






A Holistic Security Analysis of Monero Transactions

Cas Cremers¹(✉) , Julian Loss¹ , and Benedikt Wagner^{1,2} 

¹ CISA Helmholtz Center for Information Security, Saarbrücken, Germany

{cremers,loss,benedikt.wagner}@cispa.de

² Saarland University, Saarbrücken, Germany

Abstract. Monero is a popular cryptocurrency with strong privacy guarantees for users' transactions. At the heart of Monero's privacy claims lies a complex transaction system called RingCT, which combines several building blocks such as linkable ring signatures, homomorphic commitments, and range proofs, in a unique fashion. In this work, we provide the first rigorous security analysis for RingCT (as given in Zero to Monero, v2.0.0, 2020) in its entirety. This is in contrast to prior works that only provided security arguments for parts of RingCT.

To analyze Monero's transaction system, we introduce the first holistic security model for RingCT. We then prove the security of RingCT in our model. Our framework is modular: it allows to view RingCT as a combination of various different sub-protocols. Our modular approach has the benefit that these components can be easily updated in future versions of RingCT, with only minor modifications to our analysis.

At a technical level, we split our analysis in two parts. First, we identify which security notions for building blocks are needed to imply security for the whole system. Interestingly, we observe that existing and well-established notions (e.g., for the linkable ring signature) are insufficient. Second, we analyze all building blocks as implemented in Monero and prove that they satisfy our new notions. Here, we leverage the algebraic group model to overcome subtle problems in the analysis of the linkable ring signature component. As another technical highlight, we show that our security goals can be mapped to a suitable graph problem, which allows us to take advantage of the theory of network flows in our analysis. This new approach is also useful for proving security of other cryptocurrencies.

Keywords: Monero · RingCT · Transaction Scheme Security · Algebraic Group Model · Network Flows

1 Introduction

In the rapidly growing zoo of cryptocurrencies, Monero¹ [33,52] is among the largest and most well-known systems, with a market capitalization of about

¹ See <https://www.getmonero.org>.

three billion USD at the time of writing. One of Monero’s distinguishing features is its unique transaction scheme RingCT (“Ring Confidential Transactions”) which offers users a high degree of privacy for on-chain transactions. To this end, RingCT provides an efficient means of hiding how funds are transferred between users. The core property that users of a currency rely on, however, is *transaction security*. Namely, it should not be possible to spend funds twice, create money out of thin air, or steal coins from other users. To achieve transaction security, decentralized currencies require that the validity of transactions can be verified publicly, which seemingly contradicts the privacy goals of currencies like Monero.

Monero’s Complexity. To achieve the challenging goal of reconciling privacy and security, RingCT combines several simpler building blocks such as linkable ring signatures, homomorphic commitments, and range proofs into a highly complex protocol. The building blocks are combined with a key derivation process in a unique way. This is in contrast to simpler currencies, e.g. Bitcoin, which merely rely on standard signatures. Unfortunately, it is not obvious at all that RingCT’s complex system is indeed secure. For example, when a user Alice sends coins to a user Bob, Alice (who may be adversarial) derives new keys for Bob using Bob’s long-term address. This implies that Alice has non-trivial knowledge of relations between the keys of Bob, potentially opening the door for related-key attacks. Such related-key attacks are not considered by the standard security notions of the components. Even worse, the complex nature of RingCT has led to concrete attacks [37, 42, 43] in the past, which were not captured by the limited prior analyses. This raises the following question:

Is Monero’s transaction scheme secure?

Our Contribution. In this work, we provide the first comprehensive and holistic security analysis of RingCT. Our contributions are

- We show that RingCT as a whole achieves transaction security.
- We thereby identify which security properties of components are sufficient to imply security for the entire transaction scheme. Thus, our analysis is modular, which makes it easy to adapt to changes of RingCT in the future.
- We introduce a new proof technique which reduces a game-based security notion to a combinatorial problem of network flows. This combinatorial argument allows us to prove that no adversary can create money out of thin air. We are confident that it can be applied when analyzing other currencies as well.

Along the way, we face several technical challenges, arising from composition in the algebraic group model (AGM) [21] and the insufficiency of established security notions for building blocks. For example, we observe that the established notion of linkability for linkable ring signatures has to be strengthened significantly (see Sect. 1.2).

Due to space limitations, we refer to the full version [12] for formal details of our analysis and full proofs, and mostly give an overview here.

1.1 Our Approach: A Modular Analysis of RingCT

We provide the first rigorous security analysis of Monero’s transaction system RingCT as a whole. Our framework is modular and abstracts many of the components of RingCT into stand-alone building blocks. We believe that these components naturally reflect the design ideas of RingCT, and lead to an improved understanding of the ideas at its core. In addition, this approach makes it possible to easily replace a given part of the scheme in future system updates. For example, should Monero decide to use another ring signature scheme in the future, one just needs to redo the parts of our analysis that deal with the ring signature component. Conversely, our results may also serve as guidelines for the required security properties of the components in the event of such an update.

We begin by introducing syntax and model for the desired security properties of the top-level transaction scheme (i.e., RingCT). We define a single security experiment that can be summarized as follows:

1. Whenever an honest user receives coins, they can later spend these coins. That is, an adversary can neither steal the coins that an honest user received, nor prevent the honest user from spending them.
2. An adversary can not create coins out of thin air. That is, the adversary can never spend more coins than it ever received.

In contrast to prior models for RingCT-like transaction schemes, our model is not only holistic, but also takes subtleties such as the reuse of randomness or adversarially generated keys into account.

Having defined the security properties we aim for, we then prove that our model of RingCT meets these properties. This consists of the following steps:

1. *Syntax and Security for Subcomponents.* We identify the structural components of RingCT and introduce appropriate syntax for them. Then, we define several new security notions that are tailored to the interplay of these building blocks within RingCT. For example, due to potential related-key attacks, it is necessary to define security of the ring signature component with respect to the key derivation mechanism. Thus, we require security notions that differ from well-established ones from the literature.
2. *System Level Analysis.* The next step of our analysis is to prove the security of any top-level transaction scheme that follows our syntax. Our proof is generic and only assumes that subcomponents satisfy our novel security notions. A technical highlight of our proof is the utilization of the theory of network flows. Concretely, after applying the security notions of subcomponents to extract the hidden flow of money in the system, we define a graph based on it. Then, we use further notions of subcomponents to argue that this graph constitutes a flow network. Finally, we show that no money can be created by

using the fact that every cut in such a flow network has the same flow passing through it. We are confident that this new technique is also applicable in the context of other currencies such as Bitcoin or Ethereum.

3. *Component Level Analysis.* Finally, we instantiate the components as in Monero and prove that they satisfy our security notions. Here, the biggest challenge lies with the linkable ring signature component, for which we provide an analysis in the Algebraic Group Model (AGM) [21]. We encounter several subtle issues that arise from composing different building blocks. As such, we believe that our proof sheds further light on the pitfalls of naively composing proofs in the AGM.

1.2 Technical Highlights and Findings

In this section, we give an overview of some of our findings.

