Spats: confidential assets and non-fungible tokens

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

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

June 16, 2023

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 or non-fungible tokens. Here we describe Spats, a new design for confidential assets and serialized tokens 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 identifier, and asserting in zero knowledge that they are 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.

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

*Corresponding author: aram@firo.org

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, with the advantage that multiple types may be included in a single transaction with balance separation; 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. This design unfortunately inherits the range proof and type validity requirements of [14].

In this paper, we propose Spats, a new design for confidential assets that offers different tradeoffs in efficiency and information leakage, and also admits functionality suitable for private non-fungible tokens, where uniqueness and atomicity are required. Unlike protocols that rely on asset-specific commitment generators to separate assets for balance integrity, Spats coins bind to asset types and (optionally) token identifiers 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. Spats transactions require separate balance assertions for each hidden asset type or non-fungible token present in a transaction. While this approach limits flexibility somewhat, it has the advantage of replacing separate expensive type validity proofs with a simpler and smaller fixed-size proof that applies to all consumed and generated coins of a given type or token.

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 require homomorphic commitments with multiple 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 $pp_{\text{com},n} = (\mathbb{G}, \mathbb{F}, \{G_i\}_{i=0}^{n-1}, H)$, where $n > 1$ and $\{G_i\}_{i=0}^{n-1} \cup \{H\} \subset \mathbb{G}$ is a set of independent public generators; that is, they have no efficiently-computable discrete logarithm relationship. The commitment scheme consists of a function $\mathsf{Com}_n : \mathbb{F}^{n+1} \to \mathbb{G}$ that is additively homomorphic. For our purposes, we let Com_n be a Pedersen-type commitment, where

$$\mathsf{Com}_n(\{v_i\}_{i=0}^{n-1}, m) = \sum_{i=0}^{n-1} v_i G_i + mH$$

for all values $\{v_i\}_{i=0}^{n-1} \subset \mathbb{F}$ and masks $m \in \mathbb{F}$.

Because the generators are independent, such a commitment scheme is computationally binding. It is also perfectly hiding.

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 $pp_{\text{par}} = (n, m, pp_{\text{com},2}, pp_{\text{com},3})$, where $pp_{\text{com},2}$ and $pp_{\text{com},2}$ are the public parameters for extended commitment constructions. We note that while commitment generators must be independent within a parameter set, they need not be independent across distinct parameter sets. The algorithm tuple $(\mathsf{ParallelProve}, \mathsf{ParallelVerify})$ is modified to support the following relation:

$$\{pp_{\text{par}}, \{S_k, V_k\}_{k=0}^{N-1} \subset \mathbb{G}^2, S', V' \in \mathbb{G}; l \in \mathbb{N}, (s, v) \in \mathbb{F} : \\ 0 \leq l < N, S_l - S' = \mathsf{Com}_2(0, 0, s), V_l - V' = \mathsf{Com}_3(0, 0, 0, v)\}$$

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

$$\{pp_{\text{rp}}, \{C_j\}_{j=0}^{m-1} \in \mathbb{G}; \{(v_{j,i}, m_j)\}_{j,i=0}^{m-1,n-1} \in \mathbb{F} : \\ \forall j \in [0, m), 0 \leq v_j < v_{\max}, C_j = \mathsf{Com}_n(\{v_{j,i}\}_{i=0}^{n-1}, m_j)\}$$

It is possible to produce a straightforward modification to the Bulletproofs+ range proving system to support this construction. Zarcanum [15] describes such a modification for such a proving system, and updates the corresponding security proofs.

We note that the modification adds only $n-1$ additional scalar elements to the overall proof structure, compared to a standard Bulletproofs+ range proof.

2.4 Base asset 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 certain zero components. In the context of Spats, this is used to show that certain extended commitments represent a base asset type reducing to standard Pedersen commitments by setting certain discrete logarithms to zero, and is useful by allowing optimal aggregation of commitment range proofs for space efficiency. Let $pp_{\text{base}} = (pp_{\text{com},3})$ be the public parameters of such a proving system, where pp_{com_3} are the public parameters for an extended commitment construction.

