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

koe[1,2] ukoe@protonmail.com

August 26, 2021
Draft v0.0.2

1 Abstract

Seraphis[3] is a privacy-focused transaction protocol abstraction for p2p electronic cash systems that use the transaction output model (the *e-note* model in this paper). Seraphis e-notes 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 CT (Confidential Transactions), where e-note 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 allows the security model for membership proofs to be abstracted away from any specific proving system, which both allows relatively simpler proving structures to be used, and greatly simplifies the overall security model of Seraphis compared to its predecessors. Doing so also allows a linking tag (a.k.a. key image) construction with a number of favorable properties. Most notably, implementers of Seraphis can use an 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 Introduction

Seraphis is a transaction protocol abstraction for p2p electronic cash systems. What is a 'p2p electronic cash system' and why does it need a 'transaction protocol'?

A p2p (peer-to-peer) electronic cash system is a monetary system where the entire supply of currency is stored/recorded in digital form, and transactions (attempts to transfer money to new owners) are mediated by a network of *peers* (usually called *nodes*). Such systems are typically designed so 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.

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

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

[3] The name 'Seraphis' is based on 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.

To achieve those design goals, 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 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 mutations of the 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.

2.1 Monetary state

Most modern p2p electronic cash systems are so-called 'cryptocurrencies' in the tradition of Bitcoin [19]. In Bitcoin, each (archival) node maintains a full copy (called a *ledger*) of all mutations to the monetary state that led from Bitcoin's inception up to the current moment.

The monetary state of a cryptocurrency is defined by all the 'money creation' and 'amount transfer' events that have occurred since the currency was created. Almost universally, those events are defined in the *transaction output* model (henceforth called the *e-note* model). An e-note is a small message that records an 'amount of money', an 'ownership address' that gives someone (the recipient) the authority to spend the e-note, and an optional arbitrary memo.

- **Money creation event**: Create a new 'coinbase e-note', which increases the total supply of money (see Section 5.1).
- **Money transfer event (transaction)**: Consume one or more previously unspent e-notes to transfer the amounts they record to one or more new e-notes (see Section 4).

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

2.2 Transaction protocols

Transaction protocols must always codify a basic set of rules.

- **Membership**: E-notes spent by a transaction must already exist in the ledger.
- **Unspentness**: E-notes spent by a transaction must not have been already spent.
- **Ownership**: The transaction author must have the authority to spend those e-notes.
- **Amount balance**: The total amount recorded in e-notes spent by a transaction must equal the total amount in new e-notes created (plus a transaction fee, usually).

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

- **Membership**: Reference existing e-notes with their indices in the ledger. Transaction validators can look up those e-notes directly.
- **Unspentness**: When an e-note is spent by a transaction, set a bit flag next to that e-note in the ledger. Reject transactions that reference spent e-notes.
- **Ownership**: Define ownership with public key cryptography. Let each e-note's address record a public key specified in advance by the intended recipient. To spend an e-note, its owner must create a cryptographic signature with its public key and add the signature to their transaction.
- **Amount balance**: Record e-note 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 e-notes and transaction events are public knowledge. If amounts are in cleartext, addresses can be reused, and e-notes 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 e-notes to be more valuable than other e-notes just based on differences in who owns them or where they originated (i.e. the history/transaction-graph that led to those e-notes 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.

[4] A transaction author inherently knows who they send e-notes 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 (except 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) [26] and Confidential Transactions [16], which were first combined in the protocol RingCT [22]. 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 [21], Lelantus [9] [[[change citation?]]], Omniring [14], and RingCT3.0 [27]) have 'anonymity sets'. A transaction author proves that each e-note spent by their transaction exists in a small set of e-notes, and further proves that that small set is a subset of e-notes that exist in the ledger. With this method, observers know there is more likely to be a relationship between two transactions if one of them references an e-note created by the other, than if no such connection exists. This probabilistic knowledge is stronger than the 'pure/ideal' case where membership proofs show that e-notes exist in the ledger without giving any hints about which ones they 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 'images' of e-note addresses produced when the e-notes are spent. Linking tags are the core element of unspentness proofs in privacy-focused transaction protocols. 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 e-note that has already been spent.

As one example, the transaction protocol Triptych [21], which allows a proving structure one or two orders 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 e-note, 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.

2.3 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 e-note corresponds with an e-note in some reference set. The prover then operates on the e-note 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. Linking tags are created by inverting 'some' of the private key material associated with an e-note's address.

   If, in a multisignature scheme, all private key material related to linking tags are known by all signing participants in advance, then it is not a concern if that material is inverted to create linking tags.

   Proving knowledge of the address's 'other' private key material (as part of an ownership proof; see Section 4.5) is trivially linear, so if that material is divided amongst multisig participants, then simple multisignature schemes are possible.

2. Linking tags make it possible to implement a user addressing scheme with three tiers of permissions (see Sections 4.7 and 5.5).

   In that scheme, users can isolate parts of their personal private key material to create wallets that can view received e-notes only, recover the user's full balance (i.e. recompute linking tags to detect spent e-notes), or both recover the user's full balance and also spend owned e-notes. The second permission tier is uniquely enabled by Seraphis among protocols in the CryptoNote tradition.

In Appendix A.1 we introduce a more-restrictive membership proof model layered on the primary model in this paper. We call it the 'squashed e-note' 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.

Also note that Seraphis permits 'transaction chaining', where it is possible to construct a transaction B that spends an e-note from transaction A before A has been added to the ledger (Section 5.6), and 'membership proof delegation', where constructing a membership proof is delegated to a third party without revealing any wallet private keys. Combining these yields 'transaction chaining by delegation', where the author of transaction B authorizes the transfer of funds from an

e-note that doesn't exist in the chain, then gives their partial transaction (without a membership proof) to the author of transaction A (which creates that e-note), who can complete transaction B and submit it after they have completed and submitted transaction A.[5]

3 Preliminaries

3.1 Public parameters

[[[PLAGIARIZED FROM TRIPTYCH PAPER]]] Let $\mathbb{G}$ be a cyclic group of prime order $l > 3$ in which the discrete logarithm problem is hard, and let $\mathbb{Z}_l$ be its scalar field. Let $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_l$ 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 $\mathcal{H}(\text{“}sender\_receiver\_secret\text{”}, [\text{hash input}])$). Let G, H, and U be generators of $\mathbb{G}$ whose discrete logarithm relationship to each other is unknown. 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 such 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.

3.2 Notation

no exponents, only superscripts

Pedersen commitments? vG + aH

non-pedersen commitments? vG + P

range proof? for pedersen commitment, demonstrate composition (DL known on G and H), a /in [0, 2ĝ64 - 1]

4 Seraphis

In this section we discuss the various components of Seraphis, with security theorems and proofs introduced where appropriate.

[5] Transaction chaining by delegation gets even more interesting if combined with something like Monero's lock-time. A Monero transaction author can define the earliest blockchain height where the new e-notes they create can be spent. Suppose transaction B spends an e-note that doesn't exist in the ledger (blockchain) *and* an e-note with a certain lock-time, then author B delegates completing the transaction to author A. Author A cannot submit transaction B until that locked e-note can be spent. If they submit transaction A right away, then there will be a period of time where the e-note it creates can't be spent by transaction B. Author B could make another transaction B' in that time that spends the e-note. We leave the application of these ideas to a useful protocol as an exercise for the reader.

