Spats: User-defined Confidential Assets and Tokens

Aaron Feickert, Ph.D.

Monero Konferenco
23 June 2023

Acknowledgments and disclaimer

Spats is joint work with **Aram Jivanyan** of Firo and Yerevan State University.

The author is head of research at **Cypher Stack**, and research for Spats was conducted for **Firo**.

The material in this presentation is for informational use only, has not undergone formal external review, and does not constitute professional advice of any kind.

Overview

Protocols like **Spark** and **Seraphis** and **RingCT** already do neat tricks. They can:

- Dissociate recipient addresses from generated coin data
- Provide ambiguity about generated coin spend status
- Protect coin values and other metadata

But they only support a **single asset type**.

We want to support:

- Multiple asset types that have independent supply
- Protection of asset types and values in transactions
- Non-fungible tokens whose identifiers are protected

Commitments

How do we represent value in a safe way?

A coin with value v can be represented in part by a **commitment** that both hides and binds the value by including a random mask m.

We can write this using Pedersen commitments as $C = \text{Com}(v, m) = mG + vH$, where G and H are fixed group elements.

It's not possible to look at C and extract the value or mask.

We can show that a transaction balances in zero knowledge using commitment sums and differences via a Schnorr proof without revealing values or masks.

Multiple assets

But suppose we want to support **multiple asset types** with independent supply in a safe way.

Some asset types might be **fungible**, where coins are divisible with no particular differentiation.

Others might be **non-fungible**, where tokens have identifiers and must be differentiated.

In all cases, transactions should be as uniform as possible, and protect data and metadata until or unless the user chooses to reveal it.

Spats

Spats is a protocol intended as an extension to confidential transaction protocols like Spark, Seraphis, or RingCT to meet these goals.

The basic idea is to extend value commitments to include additional data: an **asset type** a and an **identifier** ι. This is easy with Pedersen commitments.

All assets, whether fungible or non-fungible, have a unique asset type.

Fungible assets don't use identifiers, but each token of a non-fungible type has a unique identifier within its type.

Commitments

Previously, a value commitment looked pretty simple:

$$\mathsf{Com}(v, m)$$

Now, it looks a little different, with the form $\mathsf{Com}(a, \iota, v, m)$:

Fungible	**Non-fungible**
$\mathsf{Com}(a, 0, v, m)$	$\mathsf{Com}(a, \iota, 1, m)$

Fungible asset types always set the identifier $\iota = 0$ (more on this later).

Non-fungible asset types always set the value $v = 1$ for atomicity.

Minting

Right now, a minted coin of value v can include a Schnorr proof that its commitment is to the expected value, but without revealing the secret mask m. This protects supply.

With Spats, we simply extend this.

A minted coin or token reveals its **type**, **identifier**, and **value** publicly. This is important for ensuring supply is correct across types according to whatever consensus rules the protocol chooses.

This still supports dissociating minted assets from recipient addresses.

Transactions

So how do transactions work? There's a Simple Rule:

All consumed and generated assets in a transaction must share the same asset type and identifier.

For fungible assets, a transaction might consume several coins of the same type, and generate several coins of the same type (including change). Remember, $\iota = 0$ here!

For non-fungible assets, the transaction will transfer only a single token; we can include zero-value dummy tokens for uniformity too. Remember, $v = 1$ here!

The catch: fees

If you think about it for a while, there's a problem.

How can you issue fees if the asset type is protected?

The answer is that you can't, so Spats makes a shitty tradeoff by adding a second Simple Rule:

Transactions include a "fee side" that consumes and generates coins of a fixed base asset type.

This means a transaction really does two things:

- Issues fees in a fixed base asset type
- Transfers coins or tokens of an arbitrary protected asset type

The slightly gory details

The two Simple Rules need to be enforced cryptographically. We can do this by adding a few zero-knowledge proofs (ZKPs) into the mix.

The Fee-Side ZKPs:

- All consumed and generated coins on the fee side have $a = \iota = 0$
- The fee side balances

The Asset-Side ZKPs:

- All consumed and generated coins/tokens on the asset side have the same a and ι
- The asset side balances

This asserts that minimal information is leaked. On the fee side, we need to know what asset is being used. On the asset side, we must **not** know what asset is being used.

The slightly more gory details

Fortunately, we know how to build these zero-knowledge proofs!

Each is a Schnorr-type proof, but with nice properties:

- Size is independent of the number of coins/tokens involved
- Verification is efficient and can be batched
- There are no icky trusted setups
- Security proofs are straightforward

They can be built using a power-of-challenge design.

We also need to modify range proofs to support vector commitments, but this is easy to do with Bulletproofs and Bulletproofs+ with only minor changes.

Efficiency

Spats transactions require modifications to existing proofs, as well as new proofs.

Component	**Δ (bytes)**
Range proof	64
Balance (asset)	128
Type (base)	128
Type (asset)	256
Total	576

(The total does not include encrypted encodings of asset types and identifiers, which depend on protocol rules and transaction structure.)

New proofs are speedy, adding minimal verification complexity.

Migration

Spats needs to play nicely with existing protocols like Spark or Seraphis or RingCT. You shouldn't need to migrate assets from one pool to another, since this is risky.

Fortunately, the design makes this a snap.

Because base assets require $a = \iota = 0$, this means **all existing coins** are automatically Spats base asset coins! That is, $\mathrm{Com}(0, 0, v, m) = \mathrm{Com}(v, m)$ holds between new and existing commitments.

Further, because the two Simple Rules are enforced using ZKPs on coin commitments, there is always a **single pool** of assets available for transactions, regardless of asset type!

This means that Spats supports clean migrations from existing protocols.

Ownership proofs

Support for non-fungible tokens implies something a bit unique: **ownership proofs**.

It's likely that you'll need to prove ownership of a token, but this really involves showing several things:

- The token's commitment is to a given type a and identifier ι (and value $v = 1$)
- You know the secret data required to transfer the token
- The proof isn't being replayed as a trick
- The token has not already been transferred elsewhere

The first two are straightforward using Schnorr-type proofs.

The third is easy using Fiat-Shamir message binding.

Ownership proofs

Showing that a token hasn't been transferred is much trickier.

One approach is to do the following:

- Reveal the **linking tag** associated to the token
- Prove that the linking tag is correct
- Assert that the linking tag does not appear in any subsequent transaction

Unfortunately, this sucks. Having the tag means the verifier can see if/when the token is later transferred, which is leaky.

A better way is to prove the tag doesn't appear on chain without revealing it, which requires more clever proof techniques and is work in progress.

The gist

Spats is a new design for user-defined confidential assets and tokens with nice properties:

- Fungible and non-fungible asset types are supported
- Transactions can be made uniform
- Asset types, identifiers, and values are protected
- Fees can be denominated in a fixed base type
- Users can prove token ownership (in a leaky way)
- Proving systems are straightforward
- Migration is clean
- All coins and tokens can share a single pool

Questions?

Preprint ia.cr/2022/288
github.com/AaronFeickert/spats

Slides github.com/AaronFeickert/monkon2023

Email aaron@cypherstack.com