A Report on Seraphis

coinstudent2048

September 22, 2021

Abstract

This document contains a concise description of Seraphis [9], a novel privacy-preserving transaction protocol abstraction, and its security model. This document will also serve as a suggestion to the organization of the contents of the final Seraphis paper.

1 Introduction

In a p2p (peer-to-peer) electronic cash system, the entire supply of currency exists as a digital record that 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*)... ...An unfortunate consequence of cryptocurrencies being decentralized is that the ledger is *public*, implying that all e-notes and transaction events are public knowledge. If amounts are in cleartext, addresses are trivially traceable, and e-notes to be spent are referenced directly, then observers can discern many details about users' finances.

Hence, several confidential transaction protocols have been proposed to ensure financial privacy and currency fungibility. [List privacy technologies and limitations]

1.1 Our contribution

We introduce Seraphis, privacy-preserving transaction protocol abstraction, which means the unlike previous... Moreover...

1.2 Acknowledgments

Lelantus Spark

2 Preliminaries

2.1 Public parameters

Let λ be the security parameter. Let $\mathbb{G}$ be a prime order group based on λ where the Discrete Logarithm (DL), Computational Diffie-Hellman (CDH), and Decisional Diffie-Hellman (DDH) problems are hard, and let $\mathbb{F}$ be its scalar field. Let G, H, X, U be generators of $\mathbb{G}$ with unknown DL relationship to each other. Note that these generators may be produced using public randomness. Let $a_{max} \in \mathbb{F}$ be the maximum amount value. Let $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$ 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. All these public parameters are collected as pp, and we now define the setup algorithm: $pp \leftarrow \mathsf{Setup}(1^\lambda)$. Setup is implicitly executed by all players involved in the beginning, hence it can be omitted in protocol descriptions.

The notation $\xleftarrow{\$}$ will be used to denote for a uniformly randomly chosen element, and $(1/x)$ for the modular inverse of $x \in \mathbb{F}$. Lastly, we use additive notation for group operations.

2.2 E-notes and e-note images

Definition 2.1. *An* **e-note** *for scalars $k_a^o, k_b^o, a \in \mathbb{F}$ is a tuple (C, K^o, m) such that $C = xG + aH$ for $x \xleftarrow{\$} \mathbb{F}$, $K^o = k_a^o X + k_b^o U$, and m is an arbitrary data.*

C is called the **amount commitment** for the amount a with blinding factor x, K^o is called the **one-time address** for (one-time) private keys k_a^o and k_b^o (the o superscript indicates "one-time"), and m is the **memo field**. We say that someone *owns* an e-note if they know the corresponding scalars $k_a^o, k_b^o, a, x \in \mathbb{F}$.

Definition 2.2. *An* **e-note image** *for an e-note (C, K^o, m) is a tuple $(C', K'^o, \tilde{K})$ such that*

$$\begin{aligned} C' &= t_c G + C \\ &= (t_c + x)G + aH \\ &= v_c G + aH \ , \\ K'^o &= t_k G + K^o \\ &= t_k G + k_a^o X + k_b^o U \ , \ \textit{and} \\ \tilde{K} &= (k_b^o / k_a^o) U \end{aligned}$$

for $t_c, t_k \xleftarrow{\$} \mathbb{F}$ and independent to each other.

C' is called the **masked amount commitment**, K'^o is called the **masked address**, and $\tilde{K}$ is called the **linking tag**.

Definition 2.3. *A* **receiver address** *is a tuple (K^{dh}, K^v, K^s) such that $K^{dh} \in \mathbb{G}$, $K^v = k^v K^{dh}$, and $K^s = k_a^s X + k_b^s U$.*

K^{dh} is called the **Diffie-Hellman base public key**, the v superscript indicates "view", and the s superscript indicates "spend". The reason for the name of K^{dh} will be clear in the next subsection, while the reason for the names of superscripts will be discussed in the addressing schemes (Subsection 4.1). We say that someone *owns* a receiver address if they know the corresponding scalars $k^v, k_a^s, k_b^s \in \mathbb{F}$.