4.1 Transaction overview

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

- **Inputs**: The transaction spends old e-notes.
  - **E-note-images**: Representations of the e-notes spent by this transaction, including their linking tags (Section 4.3).
  - **Membership proofs**: Proof structures demonstrating that each e-note-image corresponds with a real e-note in the ledger (Section 4.4).
  - **Ownership and unspentness proofs**: Proof structures that use e-note-images to demonstrate ownership and unspentness for each spent e-note (Section 4.5).
- **Outputs**: The transaction creates new e-notes.
  - **E-notes**: New e-notes (Section 4.2). The total amount they contain equals the total amount in spent e-notes (Section 4.6).
  - **Range proofs**: Proof structures demonstrating that amount commitments in new e-notes are legitimate (part of the Confidential Transactions technique) (Section 4.6).
- **Miscellaneous**: Miscellaneous other data included with a transaction, such as a transaction fee (Section 5).

4.2 E-notes

Seraphis e-notes 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 possessed by the e-note.
$$C = xG + aH$$
- **Address**: A public key K^o composed of two generators U and G, and two corresponding private keys k_a^o and k_b^o. The e-note's owner must prove knowledge of those private keys if they want to transfer the amount a to new e-notes.
$$K^o = k_a^o * U + k_b^o * G$$
- **Memo**: An arbitrary memo field. This usually includes information that helps the e-note's owner identify that they own it, learn the private keys k_a^o and k_b^o, and reconstruct the amount commitment. See Section 5.4.

4.3 E-note-images

An e-note-image is a representation of an e-note.

- **Masked commitment**: The e-note's commitment with an additional masking factor.
$$C' = t_c G + C$$
$$C' = (t_c + x) * G + aH$$
$$C' = v_c G + aH$$
- **Masked address**: The e-note's address with a masking factor.
$$K'^o = t_k G + K^o$$
$$K'^o = (t_k + k_b^o) * G + k_a^o * U$$
$$K'^o = v_k G + k_a^o * U$$
- **Linking tag**: The e-note's linking tag.
$$\tilde{K} = (1/k_a^o) * G$$

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 cannot look at an e-note-image and discern what e-note it was created from.

We describe how a transaction author can prove that e-note-image addresses and commitments are constructed properly from real e-notes in Section 4.4, and further prove that linking tags are constructed properly from e-note-image masked addresses in Section 4.5.

4.3.1 Sender-receiver anonymity

If a person spends an e-note, they should expect that the person who originally sent them that e-note 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 e-note by inspecting the e-note-image's commitment and address.

We further argue in Sections 4.4, 4.5, and 4.7 that input proofs and linking tags will not break sender-receiver anonymity.

4.3.2 Linking tags

Linking tags are uniquely defined by the private key k_a^o. This means for a user to create two distinct linking tags from the same address, they must be able to solve the DLP between generators U and G, 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 e-note-image with a linking tag that has appeared in the ledger, then that transaction is invalid.

Note that if two e-notes 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 discussed in Sections 4.7 and 5.4.

4.4 Membership proofs

Every input to a transaction must have a membership proof. The proof must demonstrate that the input's e-note-image was built from an e-note 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 $G_1, ..., G_n$ and $H_1, ..., H_n$ be generators whose discrete logarithm relations with each other and with G are unknown.
2. Let $\mathbb{S}$ represent a set of tuples $\{K_i, C_i\}$, where
$$K_i = z_i G + s_{i,1} G_1 + s_{i,2} G_2 + ... + s_{i,n} G_n$$
$$C_i = x_i G + a_{i,1} H_1 + a_{i,2} G_2 + ... + a_{i,n} H_n$$
3. Let $\tilde{S}$ represent a tuple $\{K', C'\}$, where
$$K' = z' G + s'_1 G_1 + s'_2 G_2 + ... + s'_n G_n$$
$$C' = x' G + a'_1 H_1 + a'_2 H_2 + ... + a'_n H_n$$
4. 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 π is unknown to the verifier, such that:
   (a) $s'_j == s_{\pi,j}$ for $j \in 1, ..., n$
   (b) $a'_j == a_{\pi,j}$ for $j \in 1, ..., n$
   (c) If the prover later demonstrates knowledge of z' in K', then they must also have knowledge of z_π.
5. The proving system should be considered unusable if, given a proof σ that $\tilde{S}$ corresponds to some S_π in the set $\mathbb{S}$, an observer can guess the index π with probability $> 1/\text{size}(\mathbb{S}') + \epsilon(k)$, 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 e-notes, then for example he owns the e-notes in $\mathbb{S}_O$), and the observer has no special knowledge about the elements in $\mathbb{S}'$ (however, he can know $z_i G$, x_i, and $a_{i,j}$ for all $S_i \in \mathbb{S}'$).[6]

In the context of Seraphis, we straightforwardly construct tuples S directly from e-notes, which can be referenced with simple ledger indices for verifiers to find (in a naive implementation),[7] and tuples $\tilde{S}$ from e-note-images. Readers will note that a membership proof says nothing about how

[6] In practice, π can often be guessed with probability at least marginally above $1/\text{size}(\mathbb{S}')$. This is because the circumstances around when e-notes 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.

[7] If the size of $\mathbb{S}$ is small, then it may be practical to reference e-notes 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 [3].

K and C are constructed (i.e. the values of $G_1, ..., H_1, ...$, etc.). In future sections we will add more constraints to guarantee that e-notes and e-note-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 $K' - K$ and $C' - C$, given a reference set $\mathbb{S}$ that contains only one tuple $\{K, C\}$. More interesting solutions include a CSAG (a CLSAG [8] without linking) on a ring of such commitment to zero pairs, a Groth/Bootle [**?**] one-of-many proof (see Appendix **??**) on a collection of those pairs, or a Groth/Bootle one-of-many proof applied to the squashed e-note model (see Appendix A).

4.5 Ownership and unspentness proofs

Alongside each membership proof must be an ownership and unspentness proof. In Seraphis, these are done simultaneously to ensure the linking tag (which forms the basis of unspentness) is derived from the relevant e-note 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 $K = xG + yU$. Let
$$\tilde{K} = (1/y) * G$$
$$K_{t1} = (1/y) * K$$
$$K_{t2} = K_{t1} - U$$
2. Demonstrate the simultaneous discrete log of K_{t1} and $\tilde{K}$ with respect to K and G. A simultaneous discrete log proof shows knowledge of a single scalar that is the discrete log between two pairs of group elements (in this case $(1/y)$). [[[formalize better?]]]
3. Demonstrate the discrete log of $K_{t2} = (x/y) * G$ with respect to G.

This 'composition' proof system shows that:

- The prover knows values x and y such that $K = xG + yU$.
- $\tilde{K} == (1/y) * G$

We can apply composition proofs to Seraphis in the following way.

Suppose a membership proof σ_{mp} shows that $\tilde{S}$ corresponds to some S_π in the set $\mathbb{S}$. Then suppose the key $K' = z'G + s'_1 G_1$ 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, $G_1 == U$ must be true, and all $s'_x == 0$ for $x \geq 2$.
- The key K_π from S_π must contain the generators G and U like so: $K_\pi = z_\pi G + s_{\pi,1} U$.