Composing Extractors in the Algebraic Group Model. To show that our security notions for components imply security for the entire transaction scheme, we make use of knowledge extractors. Namely, we consider each transaction that the adversary submits to the system, and run a knowledge extractor to get the secret signing key that the adversary used to create the transaction. The existence of such an extractor should be guaranteed by our notions for the linkable ring signature components. As we extract for each submitted transaction, it is crucial that our extractor does not rewind the adversary. A common way to design such a non-rewinding extractor for a given scheme is to leverage the algebraic group model (AGM) introduced by Fuchsbauer, Kiltz, and Loss [21]. In this model, whenever an adversary submits a group element $X \in \mathbb{G}$ (e.g., as part of transaction), it also submits exponents $(\gamma_i)_i$ such that $X = \prod_i A_i^{\gamma_i}$, where $A_i \in \mathbb{G}$ are all group elements the adversary ever received. We say that $(\gamma_i)_i$ is a representation of X over basis $(A_i)_i$. A carefully crafted extractor can now use the representation to compute the secret signing key the adversary used. Unfortunately, formally defining under which conditions such an extractor has to succeed turns out to be non-trivial. The naive way of doing it would be to define an isolated notion for the linkable ring signature as follows: The adversary gets system parameters as input (including a generator $g \in \mathbb{G}$), and may output a signature and algebraic representations of all group elements over basis g , and it wins if the extractor fails to output a secret key, but the signature is valid. In fact, such an isolated approach has been used in the literature for other primitives [10, 38]. However, this extractor does not compose well. Concretely, in the isolated notion, the extractor expects that all representations are over basis g . On the other hand, if we use our extractor in the wider context, i.e., in the proof of RingCT, the representations are over much more complicated bases, because the adversary receives group elements in signatures, hash values, and keys. Formally, the security game (and subsequent reductions) would have to translate all representations into a representation over basis g first. It turns out that such a translation is not compatible with our subsequent proof steps.

For example, if the adversary just forwards a signature that it obtained from a signing oracle, there is no way that we can extract a secret key from it.

The solution we opt for is to change the isolated notion for the linkable ring signature into a more involved notion resembling simulation-extractability, in which we give the adversary oracles that output signatures, hash values, and keys. We require that the extractor is able to extract a valid secret key only under certain conditions, e.g., if the adversary did not obtain the signature from an oracle. At the same time, the extractor is not allowed to share any internal state with the oracles. While this makes our extractor usable in the proof of RingCT, it substantially complicates the AGM proof of the extractor.

Notions of Linkability. In a ring signature scheme, a signer holding a secret key sk can sign a message with respect to a so-called key ring $R = (pk_1, \dots, pk_N)$, where sk is associated with one of the public keys, say pk_{i^*} . Crucially, the signature does not reveal the index i^* , so that the signer stays anonymous. Linkable ring signatures additionally allow to publicly identify whether two signatures have been computed using the same secret key. More precisely, they are required to satisfy a property called linkability. It states that there is an efficient algorithm $Link$, such $Link$ outputs 1 on input σ, σ' (resp. 0) if and only if the signatures σ, σ' have been computed with the same (resp. different) secret key. In terms of security, no adversary should manage to compute two signatures σ, σ' using the same secret key, such that $Link(\sigma, \sigma')$ outputs 0. In other words, $Link$ detects if two signatures are computed using the same secret key, and can not be cheated by an adversary. In RingCT, each unspent transaction output is associated to a fresh secret key, which implies that $Link$ can detect double spending of outputs. Formally defining linkability is a non-trivial task. As already noted in [27], there are several independent notions of linkability. One of the more established notions is so-called pigeonhole linkability. It is defined in the following way: An adversary breaks pigeonhole linkability if it outputs $N + 1$ valid non-linking signatures, where all rings have size at most N . Unfortunately, pigeonhole linkability seems to be insufficient for our purposes. Concretely, suppose an adversary uses a key ring (o_1, o_2) consisting of two outputs o_1 and o_2 in two distinct valid transactions. Now, recall from our previous paragraph that we use a knowledge extractor that gives us the secret key that the adversary used. Assume this knowledge extractor returns the secret key sk_1 associated to o_1 in both cases, but the two signatures do not link. Intuitively, linkability should say that this is not possible, because the adversary used sk_1 to compute both signatures. However, pigeonhole linkability is not applicable, as we only have two signatures on rings of size two. Instead, we need a notion of linkability that is tied to our knowledge extractor, and rules out this case. More precisely, it should guarantee that if the extractor outputs the same secret key for two signatures, then the signatures link.

1.3 Related Work

In this section, we give an overview of related work.

Related Security Models. Prior to our work, security models for systems similar to RingCT have been given [18, 19, 35, 49, 56]. Notably, all of them analyze new constructions and not RingCT as it is. Further, some of these models [18, 19, 49, 56] omit important non-trivial aspects of RingCT, e.g., adversarial key derivation. Some of them [35, 49, 56] do not give a security definition for the whole scheme, but instead present a set of notions, somewhat similar to the component-wise notions we present as an intermediate step. It remains unclear how these notions relate to each other and the security of the transaction scheme as a whole. We provide a more detailed discussion on these models and how they relate to our model in Sect. 5.

History of Monero. Monero’s transaction scheme RingCT originates in the CryptoNote protocol [52], which is based on a linkable ring signature presented in [24]. Noether [44] introduced a way to hide transaction amounts using Pedersen commitments [45] and range proofs, and also presented a compatible new ring signature component, called MLSAG. The construction of MLSAG is mostly based on [36]. Later, MLSAG was replaced by a more concise ring signature component, called CLSAG [27], and Bulletproofs/Bulletproofs+ [7, 11, 26] are being used as range proofs. Bulletproofs++ [16] are investigated for potential use [46]. Overviews of Monero and its transaction system can be found in [1, 33]. Prior work has studied the security of some of RingCT’s building blocks in isolation [7, 26, 27, 45], but no rigorous security argument has been given for RingCT as a whole.

New Constructions and Functionality Enhancements. Several works presented new constructions of transaction schemes similar to RingCT. These range from efficiency and anonymity improvements [31, 35, 49, 56] to the use of post-quantum assumptions [18, 19]. Also, some works modify RingCT with the motivation to increase compatibility with other protocols, e.g., second-layer protocols [40] or proof of stake consensus [39]. A variety of protocols has been designed add new functionality to the Monero ecosystem. Examples include proofs of reserve [14, 15], payment channels [40, 48, 50], and protocols atomic swaps [28, 50].

Attacks on Monero. Researchers have also studied attacks against Monero and their mitigations. These target privacy [13, 17, 20, 32, 34, 41, 47, 53–55], centralization [9] and security aspects [37, 42, 43]. In terms of privacy, attacks reach from passive attacks [41, 53] to active attacks [54, 55], and temporal attacks [34] that make users traceable. These attacks are purely combinatorial in nature. The works [17, 47] study how to mitigate such combinatorial attacks.

Related Currencies and Their Analysis. ZCash [29] is one of the most prominent privacy-focused cryptocurrencies. It is based on the Zerocash protocol [3], which comes with a cryptographic security analysis. The current protocol specification of ZCash [29] suggests that ZCash deviates from Zerocash in multiple ways. Mumblewimble [30] is a currency prototype that uses homomorphic commitments for efficiency reasons and to hide transaction amounts. In contrast to Monero’s transaction scheme, Mumblewimble does not rely on ring signatures or stealth addresses. A security model and analysis of Mumblewimble has been given in [22, 23].