2.3 Authenticated symmetric encryption scheme

We require the use of an authenticated symmetric encryption scheme. The Diffie-Hellman base public key enables shared secrets between the sender and the receiver, which can be used to produce the key for encryption and the authentication tag. We denote the encryption and decryption of data x with the input k for Key Derivation Function (KDF) as $\texttt{enc}[k](x)$ and $\texttt{dec}[k](x)$, respectively. We put overlines (e.g. $\overline{x}$) to indicate encrypted data.

The required security properties for application to Seraphis are described in Section 5.2.

3 A Seraphis transaction

We now describe a simple Seraphis transaction. This will be used as the basis for further instantiations and modifications (Section 4) and for the security model (Section 5).

Suppose that Alice would send $a_t \in \mathbb{F}$ amount of funds to Bob. Alice owns a set of e-notes $\{(C_i, K_i^o, m_i)\}_{i=1}^n$ with a total amount of $\left(\sum_{i=1}^n a_i\right) \geq a_t$, all *connected* to a receiver address $(K_{ali}^{dh}, K_{ali}^v, K_{ali}^s)$. This "connection" will be elaborated later on. On the other hand, Bob owns a receiver address $(K_{bob}^{dh}, K_{bob}^v, K_{bob}^s)$. For Bob to receive the funds, he will now send his receiver address to Alice. Alice will actually send funds to two addresses: to Bob's and to herself (for the "change" $a_c = \sum_{i=1}^n a_i - a_t$ *even if* $a_c = 0$). Hence, Alice must create 2 new e-notes. She starts the transaction by doing the following:

1. Generate $r_{ali}, r_{bob} \xleftarrow{\$} \mathbb{F}$ and independent to each other.
2. Compute $R_{ali} = r_{ali} K_{ali}^{dh}$ and $R_{bob} = r_{bob} K_{bob}^{dh}$, then store R_{ali} and R_{bob} to new (empty) memos m_{ali} and m_{bob}, respectively. The name for K^{dh} should now be clear.

3. Compute the sender-receiver shared secrets $q_{ali} = \mathcal{H}_1(r_{ali}K^v_{ali})$ and $q_{bob} = \mathcal{H}_1(r_{bob}K^v_{bob})$.

4. Compute the one-time addresses $K^o_{ali} = \mathcal{H}_2(q_{ali})X + K^s_{ali}$ and $K^o_{bob} = \mathcal{H}_2(q_{bob})X + K^s_{bob}$. It is easy to see that $\mathcal{H}_2(q_{ali})$ and $\mathcal{H}_2(q_{ali})$ are uniformly random in the random oracle model.

5. Compute the amount commitments $C_{ali} = \mathcal{H}_3(q_{ali})G + a_cH$ and $C_{bob} = \mathcal{H}_3(q_{bob})G + a_tH$. It is easy to see that the blinding factors $\mathcal{H}_3(q_{ali})$ and $\mathcal{H}_3(q_{bob})$ are uniformly random in the random oracle model.

6. Encrypt the amounts: $\overline{a_c} = \texttt{enc}[q_{ali}](a_c)$ and $\overline{a_t} = \texttt{enc}[q_{ali}](a_t)$, and store $\overline{a_c}$ and $\overline{a_t}$ to memos m_{ali} and m_{bob}, respectively.

Alice now has two new e-notes: $\texttt{enote}_{ali} = (C_{ali}, K^o_{ali}, m_{ali})$ and $\texttt{enote}_{bob} = (C_{bob}, K^o_{bob}, m_{bob})$. These will then be stored to a new (empty) *whole transaction* T. Other objects that will be stored to the whole transaction are from proving systems, which can be executed in *any* order. Proving systems are discussed in the next subsections.

For specific instances of Seraphis, there might be changes in some parts of the above steps, and by reflection, in some parts of the Receipt. Here are some notable changes:

- For some addressing schemes, the input to $\mathcal{H}_2$, the input to $\mathcal{H}_3$, and the key for both `enc` and `dec` may be constructed differently and different to each other. Nevertheless, these inputs and key must still be sender-receiver shared secrets.

