming view key $\mathsf{addr}_{\mathrm{in}}$, coin Coin

Outputs: Value v, memo m, asset type a, diversifier i, nonce k

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

3.4 Spend

This algorithm allows a recipient to generate a transaction that consumes coins it controls, and generates new coins destined for arbitrary public addresses. A spend transaction spends coins of two asset types separately; one is a generic asset type (where the type itself is hidden), and the other is the base asset type. This ensures that fees can be denominated consistently and publicly.

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

Inputs:

- A full view key $\mathsf{addr}_{\text{full}}$
- A spend key $\mathsf{addr}_{\text{sk}}$
- A set of N input coins $\mathsf{GenericInCoins}$, of generic asset type, as part of a cover set
- A set of N input coins $\mathsf{BaseInCoins}$, of base asset type, as part of a cover set
- For each $u \in [0, \underline{w})$ coin of generic type $a \neq 0$ to spend, the index in $\mathsf{GenericInCoins}$, serial number, tag, value, and nonce: $(\underline{l}_u, \underline{s}_u, \underline{T}_u, \underline{v}_u, \underline{k}_u)$
- For each $u \in [0, w)$ coin of base type $a = 0$ to spend, the index in $\mathsf{BaseInCoins}$, serial number, tag, value, and nonce: $(l_u, s_u, T_u, v_u, k_u)$
- An integer fee value $f \in [0, v_{\max})$
- A set of $\underline{t}$ generic-type output coin public addresses, values, and memos:

$$\{\underline{\mathsf{addr}}_{\text{pk},j}, \underline{v}_j, \underline{m}_j\}_{j=0}^{t-1}$$

- A set of t base-type output coin public addresses, values, and memos:

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

Outputs: Spend transaction tx_{spend}

1. Parse the required full view key component D from $\mathsf{addr}_{\text{full}}$.
2. Parse the spend key $\mathsf{addr}_{\text{sk}} = (s_1, s_2, r)$.
3. Parse the cover set serial number commitments and value commitments $\{(S_i, C_i)\}_{i=0}^{N-1}$ from $\mathsf{BaseInCoins}$ and $\{(\underline{S}_i, \underline{C}_i)\}_{i=0}^{N-1}$ from $\mathsf{GenericInCoins}$.
4. For each $u \in [0, w)$:

   (a) Compute the serial number commitment offset:

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

(b) Compute the value commitment offset:

$$C'_u = \mathsf{Comm}(0, v_u, \mathcal{H}_{\mathrm{val}'}(s_u, D))$$

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

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

5. For each $u \in [0, \underline{w})$:

(a) Compute the serial number commitment offset:

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

(b) Compute the value commitment offset:

$$\underline{C}'_u = \mathsf{Comm}(a, \underline{v}_u, \mathcal{H}_{\mathrm{val}'}(\underline{s}_u, D))$$

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

$$(\underline{\Pi}_{\mathrm{par}})_u = \mathsf{ParProve}(pp_{\mathrm{par}}, \{\underline{S}_i, \underline{C}_i\}_{i=0}^{N-1}, \underline{S}'_u, \underline{C}'_u; \\ (\underline{l}_u, \mathcal{H}_{\mathrm{ser}'}(\underline{s}_u, D), \mathcal{H}_{\mathrm{val}}(\underline{k}_u) - \mathcal{H}_{\mathrm{val}'}(\underline{s}_u, D)))$$

6. Generate a set $\mathsf{BaseOutCoins} = \{\mathsf{CreateCoin}(\mathsf{addr}_{\mathrm{pk},j}, v_j, m_j, 0, 0)\}_{j=0}^{t-1}$ of base-type output coins.

7. Generate a set $\mathsf{GenericOutCoins} = \{\mathsf{CreateCoin}(\underline{\mathsf{addr}}_{\mathrm{pk},j}, \underline{v}_j, \underline{m}_j, a, 0)\}_{j=0}^{t-1}$ of generic-type output coins.

