# Seraphis: A Privacy-Preserving Transaction Protocol Abstraction (WIP)*

Draft v0.0.18†

koe‡ `ukoe@protonmail.com`

April 29, 2023

**Abstract**

Seraphis[1] is a privacy-focused transaction protocol abstraction for p2p electronic cash systems. Seraphis transaction outputs, called *e-notes* in this paper, are amount-transfer devices in the RingCT tradition, which record an 'amount' as a Pedersen commitment and an 'address with transfer-authority' as a specially-designed prime-order group point (similar to CryptoNote one-time addresses). Unlike previous protocols compatible with Confidential Transactions (CT), where enote membership, ownership, and unspentness proofs were highly integrated into one large proving structure (such as MLSAG or CLSAG in the case of standard RingCT), Seraphis separates membership proofs from ownership and unspentness proofs. This greatly simplifies the requirements for membership proofs, allowing more efficient proving structures to be used. Doing so also allows a linking tag (a.k.a. key image) construction with a number of favorable properties. Most notably, it is possible to have a Seraphis-compatible addressing scheme which permits wallets with three tiers of permissions (view received amounts, full balance recovery, full balance recovery with spend authority). The second permission tier is unique to Seraphis among protocols in the CryptoNote tradition.[2]

## 1 Introduction

A p2p (peer-to-peer) electronic cash system is a monetary system where the entire supply of currency exists as a set of digital records describing money amounts and their owners. Those records can be stored by any person, and transactions (attempts to transfer money to new owners) are mediated by a network of *peers* (usually called *nodes*). Such systems are typically designed with the goal that no participant in the system has the power to easily censor transactions, re-spend funds that have been spent before, or increase the total money supply at will.

To achieve that goal, it is necessary for such systems to be decentralized. The peers who mediate transactions (checking their validity with respect to the existing state of the money supply, and

*__License__: Seraphis is released into the public domain.

†This is just a draft, so it may not always be available wherever it is currently hosted.

‡Author 'koe' worked on this document partly as an employee of MobileCoin, Inc.

[1] The name 'Seraphis' is derived from Serapis, a Graeco-Egyptian syncretistic deity. Syncretism is the combination/reconciliation of different ideas/ways of thinking, similar to how Seraphis is a protocol that brings together many ideas and permits a variety of proving systems.

[2] An example of such a scheme is discussed in this paper's companion paper [citation TBD].

deciding which of N conflicting transactions to accept) do not necessarily trust each other. It is therefore beneficial to have a common rule-set and format for constructing transactions, so that any peer can validate any transaction and reach consensus with other peers/nodes about changes to the recorded monetary state. The transaction rule-set and format used by any given p2p electronic cash system is called its *transaction protocol*.

Seraphis is a transaction protocol *abstraction*, which means it defines the rule-set that a transaction protocol must satisfy (and the corresponding security model) without specifying any concrete proving systems or semantic constraints.

## 1.1 Monetary state

Most modern p2p electronic cash systems are 'cryptocurrencies' in the tradition of Bitcoin [17]. In Bitcoin, each (archival) node maintains a full copy of all mutations to the monetary state that led from Bitcoin's inception up to the current moment (called a *ledger*).

The monetary state of a cryptocurrency arises from all the 'money creation' and 'amount transfer' events that have occurred since the currency was born. Almost universally, those events are defined in the *transaction output* model (henceforth called the *enote* model). An enote is a small message containing an 'amount' of money, an 'ownership address' that encapsulates the authority to spend the enote, and an optional arbitrary memo.

- **Money creation event**: Create a new 'coinbase enote', which increases the total supply of money.
- **Money transfer event (transaction)**: Consume one or more previously unspent enotes to transfer the amounts they record to one or more new enotes (see Section 3).

The 'current monetary state' of a cryptocurrency is therefore the set of spent and unspent enotes recorded in the ledger.

## 1.2 Transaction protocols

Transaction protocols must always codify a basic set of rules.

- **Membership**: enotes spent by a transaction must already exist in the ledger.
- **Unspentness**: enotes spent by a transaction must be unspent prior to that transaction.
- **Ownership**: A transaction author must have the authority to spend the transaction's input enotes.
- **Amount balance**: The total amount in enotes spent by a transaction must equal the total amount in new enotes created (plus a transaction fee, usually).

A very simple transaction protocol could implement those rules like this:

- **Membership**: Reference existing enotes with their indices in the ledger. Transaction validators can look up those enotes directly.
- **Unspentness**: When an enote is spent by a transaction, set a bit flag next to that enote in the ledger. Reject transactions that reference spent enotes.
- **Ownership**: Define ownership with public key cryptography. Let each enote's address record a public key specified in advance by the intended owner. To spend an enote, its owner must create a cryptographic signature with the address key and add the signature to their transaction.[3]
- **Amount balance**: Record enote amounts in clear text and use simple sums to check that input amounts equal output amounts (disallowing integer overflow).

An unfortunate consequence of cryptocurrencies being decentralized is that the ledger is 'public'. This means all enotes and transaction events are public knowledge. If amounts are in cleartext, addresses can be reused, and enotes to be spent are referenced directly, then observers can discern many details about users' finances.

A lack of privacy in the design of a transaction protocol has two main drawbacks, which lead to a competitive disadvantage versus protocols that include elements of privacy (all else being equal).

1. Privacy is valuable to real people. Typically, it is preferable to choose when others obtain information about you than for that information to be available automatically.
2. Fungibility and privacy go hand-in-hand. If observers have detailed information about the ledger, then it is possible for some enotes to be more valuable than other enotes just based on differences in who owns them or where they originated (i.e. the history/transaction-graph that led to those enotes being created), even if the amounts they contain are the same.

We believe an ideal transaction protocol should satisfy the following informal privacy matrix.

- **Recipients**
  - **Know**: Amounts received, and when they were received.
  - **Don't know**: Who sent them any given amount.
- **Senders**
  - **Know**: Amounts sent, when they were sent, and who they were sent to.[4]
  - **Don't know**: If an amount sent to someone else has been spent.

[3] The 'crypto' in 'cryptocurrency' refers to the use of cryptography to control enote ownership.

[4] A transaction author inherently knows who they send enotes to. This information does not need to be stored in the ledger to satisfy this privacy matrix.

- **Observers**
  - **Know**: The number of inputs/outputs in each transaction, fees paid by each transaction, and when each transaction was added to the ledger.
  - **Don't know**: The amounts involved in any transaction (other than fees), the relationships between any transactions, or the amounts owned by any user.

Most of these requirements are relatively easily met by CryptoNote-style addressing and linking tags (a.k.a. key images) [27] and Confidential Transactions [15], which were first combined in the protocol RingCT [20]. There are two areas of weakness in existing protocols based off RingCT.

- Observers can, to some extent, discern when a transaction was constructed, which is stronger information than simply 'when a transaction was added to the ledger'. The biggest culprit for this lies in transaction fees, which are often a function of real-world time. The problem of transaction timing is out of scope for this paper.

- Observers can, to some extent, discern relationships between transactions. Membership proofs defined in RingCT (and those used in related protocols like Triptych [19], Lelantus-Spark [11], Omniring [13], and RingCT3.0 [28]) have 'anonymity sets'. A transaction author proves that each enote spent by their transaction exists in a small set of enotes, and further proves that that small set is a subset of enotes that exist in the ledger. Unfortunately, if observers see that one transaction references an enote created by another transaction, then they know there is more likely to be a relationship between those transactions than if no such connection exists. This probabilistic knowledge is stronger than the 'pure/ideal' case where a membership proof shows that an enote exists in the ledger without giving any hints about which one it might be.

Increasing the anonymity set size of membership proofs naturally reduces how much information observers can glean from transactions. However, combining membership proofs with ownership and unspentness proofs in one large proving structure, a ubiquitous pattern in previous RingCT-inspired protocols, has led to some challenges around increasing that size.

Most importantly, proving structures suitable for both membership proofs and ownership/unspentness proofs place constraints on the construction of linking tags, which are the core element of unspentness proofs in privacy-focused transaction protocols. A linking tag is an 'image' of an enote's address produced when trying to spend the enote. If a transaction's input proofs contain a linking tag that already exists in the ledger, then the transaction is trying to re-spend an enote that has already been spent.

As one example of those constraints, the transaction protocol Triptych [19], which allows a proving structure an order of magnitude more efficient than those allowed by standard RingCT, features a linking tag construction that looks like $\tilde{K} = (1/k^o) * U$. Here $k^o$ is the private key of the address that owns a given enote, and $U$ is a generator of a prime-order cyclic group. By inverting $k^o$ to create linking tags, it becomes relatively more difficult to design a multisignature scheme where multiple individuals collaborate to sign transactions, compared to a construction that is linear in $k^o$. This is because a linear construction would allow a simple sum of components provided by signature participants (such as in [21]), whereas an inverted shared key requires a non-trivial signing protocol (e.g. involving Paillier encryption [24]).

## 1.3 Our contribution

The main innovation of Seraphis compared to its predecessors is separating ownership and unspentness proofs from membership proofs. Seraphis membership proofs only say (more-or-less) that a commitment to an enote corresponds with an enote in some reference set. The prover then operates on the enote commitment to demonstrate ownership and unspentness and to connect it with the proof that amounts balance.

This separation allows the definition of linking tags to be fairly open-ended. We designed a linking tag construction with the following (informal) properties.

1. **Multisig compatibility**: Linking tags are composed of two pieces of private key material associated with an enote's address. One piece is transformed linearly from the address to the tag, while the other piece is inverted (see Section 3.5).

   A multisignature group may share knowledge of the inverted component and split knowledge of the linear component. This facilities straightforward multisignature schemes for ownership and unspentness proofs.