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

$$\{pp_{\text{base}}, \{C_i\}_{i=0}^{n-1}; (\{y_i, z_i\}_{i=0}^{n-1}) : \forall i \in [0, n), C_i = \mathsf{Com}_3(0, 0, y_i, z_i)\}$$

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

2.5 Type equality proof

We introduce a new proving system to prove in zero knowledge that a set of extended commitments share common single discrete logarithms in certain components. 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 and identifier. Let

$$pp_{\text{type}} = (pp_{\text{com},3})$$

be the public parameters of such a proving system, where $pp_{\text{com},3}$ 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:

$$\big\{pp_{\text{type}}, \{C_i\}_{i=0}^{n-1} \in \mathbb{G}; (w, x, \{y_i, z_i\}_{i=0}^{n-1}) \in \mathbb{F} : \\ \forall i \in [0, n), C_i = \mathsf{Com}_3(w, x, y_i, z_i)\}$$

The protocol proceeds as follows:

1. The prover selects $r_w, r_x, r_y, r_z, s_y, s_z \in \mathbb{F}$ uniformly at random.

2. The prover computes $A = \mathsf{Com}_3(r_w, r_x, r_y, r_z)$ and $B = \mathsf{Com}_3(0, 0, s_y, s_z)$, 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

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

   and sends them to the verifier.

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

$$\mathsf{Com}_3(t_w, t_x, t_y, t_z) = A + cC_0 \tag{1}$$

$$\mathsf{Com}_3(0, 0, u_y, u_z) = B + \sum_{i=1}^{n-1} c^i (C_i - C_0) \tag{2}$$

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 $\{(t_w^j, t_x^j, t_y^j, t_z^j, u_y^j, u_z^j)\}_{j=0}^{n-1}$, 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:

$$\mathsf{Com}_3(t_w^1 - t_w^0, t_x^1 - t_x^0, t_y^1 - t_y^0, t_z^1 - t_z^0) = (c_1 - c_0)C_0$$

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

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

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

$$\begin{aligned} B + \sum_{i=1}^{n-1} c_0^i (C_i - C_0) &= \mathsf{Com}_3(0, 0, u_y^0, u_z^0) \\ B + \sum_{i=1}^{n-1} c_1^i (C_i - C_0) &= \mathsf{Com}_3(0, 0, u_y^1, u_z^1) \\ &\vdots \\ B + \sum_{i=1}^{n-1} c_{n-1}^i (C_i - C_0) &= \mathsf{Com}_3(0, 0, u_y^{n-1}, u_z^{n-1}) \end{aligned} \tag{3}$$

Subtract each equation from the first:

$$\begin{aligned} \sum_{i=1}^{n-1} (c_1^i - c_0^i)(C_i - C_0) &= \mathsf{Com}_3(0, 0, u_y^1 - u_y^0, u_z^1 - u_z^0) \\ \sum_{i=1}^{n-1} (c_2^i - c_0^i)(C_i - C_0) &= \mathsf{Com}_3(0, 0, u_y^2 - u_y^0, u_z^2 - u_z^0) \\ &\vdots \\ \sum_{i=1}^{n-1} (c_{n-1}^i - c_0^i)(C_i - C_0) &= \mathsf{Com}_3(0, 0, u_y^{n-1} - u_y^0, u_z^{n-1} - u_z^0) \end{aligned}$$

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 $C_i - C_0 = \mathsf{Com}_3(0, 0, y_i - y_0, z_i - z_0)$ satisfy the original linear system in Equation 3 for all $i \in [1, n)$. From our earlier extraction we have $C_0 = \mathsf{Com}_3(w, x, y_0, z_0)$, so for $i \in [1, n)$ we have $C_i = \mathsf{Com}_3(w, x, y_i, z_i)$ as required.