2 Informal Overview of Monero Transactions

In this section, we give an informal overview of the Monero transaction scheme. The purpose of this is twofold. On the one hand, it should explain the complex structure of transactions for readers not familiar with Monero. On the other hand, Monero versed readers may use this section as a first introduction to our modularization. We assume familiarity with common cryptographic tools such as commitments, ring signatures, and zero-knowledge proofs.

User Addresses. Before diving into the structure of transactions, we first clarify what constitutes an address of a user, i.e., its long-term key material. Namely, each user holds a triple $(\text{ipk}, \text{ivk}, \text{isk})$. We call these the identity public key, identity view key, and identity signing key, respectively. While ipk serves as a public address of the user, the keys ivk and isk should remain secret and provide the following functionality:

- The identity view key ivk allows to identify payments that the user receives and decrypt the associated amounts.
- The identity signing key isk allows to spend funds, i.e., sign transactions.

Readers familiar with simpler currencies such as Bitcoin should think of isk as a secret key as in Bitcoin, and ivk as being an additional key related to privacy. Namely, leaking ivk should only compromise the privacy, but not the security of users. In the concrete implementation of Monero, the identity public key ipk contains two group elements $K_v = g^{k_v} \in \mathbb{G}$ and $K_s = g^{k_s} \in \mathbb{G}$, where $\text{isk} = k_s \in \mathbb{Z}_p$, and $\text{ivk} = k_v \in \mathbb{Z}_p$, i.e., we have $\text{ipk} = (g^{\text{ivk}}, g^{\text{isk}})$.

Key Concepts of Transactions. Transactions in Monero follow the widely used UTXO (“unspent transaction output”) model. In this model, each transaction spends some inputs into some outputs, and all inputs are unused outputs of previous transactions. As our running example, we consider the case of a transaction with two inputs and three outputs. A transaction is visualized in Fig. 1. We refer to the sender of a transaction as Alice, and to the recipient of an output as Bob. A naive transaction (as used in other currencies) would simply contain references to the inputs, and a digital signature per input. Each output

would contain the address of the receiver Bob and the amount that it is worth. In contrast, Monero uses the following core ideas:

- To hide the sender, the actual inputs are grouped with decoy inputs. Ring signatures are used for each input.
- To hide the recipient, addresses contained in outputs are rerandomized. These rerandomized addresses are also known as stealth addresses.
- Amounts contained in outputs are hidden in homomorphic commitments.

Next, we explain how these ideas are implemented in more detail.

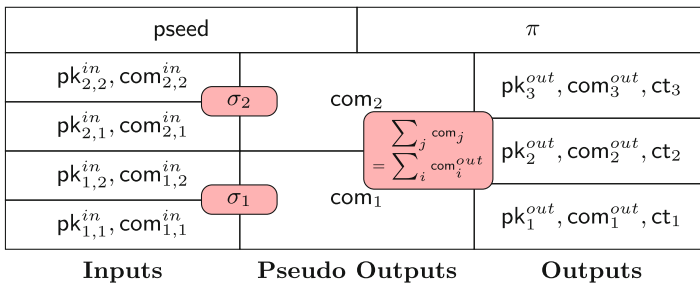


Fig. 1. Schematic overview of an example transaction in Monero with two inputs and three outputs. Inputs are actually references to previous outputs. Signatures σ_i connect inputs and pseudo outputs. The homomorphic property of commitments ties pseudo outputs to outputs. In addition to inputs, outputs, and signatures, a transaction also contains a public seed **pseed** and a range proof π .

Outputs. We start by describing what constitutes an output of a transaction, and how it is generated. Recall that in a naive transaction, an output would just be the address of the recipient and an amount. Monero hides amounts in commitments **com**, and recipients by using rerandomizations **pk** of their actual address **ipk**. To ensure that the recipient Bob can (1) recognize that he receives an output, and (2) use that output, the randomness for commitments and rerandomization has to be recovered by Bob. This is implemented using a Diffie-Hellman-style derivation of shared secrets: The sender Alice first includes a public seed (also called transaction public key) $pseed = g^r$ in the transaction. The public seed will be used for the entire transaction, and not just for one output. Then, she derives $ok = (K_v)^r$, where K_v is the view key part of Bob’s identity public key $ipk = (K_v, K_s)$, i.e., $ok = g^{ivk \cdot r}$. Thus, ok serves as a shared secret between Alice and Bob. The randomness for the rerandomization and the commitment is derived from ok and the position of the output. Namely, the first component of an output is $pk = K_s \cdot g^\tau$, where the exponent $\tau \in \mathbb{Z}_p$ is deterministically derived from ok and the position of the output. The second component is a commitment $com = Com(amt, cr)$, where the randomness cr is deterministically derived from ok and the position of the output. Finally, the output also contains a symmetric

encryption ct of the amount amt . Most importantly, the values τ, cr , and the key for the encryption are all deterministically derived from ok and the position of the output. Let us point out the implications of this: The recipient Bob can derive the shared secret ok using his view key $ivk = k_v$ and the public seed $pseed$. Then, he can also derive τ and cr from ok , decrypt ct , and check whether the equations $pk = K_s \cdot g^\tau$ and $com = Com(amt, cr)$ hold. If so, Bob knows that he just received amt coins. This is possible even if isk is unknown. If $isk = k_s$ is known, then Bob can recover a secret key $sk = k_s + \tau$ for pk . This allows Bob to spend the output in a future transaction. To emphasize, ok is a shared secret between Alice and Bob, and no other party learns τ, cr , or the decryption key for ct .

Inputs. Assume Alice owns an output $o^* = (pk^*, com^*, ct^*)$ of a previous transaction. Especially, she knows the secret key sk^* corresponding to pk^* . Assume she wants to use this output as an input in the current transaction. A naive way for Alice to do that would be to include (a reference to) o^* , and a signature with respect to pk^* to prove ownership. In order to obfuscate the link between the transaction and o^* , Monero uses a different approach. Namely, in a first step, Alice selects some random outputs $o' = (pk', \cdot, \cdot)$ of previous transactions in the system. These are not necessarily owned by Alice, and will serve as decoys. For simplicity, assume she only selects one such decoy output. Then, (references to) the outputs o^* and o' are included in the transaction. Finally, Alice does not use a standard signature, but instead she uses a ring signature for ring $R = \{pk^*, pk'\}$. This signature proves that Alice owns one of the outputs o^*, o' , but does not reveal which one. However, this implies that after the transaction is accepted by the system, there has to be some mechanism that ensures that the output o^* can no longer be spent, while the decoy output o' can. We will see how to solve this later.

Homomorphic Commitments. So far, we discussed how to include outputs in transactions, and use previously received outputs as inputs for a transaction. However, we did not discuss how it is ensured that combination of inputs and outputs is valid, i.e., no money is created. In other words, we have to ensure that $\sum_j amt_j^{in} = \sum_i amt_i^{out}$, where amt_j^{in} and amt_i^{out} are the amounts encoded in inputs and outputs, respectively. To do this without revealing the amounts itself, Monero leverages homomorphic properties of the commitment scheme (i.e., the Pedersen commitment scheme). Namely, ignoring decoys for a moment, if com_j^{in} are the commitments contained in the inputs, and com_i^{out} are the commitments in the outputs, then we would ensure that $\sum_j com_j^{in} = \sum_i com_i^{out}$. Intuitively, the binding property of the commitment scheme should tell us that this equality implies the equality over the amounts that we want. However, this only holds if we avoid overflows. To do that, we ensure that the amt_j^{in} and amt_i^{out} are in a certain range. For that reason, Alice includes a range proof π in the transaction.