2. **Address scheme flexibility**: The linking tag construction makes it is possible to implement user addressing schemes with multiple tiers of permissions (see this paper's companion paper [citation TBD]). Most significantly, it is possible to have a 'view entire balance' tier that is fully separated from the 'spend authority' tier.

In Appendix B we introduce a more-restrictive membership proof model layered on the primary model in this paper. We call it the 'squashed enote' model. Concrete proving structures in that model are non-trivially more efficient than structures in the plain model, allowing relatively larger anonymity set sizes as a function of proof complexity and size compared to structures in the plain model, when comparing structures based on the same proving systems (see an efficiency analysis in Section 4).

Appendix A introduces Seraphis composition proofs, which are a Seraphis-compatible ownership/unspentness proof. Appendix C introduces Grootle proofs, a Seraphis-compatible membership proof for the squashed enote model.

## 1.4 Acknowledgements

We would like to thank Aaron Feickert and Aram Jivanyan (who wrote Lelantus-Spark [11], a transaction protocol very similar to Seraphis) for critical discussions that led to Seraphis's linking tag construction, and Nikolas Krätzschmar for pointing out a flaw in an earlier version of that construction.

## 2 Preliminaries

### 2.1 Public parameters

[5]Let $\mathbb{G}$ be a cyclic group of prime order $l > 3$ in which the discrete logarithm problem is hard and the decisional and inverse decisional Diffie-Hellman assumptions hold, and let $\mathbb{Z}_l$ be its scalar field. Let $\mathcal{H} : [0, 1]^* \to \mathbb{Z}_l$ be a cryptographic hash function. We add a subscript to $\mathcal{H}$, such as $\mathcal{H}_1$, in lieu of domain-separating the hash function explicitly; any domain-separation method may be used in practice (e.g. an ASCII string corresponding to a domain-separated use case, such as $\mathcal{H}$("*sender_receiver_secret*" || [hash input])).

Let each of the following sets contain generators of $\mathbb{G}$, where each generator's discrete logarithm with respect to other generators in the same set is unknown (there may be intersections between sets): $\{G_0, G_1, ..., G_n\}$, $\{H_0, H_1, ..., H_m\}$, and $\{J\}$ (for arbitrary integers $n$, $m$). Note that all such generators may be produced using public randomness. For example, the use of a hash function with domain separation may be appropriate. All public parameters are assumed to comprise a global reference string known to all players. For readability, we generally exclude explicit reference to public parameters in algorithm definitions and Fiat-Shamir transcript hashes.

### 2.2 Notation

- We use additive notation for group operations on $\mathbb{G}$. This means, for example, that the binary group operation between $G$ and $H$ is denoted $G + H$.
- This paper contains no exponentiation outside Appendix C. Superscripts, such as the $o$ in $k^o$, are merely for descriptive purposes and have no mathematical significance.
- For group element $P$ and scalar $x \in \mathbb{Z}_l$, $xP$ and $x * P$ both indicate scalar multiplication. The use of asterisks ($*$) in some places but not others is meant to aid visual clarity where appropriate (usually when multiplying by a parenthesized scalar or by a scalar that has a superscript).
- Modular multiplicative inverse group operations use the notation $(1/x) * P$.
- Tuples are indicated with brackets, e.g. $[A, B, C]$. To avoid confusion, we always explicitly refer to tuples as tuples wherever they appear (e.g. 'the tuple $[A, B, C]$').

## 3 Seraphis

In this section we discuss the various components of Seraphis, with security theorems and proofs introduced where appropriate [[[TODO]]]. Seraphis is an *abstract* protocol, so various implementation details are left undefined. An implementer needs to define the generators $G_0, G_1, G_2, H_0, H_1, J$,

[5] This section was copied mostly verbatim from the Triptych preprint [19].

specify a membership proof structure (e.g. Appendix C), specify an ownership/unspentness proof structure (e.g. Appendix A), define how to check that transaction amounts balance, define an address scheme, and define an information recovery scheme. See this paper's companion paper [citation TBD] for a complete instantiation of Seraphis.

## 3.1 Transaction overview

For context, we outline the content of a transaction here.

- **Inputs**: A transaction spends old enotes.
  - **Enote images**: Representations of the enotes spent by this transaction, including their linking tags (Section 3.3).
  - **Membership proofs**: Proof structures demonstrating that each enote image is constructed properly from a real enote in the ledger (Section 3.4).
  - **Ownership and unspentness proofs**: Proof structures that use enote images to demonstrate ownership and unspentness for each spent enote (Section 3.5).
- **Outputs**: A transaction creates new enotes.
  - **Enotes**: New enotes (Section 3.2). The total amount they contain equals the total amount in spent enotes (Section 3.6).
  - **Range proofs**: Proof structures demonstrating that amount commitments in new enotes are legitimate (part of the Confidential Transactions technique) (Section 3.6).
- **Balance proof**: A proof that the sum of input amounts equals the sum of output amounts (Section 3.6).
- **Miscellaneous**: Miscellaneous other data included with a transaction, such as a transaction fee.

## 3.2 Enotes

Seraphis enotes are composed of an amount commitment, an address, and a memo.

- **Amount commitment**: A Pedersen commitment $C$ (with blinding factor $x$) to the amount $a$ contained in the enote [22, 15].
$$C = xH_0 + aH_1$$
- **Address**: A public key $K^o$ composed of three generators $G_0$, $G_1$, and $G_2$, and three corresponding private keys $k_0^o$, $k_1^o$, and $k_2^o$. The enote's owner must prove knowledge of those private keys if they want to transfer the amount $a$ to new enotes (it is allowed for $k_0^o$ to equal zero).
$$K^o = k_0^o * G_0 + k_1^o * G_1 + k_2^o * G_2$$

- **Memo**: An arbitrary memo field. This usually includes information that helps the enote's owner identify that they own it, learn the private keys $k_i^o$, and reconstruct the amount commitment. See Sections 3.9 and **??**.

### 3.3 Enote images

An enote image is a representation of an enote.

- **Masked commitment**: The enote's commitment with an additional masking factor.

$$C' = t_c H_0 + C$$
$$C' = (t_c + x) * H_0 + a H_1$$
$$C' = v_c H_0 + a H_1$$

- **Masked address**: The enote's address with a masking factor.

$$K' = t_k G_0 + K^o$$
$$K' = (t_k + k_0^o) * G_0 + k_1^o * G_1 + k_2^o * G_2$$

- **Linking tag**: The enote's linking tag.

$$\tilde{K} = (k_2^o / k_1^o) * J$$

The blinding factors $t_c$ and $t_k$ must be statistically independent and selected at random from a uniform distribution. [[[formalize better?]]] Note that observers who don't know $t_c$ and $t_k$ cannot look at an enote image and discern what enote it was created from.

We describe how a transaction author can prove that enote image addresses and commitments are constructed properly from real enotes in Section 3.4, and further prove that linking tags are constructed properly from enote image masked addresses in Section 3.5.

#### 3.3.1 Sender-receiver anonymity

If a person spends an enote, they should expect that the person who originally sent them that enote will not know it is spent.

If $t_c$ and $t_k$ are randomly selected and unknown to the original sender, then the sender cannot detect the original enote by inspecting the enote image's commitment and address.

We further argue in Sections 3.4, 3.5, and 3.8 that input proofs and linking tags will not break sender-receiver anonymity.

#### 3.3.2 Linking tags

Linking tags are uniquely defined by the private keys $k_1^o$ and $k_2^o$ (as proven in Section 3.5). This means for a user to create two distinct linking tags from the same address, they must be able to solve one of the DLPs between generators $G_0$, $G_1$, and $G_2$, which we assume to be a hard problem [[[elaborate this proof?]]].

Since linking tags are assumed to be unique for each unique address $K^o$, they can be used to prove unspentness. If a transaction contains an enote image with a linking tag that has appeared in the ledger, then that transaction is invalid. The word 'linking' refers to the ability of observers to link attempts to spend the same enote.

Note that if two enotes have the same address $K^o$, then only *one* of them can be spent, hence the superscript $o$. Going along with the CryptoNote tradition, $K^o$ can be referred to as a *one-time address*. The construction of one-time addresses is further discussed in Section 3.8.

## 3.4 Membership proofs

Every input to a transaction must have a membership proof. The proof must demonstrate that the input's enote image was built from an enote that exists in the ledger.

A proving system/structure is only eligible to be used as a Seraphis membership proof if it can satisfy the following abstract model. [[[formalize better?]]]

1. Let $\mathbb{S}$ represent a set of tuples $[K_i, C_i]$, where
$$K_i = z_i G_0 + s_{i,1} G_1 + s_{i,2} G_2 + ... + s_{i,n} G_n$$
$$C_i = x_i H_0 + a_{i,1} H_1 + a_{i,2} H_2 + ... + a_{i,m} H_m$$
2. Let $\tilde{S}$ represent a tuple $[K', C']$, where
$$K' = z' G_0 + s'_1 G_1 + s'_2 G_2 + ... + s'_n G_n$$
$$C' = x' H_0 + a'_1 H_1 + a'_2 H_2 + ... + a'_m H_m$$
3. The proving system must be able to demonstrate that, within a security parameter $k$, $\tilde{S}$ corresponds to some $S_\pi \in \mathbb{S}$, where $\pi$ is unknown to the verifier, such that:
   (a) $s'_p == s_{\pi,p}$ for $p \in 1, ..., n$
   (b) $a'_q == a_{\pi,q}$ for $q \in 1, ..., m$
4. The proving system should be considered unusable if, given a proof $\sigma$ that $\tilde{S}$ corresponds to some $S_\pi$ in the set $\mathbb{S}$, an observer can guess the index $\pi$ with probability $> 1/|\mathbb{S}'| + \epsilon(k)$, where $\mathbb{S}' = \mathbb{S} \backslash \mathbb{S}_O$ and $S_\pi \in \mathbb{S}'$, $\mathbb{S}_O$ are tuples the observer knows can't have been subjects of the proof (in the context of Seraphis, if tuples $S$ represent enotes, then for example he owns the enotes in $\mathbb{S}_O$), and the observer has no special knowledge about the elements in $\mathbb{S}'$ (however, he can know $z_i G_0$, $x_i$, and $a_{i,q}$ for all $S_i \in \mathbb{S}'$).[6]

[6] In practice, $\pi$ can often be guessed with probability at least marginally above $1/|\mathbb{S}'|$. This is because the circumstances around when enotes are recorded in the ledger are often observable. Things like timing information, patterns of behavior, IP addresses of transaction submitters, transaction fees, etc., can all form the basis of heuristics for analyzing the true member referenced by a membership proof. See the discussions in [16] and [23] for example.