It remains to show that these solutions are unique; that is, that no C_i has a different representation with coefficients w', 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 $w = w'$ 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 $t_w, t_x, t_y, t_z \in \mathbb{F}$ uniformly at random, and sets $A = \mathsf{Com}_3(t_w, t_x, t_y, t_z) - cC_0$ using these values. It selects $u_y, u_z \in \mathbb{F}$ uniformly at random, and sets

$$B = \mathsf{Com}_3(0, 0, u_y, u_z) - \sum_{i=1}^{n-1} c^i(C_i - C_0)$$

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 Extended balance proof

We require a representation proof for use in balance assertion. It proves that an 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

$$pp_{\text{bal}} = (pp_{\text{com},3})$$

be the public parameters of such a proving system, where $pp_{\text{com},3}$ 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:

$$\{pp_{\text{bal}}, C \in \mathbb{G}; (w, x, z) \in \mathbb{F} : C = \mathsf{Com}_3(w, x, 0, z)\}$$

The protocol proceeds as follows:

1. The prover selects $r_w, r_x, r_z \in \mathbb{F}$ uniformly at random.
2. The prover computes $A = \mathsf{Com}_3(r_w, r_x, 0, r_z)$ 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

$$\begin{aligned} t_w &= r_w + cw \\ t_x &= r_x + cx \\ t_z &= r_z + cz \end{aligned}$$

   and sends them to the verifier.
5. The verifier accepts the proof if and only if $\mathsf{Com}_3(t_w, t_x, 0, t_z) = A + cC$ 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 addition to binding to a value and mask, each coin in Spats also binds to an asset type $a \in \mathbb{F}$ and identifier $\iota \in \mathbb{F}$. All coins of a common asset type share a common a. In cases where coins of a given type are intended to be fungible and divisible, we set $\iota = 0$. In cases where coins of a given type are intended to represent non-fungible tokens that are atomic, each such token is represented by a coin whose identifier ι is unique within its type. Consensus

rules must determine the conditions under which new coins 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; if we do not specify a particular a, we call it a *generic asset* coin. While Spats can be easily generalized such that this distinction is irrelevant, the implementation we describe here requires differentiation; this is to allow for transactions to include fees that are always denominated in the base asset type.

We note a crucial requirement for non-fungible token issuance that is essential to ensure uniqueness and atomicity. When a new non-fungible token is issued for a particular asset type, its value must be set to $v = 1$. This ensures that when the token is transferred, it can only be done atomically. Spend transactions are constructed intended to allow consumption and production of multiple coins of generic asset type, and the associated proofs require that all such coins share the same asset type and identifier. The requirement that a given non-fungible token have $v = 1$ at the time of issuance ensures that if multiple coins are produced in a transaction transferring the token, exactly one such coin will also have $v = 1$. This has the advantage of making spend transactions uniform, regardless of whether or not they consume and generate non-fungible tokens.

3.1 CreateCoin

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

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

Outputs: Coin Coin, nonce k

1. Parse the recipient address $\mathsf{addr}_{\mathrm{pk}} = (d, Q_1, Q_2)$.
2. Sample a nonce $k \in \mathbb{F}$.
3. Compute the recovery key $K = \mathcal{H}_k(k)\mathcal{H}_{\mathrm{div}}(d)$.
4. Compute the serial number commitment
$$S = \mathsf{Com}_2(\mathcal{H}_{\mathrm{ser}}(k), 0, 0) + Q_2.$$
5. Generate the value commitment $C = \mathsf{Com}_3(a, \iota, v, \mathcal{H}_{\mathrm{val}}(k))$.
6. If $b = 0$, set the recipient data $r = (a, \iota, v, d, k, m)$; otherwise, set $r = (d, k, m)$.
7. Generate an AEAD encryption key $k_{\mathrm{aead}} = \mathsf{AEADKeyGen}(\mathcal{H}_k(k)Q_1)$; encrypt the recipient data
$$\overline{r} = \mathsf{AEADEncrypt}(k_{\mathrm{aead}}, \mathtt{r}, r).$$