Pseudo Outputs. In the previous paragraph, we oversimplified our explanation. Namely, the following two obstacles remain:

- How can Alice ensure that the equation $\sum_j \text{com}_j^{\text{in}} = \sum_i \text{com}_i^{\text{out}}$ holds? Namely, for the Pedersen commitment, this not only requires $\sum_j \text{amt}_j^{\text{in}} = \sum_i \text{amt}_i^{\text{out}}$, but also $\sum_j \text{cr}_j^{\text{in}} = \sum_i \text{cr}_i^{\text{out}}$, where $\text{com}_* = \text{Com}(\text{amt}_*, \text{cr}_*)$. Given the structure of outputs, Alice has no way to ensure this.
- If we insist on the equation $\sum_j \text{com}_j^{\text{in}} = \sum_i \text{com}_i^{\text{out}}$, then actual inputs are distinguishable from the decoys, as they most likely do not satisfy the equation.

To get around these two problems, a level of indirection, called pseudo outputs, is used. In a nutshell, a pseudo output is just another commitment that Alice computes to connect inputs to outputs. Namely, for each of her inputs with amount amt_j^{in} , Alice computes a new commitment $\text{com}_j = \text{Com}(\text{amt}_j^{\text{in}}, \text{cr}_j)$, with freshly sampled randomness cr_j , and such that $\sum_j \text{cr}_j = \sum_i \text{cr}_i^{\text{out}}$. Then, instead of homomorphically checking equality between inputs and outputs, we now check equality between pseudo outputs and outputs using the equation $\sum_j \text{com}_j = \sum_i \text{com}_i^{\text{out}}$. This works out, because Alice now has the freedom to choose the values cr_j . In this way, we ensure that no money is created on the transition from pseudo outputs to outputs. What remains is to ensure that this also holds for the transition from inputs to pseudo outputs. To do that, for each input j , Alice needs to prove that she indeed used amt_j^{in} to compute com_j , where amt_j^{in} is the amount associated to her input $(\text{pk}^*, \text{com}^*, \text{ct}^*)$. Recall that in our running example, this input is grouped with a decoy $(\text{pk}', \text{com}', \text{ct}')$. We can not just insist on $\text{com}^* = \text{com}_j$, because this reintroduces the two problems from above. Instead, Alice could prove that $\text{com}^* - \text{com}_j$ or $\text{com}' - \text{com}_j$ is a commitment to 0. For Pedersen commitments with basis g, h , this is equivalent to proving that Alice knows some r such that $\text{com}^* - \text{com}_j = g^r$ or $\text{com}' - \text{com}_j = g^r$. Interestingly, this proof can be implemented as part of the ring signature that is used: We introduce a second dimension to the public keys, and Alice signs for the ring $R = \{(\text{pk}^*, \text{com}^* - \text{com}_j), (\text{pk}', \text{com}' - \text{com}_j)\}$ using the secret key (sk^*, r) . In this way, the signature not only proves ownership of inputs, but also consistency between the amounts encoded in input and pseudo output.

Double-Spending Detection. When we discussed the structure of inputs, we claimed that ring signatures are used for each input. We already saw that this claim is just a simplification, because pseudo outputs require us to use two-dimensional ring signatures. What we did not solve yet is the problem raised in our discussion of inputs. Namely, after a transaction is accepted, the actual inputs should no longer be spendable, while the decoy outputs should be. Intuitively, if we were able to detect that two signatures are computed using the same secret key, then we could solve this problem. Namely, we force that each pk is only used once, and a transaction is only accepted if no signature conflicts with a previous one, in the above sense. Fortunately, there is a variant of ring signatures, called linkable ring signatures, that allows us to do exactly that.

More precisely, there is an algorithm $\text{Link}(\sigma, \sigma')$ which outputs 1 if and only if σ and σ' were computed using the same key sk . This does not reveal which sk was used.

Summary: Transaction Generation. A user Alice can generate a transaction as follows:

1. Alice computes a public seed $\text{pseed} = g^{\text{sseed}}$ and includes it in the transaction.
2. Alice computes outputs. That is, for each recipient Bob with identity public key $\text{ipk} = (K_v, K_s)$ that should receive amt coins, she does the following:
 - (a) Derive the shared secret ok from K_v and sseed .
 - (b) Using ok and the position of the output, derive commitment randomness and a rerandomization term.
 - (c) Use these to compute a commitment com to amt and a rerandomization pk of K_s .
 - (d) Encrypt amt into a ciphertext ct using a key derived from ok .
 - (e) The output is $(\text{pk}, \text{com}, \text{ct})$.
3. For each of her inputs, Alice selects other outputs of previous transactions as decoys, and groups her actual input with these decoys.
4. For each of her inputs, Alice computes a pseudo output com_j , such that the pseudo outputs sum up to the sum of the output commitments.
5. Alice computes a range proof π showing that the amounts in output commitments and pseudo outputs do not cause overflows.
6. For each of the inputs, Alice signs the transaction using a two-dimensional linkable ring signature.

Summary: Transaction Verification. Throughout the last paragraphs, we introduced a lot of conditions that a valid transaction has to satisfy implicitly. Now, we explicitly summarize them. Namely, to verify the validity of a transaction, the following has to be checked:

1. All inputs (including the decoys) are outputs of previous transactions.
2. All signatures are valid with respect to the given rings.
3. There is no signature that links to another signature in this or a previous transaction.
4. We have $\sum_j \text{com}_j = \sum_i \text{com}_i^{\text{out}}$, where com_j are the pseudo outputs, and $\text{com}_i^{\text{out}}$ are the output commitments.
5. The range proof for the commitments verifies.

3 Model for Private Transaction Schemes

In this section, we present our formal model for a private transaction scheme, such as RingCT. We first specify the components of a private transaction scheme. Then, we define how transactions are constructed using these components. Finally, we define the security of private transaction schemes.

3.1 Syntax

We introduce our syntax for private transaction schemes. A more detailed version is given in our full version [12]. Throughout, we assume that some system parameters $\text{par} \leftarrow \text{Setup}(1^\lambda)$ are generated using a setup algorithm Setup . These are given implicitly to all algorithms and define certain data types. A private transaction scheme consists of several components, which we introduce below. For the informal explanation, we assume that a user Alice wants to spend coins to a user Bob.