In the context of Seraphis, we straightforwardly construct tuples $S$ directly from enotes, which can be referenced with simple ledger indices for verifiers to find (in a naive implementation),[7] and tuples $\tilde{S}$ from enote images. Readers will note that a membership proof says nothing about how $K$ and $C$ are constructed (i.e. the values of $G_0, ..., H_0, ...,$ etc.). In future sections we will add more constraints to guarantee that enotes and enote images found in transactions have the expected forms (within a security parameter).

A trivial proof that satisfies the above model would be a pair of signatures on commitments to zero $K' - K = t_k G_0$ and $C' - C = t_c H_0$, given a reference set $\mathbb{S}$ that contains only one tuple $[K, C]$. More interesting solutions include a CSAG (a CLSAG [9] without linking) on a ring of such commitment to zero pairs (assuming $G_0 == H_0$), a Groth/Bootle one-of-many proof [10, 2, 19, 11] on a collection of those pairs, or a Groth/Bootle one-of-many proof applied to the squashed enote model (see Appendices B and C).

## 3.5 Ownership and unspentness proofs

Alongside each membership proof must be an ownership and unspentness proof. In Seraphis, 'unspentness' is checked by looking for linking tag duplicates in the ledger. However, it is necessary to prove that linking tags are properly constructed. This is done simultaneously with the ownership proof to ensure the linking tag is derived from the relevant enote address.

A proof structure is only eligible to be used for Seraphis ownership/unspentness proofs if it can accomplish the following.

1. Assume there is a group point $K = xG_0 + yG_1 + zG_2$.
2. Demonstrate knowledge of values $x, y, z$ such that $K = xG_0 + yG_1 + zG_2$ and $y \neq 0, z/y \neq 0$.
3. Demonstrate that a key $\tilde{K}$ satisfies $\tilde{K} == (z/y) * J$.
4. A prover who knows $x$ and/or $y$, but not $z$, must be unable to create a proof that contains $\tilde{K} = (z/y) * J$ (within a security factor).

We integrate that 'composition proof' model into Seraphis in the following way (see Appendix A for an example proof structure that satisfies the model).

Suppose a membership proof $\sigma_{mp}$ shows that $\tilde{S}$ corresponds to some $S_\pi$ in the set $\mathbb{S}$. Then suppose the key $K' = z'G_0 + s'_1 G_1 + s'_2 G_2 + ...$ from $\tilde{S}$ is passed as input to the above proof system, and a valid proof is created. Observe the following.

- For the composition proof to succeed, it must be the case that all $s'_p == 0$ for $p \geq 3$. This implies the key $K_\pi$ from $S_\pi$ (which $K'$ is based on) has the following form: $K_\pi = z_\pi G_0 + s_{\pi,1} G_1 + s_{\pi,2} G_2$. Moreover, the prover must know $z_\pi, s_{\pi,1}, s_{\pi,2}$.

[7] If the size of $\mathbb{S}$ is small, then it may be practical to reference enotes with simple indices. As $\mathbb{S}$ gets large, more sophisticated data-compression techniques are advisable to minimize transaction sizes. For example, deterministically selecting members of the anonymity set using public entropy and a hash function [4].

- It must be the case that $\tilde{K} = (s_{\pi,2}/s_{\pi,1}) * J$.
- The verifier will not be able to discern which $S_i$ in the set $\mathbb{S}$ corresponds with $K'$.

Now suppose a transaction spends an enote. Let them set $S_\pi = [K^o_\pi, C_\pi]$ using the enote's one-time address and amount commitment, give $S_\pi$ a membership proof $\sigma_{mp}$, and give the resulting image $S' = [K', C']$ a composition proof $\sigma_{cp}$. With this proof pair, the verifier can be confident that the transaction author owns an enote in the set $\mathbb{S}$ (i.e. they know the keys $z_\pi = k^o_o$, $s_{\pi,1} = k^o_1$, and $s_{\pi,2} = k^o_2$ for some unknown index $\pi$), and that the linking tag $\tilde{K} = (k^o_2/k^o_1) * J$ in the composition proof is valid and can be used to check if the enote at index $\pi$ is unspent. Importantly, linking tag $\tilde{K}$ is independent of $z_\pi$ and $z'$ and hence is unaffected by the transformation from enote to enote image.

The transaction's enote image structure records $\tilde{S} = [K', C']$ and $\tilde{K}$ for observers/verifiers to reference (recall Section 3.3).

## 3.6 Amount balance proofs

In accordance with the Confidential Transactions technique [15], Seraphis amounts are recorded as *Pedersen commitments*, which hide the amounts involved from observers (they have the 'perfectly hiding' property). Even though observers cannot see transaction amounts directly, they should still be able to verify that the sum of input amounts always equals the sum of output amounts in every transaction.

First note that, thanks to our membership proof model (Section 3.4), the commitment $C'$ in an enote image will contain the same values $a_{\pi,1}, ..., a_{\pi,m}$ as the commitment $C$ in the enote being spent. In Section 3.6.1 we will prove that enote commitments have the form $C = xG_0 + aG_1$ as expected (i.e. prove that $a_{\pi,q} == 0$ for $q \geq 2$), and hence the amount $a$ in $C' = v_c G_0 + aG_1$ equals the amount in the original commitment.

Pedersen commitments have the 'homomorphic' property, which means, for example, that if $P_1 = p_1 G$ and $P_2 = p_2 G$, then $P_1 + P_2 == (p_1 + p_2) * G$. We should therefore expect that if we sum together enote image commitments (inputs) and sum together new enote commitments (outputs), then $\sum C'_j - \sum C_t$ will contain no $H_1$ component only if the sum of input amounts equals the sum of output amounts.

An implementation of Seraphis must do the following.

1. Demonstrate knowledge of the 'remainder' $p_r$ in the commitment to zero $\sum C'_j - \sum C_t = p_r H_0$. It is acceptable if $p_r == 0$, such that $\sum C'_j == \sum C_t$ can be checked directly.

### 3.6.1 Range proofs

Since Pedersen commitments are elements of a cyclic group, it is conceivable that the sum of amounts modulo the group order is less than the absolute sum of amounts.[8] To properly convince observers that transaction amounts balance, transaction authors must provide a 'range proof' for each new enote's commitment.

A range proof must demonstrate the following for a given commitment $C = xH_0 + aH_1$.[9][[[formalize this better?]]]

- Prove knowledge of $x$ and $a$ such that $C = xH_0 + aH_1$.
- Show that the value $a$ is in the range $[0, 2^z - 1]$.

The maximum number of elements $n$ that can be summed together must be $n < l/(2^z - 1)$, otherwise range proofing those elements is pointless. Typically $z = 64$ and $l \approx 2^{252}$ (e.g. in Ed25519 [1]), so $n$ can be as large as $\approx 2^{192}$. However, usually $n << 2^{64}$ for practical reasons.

In a real system based on Seraphis, only new enote commitments need range proofs, not enote image commitments (see Appendix B for an exception). If all new enotes added to the ledger are range proofed, and membership proofs only reference enotes from the ledger, then enote image commitments are guaranteed (within a security factor) to contain legitimate amounts.

Finally, note that range proofing a commitment 'locks in' the structure $C = xH_0 + aH_1$. This means in any membership proof that acts on only range proofed commitments, it must be the case that $a_{\pi,q} == 0$ for $q \geq 2$.

## 3.7 Transaction teleology

Since Seraphis is a transaction protocol, there is a 'teleological' dimension to transaction contents. In other words, a transaction author acts with 'purpose' or 'intent' when writing a transaction. It is important to embed those intentions in transactions, so a transaction only contains statements intended by transaction authors, and doesn't reflect the intentions of arbitrary third parties.

There are three primary 'intentions' that a Seraphis transaction must capture.

1. If an enote's owner authorizes transfer of funds out of that enote, they must commit to the full set of destinations for those funds (i.e. the full set of new enotes created by the transaction), and the full set of messages (i.e. memos) attached to that transfer.

   This way an enote's funds cannot be transferred to new enotes without the enote owner's full consent, and no message can be affiliated with an enote owner (i.e. supposedly endorsed by that owner) without their explicit approval.

[8] For example, in a cyclic group of order 11, $7 + 7 \equiv 3 \pmod{11}$. If the input amount is 3, then the output amount could be 14!

[9] At this time, Bulletproofs+ by Chung et. al [5] (based on Bulletproofs by Bünz et. al [3]) is thought to be the most efficient zero-knowledge proving structure for range proofs, without a trusted setup.

2. Only the owners of enotes spent by a transaction, or their proxies, should be able to decide the transaction's contents.
3. A transaction recorded in the ledger should be unmalleable (i.e. 'permanent').