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

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

3.2 Mint

This algorithm generates new coins of arbitrary asset type and identifier 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, asset types, and identifiers:

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

Outputs: Mint transaction $\mathrm{tx}_{\mathrm{mint}}$

1. For any $j \in [0, t)$, if $\iota_j \neq 0$ and $v_j \neq 1$, abort.
2. For any $j \in [0, t)$, if $a_j = 0$ and $\iota_j \neq 0$, abort.
3. Generate a set $\mathsf{OutCoins} = \{\mathsf{CreateCoin}(\mathsf{addr}_{\mathrm{pk},j}, v_j, m_j, a_j, \iota_j, 1)\}_{j=0}^{t-1}$ of output coins.
4. 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.
5. Generate a representation proof for value, asset type, and identifier assertion:

$$\Pi_{\mathrm{val}} = \mathsf{RepProve}\,(pp_{\mathrm{rep}}, H, \{\overline{C}_j - \mathsf{Com}_3(a_j, \iota_j, v_j, 0)\}_{j=0}^{t-1}; \{\mathcal{H}_{\mathrm{val}}(k_j)\}_{j=0}^{t-1})$$

6. 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, asset type, identifier, 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 assocated to the incoming view key, 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, identifier ι, 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 its components as $\mathsf{Coin} = (S, K, C, a, \iota, v, \overline{r})$; 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 $r = (a, \iota, v, d, k, m)$.
5. Check that $K = \mathcal{H}_k(k)\mathcal{H}_{\mathrm{div}}(d)$, and return failure otherwise.
6. Check that $C = \mathsf{Com}_3(a, \iota, v, \mathcal{H}_{\mathrm{val}}(k))$, and return failure otherwise.
7. Decrypt the diversifier $i = \mathsf{SymDecrypt}(\mathsf{SymKeyGen}(s_1), d)$.
8. Check that
$$S = \mathsf{Com}_2(\mathcal{H}_{\mathrm{ser}}(k), 0, 0) + \mathsf{Com}_2(\mathcal{H}_{Q_2}(s_1, i), 0, 0) + P_2,$$
and return failure otherwise.
9. Output (v, m, a, ι, 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 $a = a'$ for some a', and the other is the base asset type $a = 0$. This ensures that fees can be denominated consistently and publicly. It is required that all consumed and generated coins of type $a = a'$ share a common identifier $\iota = \iota'$ for some ι', and that all generated coins of type $a = 0$ have identifier $\iota = 0$.

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 $\mathsf{addr}_{\mathrm{full}}$
- A spend key $\mathsf{addr}_{\mathrm{sk}}$
- A set of N input coins $\mathsf{InCoins}$ as a cover set, each of which was generated in a previous valid transaction

- A set of indexes in $\mathsf{InCoins}$, serial numbers, tags, values, and nonces $\{l_u, s_u, T_u, v_u, k_u\}_{u=0}^{w+w'-1}$ such that the $[0, w)$-subset corresponds to base coins with $a = 0$ to spend and the $[w, w')$-subset corresponds to generic coins of a common type a' to spend
- An integer fee value $f \in [0, v_{\max})$
- A set of output coin public addresses, values, and memos:

$$\{\mathsf{addr}_{\mathrm{pk},j}, v_j, m_j\}_{j=0}^{t+t'-1}$$

  such that the $[0, t)$-subset corresponds to base coins with $a = 0$ to generate and the $[t, t')$-subset corresponds to coins of type a' to generate