- Since the composition proof shows that the prover knows z', according to the membership proof model they must also know z_π.
- We know that $s_{\pi,1} == s'_1$, which means $\tilde{K} = (1/s_{\pi,1}) * G$.
- 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 e-note. Let them convert it into S_π, give it a membership proof σ_{mp}, and give the resulting image S' a composition proof σ_{cp}. With this proof pair, the verifier can be confident that the transaction author owns an e-note in the set $\mathbb{S}$ (i.e. they know the keys $s_{\pi,1} = k^o_a$ and $z_\pi = k^o_b$ for some unknown index π), and that the linking tag $\tilde{K} = (1/k^o_a) * G$ is valid and can be used to check if the e-note at index π is unspent.

The transaction's e-note-image structure records $\tilde{S} = \{K', C'\}$ and $\tilde{K}$ for observers/verifiers to reference (recall Section 4.3).

4.6 Confidential Transactions

In accordance with the Confidential Transactions technique [16], 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 4.4), the commitment C' in an e-note-image will contain the same values $a_{\pi,1}, ..., a_{\pi,n}$ as the commitment C in the e-note being spent. In Section 4.6.1 we will prove that e-note commitments have the form $C = xG + aH$ as expected (i.e. prove that $H_1 == H$ and $a_{\pi,x} == 0$ for $x \geq 2$), and hence the amount a in $C' = v_c G + aH$ equals the amount in the original commitment.

Pedersen commitments have the 'homomorphic' property, which means if the sum of e-note-image commitments (inputs) equals the sum of new e-note commitments (outputs), then the sum of input amounts must equal the sum of output amounts. To achieve this, the sums of blinding factors must also match. Perform the following steps before constructing any membership proofs for a transaction.

1. Let a transaction spend $j \in 1, ..., m$ old e-notes and create $t \in 1, ..., p$ new ones. Let the masked commitments in e-note-images be denoted $C'_j = v_{c,j} G + a_j H$. Let the commitments in new e-notes be denoted $C_t = y_t G + b_t H$.
2. For $j \in 1, ..., m-1$, randomly select $v_{c,j} \in_R \mathbb{Z}_l$. For $t \in 1, ..., p$, randomly select $y_t \in_R \mathbb{Z}_l$.
3. Define $v_{c,m} = [\sum_{t=1}^{p} y_t] - [\sum_{j=1}^{m-1} v_{c,j}]$.

If the following equality holds for the transaction, then, within a security factor, there must be a balance on both generators (G and H) in the commitments.

$$\sum C'_j == \sum C_t$$

In conclusion, the amounts must balance between input and output e-notes.

Note: The values $t_{c,j} = v_{c,j} - x_j$ (recall Section 4.3) will be uniformly distributed because $v_{c,j}$ are uniformly distributed.

4.6.1 Range proofs

Since Pedersen commitments are elements of a cyclic group, it is conceivable that the amounts represented by individual elements are larger than the amount represented by a sum of those elements.[8] To properly convince observers that transaction amounts balance, transaction authors must provide a 'range proof' for each new e-note's commitment.

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

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

The maximum number of elements n allowed on one side of a balance check must be $n < l/(2^z - 1)$, otherwise range proofing those elements is pointless. Typically $z = 64$ and $l \approx 2^{252} - 2^{256}$, 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 e-note commitments need range proofs, not e-note-image commitments. Membership proofs should only reference e-notes from the ledger, which should all have range proofs, so it is guaranteed (within a security factor) that e-note-image commitments contain legitimate amounts.

Importantly, range proofing new e-note commitments 'locks in' the structure $C = xG + aH$. Since Seraphis membership proofs act on e-notes found in the ledger (i.e. as converted into $\mathbb{S}$), which should all have range proofs, it must be the case that in any Seraphis membership proof, $H_1 == H$ and $a_{\pi,x} == 0$ for $x \geq 2$.

4.6.2 Sender-receiver anonymity

If a transaction only has one input ($m = 1$) and all its y_t are known by an observer (e.g. they received all e-notes produced by the transaction), then the observer will know the value $v_{c,1} = [\sum_{t=1}^{p} y_t]$.

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

[9] At this time, Bulletproofs+ by Chung et. al [4] (based on Bulletproofs by Bünz et. al [2]) is thought to be the most efficient proving structure for range proofs, without a trusted setup.

However, even if the observer is the original sender of the e-note that the transaction author is spending, they won't necessarily know any more information about the transaction's input than if they weren't the original sender.

First note that the observer, by knowing all y_t, will presumably also know the total amount output by the transaction (assuming they know the transaction fee, if relevant), and hence will know the input amount a_1.

Second, even if they were the original sender, the input could have been sent to the transaction author by someone else. Despite knowing both x_1 and $v_{c,1}$, the observer has no way to know if the real input actually had a different blinding factor x'_1, since t_c is uniformly distributed at random.

There are two problems to consider.

1. If the amount a_1 is 'unusual' (i.e. unlikely to have been created by someone else), then the observer can guess with high probability of success that they created the e-note being spent, assuming that e-note was referenced by the input's membership proof. This problem may extend to multi-input transactions if the 'low bits' of the total amount are unusual (e.g. because one input has a fingerprint recorded in low bits of its amount value, and other inputs' amounts have low bits set to zero).

   Even if the amount isn't unusual, if the anonymity set size of membership proofs is relatively small, then there is a very low probability that the observer's e-note was randomly selected as a decoy and just happened to have the same amount as the real e-note being spent.

2. If $v_{c,1}$ is used as a secret input to a proof (e.g. a discrete log proof of the commitment to zero $C' - C$ with respect to G), then the observer may be able to guess and check the proof structure to see if $v_{c,1} = [\sum_{t=1}^{p} y_t]$ is in fact that secret input (depending on the proof structure used).

Both problems are mitigated or solved by including a 'change e-note' in each transaction, even if its amount must be zero.[10] A change e-note is an e-note the transaction author sends to himself if the total output amount of their transaction exceeds the amount they intend to send to other people (unavoidable if no combination of owned e-notes' amounts equals the intended total output amount of their transaction).

4.7 E-note address model

An e-note is sent from one person (a transaction author) to another (the recipient). To spend an e-note, the recipient must know k_a^o and k_b^o in the address $K^o = k_a^o * U + k_b^o * G$. However, it isn't feasible for the recipient to define both of those values in advance, for example by requesting that the transaction author place a pre-defined public key in the e-note address slot.

[10] There are niche cases where the first problem is unsolvable. For example, the sender could allow a 'low bit' fingerprint to propagate from an input to an output. The observer may also be able to infer, by the mere fact an e-note he created was referenced by a membership proof, that his e-note is being spent.

The reason for this is only one e-note with a given value k_a^o can ever be spent, since linking tags have the form $\tilde{K} = (1/k_a^o)G$. One trivial solution would be for recipients to randomly generate a new address $K = k_a U + k_a G$ for each e-note they want to receive. However, that is very inefficient and impractical.

Instead, we recommend the following e-note address model inspired by CryptoNote addresses.

1. Let each recipient have a *spend key* K^s for spending e-notes:
$$K^s = k_{a,recipient} U + k_{b,recipient} G$$
2. When sending an e-note, the sender generates a random scalar $k_{a,sender} \in_R \mathbb{Z}_l$.
3. The sender defines the e-note's one-time address based on the recipient's spend key:
$$K^o = k_{a,sender} U + K^s$$
$$K^o = (k_{a,sender} + k_{a,recipient}) * U + k_{b,recipient} G$$

E-note recipients must learn $k_{a,sender}$ in order to spend their e-notes. We discuss that topic in Section 5.4.