In practice, these 'intentions' can be implemented with the following general rules.

1. **Transcript dependencies**: Assume that ownership/unspentness proofs, membership proofs, and balance proofs are implemented with Sigma protocols using the Fiat-Shamir transform [8]. Each of a transaction's proofs' Fiat-Shamir challenges (transcript hashes) should depend on...

   (a) **Ownership/unspentness proof**: The relevant enote image, the full set of output enotes, and all the memos found in the transaction.

   (b) **Membership proof**: The relevant enote image and the set of enotes referenced by the proof.

   (c) **Balance proof**: The full set of input enote images, the full set of new output enotes, and the transaction fee.

   Any miscellaneous transaction data not mentioned above (e.g. transaction version numbers, transaction fee, etc.) should be included in all of the transaction's ownership/unspentness proof transcript hashes.

2. After a transaction's proofs have been constructed, any change to the byte serialization of the transaction (without replacing any of the proofs) should invalidate at least one of the proofs.
3. After a transaction has been added to the ledger, any change to its byte serialization should cause a change in the transaction hash (i.e. canonical hash of all transaction content). In particular, it should be impossible to switch out a transaction proof for a different one without changing the transaction hash.

## 3.8 Enote address model

An enote is created by one person (a transaction author) for another (the recipient of funds). To spend an enote, the recipient must know private keys $k^o_i$ in the address $K^o = k^o_0 * G_0 + k^o_1 * G_1 + k^o_2 * G_2$. However, it isn't feasible for the recipient to define those values in advance, for example by requesting that the transaction author place a pre-defined public key in the enote address slot.

The reason for this is only one enote with a given pair $[k^o_1, k^o_2]$ can ever be spent, since linking tags have the form $\tilde{K} = (k^o_2/k^o_1) * J$. Recipients could randomly generate a new address $K = k_0 * G_0 + k_1 G_1 + k_2 G_2$ for each enote they want to receive, but that is very inefficient and impractical.

Instead, we recommend the following enote address model inspired by CryptoNote addresses [27].

1. Let each recipient have a *spend key* $K^s$ for spending enotes:
$$K^s = k_{0,recipient} G_0 + k_{1,recipient} G_1 + k_{2,recipient} G_2$$

2. When sending an enote, the sender generates random scalars $k_{0,sender}, k_{1,sender}, k_{2,sender} \in_R \mathbb{Z}_l$.

3. The sender defines the enote's one-time address based on the recipient's spend key:
$K^o = k_{0,sender} G_0 + k_{1,sender} G_1 + k_{2,sender} G_2 + K^s$
$K^o = (k_{0,sender} + k_{0,recipient}) * G_0 + (k_{1,sender} + k_{1,recipient}) * G_1 + (k_{2,sender} + k_{2,recipient}) * G_2$

Enote recipients must learn private keys $k_{i,sender}$ in order to spend their enotes. We discuss that topic in Section 3.9.

**Comments**

- Transaction authors cannot spend enotes they created unless they know private keys $k_{i,recipient}$ in addition to $k_{i,sender}$. They cannot create linking tags unless they know $k_{1,sender}$, $k_{1,recipient}$ and $(k_{2,sender} + k_{2,recipient})J$.

- Observers will not be able to associate a one-time address $K^o$ with a spend key $K^s$ unless they know the terms $k_{i,sender} G_i$. We assume $k_{i,sender}$ are randomly generated every time an enote is created, so there will be no 'key re-use' patterns that allow observers to derive $K^s$ from $K^o$.

- Linking tags will have the form $((k_{2,sender} + k_{2,recipient})/(k_{1,sender} + k_{1,recipient})) * J$. Even if a transaction author sends many enotes to the same spend key $K^s$, and all of those enotes are spent, the author cannot use linear algebra on the resulting linking tags to associate those linking tags with the enotes they sent out. This avoids the 'linearity' problem for fixed-base-point linking tag constructions noted by the CryptoNote whitepaper [27], which breaks sender-receiver anonymity. [[[formalize better? proof?]]]

- In the context of transaction protocols, multisignature schemes allow a group of N users to 'co-own' enotes (see [21] for example). Only a collaborating subgroup of participants of size M (M <= N) may spend any enote. This is called 'M-of-N multisig'.

  Ideally, multisig schemes should allow all participants to view the group's balance (amount of money currently owned). In Seraphis, this means being able to identify all owned enotes (see Section 3.9) and recreate all their linking tags to check in the ledger if they have been spent.

  Conveniently, the distinction between $k_{1,recipient}$ and $k_{2,recipient}$ makes our addressing model very 'multisig-friendly'. If all multisig participants have full knowledge of $k_{1,recipient}$ (and $k_{1,sender}, k_{2,sender}, k_{2,recipient}J$), then they can easily recompute all linking tags to identify spent enotes, and can identify newly acquired enotes and recover their amounts with a method such as the one described in the companion paper [citation TBD]. Meanwhile,

$k_{2,recipient}$ can be divided among participants so a collaborating subgroup of size M is required to make a composition proof (Section 3.5).

With, for example, the proof structure in Appendix A, proving knowledge of $k_{2,recipient}$ only requires a discrete-log proof between points $\tilde{K}$ and $J$, where $\tilde{K} = (z/y) * J = ((k_{2,sender} + k_{2,recipient})/(k_{1,sender} + k_{1,recipient})) * J$. For multisig, this can be achieved with a simple thresholded Schnorr signature (e.g. [18, 12, 6]), assuming $k_{1,sender}, k_{2,sender}$ and $k_{1,recipient}$ are known by all M co-signers.

## 3.9 Information recovery

Enote owners need to discover the enotes they own in the ledger, read the amounts in those enotes, learn the amount commitment blinding factors in order to create balance proofs for new transactions, and acquire the sender keys $k_{i,sender}$ so they can construct linking tags.

A Seraphis-compatible information-recovery scheme must satisfy the following requirements.

1. Any user who has the private keys $k_{i,recipient}$, or a proxy of that user, should be able to acquire the following 'nominal' secrets corresponding to each enote in the ledger, using information stored in the ledger.
   - **Sender secrets**: $k^{nom}_{i,sender}$
   - **Amount**: $a^{nom}$
   - **Amount commitment blinding factor**: $x^{nom}$
2. The user is considered an enote's owner if the following two tests succeed (using the values $K^o$ and $C$ recorded in the enote).
$$K^o \stackrel{?}{=} (k^{nom}_{i,sender} + k_{i,recipient}) * G_i$$
$$C \stackrel{?}{=} x^{nom} H_0 + a^{nom} H_1$$
3. An observer/user who knows none of the keys in the tuple $[k_{i,sender}, k_{i,recipient}]$ must always fail the first equality test (within a security factor). If they don't know the true value of $x$, then they must always fail the second equality test, and must be unable to guess the true value of $a$ with probability of success better than random chance.

   Furthermore, any observer who fails the first equality test must not be able to guess with probability better than random chance that $K^o$ is constructed from a composition between $k^{nom}_{i,sender} G_i$ and $k_{i,recipient} G_i$. In other words, even though it is trivial to compute $k^{nom}_{i,sender} G_i = K^o - k_{i,recipient} G_i$ (if the spend key $K^s = k_{i,recipient} G_i$ is public information), the observer cannot guess with probability better than random chance that the discrete log composition of $k^{nom}_{i,sender} G_i$ with respect to $G_0, G_1, G_2$ is known to the user who has spend key $K^s$. [[[This point feels a bit shaky - I want to say that anonymity can't be broken by observers, i.e. that observers can't identify the recipient of an enote nor the amount involved]]]

We present a concrete approach to information recovery in this paper's companion paper [citation TBD] based on a Diffie-Hellman shared secret between sender and receiver.

An implementation of this section and the Seraphis address model is 'unverifiable'. It isn't possible for a transaction verifier to know if a transaction's author has in fact followed these recommendations. Instead, these sections can be enforced by user choice. If, in practice, a user only creates transactions using a transaction-builder implementation that satisfies all Seraphis requirements, then the privacy model we laid out will be achieved for that user (with some caveats around 'fingerprinting' if there are multiple implementations with different semantic conventions).

## 4 Efficiency

In this section we discuss performance results (transaction sizes and verification costs) from a test of several proof-of-concept implementations of Seraphis variants, along with mock-ups of prominent alternatives (RingCT [20] and Triptych [19]).[10] The results are not comprehensive (see [25] for more in-depth results), but should be sufficient to give readers a sense of the trade-offs between different protocols and design choices.

### 4.1 Test details

Here is a brief overview of the transaction protocols tested, which were implemented on Ed25519 [1] using a fork of the Monero codebase [26]. All protocols use Confidential Transactions [15], so all mock-ups essentially had the same balance proof approach (with slight variations). The 'Concise-Grootle' proof is simply a Triptych [19] proof with the linking tag components removed and adapted to our membership proof model. 'Plain-Grootle' is very similar to 'Concise-Grootle', but uses an expanded structure.[11] A Grootle proof is the degenerate form of the Concise-Grootle proof when there is only one proof layer (one proof key).

**Table 4-1:** Transaction Protocols Overview

| **Protocol** | Membership Proof | Ownership Proof | Balance Proof |
|---|---|---|---|
| **RingCT** [20] | CLSAG [9] | CLSAG | A $p = 0$ balance check with aggregated Bulletproofs+ range proofs [5] for output enote amount commitments.[12] |

[10] A mock-up of Lelantus-Spark [11] was not built since it is very similar to Seraphis-Concise, aside from the use of expanded Grootle proofs instead of Concise-Grootle. Concise-Grootle proofs are either equivalent-to, or better-than, expanded Grootle proofs in terms of proof size and verification cost.