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

1. Parse the required full view key component D from $\mathsf{addr}_{\mathrm{full}}$.
2. Parse the spend key $\mathsf{addr}_{\mathrm{sk}} = (s_1, s_2, r)$.
3. Parse the cover set serial number commitments and value commitments $\{(S_i, C_i)\}_{i=0}^{N-1}$ from $\mathsf{InCoins}$.
4. For each $u \in [0, w + w' - 1)$:
   (a) Compute the serial number commitment offset:

$$S'_u = \mathsf{Com}_2(s_u, 0, -\mathcal{H}_{\mathrm{ser}'}(s_u, D)) + D$$

   (b) Compute the value commitment offset:

$$C'_u = \mathsf{Com}_3(a, \iota, v_u, \mathcal{H}_{\mathrm{val}'}(s_u, D))$$

   (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. Generate a set $\mathsf{OutCoins} = \{\mathsf{CreateCoin}(\mathsf{addr}_{\mathrm{pk},j}, v_j, m_j, a, \iota, 0)\}_{j=0}^{t+t'-1}$ of output coins.
6. Parse the coin value commitments $\{\overline{C}_j\}_{j=0}^{t+t'-1}$ from $\mathsf{OutCoins}$, where each $\overline{C}_j$ is associated to nonce $\overline{k}_j$.
7. Generate an aggregated range proof for all output coins:

$$\Pi_{\mathrm{rp}} = \mathsf{RangeProve}\left(pp_{\mathrm{rp}}, \{\overline{C}_j\}_{j=0}^{t+t'-1}; \{(a, \iota, v_j, \mathcal{H}_{\mathrm{val}}(\overline{k}_j))\}_{j=0}^{t+t'-1}\right)$$

8. Generate a proof that all base-type assets have $a = \iota = 0$:

$$\Pi_{\text{base}} = \mathsf{BaseProve}\left(pp_{\text{agg}}, \{C'_u\}_{u=0}^{w-1} \cup \{\overline{C}_j\}_{j=0}^{t-1}; \left(\{v_u, \mathcal{H}_{\text{val}'}(s_u, D)\}_{u=0}^{w-1} \cup \{v_j, \mathcal{H}_{\text{val}}(\overline{k}_j)\}_{j=0}^{t-1}\right)\right)$$

9. Generate a proof that all generic-type coins have the same type and identifier:

$$\Pi_{\text{type}} = \mathsf{TypeProve}\left(pp_{\text{type}}, \{C'_u\}_{u=w}^{w+w'-1} \cup \{\overline{C}_j\}_{j=t}^{t+t'-1}; \left(a, \iota, \{v_u, \mathcal{H}_{\text{val}'}(s_u, D)\}_{u=w}^{w+w'-1} \cup \{v_j, \mathcal{H}_{\text{val}}(\overline{k}_j)\}_{j=t}^{t+t'-1}\right)\right)$$

10. Generate proofs for balance assertion:

$$\Pi_{\text{bal}} = \mathsf{RepProve}\left(pp_{\text{rep}}, H, \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \mathsf{Com}_3(0, 0, f, 0); \sum_{u=0}^{w-1} \mathcal{H}_{\text{val}'}(s_u, D) - \sum_{j=0}^{t-1} \mathcal{H}_{\text{val}}(\overline{k}_j)\right)$$

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

11. Define the following binding hash:

$$\mu = \mathcal{H}_{\text{bind}}(\mathsf{InCoins}, \mathsf{OutCoins}, \{S'_u, C'_u, T_u, (\Pi_{\text{par}})_u, \}_{u=0}^{w+w'-1}, \Pi_{\text{base}}, \Pi_{\text{rp}}, \Pi_{\text{bal}}, \Pi_{\text{bal}'}, \Pi_{\text{type}})$$

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

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

13. Output the tuple:

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

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 that asset type representations are restricted to τ_a bytes, and that identifier representations are restricted to τ_ι bytes.