Comments

- Transaction authors cannot spend e-notes they created unless they know $\{k_{a,recipient}, k_{b,recipient}\}$. They cannot create linking tags unless they know $k_{a,recipient}$.
- Observers will not be able to associate a one-time address K^o with a spend key K^s unless they know the term $k_{a,sender} U$. We assume $k_{a,sender}$ is randomly generated every time an e-note 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 $(1/(k_{a,sender} + k_{a,recipient})) * G$. Even if a transaction author sends many e-notes to the same spend key K^s, and all of those e-notes are spent, they cannot use linear algebra on the resulting linking tags to associate those linking tags with the e-notes they created. This avoids the 'linearity' problem for fixed-base-point linking tag constructions noted by the CryptoNote whitepaper [26]. [[[formalize better, proof?]]]
- In the context of transaction protocols, multisignature schemes allow a group of N participants to 'co-own' e-notes (see [23] for example). Only a collaborating subgroup of participants of size M (M <= N) may spend any e-note. This is called 'M-of-N multisig'.

  Ideally, multisig schemes allow all participants to view the group's balance (amount of money currently owned). In Seraphis, this means being able to identify all owned e-notes (see Section 5.4) and recreate all the corresponding linking tags to check if they have been spent.

  Conveniently, the distinction between $k_{a,recipient}$ and $k_{b,recipient}$ makes our addressing model very 'multisig-friendly'. If all multisig participants have full knowledge of $k_{a,recipient}$, then they can easily recompute all linking tags to identify spent e-notes, and can identify newly

acquired e-notes and recover their amounts with the method described in Section 5.4. Meanwhile, $k_{b,recipient}$ can be divided among participants so proving ownership (Section 4.5) requires a collaborating subgroup of size M.

Importantly, in Seraphis, proving knowledge of $k_{b,recipient}$ only requires a discrete-log proof between K_{t2} and G from Section 4.5, where $K_{t2} = (x/y) * G = (t_k + k^o_{b,recipient})/(k^o_{a,sender} + k^o_{a,recipient}) * G$. For multisig, this can be achieved with a simple thresholded Schnorr signature (e.g. [**?**]), assuming t_k, $k^o_{a,sender}$, and $k^o_{a,recipient}$ are known by all M co-signers.

5 Considerations for implementers

There are a number of details to consider when implementing Seraphis in a real cryptocurrency. This section is comprised of 'recommendations' inspired by historical privacy-focused cryptocurrency implementations.

5.1 Coinbase e-notes

For a cryptocurrency to be widely adopted, observers should be able to verify that the total supply of money matches their expectations, based on looking at coinbase e-notes and transactions recorded in the ledger.[11]

However, Seraphis amounts are hidden using Pedersen commitments. How can transactions spend coinbase e-notes, while allowing coinbase amounts to be visible to observers? There are two approaches.

1. Construct coinbase e-notes the same as normal e-notes. Coinbase e-note authors must publicize the e-note commitments' blinding factors and amounts so observers can verify all coinbase e-note amount commitments are well-made.[12]

2. Let coinbase e-notes have a special format. Instead of recording amount commitments, they should record the amounts in cleartext. For a coinbase e-note to be referenced in a membership proof's input set $\mathbb{S}$, then it must be 'converted' into a normal e-note first.[13]

   Converting a coinbase e-note to a normal e-note is very simple.

   - Set the e-note's address equal to the coinbase e-note's address: $K^o_{e\text{-}note} = K^o_{coinbase}$.
   - Set the e-note's commitment equal to an unmasked commitment to the coinbase e-note's amount a: $C_{e\text{-}note} = aH$.

[11] Observers should also expect that coinbase e-notes only appear in the ledger when well-defined rules have been satisfied (e.g. they were created in the genesis block, or via PoW/PoS 'mining').
[12] This approach was taken in MobileCoin, where the 'origin' account's private keys were publicized [13].
[13] This approach was taken in Monero and its various forks [12].

When a transaction's membership proof references e-notes in the ledger, it is common to reference them by index. Verifiers look up those indices, then copy the e-notes they find into $\mathbb{S}$. If a verifier finds a coinbase e-note at a lookup index, they should convert it into a normal e-note before copying it into $\mathbb{S}$.[14]

If transaction spends a coinbase e-note, then its e-note-image's commitment will hide the amount involved even though the original amount had no blinding factor.

5.2 Transaction fees

Most (or perhaps all) cryptocurrencies have a so-called 'transaction fee'. Each transaction must send a small fee to a third-party. Fees disincentivize creating large numbers of transactions, which could cause the ledger to become excessively large. They also allow transaction authors to 'prioritize' their transactions. Transactions with high fees will typically be added to the ledger faster than those with low fees if the p2p network is congested.

To ensure fees are publicly verifiable, they are usually recorded in cleartext in transactions. Fee amounts are then converted into e-notes and added to the ledger at a later date. The rules around this conversion process are minutiae defined by each cryptocurrency.[15]

Transaction fees must be incorporated into amount balances. Verifiers can use the following simple procedure.

1. Convert the fee amount f into an unmasked commitment: fH. Require that $0 \leq f < 2^z$.
2. Test that amounts balance:
$$\sum_j C' \stackrel{?}{=} \sum_t C + fH$$

5.3 Non-prime groups

This paper requires $\mathbb{G}$ to be a prime group, however in practice it may be implemented as a prime subgroup of a non-prime group. One prominent example, used in CryptoNote [26] and its progeny, is the elliptic curve Ed25519 [1], which has order $8 * l$ (l is a prime number $\approx 2^{252}$). CryptoNote e-notes and proofs are designed to only use curve points from the subgroup of size l.

All uses of curve points in an implementation of Seraphis based on a non-prime group must take into account the possibility that a point recorded in a transaction may not be in the prime subgroup.

[14] In practice, transaction verifiers can store converted coinbase e-notes directly in/alongside a local copy of the ledger, so they don't have to be converted each time they are referenced by a transaction.

[15] In PoW cryptocurrencies, each block's miner typically adds the fee amounts from the block's transactions into their coinbase e-note (i.e. the output of a so-called 'miner transaction') as part of their 'block reward' (which usually includes newly minted money).

In particular, linking tags recorded in e-note-images *must* be points in the prime subgroup [7], since checking if a linking tag has appeared in the ledger usually involves a byte-wise lookup. There are several ways to ensure non-prime points are detected by transaction validators. From least to most efficient, they are:

- Test $l * \tilde{K} \stackrel{?}{=} I$, where I is the curve's identity element.
- Let the linking tag recorded in e-note-images (and the ledger) be $\tilde{K}_{record} = (1/h) * \tilde{K}$, where h is the curve's cofactor (8 in the case of Ed25519). To validate a transaction, compute $\tilde{K} = h * \tilde{K}_{record}$ before verifying the composition proof from Section 4.5.
- Use an encoding abstraction such as Ristretto [5] to ensure that all points recorded in a transaction (in e-notes, e-note-images, and proof elements) are in the prime subgroup.[16]

5.4 Information recovery

How can e-note owners discover the e-notes they own, read the amounts in those e-notes, reconstruct commitments in order to perform balance proofs in new transactions, and learn the sender keys $k_{a,sender}$ so they can construct linking tags?

The answer first pioneered by CryptoNote [26] for privacy-focused transaction protocols revolves around a Diffie-Hellman shared secret between the sender and receiver of an e-note.