[11] 'Grootle' refers to papers by Groth [10] and Bootle [2], which were precursors to Triptych [19]. 'Concise' refers to the use of $\mu_\alpha$ to reduce proof sizes in Triptych, which is in contrast to the 'expanded' proof structure used by Lelantus-Spark [11] ('Plain-Grootle' in our case). Note that our Grootle proofs use the A/B optimization from Section 1.3 of this paper: [7].

| | | | |
|---|---|---|---|
| **Triptych** [19] | Triptych | Triptych | A $p = 0$ balance check with aggregated Bulletproofs+ range proofs [5] for output enote amount commitments. |
| **Seraphis-Concise** | Concise-Grootle | Seraphis composition proofs for each input image's masked address and linking tag (Appendix A). | A $p > 0$ balance check[13] with aggregated Bulletproofs+ range proofs for output enote amount commitments. |
| **Seraphis-Squashed** (Appendix B) | Grootle on squashed enotes. | Seraphis composition proofs for each input image's masked address and linking tag. | A $p > 0$ balance check with aggregated Bulletproofs+ range proofs for input image and output enote amount commitments. |
| **Seraphis-Merge** | Concise-Grootle | Aggregate Seraphis composition proof for all input images' masked addresses and linking tags. | A $p = 0$ balance check with aggregated Bulletproofs+ range proofs for output enote amount commitments. |
| **Seraphis-Plain** | Plain-Grootle | Seraphis composition proofs for each input image's masked address and linking tag. | A $p > 0$ balance check with aggregated Bulletproofs+ range proofs for output enote amount commitments. |

The test was run single-threaded on a `zenith2alpha` motherboard with an AMD Ryzen Threadripper 3970X 32-Core processor and 256GB RAM. It was run on author koe's Seraphis performance test branch [26] at commit `ff4b862be0fbb5686067957628db401703fbfe19`.[14]

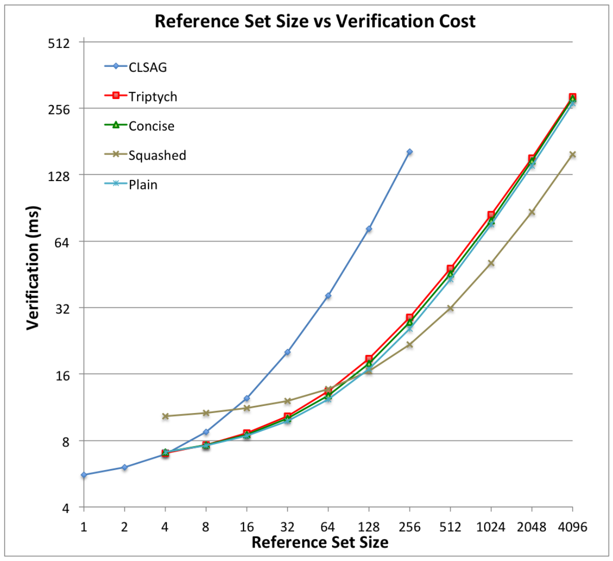
[12] 'Aggregated' Bulletproofs+ range proofs are range proofs for multiple commitments that are combined in one proof structure (i.e. byte blob).

[13] Seraphis-Concise, Seraphis-Squashed, and Seraphis-Plain (the non-merged variants) use a $p > 0$ balance check to better support collaborative funding. Since $p = 0$ is slightly more efficient (by one scalar-mult-public-key elliptic curve operation, one group addition, and 32 bytes per transaction), $p = 0$ is used for the merged variant.

[14] The test command was `./build/Linux/seraphis_perf/release/tests/performance_tests/performance_tests --filter=mock_tx --stats --loop-multiplier=10 --timings-database=/home/user/seraphis_perf/test_results/perf_run.txt > /home/user/seraphis_perf/test_results/stdout_perf_run.txt`.

## 4.2 Results and discussion

Here we discuss the test results. Note that, since the only big difference between Seraphis-Merge and Seraphis-Concise is how transaction size changes as the number of transaction inputs changes, Seraphis-Merge is only seen in the plot of input count vs transaction size.



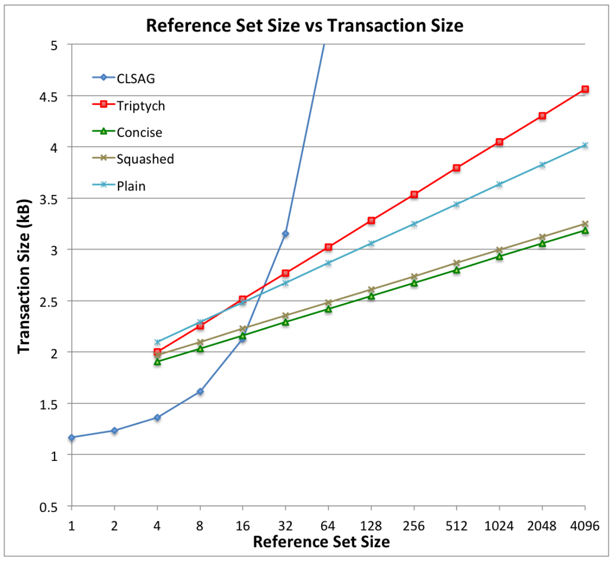


**Figure 4-1:** [Left] Membership proof reference set size vs verification cost for 2-input/2-output transactions and no batch-verification. Note that both axes are logarithmic.

**Figure 4-2:** [Right] Membership proof reference set size vs verification size for 2-input/2-output transactions and no batch-verification.

In Figure 4-1 we see that Seraphis-Concise and Triptych have practically identical verification costs. This is because the modifications to Triptych to get Concise-Grootle proofs have minimal effect on verification costs when reference sizes are reasonably large, and because Seraphis composition proofs have negligible verification cost compared to Concise-Grootle proofs.

Seraphis-Plain is only marginally faster than Seraphis-Concise (less than 10%), because the verification optimizations that can be used by Seraphis-Plain are relatively inefficient.[15]

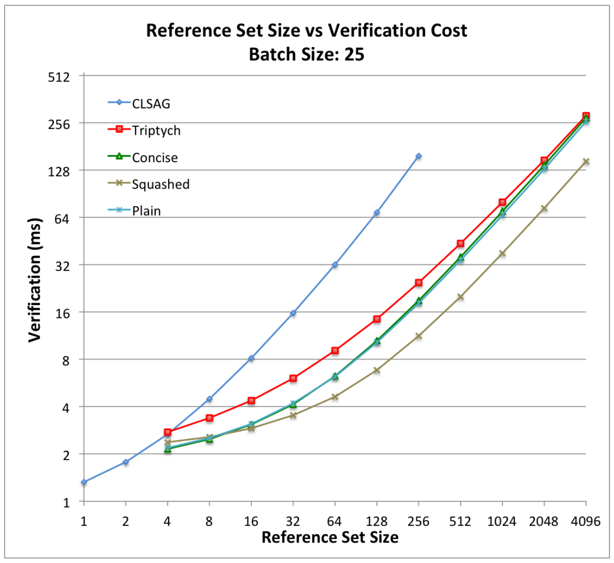
We also see that above reference set size 4, CLSAG performs significantly worse than Seraphis-Concise and Triptych. The performance cost of CLSAG for 'large' reference set sizes (above around 5-15) is the driving motivator behind research into protocols like Triptych and Seraphis.

It would seem that Seraphis-Squashed is unfavorable below a reference set size of 128, but Figures 4-3 and 4-4 will expose the advantages of that protocol variant.

Figure 4-2 further reinforces the costliness of CLSAG, and reveals that Seraphis transaction sizes scale better than Triptych with reference set sizes, which is thanks to the relative simplicity of

[15] For those readers familiar with Lelantus-Spark's Grootle proofs, we used 2-byte weights to aggregate keys during verification.

Concise-Grootle proofs. Here we also see that the marginal performance gains of Seraphis-Plain compared to Seraphis-Concise come at a significant transaction size cost.



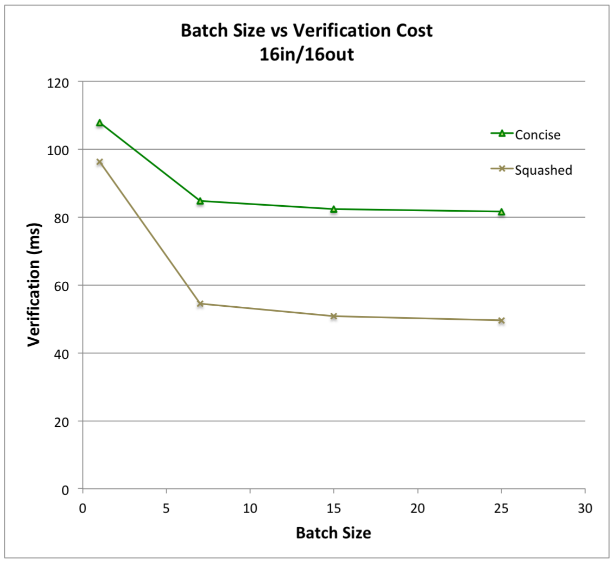


**Figure 4-3:** [Left] Membership proof reference set size vs per-transaction verification cost for 2-input/2-output transactions verified in batches of 25. Note that both axes are logarithmic.

**Figure 4-4:** [Right] Batch size vs per-transaction verification cost for 16-input/16-output transactions with 128-member reference sets.

Figures 4-3 and 4-4 show what happens when transactions are verified in batches. Multiple Grootle proofs can be verified together in batches, and likewise the aggregated Bulletproofs+ range proofs in a transaction can be batch-verified with range proofs from other equivalent transactions.