Component	$\Delta\mathbb{G}$	$\Delta\mathbb{F}$	**Δ bytes**
a			$\tau_a t'$
ι			$\tau_\iota t'$
Total			$(\tau_a + \tau_\iota)t'$

Table 1: Mint transaction size change by component (t' generated generic-type coins)

Component	$\Delta\mathbb{G}$	$\Delta\mathbb{F}$	**Δ bytes**
Π_{rp}		2	
$\Pi_{\text{bal}'}$	1	3	
Π_{base}	2	2	
$\{\overline{r}_j\}$			$(\tau_a + \tau_\iota)t'$
Π_{type}	2	6	
Total	5	13	$(\tau_a + \tau_\iota)t'$

Table 2: Spend transaction size change by component (t' generated generic-type coins)

A mint transaction with t' generated generic-type coins increases by $(\tau_a+\tau_\iota)t'$ bytes. A spend transaction with t' generated generic-type coins increases by $576 + (\tau_a + \tau_\iota)t'$ bytes, if we assume a 32-byte representation for group and scalar elements.

5 Ownership of non-fungible tokens

5.1 Overview

A common requirement for the use of non-fungible tokens is that the entity controlling the token be able to prove ownership.

In the case of non-private transaction protocols, this is typically straightforward, as token ownership can be viewed on a public ledger, and the owner can sign arbitrary messages with a key associated to its address.

For private transaction protocols like Spats, ownership is more complex. This is in part because token transfers are not publicly associated with addresses on a ledger. Further, the spend status of tokens is generally unknown without external information; this means that even if a token owner proves that

a particular transaction transfers control of a token to it, a verifier cannot be certain that the token was not later transferred elsewhere.

We take an approach similar to the Spark authorizing proof in [12], which proves knowledge of the spend key corresponding to a coin's destination address in zero knowledge, and also asserts that a given tag binds uniquely and correctly to the coin. Unlike the authorizing proof, which operates on a re-randomized version of the coin's serial commitment to avoid linking directly to the coin, we will produce a proof that operates on the serial commitment directly.

By providing such a proof (and the tag), a token owner can assert ownership in zero knowledge. After verifying the proof against statement values from its own view of the ledger, the verifier can check its list of tags revealed in transactions. If it sees the token tag in this list, the ownership proof is invalid, as the prover has transferred the token; otherwise, it is valid.

The prover also provides a claimed asset type and identifier for the token, and proves in zero knowledge that these are correctly bound to the token via the corresponding coin's value commitment, and that the value bound to this commitment is $v = 1$.

We note an important downside to this approach to ownership assertion. Once the verifier knows the tag corresponding to the token, it can continue to watch the ledger for this tag. If it sees the tag in a later valid transaction, it knows the transaction corresponds to a transfer of the token. However, the verifier will be unable to determine the destination of the transfer without external information.

5.2 Proving system

For convenience, let $s = \mathcal{H}_{\text{ser}}(k) + \mathcal{H}_{Q_2}(s_1, i) + s_2$. Then recall from [12] that a coin serial commitment is of the form $S = \mathsf{Com}_2(s, r, 0)$. Here s_1 and s_2 are part of the owner's view key, and r is part of the owner's spend key. Further, recall that the coin's tag is the unique value T such that $U = sT + rG$ for globally-fixed generators U and G.

Consider the following relation, where we assume the public parameters used in Spats:

$$\{S, C, T, a, \iota; (s, r, m) : \\ S = \mathsf{Com}_2(s, r, 0), U = sT + rG, C = \mathsf{Com}_3(a, \iota, 1, m)\}$$

Note that a sigma protocol for this relation can be easily constructed as a conjunction of a modification of the Chaum-Pedersen-type proving system used for authorizing proofs in [12] (for the first two conditions in the relation) and a Schnorr-type sigma protocol (for the last relation). We omit the details here.

Such a sigma protocol may be made non-interactive using the Fiat-Shamir technique, and bound to an arbitrary message. This allows a prover to provide a context-dependent proof of ownership to the verifier.

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`.