8. Parse the coin value commitments $\{\overline{C}_j\}_{j=0}^{t-1}$ from $\mathsf{BaseOutCoins}$, where each $\overline{C}_j$ is associated to nonce $\overline{k}_j$.

9. Parse the coin value commitments $\{\underline{\overline{C}}_j\}_{j=0}^{t-1}$ from $\mathsf{GenericOutCoins}$, where each $\underline{\overline{C}}_j$ is associated to nonce $\underline{\overline{k}}_j$.

10. Generate an aggregated range proof for all output coins:

$$\Pi_{\mathrm{rp}} = \mathsf{RangeProve}\Big(pp_{\mathrm{rp}}, \{\overline{C}_j\}_{j=0}^{t-1} \cup \{\underline{\overline{C}}_j\}_{j=0}^{t-1}; \\ \{(0, v_j, \mathcal{H}_{\mathrm{val}}(\overline{k}_j))\}_{j=0}^{t-1} \cup \{(a, \underline{v}_j, \mathcal{H}_{\mathrm{val}}(\underline{\overline{k}}_j))\}_{j=0}^{t-1}\Big)$$

11. Generate a proof that all base-type assets have $a = 0$:

$$\Pi_{\mathrm{base}} = \mathsf{AggProve}\left(pp_{\mathrm{agg}}, \{\overline{C}_j\}_{j=0}^{t-1}; (\{v_j, \mathcal{H}_{\mathrm{val}}(\overline{k_j})\}_{j=0}^{t-1})\right)$$

12. Generate a proof that all generic-type coins have the same type:

$$\Pi_{\text{type}} = \mathsf{TypeProve}\Big(pp_{\text{type}}, \{\underline{C}'_u\}_{u=0}^{\underline{w}-1} \cup \{\underline{\overline{C}}_j\}_{j=0}^{\underline{t}-1}; \\ \Big(a, \{\underline{v}_u, \mathcal{H}_{\text{val}'}(\underline{s}_u, D)\}_{u=0}^{\underline{w}-1} \cup \{\underline{v}_j, \mathcal{H}_{\text{val}}(\underline{\overline{k}}_j)\}_{j=0}^{\underline{t}-1}\Big)\Big)$$

13. Generate representation proofs for balance assertion:

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

$$\underline{\Pi}_{\text{bal}} = \mathsf{RepProve}\left(pp_{\text{rep}}, H, \sum_{u=0}^{\underline{w}-1} \underline{C}'_u - \sum_{j=0}^{\underline{t}-1} \underline{\overline{C}}_j; \\ \sum_{u=0}^{\underline{w}-1} \mathcal{H}_{\text{val}'}(\underline{s}_u, D) - \sum_{j=0}^{\underline{t}-1} \mathcal{H}_{\text{val}}(\underline{\overline{k}}_j)\right)$$

14. Define the following binding hash:

$$\mu = \mathcal{H}_{\text{bind}}(\mathsf{BaseInCoins}, \mathsf{GenericInCoins}, \mathsf{BaseOutCoins}, \mathsf{GenericOutCoins}, \\ \{S'_u, C'_u, T_u, (\Pi_{\text{par}})_u,\}_{u=0}^{w-1}, \{\underline{S}'_u, \underline{C}'_u, \underline{T}_u, (\underline{\Pi}_{\text{par}})_u,\}_{u=0}^{\underline{w}-1}, \\ \Pi_{\text{base}}, \Pi_{\text{rp}}, \Pi_{\text{bal}}, \underline{\Pi}_{\text{bal}}, \Pi_{\text{type}})$$

15. Generate a modified Chaum-Pedersen proof, where we additionally bind μ to the initial transcript:

$$\Pi_{\text{chaum}} = \mathsf{ChaumProve}((pp_{\text{chaum}}, \mu), \{S'_u, T_u\}_{u=0}^{w-1} \cup \{\underline{S}'_u, \underline{T}_u\}_{u=0}^{\underline{w}-1}; \\ (\{s_u, r, -\mathcal{H}_{\text{ser}'}(s_u, D)\}_{u=0}^{w-1} \cup \{\underline{s}_u, r, -\mathcal{H}_{\text{ser}'}(\underline{s}_u, D)\}_{u=0}^{\underline{w}-1}))$$