- A Seraphis transaction can easily have multiple receivers aside from Bob, which implies that Alice will create more than 2 new e-notes. See Subsection 5.6 for a discussion of how this affects the security model.

- A Seraphis transaction can be collaboratively constructed by multiple players. This is the subject of the so-called "proof dependency" (Subsection 4.3). Also see Subsection 5.6 for a discussion of how this affects the security model.

3.1 Ownership and unspentness proofs

For each of Alice's owned e-notes in $\{(C_i, K^o_i, m_i)\}^n_{i=1}$, Alice must do the following:

1. If the masked address K'^o_i is already in the e-note image $\texttt{enimg}_i$ in T, then go to next step. Else generate K'^o_i from (C_i, K^o_i, m_i) as per definition, and insert it to $\texttt{enimg}_i$ in T.

2. If the linking tag $\tilde{K}_i$ is already in $\texttt{enimg}_i$ in T, then go to next step. Else generate $\tilde{K}_i$ from (C_i, K^o_i, m_i) as per definition, and insert it to $\texttt{enimg}_i$ in T.

3. Prepare the proof transcripts $\Pi_{\text{o\&u},i}$ for a non-interactive proving system for the following relation:

$$\{(G, X, U, K'^o_i, \tilde{K}_i \in \mathbb{G}; t_{k,i}, k^o_{a,i}, k^o_{b,i} \in \mathbb{F}) : k^o_{a,i} \neq 0 \wedge K'^o_i = t_{k,i}G + k^o_{a,i}X + k^o_{b,i}U \wedge \tilde{K}_i = (k^o_{b,i}/k^o_{a,i})U\}$$

4. Append $\Pi_{\text{o\&u},i}$ to $(\texttt{enimg}_i, \ldots)$ in T.

Aside from verifying the proof transcripts, the Verifier must confirm that the linking tags do not yet appear in the ledger.

The required security properties for application to Seraphis are described in Section 5.1. We describe our recommended composition proving system in Appendix A, and provide proof that it satisfies the required security requirements.

3.2 Amount balance

For each of Alice's owned e-notes in $\{(C_i, K^o_i, m_i)\}^n_{i=1}$, Alice must do the following:

1. If the masked amount commitment C'_i is already in $\texttt{enimg}_i$ in T, then exit this subsection. Else generate C'_i from (C_i, K^o_i, m_i) as per definition, *except* for $i = n$. For the case of $i = n$, set

$$v_{c,n} = \mathcal{H}_3(q_{ali}) + \mathcal{H}_3(q_{bob}) - \sum_{i=1}^{n-1} v_{c,i}.$$

   Note that the value of $v_{c,n}$ is still uniformly random because the values of $t_{c,i}$ for $i \in \{1, \ldots, n-1\}$ are uniformly random.

2. Insert C'_i to $\texttt{enimg}_i$ in T.

The generation of $v_{c,n}$ is as such so that the Verifier can verify the amount balance $\sum_{i=1}^{n} C'_i = C_{ali} + C_{bob}$.

3.3 Membership proofs

For each of Alice's owned e-notes in $\{(C_i, K^o_i, m_i)\}_{i=1}^n$, Alice must do the following:

1. If the masked amount commitment C'_i is already in $\texttt{enimg}_i$ in T, then go to next step. Else generate C'_i from (C_i, K^o_i, m_i) exactly like in Step 1 of Subsection 3.2, and insert it to $\texttt{enimg}_i$ in T.

2. If the masked address K'^o_i is already in $\texttt{enimg}_i$ in T, then go to next step. Else generate K'^o_i from (C_i, K^o_i, m_i) as per definition, and insert it to $\texttt{enimg}_i$ in T.

3. Collect $s-1$ number of random e-notes from the ledger and add her owned (C_i, K^o_i, m_i), for a total of s e-notes. The number s is called the **anonymity size**.

4. For each e-note in the collection (of size s), extract only the amount commitment and one-time address like this: (C_j, K^o_j). Then arrange the s e-notes in random positions. Alice now has an array (of length s) of pairs: $\mathbb{S}_i = \{(C_j, K^o_j)\}_{j=1}^s$, which is called the **ring**. Its elements (C_j, K^o_j) are called the **ring members**.