Key Derivation Scheme. We start with the definition of a key derivation scheme KDS. This component specifies how users generate their long-term address, and how other users can then derive stealth addresses from them, i.e., keys associated to outputs in the system. Concretely, to generate its long-term address, Bob runs an algorithm $\text{GenID}(\text{par})$ that outputs a triple $(\text{ipk}, \text{ivk}, \text{isk})$. As explained in Sect. 2, the identity public key ipk serves as the public address, and the identity view key ivk and identity signing key isk are kept secret. Now, suppose Alice wants to spend coins to Bob. For that, Alice first samples a public seed pseed and a private seed sseed using an algorithm $\text{Encaps}(\text{par})$. Intuitively, we think of pseed as a first message in a key exchange between Alice and Bob. Alice includes pseed in the transaction such that Bob receives it. Then, she uses these seeds and Bob’s address to derive a stealth address for Bob. To do that, algorithms SendDecaps and RecDecaps are used, where Alice runs $\text{SendDecaps}(\text{ipk}, \text{sseed})$ and Bob runs $\text{RecDecaps}(\text{ivk}, \text{pseed})$. As a result, both obtain a shared secret ok , called the output key. Alice now uses this shared secret in algorithm $\text{DerPK}(\text{ipk}, \text{ok}, \text{tag})$ to derive the public key pk , which is the stealth address. Bob uses algorithm $\text{DerSK}(\text{isk}, \text{ok}, \text{tag})$ to derive the corresponding secret key sk . Further, Bob can identify public keys pk that are derived for him by a further algorithm Track : if $\text{Track}(\text{ipk}, \text{ok}, \text{pk}, \text{tag}) = 1$, then this indicates that pk is derived for him. Here, the tag $\text{tag} \in \mathbb{N}$ is used for domain separation and ordering.

Verifiable Homomorphic Commitment Scheme. To hide the amount of a transaction while still allowing to verify consistency of inputs and outputs, a special kind of commitment scheme, and an associated proof is used. This is formalized by the notion of a verifiable homomorphic commitment scheme VHC. Namely, recall that when Alice commits to an amount she sends to Bob, then she deterministically derives the random coins cr used for the commitment from the output key ok that is shared between Alice and Bob. We model this via algorithm $\text{DerRand}(\text{ok}, \text{tag})$ that outputs cr . Given such random coins and an amount $\text{amt} \in \mathcal{D} \subseteq \mathbb{N}_0$, one can commit to amt using $\text{com} := \text{Com}(\text{amt}, \text{cr})$. Here, \mathcal{D} is the set of allowed amounts. We require that Com is homomorphic in both amt and cr . Then, Alice can prove that she knows valid amounts in \mathcal{D} that she committed to within one transaction. This is modeled by an algorithm $\text{PProve}(\text{stmt}, \text{witn})$ that takes a set of commitments as a statement stmt and the

corresponding preimages as a witness witn , and outputs a proof π . The proof can then be verified for stmt by running an algorithm $\text{PVer}(\text{stmt}, \pi)$.

Data Encryption Scheme. As amounts are hidden due to the use of a commitment scheme, Alice needs to communicate them privately to Bob. For that, a (symmetric) encryption scheme DE is used. It makes use of the shared secret ok as a key. We omit the details here, as they are not related to any of our security notions, but only relevant for privacy.

Key Conversion Scheme. Recall from our overview in Sect. 2 that Alice shows consistency between amounts in inputs and pseudo outputs by translating commitments in inputs and pseudo outputs into public keys used in the linkable ring signature scheme. If Alice knows the commitment randomness for two such commitments that commit to the same amount, then she can know the corresponding secret key. We formalize this process by defining a key conversion scheme KCS . Namely, Alice runs an algorithm $\text{auxpk} := \text{ConvertPublic}(\text{com}, \text{com}')$ to obtain an auxiliary public key auxpk from a pair of commitments com and com' , where we think of com as being part of an input, and com' as being the pseudo output. Similarly, she can run $\text{auxsk} := \text{ConvertSecret}(\text{cr}, \text{cr}')$ for the associated randomness cr, cr' to get an auxiliary secret key auxsk . The guarantee is that if com and com' commit to the same amount with randomness cr, cr' , respectively, then auxsk is a valid secret key for auxpk , and can then be used within the linkable ring signature component.

Two-Dimensional Linkable Ring Signature Scheme. Before Alice can publish the transaction, she needs to sign it, using a variant of a ring signature scheme. Recall from our overview in Sect. 2, that this has two reasons. First, it ensures that Alice holds secret keys for one output referenced by each input. Second, in combination with the key conversion scheme, it ensures that the amounts between inputs and pseudo outputs are consistent. We formalize this as a two-dimensional linkable ring signature scheme LRS , which is given by three algorithms Sig, Ver , and Link . To sign a message m , e.g., a transaction, with respect to some key ring $\text{R} = (\text{pk}_i, \text{auxpk}_i)_{i=1}^N$, Alice has to know a valid pair of secret keys sk, auxsk for one of the $\text{pk}_i, \text{auxpk}_i$. Then, she can compute a signature $\sigma \leftarrow \text{Sig}(\text{R}, \text{sk}, \text{auxsk}, \text{m})$. This signature can be verified with respect to R and m by running $\text{Ver}(\text{R}, \text{m}, \sigma)$. Also, one can check whether two signatures σ, σ' were computed using the same key by running $\text{Link}(\sigma, \sigma')$.

Generating Transactions. Suppose Alice wants to spend $\text{amt}_i^{\text{out}}$ coins to a user with identity public key ipk_i for each $i \in [K]$. Further, suppose that Alice wants to use L inputs for that, which are outputs $(\text{pk}_j^{\text{in}}, \text{com}_j^{\text{in}})$ (for $j \in [L]$) of previous transactions that she owns. Because she owns them, she knows the associated amount and commitment randomness $\text{amt}_j^{\text{in}}, \text{cr}_j^{\text{in}}$, and the corresponding secret key sk_j . We write $\text{Use}_j = (\text{pk}_j^{\text{in}}, \text{com}_j^{\text{in}}, \text{amt}_j^{\text{in}}, \text{cr}_j^{\text{in}}, \text{sk}_j)$ for $j \in [L]$ to

denote these outputs that Alice uses as inputs, along with the corresponding secret information. Finally, assume that Alice picked additional outputs (not necessarily owned by her) from previous transactions. We let Ref_j be the list of these outputs, including an entry $\text{pk}_j^{in}, \text{com}_j^{in}$. Now, we specify how Alice generates a transaction by defining an algorithm GenTx , which takes as inputs $(\text{Use}_j)_{j=1}^L, (\text{Ref}_j)_{j=1}^L$, and $(\text{ipk}_i, \text{amt}_i^{out})_{i=1}^K$. We formally present this algorithm in our full version [12].

Verifying Transactions. We specify how a user can verify a given transaction by defining an algorithm VerTx . As the validity of a transaction depends on the current state of the system, e.g., on previous transactions, this algorithm needs additional inputs that model this state. Concretely, we model all public seeds of previous transactions by a list PSeeds , all outputs in the system by a list Outputs , and all signatures contained in previous transactions by a list Signatures . Then, algorithm VerTx takes as input the lists $\text{PSeeds}, \text{Outputs}, \text{Signatures}$ and a transaction Tx . We formally define this algorithm in the full version [12].

Receiving Outputs. When a transaction Tx is published, users should be able to identify outputs that they receive. For that, we define an algorithm Receive . Concretely, write $\text{Tx} = (\text{In}, \text{Out}, \text{pseed}, \pi)$ and let its outputs be $\text{Out} = (\text{pk}_i^{out}, \text{com}_i^{out}, \text{ct}_i, \text{tag}_i)_{i=1}^K$. Then, each user with keys $\text{ipk}, \text{ivk}, \text{isk}$ runs Receive on inputs $\text{pk}_i^{out}, \text{com}_i^{out}, \text{ct}_i, \text{tag}_i$ for each $i \in [K]$. Additionally, the user inputs pseed and its keys $\text{ipk}, \text{ivk}, \text{isk}$. The algorithm outputs the received amount amt , and the commitment randomness and secret key cr, sk in case the amount is non-zero. The commitment randomness and secret key are then needed whenever the user wants to spend this output.