16. Output the tuple:

$$\text{tx}_{\text{spend}} = (\mathsf{BaseInCoins}, \mathsf{GenericInCoins}, \mathsf{BaseOutCoins}, \mathsf{GenericOutCoins}, \\ f, \{S'_u, C'_u, T_u, (\Pi_{\text{par}})_u\}_{u=0}^{w-1}, \{\underline{S}'_u, \underline{C}'_u, \underline{T}_u, (\underline{\Pi}_{\text{par}})_u\}_{u=0}^{\underline{w}-1}, \\ \Pi_{\text{base}}, \Pi_{\text{rp}}, \Pi_{\text{bal}}, \underline{\Pi}_{\text{bal}}, \Pi_{\text{type}}, \Pi_{\text{chaum}})$$

Observe that this effectively maintains two "pools" of coins available for cover sets. Coins minted with $a = 0$, or those generated in spends produced by consuming base-type coins, are the only coins used for producing $\mathsf{BaseInCoins}$ cover sets. Coins minted with $a \neq 0$, or those generated in spends produced by consuming generic-type coins, are the only coins used for $\mathsf{GenericInCoins}$ cover sets.

Also note that it is possible to aggregate all output coin range proofs using a suitable range proving system; this is particularly relevant for designs like those of [4, 6] that scale logarithmically in size for aggregation purposes.

Transaction verification is modified in a straightforward manner to verify the relevant proofs.

4 Efficiency

We briefly outline the efficiency of Spats compared to the original Spark protocol. Table 1 shows the change in size of mint transaction components, and Table 2 shows the change in size of spend transaction components. We assume throughout that both group and scalar elements have 32-byte representations, and that asset type identifiers are restricted to τ bytes.

Component	$\Delta\mathbb{G}$	$\Delta\mathbb{F}$	**Δ bytes**
a			$\tau\underline{t}$
Total			$\tau\underline{t}$

Table 1: Mint transaction size change by component ($\underline{t}$ generated generic-type coins)

Component	$\Delta\mathbb{G}$	$\Delta\mathbb{F}$	**Δ bytes**
Π_{rp}		1	
Π_{bal}		1	
$\underline{\Pi}_{\text{bal}}$	1	2	
Π_{base}	2	2	
$\{\overline{r}_j\}$			$\tau\underline{t}$
Π_{type}	2	5	
Total	5	11	$\tau\underline{t}$

Table 2: Spend transaction size change by component ($\underline{t}$ generated generic-type coins)

A mint transaction with $\underline{t}$ generated generic-type coins increases by $\tau\underline{t}$ bytes. A spend transaction with $\underline{t}$ generated generic-type coins increases by $512 + \tau t$ bytes.

References

[1] 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.

[2] 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.

[3] 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`.

[4] 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.

[5] Pyrros Chaidos and Vladislav Gelfer. Lelantus-CLA. Cryptology ePrint Archive, Report 2021/1036, 2021. `https://ia.cr/2021/1036`.

[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] 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.

[8] R. Gennaro, D. Leigh, R. Sundaram, and W. Yerazunis. Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, pages 276–292, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[9] 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.

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

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

[12] Aram Jivanyan and Aaron Feickert. Lelantus Spark: Secure and flexible private transactions. Cryptology ePrint Archive, Report 2021/1173, 2021. `https://ia.cr/2021/1173`.

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

[14] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *Financial Cryptography and Data Security*, pages 43–63, Berlin, Heidelberg, 2019. Springer Berlin Heidelberg.

[15] `sowle` and `koe`. Zarcanum: A proof-of-stake scheme for confidential transactions with hidden amounts. Cryptology ePrint Archive, Report 2021/1478, 2021. `https://ia.cr/2021/1478`.