5. Prepare the proof transcripts $\Pi_{\text{mem},i}$ for a non-interactive proving system for the following relation:

$$\{(G, C'_i, K'^o_i \in \mathbb{G}, \mathbb{S}_i \subset \mathbb{G}^2; \pi_i \in \mathbb{N}, t_{c,i}, t_{k,i} \in \mathbb{F}) : 1 \leq \pi_i \leq s \wedge C'_i - C_{\pi_i} = t_{c,i}G \wedge K'^o_i - K^o_{\pi_i} = t_{k,i}G\}$$

6. Append $(\mathbb{S}_i, \Pi_{\text{mem},i})$ to $(\texttt{enimg}_i, \ldots)$ in T.

The required security properties for application to Seraphis are described in Section 5.3. Specific proving systems satisfying the requirement include CSAG (CLSAG [5] without linking) and One-out-of-Many proving system adapted from Groth and Bootle *et al.* [6, 1].

3.4 Range proofs

For the new e-notes $\texttt{enote}_{ali}$ and $\texttt{enote}_{bob}$, Alice must do the following:

1. Prepare the respective proof transcripts $\Pi_{\text{ran},ali}$ and $\Pi_{\text{ran},bob}$ for a non-interactive proving system for the following relation:

$$\{(G, H, C \in \mathbb{G}, a_{max} \in \mathbb{F}; x, a \in \mathbb{F}) : C = xG + aH \wedge 0 \leq a \leq a_{max}\}$$

2. Store $\Pi_{\text{ran},ali}$ and $\Pi_{\text{ran},bob}$ to T.

The required security properties for application to Seraphis are described in Section 5.1. Specific proving systems satisfying the requirement include Bulletproofs [2] and Bulletproofs+ [3].

3.5 Receipt

Once the construction of T is completed, Alice sends it to the network. Its contents must now be

$$T = (\texttt{enote}_{ali}, \texttt{enote}_{bob}, \Pi_{\text{ran},ali}, \Pi_{\text{ran},bob}, \{(\texttt{enimg}_i, \Pi_{\text{o\&u},i}, \mathbb{S}_i, \Pi_{\text{mem},i})\}_{i=1}^{n}).$$

Suppose that the Verifier successfully verified T, hence T is now stored in the ledger. When Bob scans the ledger for new transactions, he must do the following for every T he encounters:

1. Get a new e-note (C, K^o, m) in T. Note that m contains $(R, \overline{a})$ (see the beginning of this whole section).
2. Compute the nominal sender-receiver shared secret: $q_{nom} = \mathcal{H}_1(k^v_{bob}R)$.
3. Compute the nominal spend public key: $K^s_{nom} = K^o - \mathcal{H}_2(q_{nom})X$. If $K^s_{nom} = K^s_{bob}$, then the e-note is *connected* to Bob's receiver address, and proceed to the next step (this is the "connection" hinted at the beginning of this whole section). Otherwise (if not equal), the e-note is not connected, and hence go to Step 1.
4. Decrypt the amount: $a = \texttt{dec}[q_{nom}](\overline{a})$.
5. Compute the nominal amount commitment: $C_{nom} = \mathcal{H}_3(q_{nom})G + aH$. If $C_{nom} \neq C$, then the e-note is malformed and cannot be spent.
6. Compute the nominal linking tag: $\tilde{K}_{nom} = (k^s_{b,bob}/(k^s_{a,bob} + \mathcal{H}_2(q_{nom})))U$. If he finds a copy of $\tilde{K}_{nom}$ in the ledger, then the e-note has already been spent.

If an e-note (C, K^o, m) is connected to Bob's receiver address, then he knows the corresponding scalars of that e-note: $(k^o_a, k^o_b, a, x) = (k^s_{a,bob} + \mathcal{H}_2(q_{nom}), k^s_{b,bob}, a, \mathcal{H}_3(q_{nom}))$. Hence, "connection" implies e-note ownership. The transaction is complete for Bob.