1. Let potential recipients define their 'public addresses' as tuples $\{K^{DH}, K^{vr}, K^{s}\}$.
   (a) Diffie-Hellman base key: K^{DH} (for now, let this be an arbitrary key)
   (b) View-received key: $K^{vr} = k^{vr} K^{DH}$
   (c) Spend key: $K^{s} = k_{a,recipient} U + k_{b,recipient} G$
2. Suppose one or more potential recipients give their public addresses to a transaction author.
3. The author constructs p e-notes. For e-note $t \in 1, ..., p$ he does the following.
   (a) Generate a random 'e-note private key' $r_t \in_R \mathbb{Z}_l$.
   (b) Compute the e-note public key: $R_t = r_t K_t^{DH}$.
      - Store R_t in the e-note's memo field.
   (c) Compute the sender-receiver shared secret: $q_t = \mathcal{H}_1(r_t K_t^{vr})$.
   (d) Define the one-time address sender key as a function of q_t: $k_{a,sender,t} = \mathcal{H}_2(q_t)$.
   (e) Define the one-time address: $K_t^o = k_{a,sender,t} U + K_t^s$.
      - Store K_t^o in the e-note's address field.
   (f) Define the commitment blinding factor as a function of q_t: $y_t = \mathcal{H}_3(q_t)$.

[16] A Ristretto point will fail to decompress into a full elliptic curve point if it is not in the prime subgroup.

(g) Define the commitment: $C_t = y_t G + b_t H$.
- Store C_t in the e-note's amount commitment field.

(h) Encrypt/encode the e-note amount b_t using q_t: $enc\_amount_t = \texttt{enc}[q_t](b_t)$.
- Store enc_amount_t in the e-note's memo field.

4. Suppose a potential recipient sees an e-note with index t in a transaction. They want to check if they own it, then uncover as much information as possible.

(a) Compute the nominal sender-receiver shared secret: $q_t^{nom} = \mathcal{H}_1(k^{vr} R_t)$.

(b) Compute the nominal spend key: $K_t^{s,nominal} = K_t^o - \mathcal{H}_2(q_t^{nom}) * U$.
- If $K_t^{s,nominal}$ matches the spend key in the recipient's public address, then they own the e-note.

(c) Decode the amount: $b_t = \texttt{dec}[q_t^{nom}](enc\_amount_t)$.

(d) Compute the commitment blinding factor: $y_t = \mathcal{H}_3(q_t^{nom})$.

(e) Verify that the e-note's commitment can be reconstructed: $y_t G + b_t H \stackrel{?}{=} C_t$. If not, then the e-note is malformed and can't be spent.

Comments

- Basing information recovery on a Diffie-Hellman exchange between sender and recipient ensures $k_{a,sender}$, y_t, and enc_amount_t will be unknown to observers (within a security factor).
- Since q_t is computed from the 'view-received' key k^{vr}, only k^{vr} and K^s are required in order to view owned e-notes (important in Section 5.5).
- Commitment blinding factors y_t and the sender key $k_{a,sender,t}$ are created in the random oracle model, instead of being generated randomly. [[[justify better?]]]
- **Optimization**: If K^{DH} is the same between multiple recipients, then those recipients can share an e-note private key r and e-note public key R.[17]
  - If r is reused, then, to ensure each q_t is unique even if multiple e-notes have the same recipient, an index t can be used to further domain-separate the hash: $q_t = \mathcal{H}_1(rK_t^{vr}, t)$.
  - If all recipients other than the transaction author himself (e.g. if he has a change e-note) share a K^{DH}, then the transaction author can 'borrow' that K^{DH} by computing a temporary view-received key $K_{temp}^{vr} = k^{vr} K^{DH}$ for e-notes he is sending himself.

  This way, if all recipients have the same K^{DH}, only one e-note public key $R = rK^{DH}$ needs to be recorded in the transaction, and users searching for owned e-notes only have to compute $k^{vr} R$ once per transaction.[18]

[17] This optimization would not be useful in a cryptocurrency like MobileCoin where only e-notes and linking tags are stored in the ledger, and transactions are discarded. Without some kind of distinct 'transaction object', it isn't possible to associate a single value R with multiple e-notes (without replication).

[18] If only a strict subset of a transaction's recipients can share a K^{DH}, then, rather than producing one R value for

5.5 Addressing schemes

Up to this point user addressing has been 'open-ended'. The values $k^{vr}, K^{DH}, k_{a,recipient}, k_{b,recipient}$ were left as implementation details.

Here we will discuss two useful schemes for defining those values.

5.5.1 Terminology

First it is worth laying out some terms.

- **Account**: Let an account be a tuple of private keys $\{k^{vr}, k^{vs}, k^s\}$. Here k^{vs} is the 'view-spent' key, which we will elaborate on later.
- **Address**: Let an address be a generic term for an address tuple $\{K^{DH}, K^{vr}, K^s\}$ that a potential recipient may transmit to transaction authors for receipt of e-notes.
- **Normal address**: Let a normal address be an address generated 'directly' from the account keys. In other words, $\{K^{DH} = K^{DH}, K^{vr} = k^{vr}K^{DH}, K^s = k^{vs}U + k^sG\}$.
- **Subaddress**: Let a subaddress $i \in 1, ..., n$ be an 'alternative' address derived from the account keys [20, 10]. A subaddress $\{K^{DH,i}, K^{vr,i}, K^{s,i}\}$ should be statistically independent of its corresponding normal address $\{K^{DH}, K^{vr}, K^s\}$ (i.e. no observer should be able to determine they are based on the same account keys, within a security factor).

5.5.2 Address scheme variant 1

In address scheme variant 1, addresses are a two-key tuple $\{K^{vr}, K^s\}$ and the Diffie-Hellman base key is implicitly defined.

- **Normal addresses**

$$\begin{aligned} K^{DH} &= G \\ K^{vr} &= k^{vr}G \\ K^s &= k^{vs}U + k^sG \end{aligned}$$

- **Subaddresses**: For any subaddress index i.

$$\begin{aligned} K^{DH,i} &= K^{s,i} \\ K^{vr,i} &= k^{vr}K^{s,i} \\ K^{s,i} &= \mathcal{H}_4(k^{vr}, i) * U + K^s \end{aligned}$$

that subset and another R_t value for the other recipients, it may be better to produce a separate R_t for all recipients. The reason is a privacy concern. Namely, if different subsets of recipients share different R values, then observers will learn some information about the difference between recipients. In Monero, there is currently a proposal from this paper's author 'koe' to standardize e-note public key use [24], such that A) a transaction must have at least two outputs, B) if a transaction has two outputs then it may only have one R value, and C) if there are more than two outputs then there must be one R_t value per output. In MobileCoin, every transaction output must have its own R_t value [11].

There are several important details to take note of.

- A potential recipient must indicate to transaction authors whether their address is a normal address or a subaddress, so the correct K^{DH} value can be used.
- Addresses from the same account are statistically independent (within a security factor). [[[formalize/justify?]]]
- An account-holder can identify any e-note that they own with just the view-received private key k^{vr} (and normal address spend key K^s), regardless of if it was sent to a normal address or any subaddress (assuming the information-recovery approach in Section 5.4 is used). Importantly, when searching the ledger for owned e-notes, an account holder only needs to compute *one* Diffie-Hellman exchange per e-note (at most) to recover funds owned by any address in the account.

  Note: An e-note can only be identified as 'owned' if the user has a local record of the public spend key of the address that owns that e-note. In practice, users must pre-generate all subaddresses that might plausibly own an e-note (which can be done if you know k^{vr}, the normal address, and a list/range of plausible subaddress indices), and should only hand out subaddresses from that set for receipt of funds.
- The view-spent private key k^{vs}, in combination with q_t (computed using k^{vr}), is required in order to compute linking tags.
- The above two points mean users have access to three tiers of wallet permissions.