3.2 Security

In this section, we introduce our security notion for private transaction schemes like RingCT. In other words, we make explicit what we aim to prove by presenting a cryptographic security game. To define security, we introduce data structures and oracles that model the state of the world and the adversary’s capabilities. For the entire section, we fix a private transaction scheme $\text{PTS} = (\text{LRS}, \text{KDS}, \text{VHC}, \text{DE}, \text{KCS})$ and an efficient adversary \mathcal{A} .

Threat Model. Before we explain the details of our security game, we provide intuition for the threat model. In our security game, the adversary is allowed to interact with honest users and a public ledger, which accepts and stores transactions whenever they are valid. The adversary can make new users appear, corrupt users, and instruct honest users to create and publish transactions. Further, the adversary can submit arbitrary transactions to the ledger. The goal of the adversary is to either create coins out of thin air, or to steal or invalidate coins from honest users. In the remainder of this section, we state the definition of this security game more precisely.

State of the World. To model the current state of the world, our game holds several data structures. First, the game should keep track of existing (honest) users, by storing identity keys and information about the outputs that these users own. For that, we introduce the following data structures.

- **Identities:** This list contains identity public keys ipk for users. These users are initially honest, but may later be corrupted by the adversary.
- $\text{ivk}[\cdot]$, $\text{isk}[\cdot]$: These maps contain the identity view key $\text{ivk}[\text{ipk}]$ and the identity signing key $\text{isk}[\text{ipk}]$ for each $\text{ipk} \in \text{Identities}$.
- $\text{corr}[\cdot]$: This map contains a value $\text{corr}[\text{ipk}] \in \{0, 1, 2\}$ for each $\text{ipk} \in \text{Identities}$. It models the corruption state of this user, i.e., $\text{corr}[\text{ipk}] = 0$ by default, $\text{corr}[\text{ipk}] = 1$ if \mathcal{A} knows $\text{ivk}[\text{ipk}]$, and $\text{corr}[\text{ipk}] = 2$ if \mathcal{A} knows $\text{ivk}[\text{ipk}]$ and $\text{isk}[\text{ipk}]$.
- $\text{Owned}[\cdot]$: This map contains a list $\text{Owned}[\text{ipk}]$ for each $\text{ipk} \in \text{Identities}$. This lists contains all outputs that user ipk owns. Additionally, it contains side information that is necessary to spend these outputs. Namely, the lists contain entries of the form $(\text{pk}, \text{com}, \text{amt}, \text{cr}, \text{sk})$. It is only kept consistent for users ipk with $\text{corr}[\text{ipk}] < 2$.

Second, the game should be able to generate and verify transactions. For that, the game has to know all previous transactions, or more precisely, previous outputs and signatures. Therefore, we introduce the following data structures.

- **TXs:** This list contains all transactions in the system, i.e., transactions that have been submitted and verified.
- **PSeeds:** This list contains all public seeds pseed that are contained in transactions in the system.
- **Outputs:** This list contains all outputs (pk, com) that are currently in the system. These may, for example, be part of previous transactions.
- **Signatures:** This list contains all signatures σ that are part of previous transactions.

Finally, we want to keep track of the amount of coins that \mathcal{A} obtained from the game, and the amount of coins that it spent to honest users. This will be necessary to define security.

- $\text{received} \in \mathbb{N}_0$: This integer models how many coins the adversary obtained from the game, e.g., via transactions generated by honest users.
- $\text{spent} \in \mathbb{N}_0$: This integer models how many coins the adversary spent to the game, e.g., via transactions received by honest users.

Adversary Capabilities. The capabilities of an adversary are modeled by a set of oracles to which the adversary has access. When the adversary calls these oracles, the current state of the world may change. This means that the oracles trigger changes to the data structures discussed before. Formally, we present all oracles using pseudocode in the full version [12].

The first capability that adversary \mathcal{A} has is to interact with honest users and corrupt them. It can populate the system with honest users, and we model

two types of corruption. This reflects that users may store their keys in different locations. Additionally, we will see that the adversary can always generate identity public keys on its own and use them in transactions.

- $\text{NEWIDENTITY}()$: This oracle generates a new honest user. For that, it generates keys $(\text{ipk}, \text{ivk}, \text{isk}) \leftarrow \text{GenID}(\text{par})$. Then, it inserts ipk into the list Identities , and sets $\text{ivk}[\text{ipk}] := \text{ivk}$, $\text{isk}[\text{ipk}] := \emptyset$, $\text{corr}[\text{ipk}] := 0$. It returns ipk to \mathcal{A} .
- $\text{PARTCORR}(\text{ipk})$: This oracle allows \mathcal{A} to learn the identity view key of an honest user. Precisely, the oracle returns \perp if $\text{ipk} \notin \text{Identities}$ or $\text{corr}[\text{ipk}] \neq 0$. Otherwise, it sets $\text{corr}[\text{ipk}] := 1$ and returns $\text{ivk}[\text{ipk}]$ to \mathcal{A} .
- $\text{FULLCORR}(\text{ipk})$: This oracle allows \mathcal{A} to learn the identity signing key of an honest user. Precisely, the oracle returns \perp if $\text{ipk} \notin \text{Identities}$ or $\text{corr}[\text{ipk}] \neq 1$. Otherwise, it sets $\text{corr}[\text{ipk}] := 2$. Then, it updates received accordingly, i.e.,

$$\text{received} := \text{received} + \sum_{(\text{pk}, \text{com}, \text{amt}, \text{cr}, \text{sk}) \in \text{Owned}[\text{ipk}]} \text{amt}.$$

It returns $\text{isk}[\text{ipk}]$ to \mathcal{A} .

Recall that valid transactions are required to use outputs in the system as inputs. Thus, we need to introduce some initial supply of outputs, as otherwise there is no way to create a valid transaction. This corresponds to mining coins in the real world. In our model, we let \mathcal{A} arbitrarily create new outputs by calling one of the following two oracles. Recalling that an output contains a public key and a commitment, we may allow \mathcal{A} to compute the commitment on its own, or to let the game compute it. However, we need to keep track of the amount of coins that \mathcal{A} spawned in this way. Therefore, if \mathcal{A} submits a (potentially maliciously computed) commitment, it is only considered valid if it can be received by an honest user.

- $\text{NEWHONSRC}(\text{pk}, \text{pseed}, \text{com}, \text{tag}, \text{ct})$: This oracle tries find an honest user to receive the given output. For that, it runs Receive for each user $\text{ipk} \in \text{Identities}$ with $\text{corr}[\text{ipk}] < 2$. If for some user, the received amount is non-zero, it inserts (pk, com) into Outputs and stores the output together with the secrets necessary to spend it in the list $\text{Owned}[\text{ipk}]$.
- $\text{NEWSRC}(\text{pk}, \text{amt}, \text{cr})$: This oracle inserts (pk, com) into Outputs , where $\text{com} := \text{Com}(\text{amt}, \text{cr})$. It also updates received accordingly, i.e., $\text{received} := \text{received} + \text{amt}$.

Finally, it is clear that we should enable the adversary to put transactions on the ledger. Additionally, honest parties may publish transactions. For that, we let adversary \mathcal{A} instruct honest users to generate transactions with some specified receivers. We allow \mathcal{A} to determine the distribution from which the users sample decoys and coins that they spend.