For Alice to receive the "change" e-note, she must do the same above steps. After that, the transcation is complete for Alice. This finishes a Seraphis trancation.

4 Instantiations and Modifications

There are a number of details to consider when implementing Seraphis in a real cryptocurrency. Aside from instances of proving systems mentioned already in the previous section, this section is comprised of 'recommendations' for instantiations and modifications of other parts of Seraphis, which are inspired by historical privacy-focused cryptocurrency implementations.

4.1 Addressing schemes

4.2 Multisignature operations

4.3 Proof dependency

Transaction Chaining

4.4 Transaction fees

4.5 Coinbase transactions

4.6 Squashed e-note model

4.7 ?????

5 Security model

For a start, we assume that the distributed ledger is immutable. Therefore, the adversary in our analysis will never be able to modify transactions already stored in the ledger. This ledger immutability can be actualized through, for instance, the Nakamoto consensus protocol [8].

Subsections 5.1 to 5.4 outline the required security properties of the cryptographic components for Seraphis, then Subsection 5.5 is the main security analysis of Seraphis, and Subsection 5.6 discusses how some instantiations and modifications described in Section 4 affect the security analysis.

5.1 Zero-knowledge proofs

We require some proving systems to be used in Seraphis to be zero-knowledge. Starting with interactive protocol description, these interative proving systems must satisfy the following three security properties. The precise definitions of these properties are common and found, for instance, in [6].

- *Perfect Completeness*: If the prover knows a witness to the statement, she should always be able to generate a proof for that statement that convinces the verifier.
- *Special Soundness*: If the prover does not know a witness to the statement, she should never be able to generate a proof for that statement that convinces the verifier. This is formalized by saying that there exists an *witness extractor* that has an ability to "rewind time" and make the prover answer several different challenges. From the valid (by verifier) answers of the prover, it should be possible for the witness extractor to extract a witness.
- *Perfect Special Honest Verifier Zero-Knowledge*: The protocol should not reveal anything about the prover's witness. This is formalized by saying that assuming that challenges are honestly generated, there exists a *simulator* such that even without knowledge of a witness, the probability distribution of the simulator's simulated transcripts (all data being sent and received in the protocol) is equal to the probability distribution of actual transcripts by a prover and a verifier.

Fiat-Shamir heuristic [4] transforms interactive proving systems with the above properties into non-interactive zero-knowledge proofs (NIZKPs) in the random oracle model. We require that the proving systems for Ownership and Unspentness proofs, and Range proofs are NIZKPs constructed through Fiat-Shamir heuristic.

5.2 Authenticated symmetric encryption scheme

We require that the authenticated symmetric encryption scheme has the following properties: indistinguishable against chosen-plaintext attack (IND-CPA), indistinguishable against adaptive chosen-ciphertext attack (IND-CCA2), and key-private under chosen-ciphertext attacks (IK-CCA). **Todo: Need to study this!**

5.3 Membership proof security properties

We require that the membership proving system has the following properties:

- *Completeness*:
- *Unforgeability*:
- *Anonymity*:

5.4 Commitment schemes

We define a commitment scheme as a tuple $(\mathsf{Setup}, \mathsf{Comm})$. Setup is the setup algorithm: $pp \leftarrow \mathsf{Setup}(1^\lambda)$, and $\mathsf{Comm} : \mathcal{M} \times \chi \to \mathcal{C}$ is the *commitment function*, where $\mathcal{M}$ is the message space, χ is the randomness space, and $\mathcal{C}$ is the commitment space. Note that $\mathcal{M}, \chi$ and $\mathcal{C}$ are all constructed from pp. To commit to a message $m \in M$, the sender selects $r \xleftarrow{\$} \chi$ and computes the commitment $C = \mathsf{Comm}(m; r)$. We define the required security properties of commitment schemes.