  1. **View received**: View e-notes received to the account. Can generate any subaddress.
     - **Requires**: k^{vr}, K^s
  2. **Balance recovery**: View received e-notes and identify which ones have been spent.
     - **Requires**: k^{vr}, k^{vs}, K^s
  3. **Full authority**: Balance recovery with the authority to spend e-notes owned by the account.
     - **Requires**: k^{vr}, k^{vs}, k^s

Adjustment: combine tiers 1 and 2

In practice it may be acceptable to merge tiers 1 and 2 (set $k^{vs} = k^{vr}$). The reason for this is anyone with an account's k^{vr} can identify e-notes in-bound to the account, which includes change e-notes.

If membership proofs rely on a fixed-size reference set construction (e.g. a ring signature or one-of-many proof), then when someone receives an e-note, they can look for intersections between the e-note's transaction's inputs' e-note reference sets and the set of prior e-notes received by the account. If the recipient believes the e-note they received was a change e-note, and sees that each of its transaction's inputs has one intersection with the owned e-note set, then it it likely that those intersections were all spent by the transaction.

Since the vast majority of transactions are likely to include a change e-note, and fixed-size reference sets in existing protocols are significantly smaller than the overall amount of transactions in the network (so the probability that someone else sends you an e-note and also references one of your owned e-notes in the same transaction is very low), this heuristic is likely powerful enough to identify most spent e-notes. In other words, it would make the distinction between k^{vr} and k^{vs} insignificant in practice.[19]

5.5.3 Janus attack

Variant 1 is not flawless. Unfortunately, the Janus attack [6] allows a malicious transaction author to discern if two subaddresses were derived from the same account.

1. Construct an e-note from components of two subaddresses.
   - Subaddress A: $K^{vr,A}, K^{s,A}$
   - Subaddress B: $K^{vr,B}, K^{s,B}$
   - Sender-receiver shared secret: $q_t = \mathcal{H}_1(r_t K^{vr,A})$
   - One-time address: $K^o = \mathcal{H}_2(q_t) * U + K^{s,B}$
   - E-note public key: $R_t = r_t K^{s,A}$
2. E-note recipient identifies they own the e-note.
   (a) Sender-receiver shared secret: $q_t^{nom} = \mathcal{H}_1(k^{vr} R_t)$
   (b) Nominal spend key: $K_t^{s,nominal} = K_t^o - \mathcal{H}_2(q_t^{nom}) * U$
   (c) If $K_t^{s,nominal} \stackrel{?}{=} K^{s,B}$, then the e-note is owned by subaddress B.

   However, in this case, R_t is based on subaddress A!
3. If the e-note recipient notifies the sender that they got an e-note, then the sender will know that subaddresses A and B belong to the same account (i.e. were constructed from the same private key tuple $\{k^{vr}, k^{vs}, k^s\}$).

5.5.4 Address scheme variant 2

Address scheme variant 2 is designed to mitigate the Janus attack. Addresses are a three-key tuple $\{K^a, K^{vr}, K^s\}$, where $K^{DH} = K^a$ is a so-called 'ancillary key'.

[19] If the membership proof reference set size is variable (i.e. equal to 'all the e-notes in the ledger'), then this heuristic would be useless. In that case, the distinction between k^{vr} and k^{vs} could be nice to have. Since the development of an efficient membership proof with variable reference set sizes could occur at any time, a conservative implementation of Seraphis may want to maintain the distinction between k^{vr} and k^{vs} 'just in case'.

- **Normal addresses**

$$K^a = k^{vr} G$$
$$K^{vr} = k^{vr} K^a$$
$$K^s = k^{vs} U + k^s G$$

- **Subaddresses**: For any subaddress index i.

$$K^{a,i} = \mathcal{H}_4(k^{vr}, i) * G + K^a$$
$$K^{vr,i} = k^{vr} K^{a,i}$$
$$K^{s,i} = \mathcal{H}_4(k^{vr}, i) * U + K^s$$

Compared to variant 1, only K^{DH} is defined differently. This new K^a value feeds into a very important rule change for e-note construction.

- When constructing an e-note, define its commitment blinding factor as $y_t = \mathcal{H}_3(q_t, r_t G)$ and its e-note public key as $R_t = r_t K_t^a$.
- After identifying that an e-note is owned by address $\{K^a, K^{vr}, K^s\}$, obtain its ancillary private key k^a (i.e. $k^a = k^{vr}$ or $k^{a,i} = \mathcal{H}_4(k^{vr}, i) + k^{vr}$) and do the following.
  1. Compute $R_{base,t}^{nom} = (1/k^a) * R_t$.
  2. Compute $y_t = \mathcal{H}_3(q_t^{nom}, R_{base,t}^{nom})$.
  3. Verify that the e-note's commitment can be reconstructed: $y_t G + b_t H \stackrel{?}{=} C_t$. If not, then the e-note is malformed and can't be spent.

Adding this rule change mitigates the Janus attack because an e-note owner will only be able to recompute C_t if the correct y_t is used, which requires the correct $R_{base,t}^{nom}$, which is only obtained if the k^a used corresponds to the key K^a used to compute R_t. Critically, the e-note owner will select k^a based on which of their addresses seems to own the e-note, which is the address where $K^s == K_t^{s,nominal}$. In other words, if R_t is based on an ancillary key from a different address/subaddress than the one whose spend key was used to construct the one-time address, then recomputing C_t will fail.

Adjustment 1: tier 1 identifies owned outputs without amounts

In variant 1, the first wallet permission tier allows a wallet to fully view all received e-notes, including the amounts contained. However, in variant 2 it is possible to adjust the address scheme so wallet tier 1 can *only* identify owned e-notes, and cannot recover amounts. This may be relatively more desirable because tier 1 is mostly useful for 'outsourcing' the task of identifying owned e-notes to a third party, who doesn't need to know the amounts involved.

1. Let an account be the tuple of private keys $\{k^{vr}, k^{vb}, k^s\}$, where k^{vb} is the 'view-balance' key (replacing the 'view-spent' key).
2. Define the normal ancillary key as $K^a = k^{vb} G$. The spend key is $K^s = k^{vb} U + k^s G$. There are no other changes to variant 2 addresses.

3. Define $q_t^{vr} = \mathcal{H}_1(r_t K^{vr})$ and $q_t^{vb} = \mathcal{H}_5(q_t^{vr}, r_t G)$.
4. Use q_t^{vr} when defining K^o, and use q_t^{vb} in all other places (e.g. $y_t = \mathcal{H}_3(q_t^{vb,nom})$, $enc\_amount_t = \texttt{enc}[q_t^{vb}](b_t)$).

This way users have to compute $R_{base,t}^{nom} = (1/k^{vb}) * R_t$ to decode the amounts in e-notes they own. The value k^{vb} is called the 'view-balance' key because it is required both to decode amounts and compute linking tags.

The one risk to this approach is that tier 1 wallets would be unable to detect the Janus attack.

Adjustment 2: tier 1 identifies owned and spent outputs without amounts

The address scheme can be further adjusted so the first permission tier can identify both owned and spent outputs. Unfortunately, like the previous adjustment, tier 1 wallets would be unable to detect the Janus attack.

Take the previous adjustment and add the following rules.