- $\text{ADDADVTRANS}(\text{Tx})$: This oracle first verifies the given transaction using algorithm VerTx and the current state of the system given by PSeeds , Outputs , and Signatures . If the transaction is invalid, it returns. Otherwise, it updates

TXs, PSeeds, Outputs, and Signatures accordingly, by inserting Tx into TXs, its public seed pseed into PSeeds, all its outputs (pk, com) into Outputs, and all its signatures into Signatures. It also updates the owned outputs of all honest users by running algorithm Receive for every honest user and every output of Tx, and then updating Owned accordingly. Finally, it sets $\text{spent} := \text{spent} + \text{spentnow}$, where spentnow is the total amount that honest users received from Tx.

- $\text{ADDHONTRANS}(\text{ipk}, (\text{ipk}_i, \text{amt}_i^{\text{out}})_{i=1}^K, \text{ISamp}, \text{RSamp})$: By calling this oracle, \mathcal{A} instructs an honest user with identity public key ipk to pay $\text{amt}_i^{\text{out}}$ coins to identity public key ipk_i for each $i \in [K]$. For that, the honest user should use distribution ISamp to determine the outputs that should be used as inputs, and distribution RSamp to determine the remaining decoys. Precisely, this oracle returns if $\text{ipk} \notin \text{Identities}$ or $\text{corr}[\text{ipk}] = 2$. Otherwise, it generates a transaction as follows.
 1. Sample inputs to use by running $(\text{Use}_j)_{j=1}^L \leftarrow \text{ISamp}(\text{Owned}[\text{ipk}])$ and $(\text{Ref}_j)_{j=1}^L \leftarrow \text{RSamp}(\text{Owned}[\text{ipk}], (\text{Use}_j)_{j=1}^L)$.
 2. Check validity of the inputs. Namely, each $\text{Use}_j = (\text{pk}_j, \text{com}_j, \text{amt}_j, \text{cr}_j, \text{sk}_j)$ should be in $\text{Owned}[\text{ipk}]$, the output $(\text{pk}_j, \text{com}_j)$ contained in Use_j should be in Ref_j , and each $(\text{pk}, \text{com}) \in \text{Ref}_j$ should be in Outputs . Also, $\sum_{j=1}^L \text{amt}_j = \sum_{i=1}^K \text{amt}_i^{\text{out}}$ should hold. If one of these conditions does not hold, the oracle returns \perp .
 3. Generate the transaction Tx by running algorithm GenTx . If the transaction is not valid, return \perp . We will see later that the adversary wins the game in this case.

Next, the oracle updates $\text{Owned}[\text{ipk}] := \text{Owned}[\text{ipk}] \setminus \{\text{Use}_j\}_{j \in [L]}$. It also updates Owned , TXs, PSeeds, Outputs as in oracle ADDADVTRANS . Then, it updates received accordingly, i.e., $\text{received} := \text{received} + \text{amt}_i^{\text{out}}$ for each $i \in [K]$ with $\text{ipk}_i \notin \text{Identities}$ or $\text{corr}[\text{ipk}] = 2$. Finally, it returns Tx to \mathcal{A} .

Security Notion. Next, we define the security notion for a private transaction scheme $\text{PTS} = (\text{LRS}, \text{KDS}, \text{VHC}, \text{DE}, \text{KCS})$. To this end, we introduce a security game $\text{UNF}_{\text{PTS}}^{\mathcal{A}}(\lambda)$ for an adversary \mathcal{A} . In the security game, \mathcal{A} interacts with all oracles defined above. Informally, \mathcal{A} breaks the security of the system, if it can create money out of thin air, or prevent honest users from spending their coins. Therefore, we say that \mathcal{A} wins the security game, if at least one of the following two events occur at any point during the game:

1. Event win-create: We have $\text{spent} > \text{received}$.
2. Event win-steal: Adversary \mathcal{A} instructs an honest user to generate a transaction using oracle ADDHONTRANS , a transaction Tx is generated accordingly, but does not verify, i.e., $\text{VerTx}(\text{PSeeds}, \text{Outputs}, \text{Signatures}, \text{Tx}) = 0$.

Consider a private transaction scheme $\text{PTS} = (\text{LRS}, \text{KDS}, \text{VHC}, \text{DE}, \text{KCS})$. For any algorithm \mathcal{A} we define the game $\text{UNF}_{\text{PTS}}^{\mathcal{A}}(\lambda)$ as follows:

1. Consider oracles $\text{O}_{id} := (\text{NEWIDENTITY}, \text{PARTCORR}, \text{FULLCORR})$, $\text{O}_{src} = (\text{NEWHONSRC}, \text{NEWSRC})$, and $\text{O}_{tx} = (\text{ADDADVTRANS}, \text{ADDHONTRANS})$ described above.

2. Run \mathcal{A} with access to oracles O_{id}, O_{src}, O_{tx} on input 1^λ .
3. Output 1, if $\text{win-create} = 1$ or $\text{win-steal} = 1$. Otherwise, output 0.

We say that PTS is secure, if for every PPT algorithm \mathcal{A} the following advantage is negligible:

$$\text{Adv}_{\mathcal{A}, \text{PTS}}^{\text{unf}}(\lambda) := \Pr \left[\text{UNF}_{\text{PTS}}^{\mathcal{A}}(\lambda) \rightarrow 1 \right].$$

4 Overview of Our Analysis

In this section, we give an overview of our formal analysis of RingCT. Due to space limitations, we keep this overview informal. We encourage the interested reader to consult our full version [12] for the detailed formal analysis. Our strategy consists of three steps. First, we introduce security notions for (combinations of) the subcomponents of a private transaction scheme. Second, we show that whenever the subcomponents satisfy these notions, the security of the private transaction scheme follows generically. Third, we prove that the instantiations of subcomponents used in Monero satisfy the respective notions.

4.1 Security Notions for Components

We introduce several security notions for the subcomponents of a private transaction scheme. For each notion, we informally sketch a security game. The formal games are given in our full version [12]. We also aim to convey an intuition for why it is needed in the context of a private transaction scheme. Throughout this section, we fix a private transaction scheme $\text{PTS} = (\text{LRS}, \text{KDS}, \text{VHC}, \text{DE}, \text{KCS})$ and an efficient adversary \mathcal{A} . We assume that \mathcal{A} gets the system parameters par at the beginning of each game.

Tracking Soundness. Recall that an honest user recognizes received outputs using algorithm Track . We want to ensure that when an honest user recognizes such an output (i.e., Track outputs 1), then this output can later be spent. In other words, if Track outputs 1, then a valid secret key will be derived. We capture this by the notion of tracking soundness. In the security game, \mathcal{A} gets as input the keys $\text{ipk}, \text{ivk}, \text{isk}$ of an honest user, which are generated via $(\text{ipk}, \text{ivk}, \text{isk}) \leftarrow \text{GenID}(\text{par})$. Then, it outputs $\text{pseed}, \text{pk}, \text{tag}$, and the honest user runs $\text{ok} := \text{RecDecaps}(\text{ivk}, \text{pseed})$, $b := \text{Track}(\text{ipk}, \text{ok}, \text{pk}, \text{tag})$ and $\text{sk} := \text{DerSK}(\text{isk}, \text{ok}, \text{tag})$ as it does when trying to receive an output. The adversary \mathcal{A} wins if $b = 1$ but sk is not a valid secret key for pk .