Definition 5.1 (Hiding Property)**.** *A commitment scheme* $(\mathsf{Setup}, \mathsf{Comm})$ *is hiding if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left| \frac{1}{2} - \mathsf{Pr}\left[b' = b \;\middle|\; \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda); (m_0, m_1) \leftarrow \mathcal{A}(pp); \\ b \xleftarrow{\$} \{0,1\}; r \xleftarrow{\$} \chi; \\ C = \mathsf{Comm}(m_b; r); b' \leftarrow \mathcal{A}(C) \end{array} \right] \right| \leq \mathsf{negl}(\lambda).$$

Definition 5.2 (Binding Property)**.** *A commitment scheme* (Setup, Comm) *is binding if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\begin{array}{c|c} \mathsf{Comm}(m_0; r_0) & pp \leftarrow \mathsf{Setup}(1^\lambda); \\ = \mathsf{Comm}(m_1; r_1) & (m_0, m_1, r_0, r_1) \leftarrow \mathcal{A}(pp) \\ \wedge\ m_0 \neq m_1 & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

5.5 Seraphis security properties

The required security properties are those in Omniring's security model [7], with modifications to fit in Seraphis. The Omniring paper presents a rigorous formalization of RingCT constructions, providing precision for security analysis against several realistic attacks. Proofs for the theorems are found in Section B.

We first define commitment functions $\mathsf{CommAmount} : \mathbb{F} \times \mathbb{F} \to \mathbb{G}$ and $\mathsf{CommTag} : \mathbb{F} \times \mathbb{F} \setminus \{0\} \times \mathbb{F} \to \mathbb{G} \times \mathbb{G}$ to be needed later on, as follows:

$$\mathsf{CommAmount}(a; x) = xG + aH \ , \ x \xleftarrow{\$} \mathbb{F} \text{ if } x \text{ is not specified.}$$
$$\mathsf{CommTag}(t_k, k_a, k_b) = (t_k G + k_a X + k_b U, (k_b/k_a)U).$$

CommAmount corresponds to the structure of amount commitment C, while CommTag corresponds to the combination of structures of one-time address K^o (if $t_k = 0$) or masked address K'^o, and linking tag $\tilde{K}$.

The first security property is **Completeness** (called *Correctness* in Omniring), which means that if an e-note appears on the ledger, then the user owning it can honestly generate a valid (by verifier) transaction spending it. Seraphis satisfying completeness immediately follows from the completeness properties of the cryptographic components and by inspection of the protocol description.

Next we consider the **Balance** property, which means that a spender adversary should never be able to spend more amounts than he truly owns, hence preventing double-spending. Balance property has two subproperties: the first involves the binding property. CommAmount is binding means that the spender adversary should not be able to change the e-note amounts when spending the e-notes he owns, while CommTag is binding means that the spender adversary should not be able to make a new linking tag from an e-note he already spent, which is an attempt of double-spending the e-note.

The second subproperty involves an experiment $\mathsf{BAL}(\mathcal{A}, \mathcal{E}_\mathcal{A}, 1^\lambda)$ between a PPT spender adversary $\mathcal{A}$ and a witness extractor $\mathcal{E}_\mathcal{A}$. $\mathcal{A}$ succeeds in the experiment (i.e. $\mathsf{BAL}(\mathcal{A}, \mathcal{E}_\mathcal{A}, 1^\lambda) = 1$) if he managed to generate a valid (by verifier) transaction such that the amounts, linking tags, private keys, etc. are *not all equal* to the data extracted by $\mathcal{E}_\mathcal{A}$. Now the second subproperty means that Seraphis must be "special sound." Moreover, if $\mathcal{A}$ succeeds in the experiment, then it is guaranteed that he broke the binding properties of CommAmount or CommTag.