1. Replace the 'view-received' key k^{vr} with a 'view-outputs' key k^{vo}.
2. Let the spend key be $K^s = k^{vo}U + k^s G$.

5.6 Other recommendations

The above recommendations are not an exhaustive list. Here are some other ideas we think implementers should consider.

- **Semantic constraints**: Transaction validation rules should contain as many 'semantic constraints' as possible. A semantic constraint is one that limits variance in how a transaction may be constructed, without affecting the underlying security model. For example, how inputs and outputs are sorted, byte serialization, memo field format/usage, etc.

  Reducing/eliminating semantic variance reduces the likelihood of 'implementation fingerprinting'. If two transaction-builder implementations use different semantic conventions, then observers can easily identify what software was used to make a given transaction. This can have undesirable privacy implications for users.

- **Decoy selection**: Membership proofs might only reference a small set of e-notes in the ledger. If 'decoy' e-notes are not selected effectively, then observers may be able to use heuristics to gain an advantage when trying to guess the real spend in a transaction input.

  Pure random selection of decoys is weak to the 'guess-newest' heuristic, where the 'newest' e-note referenced by a membership proof is most likely to be the real spend. Selecting from a gamma distribution instead is thought to best mimic the true spend distribution, and selecting 'bins' (clumps) of e-notes mitigates analysis that uses circumstantial timing knowledge about a transaction. [17]

- **View tag optimization**: To identify an owned e-note, multiple group operations are required (Section 5.4). Typically, group operations are quite expensive, so the amount of time it takes to scan the ledger for owned e-notes is a function of how many group operations are executed.

  One possible optimization is to include the first one byte of the value q_t in e-note memos as a so-called 'view tag' [25]. Before trying to compute $K_t^{s,nominal} = K_t^o - \mathcal{H}_2(q_t^{nom}) * U$ for a given e-note, users can first compute the view tag and check if it matches the value recorded in the e-note. If it does not match, then the step to get $K_t^{s,nominal}$ can be skipped.

- **Transaction chaining**: Seraphis, like other transaction protocols inspired by RingCT, does not include any advanced 'scripting' capabilities. This means transactions can only be constructed in limited scenarios. However, unlike RingCT and other protocols, a Seraphis implementation can be designed to permit 'transaction chaining', which is beneficial for atomic swap protocols [18] (among other techniques).

  Transaction chaining is the ability to construct a transaction B that spends an e-note produced by transaction A, before A has been added to the ledger. For Seraphis to support transaction chaining, the following approach can be used.

  1. The ownership/unspentness proof structure should be independent from membership proofs. This means membership proof material should not be signed by the ownership/unspentness proof structure, and the two proof structures should not share any Fiat-Shamir challenges.
  2. When constructing a transaction, first construct a 'partial transaction'. This partial transaction contains everything *except* the inputs' membership proofs. The values $t_{c,i}$ and $t_{k,i}$ should be cached alongside the partial transaction.
  3. When you want to finish a partial transaction, use the cached $t_{c,i}$ and $t_{k,i}$ values to construct membership proofs.

  Taking this approach permits transaction chaining because the second step can be executed even if the inputs being spent don't exist in the ledger. Moreover, the third step can be easily offloaded to a third party. Knowledge of the values $t_{c,i}$ and $t_{k,i}$ only allows you to identify the true spends in membership proofs, which is presumably acceptable in any situation where transaction chaining is desired.

6 Efficiency

- sample implementation - squashed vs non-squashed - aggregate CSAG & multibase/multikey signature for ownership/unspent proofs across all inputs - Grootle 1-of-many for membership proofs - bulletproofs+ range proofs

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] 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.pdf [Online; accessed 10/28/2018].

[3] 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].

[4] 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].

[5] Henry de Valance, Isis Lovecruft, and Tony Arcieri. Ristretto. https://ristretto.group/ristretto.html [Online; accessed 10/05/2020].

[6] Justin Ehrenhofer and knacc. Advisory note for users making use of subaddresses, October 2019. https://web.getmonero.org/2019/10/18/subaddress-janus.html [Online; accessed 01/02/2020].

[7] Riccardo "fluffypony" Spagni and luigi1111. Disclosure of a Major Bug in Cryptonote Based Currencies, May 2017. https://getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html [Online; accessed 04/10/2018].

[8] 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].

[9] Aram Jivanyan. Lelantus: Towards Confidentiality and Anonymity of Blockchain Transactions from Standard Assumptions. Cryptology ePrint Archive, Report 2019/373, 2019. https://eprint.iacr.org/2019/373.pdf [Online; accessed 03/04/2020].

[10] kenshi84. Subaddresses, Pull Request #2056, May 2017. https://github.com/monero-project/monero/pull/2056 [Online; accessed 02/16/2020].

[11] koe and Kurt M. Alonso. Mechanics of MobileCoin — First Edition, 2021. https://github.com/UkoeHB/Mechanics-of-MobileCoin [Online; accessed 07/29/2021].

[12] koe, Kurt M. Alonso, and Sarang Noether. Zero to Monero — Second Edition, April 2020. https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf [Online; accessed 10/03/2020].

[13] koe and Sara Drakeley. MobileCoin Governance, Fees, and Supply. https://medium.com/mobilecoin/mobilecoin-governance-fees-and-supply-60c11782eb0a [Online; accessed 06/15/2021].

[14] 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.pdf [Online; accessed 03/04/2020].

[15] 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.pdf [Online; accessed 03/04/2020].

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

[17] 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.

[18] Mahdi H. Miraz and David C. Donald. Atomic cross-chain swaps: Development, trajectory and potential of non-monetary digital token swap facilities. *CoRR*, abs/1902.04471, 2019. https://arxiv.org/ftp/arxiv/papers/1902/1902.04471.pdf [Online; accessed 07/28/2021].

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

[20] Sarang Noether and Brandon Goodell. An efficient implementation of Monero subaddresses, MRL-0006, October 2017. https://web.getmonero.org/resources/research-lab/pubs/MRL-0006.pdf [Online; accessed 04/04/2018].

[21] 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.pdf [Online; accessed 03/04/2020.

[22] 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].

[23] 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].

[24] UkoeHB. Proposal/Request: Update Supplementary Transaction Content, Issue #6456, April 2020. https://github.com/monero-project/monero/issues/6456 [Online; accessed 10/11/2020].

[25] UkoeHB. Reduce scan times with 1-byte-per-output 'view tag', Issue #73, April 2020. https://github.com/monero-project/research-lab/issues/73 [Online; accessed 11/23/2020].

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

[27] 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.pdf [Online; accessed 03/04/2020].

A Squashed e-note model

The squashed e-note 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.