Notably, Seraphis-Squashed's performance relative to the other protocols improves greatly when batching is done. This is because Seraphis-Squashed trades simpler membership proofs for range proofs on input image masked amount commitments, and because Bulletproofs+ proofs benefit more from batching than Grootle proofs.[16]

[16] Grootle proof and Bulletproofs+ proof verification use so-called 'multiexponentiation'. It turns out the multiexponentiation operations of those proofs can be combined into one operation, which is a slight optimization (on the order of 1-10% for unbatched verification).

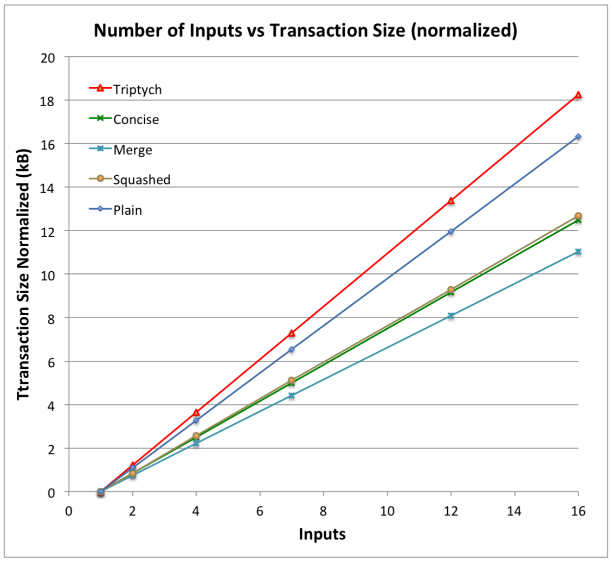


**Figure 4-5:** Input count vs transaction size (normalized to the 1-input case) for 2-output transactions with 128-member reference sets and no batch-verification.

Finally, in Figure 4-5 we see how Seraphis-Merge differs from the other variants. In explicit terms, Seraphis-Merge is $32 * (1 + 3 * (\text{num\_inputs} - 1))$ bytes smaller than Seraphis-Concise. Also note that Triptych transaction sizes scale worse with input counts than the Seraphis variants. Even Seraphis-Plain scales better than Triptych with increasing input counts.

## References

[1] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, Sep 2012. https://ed25519.cr.yp.to/ed25519-20110705.pdf [Online; accessed 03/04/2020].

[2] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on ddh. Cryptology ePrint Archive, Report 2015/643, 2015. https://ia.cr/2015/643 [Online; accessed 08/30/2021].

[3] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. https://eprint.iacr.org/2017/1066 [Online; accessed 10/28/2018].

[4] Alishah Chator and Maxwell Green. How to Squeeze a Crowd: Reducing Bandwidth in Mixing Cryptocurrencies. https://isi.jhu.edu/~mgreen/mixing.pdf [Online; accessed 08/25/2021].

[5] 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://eprint.iacr.org/2020/735 [Online; accessed 07/17/2021].

[6] Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375, 2021. https://ia.cr/2021/1375 [Online; accessed 11/19/2021].

[7] Muhammed F. Esgin, Raymond K. Zhao, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. Matrict: Efficient, scalable and post-quantum blockchain confidential transactions protocol. Cryptology ePrint Archive, Report 2019/1287, 2019. https://ia.cr/2019/1287 [Online; accessed 02/21/2022].

[8] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. https://link.springer.com/content/pdf/10.1007%2F3-540-47721-7_12.pdf [Online; accessed 03/04/2020].

[9] Brandon Goodell, Sarang Noether, and RandomRun. Concise Linkable Ring Signatures and Forgery Against Adversarial Keys. Cryptology ePrint Archive, Report 2019/654, 2019. https://eprint.iacr.org/2019/654 [Online; accessed 11/23/2020].

[10] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. Cryptology ePrint Archive, Report 2014/764, 2014. https://ia.cr/2014/764 [Online; accessed 11/18/2021].

[11] Aram Jivanyan and Aaron Feickert. Lelantus spark: Secure and flexible private transactions. Cryptology ePrint Archive, Report 2021/1173, 2021. https://ia.cr/2021/1173 [Online; accessed 09/19/2021].

[12] Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852, 2020. https://ia.cr/2020/852 [Online; accessed 11/19/2021].

[13] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Thyagarajan, and Jiafan Wang. Omniring: Scaling Private Payments Without Trusted Setup. pages 31–48, 11 2019. https://eprint.iacr.org/2019/580 [Online; accessed 03/04/2020].

[14] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. *Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups*, pages 325–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. https://eprint.iacr.org/2004/027 [Online; accessed 03/04/2020].

[15] Gregory Maxwell. Confidential Transactions. https://elementsproject.org/features/confidential-transactions/investigation [Online; accessed 11/23/2020].

[16] Andrew Miller, Malte Möser, Kevin Lee, and Arvind Narayanan. An Empirical Analysis of Linkability in the Monero Blockchain. *CoRR*, abs/1704.04299, 2017. https://arxiv.org/pdf/1704.04299.pdf [Online; accessed 03/04/2020.

[17] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. http://bitcoin.org/bitcoin.pdf [Online; accessed 03/04/2020].

[18] Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. Cryptology ePrint Archive, Report 2020/1261, 2020. https://ia.cr/2020/1261 [Online; accessed 11/19/2021].

[19] Sarang Noether and Brandon Goodell. Triptych: logarithmic-sized linkable ring signatures with applications. Cryptology ePrint Archive, Report 2020/018, 2020. https://eprint.iacr.org/2020/018 [Online; accessed 03/04/2020.

[20] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring Confidential Transactions, MRL-0005, February 2016. https://web.getmonero.org/resources/research-lab/pubs/MRL-0005.pdf [Online; accessed 06/15/2018].

[21] Shen Noether and Sarang Noether. Thring Signatures and their Applications to Spender-Ambiguous Digital Currencies, MRL-0009, November 2018. https://web.getmonero.org/resources/research-lab/pubs/MRL-0009.pdf [Online; accessed 01/15/2020].

[22] Torben Pryds Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, pages 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992. https://www.cs.cornell.edu/courses/cs754/2001fa/129.PDF [Online; accessed 03/04/2020].

[23] Viktoria Ronge, Christoph Egger, Russell W. F. Lai, Dominique Schröder, and Hoover H. F. Yin. Foundations of ring sampling. Cryptology ePrint Archive, Report 2020/1550, 2020. https://ia.cr/2020/1550 [Online; accessed 09/19/2021].

[24] UkoeHB. Inversion-style key images and aggregation (multisig), Issue #92, March 2020. https://github.com/monero-project/research-lab/issues/72 [Online; accessed 09/18/2022].

[25] UkoeHB. Seraphis Performance Results, Issue #91, November 2021. https://github.com/monero-project/research-lab/issues/91 [Online; accessed 11/18/2021].

[26] UkoeHB. `seraphis_perf` repository branch, November 2021. https://github.com/UkoeHB/monero/tree/seraphis_perf [Online; accessed 11/18/2021].

[27] Nicolas van Saberhagen. CryptoNote V2.0. https://bytecoin.org/old/whitepaper.pdf [Online; accessed 03/10/2021].

[28] Tsz Hon Yuen, Shi-feng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu. RingCT 3.0 for Blockchain Confidential Transaction: Shorter Size and Stronger Security. Cryptology ePrint Archive, Report 2019/508, 2019. https://eprint.iacr.org/2019/508 [Online; accessed 03/04/2020].

## A Composition proofs with Schnorr

We present a signature scheme that satisfies the Seraphis ownership/unspentness proof requirements (Section 3.5), assuming $J = G_2$. First we lay out the proof system that satisfies those requirements, then we describe a proof structure in that system.[17]

### A.1 Overview

1. Assume there is a group point $K = xG_0 + yG_1 + zG_2$.
2. Let
$$\begin{aligned} K_{t1} &= (1/y) * K \\ K_{t2} &= K_{t1} - G_1 - \tilde{K} \\ \tilde{K} &= (z/y) * G_2 \end{aligned}$$
3. Demonstrate the discrete log of $K_{t1} = (1/y) * K$ with respect to $K$.
4. Demonstrate the discrete log of $K_{t2} = (x/y) * G_0$ with respect to $G_0$.
5. Demonstrate the discrete log of $\tilde{K} = (z/y) * G_2$ with respect to $G_2$.

#### A.1.1 Seraphis requirements satisfaction

[[[explain how it satisfies the requirements?]]]

### A.2 Construction

Our proof is a Schnorr-like $\Sigma$-protocol between prover and verifier.

1. Suppose the prover has keys $x, y, z, K$, where $K = xG_0 + yG_1 + zG_2$.
2. The prover generates random scalars $\alpha_a, \alpha_b, \alpha \in_R \mathbb{Z}_l$.
3. The prover computes $\alpha_a G_0$, $\alpha_b G_2$, $\alpha K$, $K_{t1} = (1/y) * K$, and $\tilde{K} = (z/y) * G_2$. He sends all of those to the verifier along with the key $K$.
4. The verifier generates random challenge $c \in_R \mathbb{Z}_l$ and sends it to the prover.
5. The prover computes responses $r_a, r_b, r$ and sends them to the verifier.
$$\begin{aligned} r_a &\equiv \alpha_a - c * (x/y) \\ r_b &\equiv \alpha_b - c * (z/y) \\ r &\equiv \alpha - c * (1/y) \end{aligned}$$

[17] We leave applying the Fiat-Shamir transform [8] to the composition proof structure as an exercise for the reader.