Definition 5.3 (Balance)**.** *Seraphis is balanced if it satisfies the following two conditions:*

1. *Both* (Setup, CommAmount) *and* (Setup, CommTag) *are binding.*
2. *For every* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *extractor* $\mathcal{E}_\mathcal{A}$ *and a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathsf{BAL}(\mathcal{A}, \mathcal{E}_\mathcal{A}, 1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

where BAL *is described in Figure 1.*

Theorem 5.1 (Balance)**.** *If Ownership and Unspentness proofs are special sound, Range proofs are special sound, and Membership proofs are unforgeable, then Seraphis is balanced.*

Privacy
Non-slanderability

$\underline{\mathsf{BAL}(\mathcal{A}, \mathcal{E}_{\mathcal{A}}, 1^{\lambda})}$

- $pp \leftarrow \mathsf{Setup}(1^{\lambda})$.
- $\mathcal{A}$ is provided $k_v, k_a^s, k_b^s \in \mathbb{F}$ to construct the address $\mathtt{addr}_{\mathcal{A}} = (K_{\mathcal{A}}^{dh}, K_{\mathcal{A}}^{v}, K_{\mathcal{A}}^{s})$, $\mathtt{enote}_{\mathcal{A}} = \{(C_i, K_i^o, m_i)\}_{i=1}^n$ in the ledger which are connected to that address, and a receiver's address $\mathtt{addr}_{\mathcal{B}} = (K_{\mathcal{B}}^{dh}, K_{\mathcal{B}}^{v}, K_{\mathcal{B}}^{s})$.
- $T \leftarrow \mathcal{A}(pp, \mathtt{addr}_{\mathcal{A}}, \mathtt{addr}_{\mathcal{B}}, \mathtt{enote}_{\mathcal{A}})$. $\mathcal{A}$ spends some e-notes to $\mathtt{addr}_{\mathcal{B}}$.
- $(\{(t_{c,i}, t_{k,i}, k_{a,i}^o, k_{b,i}^o, \pi_i)\}_{i=1}^n, x_{\mathcal{B}}, a_{\mathcal{B}}) \leftarrow \mathcal{E}_{\mathcal{A}}(pp, T)$.
- $b_0 := 1$ if Verifier verified T as valid, else $:= 0$.
- $b_1 := 1$ if $\mathsf{CommAmount}(a_{\mathcal{B}}; x_{\mathcal{B}}) = C$ in $\mathtt{enote}_{\mathcal{B}}$ in T, else $:= 0$.
- $b_2 := 1$ if for all $i \in \{1, \ldots, n\}$, $\mathsf{CommAmount}(t_{c,i}; 0) = C_i' - C_{\pi_i}$ in T, else $:= 0$.
- $b_3 := 1$ if for all $i \in \{1, \ldots, n\}$, $\mathsf{CommAmount}(t_{k,i}; 0) = K_i'^o - K_{\pi_i}^o$ in T, else $:= 0$.
- $b_4 := 1$ if for all $i \in \{1, \ldots, n\}$, $\mathsf{CommTag}(t_{k,i}, k_{a,i}^o, k_{b,i}^o) = (K_i'^o, \tilde{K}_i)$ in T, else $:= 0$.
- $b_5 := 1$ if $\sum_{i=1}^n a_{\mathcal{A},i} \geq a_{\mathcal{B}}$ where $a_{\mathcal{A},i}$ is the total amount of ith spent e-note and $a_{\mathcal{B}}$ is the amount received by $\mathtt{addr}_{\mathcal{B}}$, else $:= 0$.
- Return $b_0 \wedge \neg(b_1 \wedge b_2 \wedge b_3 \wedge b_4 \wedge b_5)$.

Figure 1: Balance experiment BAL

5.6 Discussions

6 Efficiency

References

[1] 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.

[2] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Report 2017/1066, 2017. https://ia.cr/2017/1066.

[3] Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter proofs for privacy-enhanced distributed ledger. Cryptology ePrint Archive, Report 2020/735, 2020. https://ia.cr/2020/735.

[4] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[5] Brandon Goodell, Sarang Noether, and RandomRun. Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Report 2019/654, 2019. https://ia.cr/2019/654.

[6] 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.

[7] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling up private payments without trusted setup - formal

foundations and constructions of ring confidential transactions with log-size proofs. Cryptology ePrint Archive, Report 2019/580, 2019. https://ia.cr/2019/580.

[8] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[9] UkoeHB. Seraphis: Privacy-focused tx protocol. https://github.com/UkoeHB/Seraphis.

A Composition proving system

B Security proofs for Seraphis

C ?????