A.1 Model

1. Let $G_1, ..., G_n$ be generators whose discrete logarithm relations with each other, with G, and with H are unknown.
2. Let $\mathbb{S}$ represent a set of tuples $\{K_i, C_i\}$, where
$$K_i = z_i G + s_{i,1} G_1 + s_{i,2} G_2 + ... + s_{i,n} G_n$$
$$C_i = x_i G + a_{i,1} H$$
3. Let $\mathbb{S}^t$ represent a set of 'transformed' tuples $\{K_i^t, C_i^t\}$, where
$$K_i^t = \mathcal{H}_6(K_i, C_i) * K_i$$
$$C_i^t = C_i$$
4. Perform a range proof on each $C_i^t \in \mathbb{S}^t$ (recall Section 4.6.1).
5. Let $\mathbb{Q}$ represent a set of squashed tuples $\{Q_i\}$, where
$$Q_i = K_i^t + C_i^t$$
6. Let $\tilde{S}$ represent a tuple $\{K', C'\}$, where
$$K' = z' G + s'_1 G_1 + s'_2 G_2 + ... + s'_n G_n$$
$$C' = x' G + a'_1 H$$
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 π is unknown to the verifier, such that:
   (a) The discrete log relation of $\tilde{Q} - Q_\pi = [(z' + x') - (\mathcal{H}_6(K_\pi, C_\pi) * z_\pi + x_\pi)] * G$ with respect to G is known.
9. Perform a range proof on C'.
10. Demonstrate knowledge of $z', s'_1, ..., s'_n$ such that $K' = z' G + s'_1 G_1 + s'_2 G_2 + ... + s'_n G_n$.

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 [15] without linking) or Groth/Bootle one-of-many proof on the set $\{\tilde{Q} - Q\}$. The efficiency implications are discussed in Section 6, which compares possible instantiations of Seraphis using the two models.

A.2 Requirement satisfaction

A.2.1 Underlying membership proof model

We argue that the squashed e-note model satisfies the underlying membership proof model.

Let $\mathbb{S}^t$ be the input to the underlying model. We will show that the following requirements, adapted from Section 4.4, are met.

1. Demonstrate that, within a security parameter k, $\tilde{S}$ corresponds to some $S^t_\pi \in \mathbb{S}^t$, where π is unknown to the verifier, such that:
   (a) $s'_j == \mathcal{H}_6(K_\pi, C_\pi) * s_{\pi,j}$ for $j \in 1, ..., n$
   (b) $a'_1 == a_{\pi,1}$
   (c) The prover must have knowledge of $\mathcal{H}_6(K_\pi, C_\pi) * z_\pi$.

Observe the following.

1. The prover must know $[(z' + x') - (\mathcal{H}_6(K_\pi, C_\pi) * z_\pi + x_\pi)]$ and z' to satisfy the squashed e-note model, and x' and x_π to construct the range proofs on C' and C^t_π. Therefore the prover must know $\mathcal{H}_6(K_\pi, C_\pi) * z_\pi$. The point $\mathcal{H}_6(K_\pi, C_\pi)$ is considered 'public knowledge', so the prover must also know z_π.
2. Range proofing C_π and C' means they have the form $xG + aH$, implying they contain no $G_1, ..., G_n$ components. Therefore, demonstrating discrete log with respect to G in the commitment to zero $\tilde{Q} - Q_\pi$ means it must be the case that $s'_j == \mathcal{H}_6(K_\pi, C_\pi) * s_{\pi,j}$ for $j \in 1, ..., n$.
3. Suppose K_i has the form $K_i = z_i G + s_{i,1}G_1 + s_{i,2}G_2 + ... + s_{i,n}G_n + bH$. Since the model demonstrates that K' does not contain any H components, and the commitment to zero $\tilde{Q} - Q_\pi$ means all non-G components balance in those two points, it must be the case that $C' = x'G + (a_{i,1} + \mathcal{H}_6(K_\pi, C_\pi) * b) * H$. However, the term $\mathcal{H}_6(K_\pi, C_\pi)$ is both uniformly distributed and implicitly dependent on the values $a_{i,1}$ and b, so the value $a'_1 = (a_{i,1} + \mathcal{H}_6(K_\pi, C_\pi) * b)$ will be uniformly distributed in $\mathbb{Z}_l$ (assuming b is non-zero). Since the range proof on C' means that a'_1 must be in the range $[0, ..., 2^z - 1]$, if $b \neq 0$ then the probability that a range proof on a'_1 can succeed is $2^z/l$. This means $a'_1 == a_{i,1}$ can be assumed to be true within the security parameter k if $1/k > 2^z/l$.[20] [[[formalize better?]]]

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

A.2.2 Seraphis structure

1. Note that a range proof on C_i is equivalent to a range proof on C_i^t since $C_i^t = C_i$.

2. In Seraphis, range proofs on C_i are created when e-notes are first constructed (i.e. as outputs of a transaction). This means transaction authors, who reference C_i as part of transaction membership proofs, will not themselves construct range proofs on C_i. As a consequence, transaction authors won't necessarily know x_π.

   However, we do not consider this a security problem. When a transaction author sends an e-note to a recipient, they are 'delegating spend authority' to that recipient. In the context of membership proofs (i.e. step 8 above), any person who knows $(\mathcal{H}_6(K_\pi, C_\pi) * z_\pi + x_\pi)$ must have learned that value by cooperating with the original transaction author. Therefore, whoever constructs a membership proof for an e-note in the above model must be acting as a 'proxy' of that e-note's original author. Since the author knows x_π (recall that they must have range proofed C_π), the value $\mathcal{H}_6(K_\pi, C_\pi) * z_\pi$ can be derived from the combined knowledge of the author and his proxy.

   To gain further confidence in this roundabout security proof, consider the following.

   (a) An e-note's author cannot create a membership proof for that e-note (i.e. complete step 8 in the model) unless they know $\mathcal{H}_6(K_\pi, C_\pi) * z_\pi$. This trivially follows from the fact they know x_π as the one who range proved C_π, and they must know $\mathcal{H}_6(K_\pi, C_\pi) * z_\pi + x_\pi$ as the one who performed step 8 in the above model.

   (b) Suppose $p_1 + p_2 = \mathcal{H}_6(K_\pi, C_\pi) * z_\pi + x_\pi$ (p_1 or p_2 could be zero). Let the e-note author know x_π and p_2; let the prover of step 8 know p_1. In order to complete step 8 from above, the prover must learn p_2. Can the prover acquire the pair p_1, p_2 without collaborating with someone who knows x_π?

   [[[formal proof? this is giving me a lot of trouble]]]

3. Seraphis linking tags are computed from the output of a membership proof, namely the point K' in $\tilde{S}$. However, K' in the squashed e-note model applied to Seraphis has the form $K' = t_k G + \mathcal{H}_6(K_\pi^o, C_\pi) * [k_a^o * U + k_b^o * G]$. This means linking tags will have the form $\tilde{K} = (1/(\mathcal{H}_6(K_\pi^o, C_\pi) * k_a^o)) * G$ instead of $\tilde{K} = (1/k_a^o) * G$. Since $\mathcal{H}_6(K_\pi^o, C_\pi)$ is uniquely defined by each e-note (and not malleable), these modified linking tags are also unique per e-note. In other words, only one linking tag can be produced for each e-note in the ledger.

   There is one interesting side-effect. The value $\mathcal{H}_6(K_\pi^o, C_\pi)$ is dependent on the e-note commitment C_π, so it is possible for two e-notes with the same address K^o to produce different linking tags if they have different commitments.

4. 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.

A.3 Practical considerations

1. Transaction verifiers can pre-compute steps 3 and 5 from the above model for every e-note in the ledger. The squashed tuples Q_i can be stored in anticipation of new transactions that may require them.

2. In transaction chaining (Section 5.6), only step 8 in the above model needs to be deferred (assuming Q_i values have been precomputed, and range proofs on C_i already exist).

B Generalized Schnorr signatures

- generalized Schnorr signature - signing with multiple private keys in parallel (shared challenge) - concise aggregation on common base points (compressing private keys) - multi-base signing (same DL across multiple bases)

C Groth/Bootle one-of-many proofs

- Grootle (parallel vs non-parallel)