6. The verifier computes $K_{t2} = K_{t1} - G_1 - \tilde{K}$, then checks the following equalities. If any of them fail (or if $K_{t1}$ or $\tilde{K}$ equal the identity element $I$), then the prover has failed to satisfy the composition proof system.
$$\alpha_a G_0 = r_a G_0 + cK_{t2}$$
$$\alpha_b G_2 = r_b G_2 + c\tilde{K}$$
$$\alpha K = rK + cK_{t1}$$

## B Squashed enote model

The squashed enote model is a specialization of the Seraphis membership proof model that allows relatively simpler and more efficient proof structures.

First we will describe the model, then discuss how it satisfies relevant security requirements when applied to Seraphis.

### B.1 Model

1. Require $G_0 = H_0$ and that the discrete logarithm of $H_1$ with respect to any of $G_1, ..., G_n$ is unknown (in addition to all the other requirements from Section 2.1).
2. Let $\mathbb{S}$ represent a set of tuples $[K_i, C_i]$, where
$$K_i = z_i G_0 + s_{i,1} G_1 + s_{i,2} G_2 + ... + s_{i,n} G_n$$
$$C_i = x_i H_0 + a_{i,1} H_1$$
3. Let $\mathbb{S}^t$ represent a set of 'transformed' tuples $[K_i^t, C_i^t]$, where
$$K_i^t = \mathcal{H}_1(K_i, C_i) * K_i$$
$$C_i^t = C_i$$
4. Perform a range proof on each $C_i^t \in \mathbb{S}^t$ (recall Section 3.6.1).
5. Let $\tilde{S}$ represent a tuple $[K', C']$ (e.g. a masked version of $\mathbb{S}^t[\pi]$), where
$$K' = z' G_0 + s'_1 G_1 + s'_2 G_2 + ... + s'_n G_n$$
$$C' = x' H_0 + a'_1 H_1$$
6. Let $\mathbb{Q}$ represent a set of squashed tuples $[Q_i]$, where
$$Q_i = K_i^t + C_i^t$$
7. Let $\tilde{Q} = K' + C'$.
8. Demonstrate that, within a security parameter $k$, $\tilde{Q}$ corresponds to some $Q_\pi \in \mathbb{Q}$, where $\pi$ is unknown to the verifier, such that:

   (a) The discrete log relation of $\tilde{Q} - Q_\pi = [(z' + x') - (\mathcal{H}_1(K_\pi, C_\pi) * z_\pi + x_\pi)] * G_0$ with respect to $G_0$ is known.

9. Perform a range proof on $C'$.

10. Demonstrate knowledge of $z', s'_1, ..., s'_n$ such that $K' = z'G_0 + s'_1 G_1 + s'_2 G_2 + ... + s'_n G_n$.

The benefit of this specialization compared to the underlying membership proof model is you only need to prove the discrete log in one commitment to zero relation, rather than two. For example, with a SAG (e.g. LSAG [14] without linking) or Grootle proof (Appendix C) on the set $\{\tilde{Q} - Q_i\}$.[18] The efficiency implications are discussed in Section 4, which compares possible instantiations of Seraphis using the two models.

## B.2 Requirement satisfaction

### B.2.1 Underlying membership proof model

We argue that the squashed enote model satisfies the underlying membership proof model.

Let $\mathbb{S}^t$ be the input to the underlying model. We will show that the following requirement, adapted from Section 3.4, is met.

1. Demonstrate that, within a security parameter $k$, $\tilde{S}$ corresponds to some $S^t_\pi \in \mathbb{S}^t$, where $\pi$ is unknown to the verifier, such that:
   (a) $s'_p == \mathcal{H}_1(K_\pi, C_\pi) * s_{\pi,p}$ for $p \in 1, ..., n$
   (b) $a'_1 == a_{\pi,1}$

Observe the following.

1. The prover must know $[(z' + x') - (\mathcal{H}_1(K_\pi, C_\pi) * z_\pi + x_\pi)]$ and $z'$ to satisfy the squashed enote model, and $x'$ and $x_\pi$ to construct the range proofs on $C'$ and $C^t_\pi$. Therefore the prover must know $\mathcal{H}_1(K_\pi, C_\pi) * z_\pi$. The value $\mathcal{H}_1(K_\pi, C_\pi)$ is considered 'public knowledge', so the prover must also know $z_\pi$ (which may be 0).

2. Range proofing $C_\pi$ and $C'$ means they have the form $xH_0 + aH_1$, implying they contain no $G_1, ..., G_n$ components. Therefore, demonstrating discrete log with respect to $G_0$ in the commitment to zero $\tilde{Q} - Q_\pi$ means it must be the case that $s'_p == \mathcal{H}_1(K_\pi, C_\pi) * s_{\pi,p}$ for $p \in 1, ..., n$ (i.e. all factors of $G_1, ..., G_n$ in $\tilde{Q}$ and $K'$ must come from $K_\pi$).

[18] In practice, it may be simpler to implement a membership proof on the equivalent set $\{Q_i - \tilde{Q}\}$.

3. Suppose $K_i$ has the form $K_i = z_i G_0 + s_{i,1} G_1 + s_{i,2} G_2 + ... + s_{i,n} G_n + eH_1$. Since the model requires a demonstration that $K'$ does not contain any $H_1$ components, and the commitment to zero $\tilde{Q} - Q_\pi$ means all non-$G_0$ components balance in those two points, it must be the case that $C' = x' H_0 + (a_{\pi,1} + \mathcal{H}_1(K_\pi, C_\pi) * e) * H_1$. However, the term $\mathcal{H}_1(K_\pi, C_\pi)$ is both uniformly distributed and implicitly dependent on the values $a_{\pi,1}$ and $e$, so the value $a'_1 = (a_{\pi,1} + \mathcal{H}_1(K_\pi, C_\pi) * e)$ will be uniformly distributed in $\mathbb{Z}_l$ (assuming $e$ is non-zero). Since the range proof on $C'$ means that $a'_1$ must be in the range $[0, ..., 2^z - 1]$, if $e \neq 0$ then the probability that a range proof on $a'_1$ can succeed is $2^z/l$. This means $a'_1 == a_{\pi,1}$ can be assumed to be true within the security parameter $k$ if $1/k > 2^z/l$.[19] [[[formalize better?]]]

### B.2.2 Seraphis structure

1. A range proof on $C_i$ is equivalent to a range proof on $C_i^t$, since $C_i^t = C_i$. These range proofs will already exist, since membership proofs only reference enotes in the ledger (which were created by transactions whose output enotes' amount commitments were range proofed, or are coinbase enotes whose amounts are known).

2. Seraphis linking tags are computed from the output of a membership proof, namely the point $K'$ in $\tilde{S}$. However, $K'$ in the squashed enote model applied to Seraphis has the form $K' = t_k G_0 + \mathcal{H}_1(K_\pi^o, C_\pi) * [k_0^o * G_0 + k_1^o * G_1 + k_2^o * G_2]$. This means linking tags will have the form $\tilde{K} = ((\mathcal{H}_1(K_\pi^o, C_\pi) * k_2^o)/(\mathcal{H}_1(K_\pi^o, C_\pi) * k_1^o)) * J$. Since the terms $\mathcal{H}_1(K_\pi^o, C_\pi)$ cancel each other, linking tags in this model are the same as in the base model.

3. Step 10 in the above model is automatically satisfied by Seraphis because $K'$ is passed as input to the ownership/unspentness proof system, which demonstrates knowledge of the per-generator discrete log relations of its inputs.

## B.3 Practical considerations

Transaction verifiers can pre-compute steps 3 and 6 from the above model for every enote in the ledger. The squashed tuples $Q_i$ can be stored in anticipation of new transactions that may require them.

# C Grootle proof

We present a Groth/Bootle one-of-many proof [10, 2, 7] that satisfies the membership proof requirements of the squashed enote model (Appendix B). The proof construction is a simplified form of the one-of-many proof in Appendix B of the Lelantus-Spark paper [11], so we have adapted their description and security proof largely verbatim.

[19] Typically $l \approx 2^{252} - 2^{256}$, $2^z = 2^{64}$, and $k = 2^{128}$; $1/2^{128} > 2^{64}/2^{252}$ is true.

## C.1 Construction

Let there be a tuple of algorithms (GrootleProve, GrootleVerify) for the following relation, where we let $N = n^m$:

$$\{pp_{\text{grootle}}, \{S_k\}_{k=0}^{N-1} \subset \mathbb{G}, S' \in \mathbb{G}; l \in \mathbb{N}, s \in \mathbb{F} : 0 \leq l < N, S_l - S' = sG_0\}$$

Let $\delta(i, j) : \mathbb{N}^2 \to \mathbb{F}$ be the Kronecker delta function. For any integers $k$ and $j$ such that $0 \leq k < N$ and $0 \leq j < m$, let $k_j$ denote the $j$ digit of the $n$-ary decomposition of $k$. Let $\mathsf{MatrixCom} : \mathbb{F} \times \mathbb{F}^{mn} \times \mathbb{F}^{mn} \to \mathbb{G}$ be an additively-homomorphic matrix commitment construction that commits to the entries of two matrices, and is perfectly hiding and computationally binding.

The protocol proceeds as follows:

1. The prover selects
$$r_A, r_B, \{a_{j,i}\}_{j=0,i=1}^{m-1,n-1} \in \mathbb{F}$$
uniformly at random, and, for each $j \in [0, m)$, sets
$$a_{j,0} = -\sum_{i=1}^{n-1} a_{j,i}$$

2. The prover computes the following:
$$A \equiv \mathsf{MatrixCom}\left(r_A, \{a_{j,i}\}_{j,i=0}^{m-1,n-1}, \{-a_{j,i}^2\}_{j,i=0}^{m-1,n-1}\right)$$
$$B \equiv \mathsf{MatrixCom}\left(r_B, \{\delta(l_j, i)\}_{j,i=0}^{m-1,n-1}, \{a_{j,i}(1 - 2\delta(l_j, i))\}_{j,i=0}^{m-1,n-1}\right)$$

3. For each $j \in [0, m)$, the prover selects $\rho_j \in \mathbb{F}$ uniformly at random, and computes the following:
$$X_j \equiv \sum_{k=0}^{N-1} p_{k,j}(S_k - S') + \rho_j G_0$$
Here each $p_{k,j}$ is defined such that for all $k \in [0, N)$ we have
$$\prod_{j=0}^{m-1} (\delta(l_j, k_j)x + a_{j,k_j}) = \delta(l, k)x^m + \sum_{j=0}^{m-1} p_{k,j}x^j$$
for indeterminate $x$.

4. The prover sends $A, B, \{X_j\}_{j=0}^{m-1}$ to the verifier.

5. The verifier selects $x \in \mathbb{F}$ uniformly at random and sends it to the prover.

6. For each $j \in [0, m)$ and $i \in [1, n)$, the prover computes $f_{j,i} \equiv \delta(l_j, i)x + a_{j,i}$ and the following values:
$$z_A \equiv r_A + xr_B$$
$$z \equiv sx^m - \sum_{j=0}^{m-1} \rho_j x^j$$

7. The prover sends $\{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}, z_A, z$ to the verifier.

8. For each $j \in [0, m)$, the verifier sets $f_{j,0} \equiv x - \sum_{i=1}^{n-1} f_{j,i}$ and accepts the proof if and only if
$$A + xB = \mathsf{MatrixCom}\left(z_A, \{f_{j,i}\}_{j,i=0}^{m-1,n-1}, \{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}\right)$$
and
$$\sum_{k=0}^{N-1}\left(\prod_{j=0}^{m-1} f_{j,k_j}\right)(S_k - S') - \sum_{j=0}^{m-1} x^j X_j = zG_0$$
are true.

This interactive protocol can be made non-interactive using the Fiat-Shamir technique, which replaces the verifier challenge $x$ with the output of a cryptographic hash function on transcript inputs.

## C.2 Security proof

We now prove that the above protocol is complete, special sound, and honest-verifier zero knowledge.

### C.2.1 Completeness

Completeness of the protocol follows by straightforward algebra.

### C.2.2 Special honest-verifier zero knowledge

To show that the protocol is special honest-verifier zero knowledge, we construct a simulator that, when provided a valid statement and random verifier challenge $x$, produces a proof transcript identically distributed to that of a real proof.

To produce a simulated transcript on random $x$, the simulator samples
$$B, \{X_j\}_{j=1}^{m-1} \in \mathbb{G}$$
and
$$z_A, z, \{f_{j,i}\}_{j=0,i=1}^{m-1,n-1} \in \mathbb{F}$$
uniformly at random. It defines
$$f_{j,0} = x - \sum_{i=1}^{n-1} f_{j,i}$$
for each $j \in [0, m)$, and sets
$$A = \mathsf{MatrixCom}\left(z_A, \{f_{j,i}\}_{j,i=0}^{m-1,n-1}, \{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}\right) - xB$$

as well. It uses the final two verification equations to compute $X_0$:

$$X_0 = \sum_{k=0}^{N-1} \left( \prod_{j=0}^{m-1} f_{j,k_j} \right) (S_k - S') - \sum_{j=1}^{m-1} x^j X_j - zG_0$$

Since the challenge $x$ is sampled uniformly at random by construction, the commitment constructions are perfectly hiding, $\{\rho_j\}_{j=0}^{m-1}$ are sampled uniformly at random in a real proof, and the decisional Diffie-Hellman problem is hard in $\mathbb{G}$, all proof elements in both the simulation and real proofs are either independently uniformly distributed at random or uniquely determined by other transcript elements. Hence the protocol is special honest-verifier zero knowledge.

### C.2.3 Special sound

We now show that the protocol is $(m+1)$-special sound for $m > 1$. That is, we construct an extractor that, when presented with a set of $m+1$ distinct challenges and corresponding responses to the same initial statement, produces a set of extracted witness elements consistent with the statement. Consider a collection of $m+1$ distinct challenges $\{x_\iota\}_{\iota=0}^{m}$ and corresponding valid responses:

$$\left\{ \{f_{j,i}^{(\iota)}\}_{j=0,i=1}^{m-1,n-1}, z_A^{(\iota)}, z^{(\iota)} \right\}_{\iota=0}^{m}$$

Successful verification on indices $\iota \in \{0,1\}$ gives the following:

$$\begin{aligned}(x^{(0)} - x^{(1)})B = \mathsf{MatrixCom}\Big(&z_A^{(0)} - z_A^{(1)}, \\ &\{f_{j,i}^{(0)} - f_{j,i}^{(1)}\}_{j,i=0}^{m-1,n-1}, \\ &\{f_{j,i}^{(0)}(x^{(0)} - f_{j,i}^{(0)}) - f_{j,i}^{(1)}(x^{(1)} - f_{j,i}^{(1)})\}_{j,i=0}^{m-1,n-1}\Big)\end{aligned}$$

For all $j \in [0,m)$ and $i \in [0,n)$, if we let

$$b_{j,i} = \frac{f_{j,i}^{(0)} - f_{j,i}^{(1)}}{x^{(0)} - x^{(1)}}$$

and

$$c_{j,i} = \frac{f_{j,i}^{(0)}(x^{(0)} - f_{j,i}^{(0)}) - f_{j,i}^{(1)}(x^{(1)} - f_{j,i}^{(1)})}{x^{(0)} - x^{(1)}}$$

and

$$r_B = \frac{z_A^{(0)} - z_A^{(1)}}{x^{(0)} - x^{(1)}}$$

then we can express

$$B = \mathsf{MatrixCom}\left(r_B, \{b_{j,i}\}_{j,i=0}^{m-1,n-1}, \{c_{j,i}\}_{j,i=0}^{m-1,n-1}\right)$$

If for $j \in [0,m)$ and $i \in [0,n)$ we further define

$$a_{j,i} = f_{j,i}^{(0)} - x^{(0)} b_{i,j}$$

and

$$d_{i,j} = f_{j,i}^{(0)}(x^{(0)} - f_{j,i}^{(0)}) - x^{(0)} c_{j,i}$$

and $r_A = z_A^{(0)} - x^{(0)} r_B$, then we can express

$$A = \mathsf{MatrixCom}\left(r_A, \{a_{j,i}\}_{j,i=0}^{m-1,n-1}, \{d_{j,i}\}_{j,i=0}^{m-1,n-1}\right)$$

as well. Observe that since the commitment construction is computationally binding, for all $\iota \in [0,m]$ we must have $b_{j,i}x^{(\iota)} + a_{j,i} = f_{j,i}^{(\iota)}$ and $c_{j,i}x^{(\iota)} + d_{j,i} = f_{j,i}^{(\iota)}(x^{(\iota)} - f_{j,i}^{(\iota)})$ for $j \in [0,m)$ and $i \in [0,n)$. This implies in particular that for $\iota \in \{0,1,2\}, j \in [0,m), i \in [0,n)$ we have

$$c_{j,i}x^{(\iota)} + d_{j,i} = b_{j,i}(1 - b_{j,i})x^{(\iota)2} + (1 - 2b_{j,i})a_{j,i}x^{(\iota)} - a_{j,i}^2$$

and hence $b_{j,i}(1 - b_{j,i}) = 0$, so each $b_{j,i} \in \{0,1\}$.

We also have, by construction, that

$$x^{(\iota)} = \sum_{i=0}^{n-1} f_{j,i}^{(\iota)} = x^{(\iota)} \sum_{i=0}^{n-1} b_{j,i} + \sum_{i=0}^{n-1} a_{j,i}$$

for $\iota \in [0,m], j \in [0,m)$, so $\sum_{i=0}^{n-1} b_{j,i} = 1$. This means we can extract $l \in [0,N)$ such that each $b_{j,i} = \delta(l_j, i)$.

Now if we define for each $k \in [0,N)$ the polynomial

$$p_k(x) = \prod_{j=0}^{m-1} \left[\delta(l_j, k_j)x + a_{j,k_j}\right]$$

in $x$, we have $\deg(p_k) = m$ if and only if $k = l$. Verification can therefore be expressed as

$$x^{(\iota)m}(S_l - S') - \sum_{j=0}^{m-1} x^{(\iota)j}\overline{X}_j = z^{(\iota)}G_0$$

for $\iota \in [0,m]$, where the set $\{\overline{X}_j\}_{j=0}^{m-1}$ can be uniquely derived. Consider a Vandermonde matrix $V$ such that the $\iota$ row is the vector $(1, x^{(\iota)}, ..., x^{(\iota)m})$, and note since each challenge is distinct, we have $\det(V) \neq 0$ with high probability, so the rows of $V$ span $\mathbb{F}^{m+1}$. This means we can find $\{\theta_\iota\}_{\iota=0}^{m}$ such that the equation

$$\sum_{\iota=0}^{m} \theta_\iota x^{(\iota)j} = \delta(j,m)$$

holds for $j \in [0,m]$.

We can therefore build a linear combination of each of the two above verification equations, taking advantage of the Vandermonde-derived weights:

$$S_l - S' = \sum_{\iota=0}^{m} \theta_\iota x^{(\iota)m}(S_l - S') + \sum_{\iota=0}^{m} \theta_\iota \left(x^{(\iota)j}\overline{X}_j\right) = \left(\sum_{\iota=0}^{m} \theta_\iota z^{(\iota)}\right) G_0$$

This equation provides the remaining extraction

$$s = \sum_{\iota=0}^{m} \theta_\iota z^{(\iota)}$$

such that $S_l - S' = sG_0$, which completes the proof.