Key Spreadness. We introduce a notion that we call key spreadness. Roughly, it states that different public seeds $\text{pseed}, \text{pseed}'$ or different tags tag, tag' lead to different derived keys sk, sk' . Looking ahead, this ensures that no two signatures generated by honest users link. In the security game for key spreadness, \mathcal{A} gets access to an honest user's keys $\text{ipk}, \text{ivk}, \text{isk}$ and outputs seeds $\text{pseed}, \text{pseed}'$ and

tags tag, tag' . Then, the honest user derives keys sk and sk' via $\text{sk} := \text{DerSK}(\text{isk}, \text{RecDecaps}(\text{ivk}, \text{pseed}), \text{tag})$ and $\text{sk}' := \text{DerSK}(\text{isk}, \text{RecDecaps}(\text{ivk}, \text{pseed}'), \text{tag}')$. Finally, \mathcal{A} wins this game if $(\text{pseed}, \text{tag}) \neq (\text{pseed}', \text{tag}')$ but $\text{sk} = \text{sk}'$.

Conversion Soundness. Recall that the key conversion scheme KCS allows to transform pairs of commitment randomness cr, cr' for commitments com, com' to the same data amt into auxiliary keys auxsk and auxpk . Intuitively, when one then uses auxsk in the ring signature and cr' in a pseudo output commitment, this should prove that one knew cr . Our notion of conversion soundness roughly states that knowing auxsk and cr' implies (via a translation algorithm) knowing cr . In other words, if conversion soundness holds, then it is enough to show that generating a valid transaction requires knowledge of auxsk and cr' . For the formal notion, we require that there is an efficient algorithm Translate , such that (any efficient) \mathcal{A} wins the following game only with negligible probability. First, \mathcal{A} outputs $\text{amt}, \text{cr}', \text{com}, \text{com}', \text{auxpk}, \text{auxsk}$. Then Translate is run given all these elements as input and outputs cr . The adversary wins the game if its input was well formed, i.e., $\text{Com}(\text{amt}, \text{cr}') = \text{com}'$, $(\text{auxpk}, \text{auxsk})$ are a valid key pair, and $\text{ConvertPublic}(\text{com}, \text{com}') = \text{auxpk}$, but translation failed, i.e., $\text{Com}(\text{amt}, \text{cr}) \neq \text{com}$ or $\text{ConvertSecret}(\text{cr}, \text{cr}') \neq \text{auxsk}$.

Binding Commitment. Clearly, the commitment scheme should satisfy the standard notion of binding. This ensures that an adversary can not change the amount of an output. To recall, \mathcal{A} breaks binding if it outputs (amt, cr) and $(\text{amt}', \text{cr}')$ such that $(\text{amt}, \text{cr}) \neq (\text{amt}', \text{cr}')$ and $\text{Com}(\text{amt}, \text{cr}) = \text{Com}(\text{amt}', \text{cr}')$.

Commitment Knowledge Soundness. We introduce the notion of commitment knowledge soundness. Roughly, it states that the proofs π included in transactions are proofs of knowledge. Precisely, if an adversary generates pseudo output commitments and output commitments for a transaction along with a proof π , then the adversary must know the corresponding amt and commitment randomness cr . Looking ahead, the technical reason why we require a proof of knowledge is that we have to extract cr before we can reduce to binding in an overall proof of security. We stress the importance of being able to extract multiple times from the adversary. This is because we need to run the extractor for every submitted transaction in our overall proof. In our formal definition, we require the existence of an efficient extractor Ext_{VHC} such that no efficient \mathcal{A} wins the following game with non-negligible probability. The adversary \mathcal{A} gets access to an oracle O . Whenever \mathcal{A} calls O , it submits a statement $\text{stmt} = (\text{com}_i)_i$ and a proof π . Then, the game runs the extractor $\text{witn} \leftarrow \text{Ext}_{\text{VHC}}(\text{stmt}, \pi)$ to get a witness $\text{witn} = (\text{amt}_i, \text{cr}_i)_i$. If in any of these queries we have $\text{PVer}(\text{stmt}, \pi) = 1$, i.e., the proof verifies, and there is some i such that $\text{Com}(\text{amt}_i, \text{cr}_i) \neq \text{com}_i$, i.e., extraction failed, then \mathcal{A} wins.

Non-slanderability. A well-established notion for linkable ring signatures is non-slanderability [27, 51] (sometimes called non-frameability [2, 5]). This notion states that it is not possible for an adversary to come up with a signature that links to an honest user’s signature. In our setting, this means that it can not happen that an honest user computes a signature on a transaction using a valid secret key, and this transaction gets rejected because the signature links to a previous signature. However, we can not just use the standard non-slanderability notion, because the key derivation scheme KDS introduces non-trivial relations between keys. Hence, we define a game that is similar to non-slanderability, but for keys that are derived using KDS. When making signature queries, the adversary can specify the parameters with which the secret key sk is derived from an identity signing key isk . Let us now give an overview of our non-slanderability game. In this game, \mathcal{A} gets access to oracles `NEWIDENTITY`, `CORR`, and `SIGN`. When called, oracle `NEWIDENTITY` generates keys $(ipk, ivk, isk) \leftarrow \text{GenID}(\text{par})$ for a new honest user, and returns (ipk, ivk) to \mathcal{A} . The adversary \mathcal{A} can corrupt any such user and learn isk by querying `CORR(ipk)`. Further, it can ask for signatures using oracle `SIGN`. Here \mathcal{A} submits $pk, pseed, tag, R, auxsk, m$. The oracle then finds an honest user with key (ipk, ivk, isk) that owns pk , i.e., such that $\text{Track}(ipk, ok, pk, tag) = 1$ for $ok := \text{RecDecaps}(ivk, pseed)$. Then, it derives sk from ok, isk , and tag using algorithm `DerSK` and computes a signature $\sigma \leftarrow \text{Sig}(R, sk, auxsk, m)$ on the message m . This signature is returned to \mathcal{A} . When \mathcal{A} terminates, it outputs a tuple (R^*, m^*, σ^*) . It wins the game, if σ^* is valid for message m^* and R^* , \mathcal{A} never received a signature from `SIGN` by querying m^* and R^* together, and there is a non-corrupted honest user that computed a signature σ in `SIGN` such that $\text{Link}(\sigma, \sigma^*) = 1$.

Key Onewayness. We also define a weaker notion related to non-slanderability. Namely, the adversary should not be able to come up with secret keys without corrupting a user, given access to the same oracles as in the non-slanderability game. More precisely, in the key onewayness game, \mathcal{A} gets access to the same oracles as for non-slanderability, and outputs a tuple $(ipk^*, pk^*, pseed^*, tag^*, sk^*)$. It wins, if (pk^*, sk^*) are a valid key pair, ipk^* is the key of an honest and non-corrupted user, and this honest user recognizes pk^* as its key, i.e., $\text{Track}(ipk^*, ok, pk^*, tag^*) = 1$ for $ok := \text{RecDecaps}(ivk, pseed^*)$, where ivk is the identity view key of this user.

Key Knowledge Soundness. If we want to use the notion of conversion soundness introduced above, we first need to extract an auxiliary secret key $auxsk$ from an adversary submitting a transaction. Therefore, we introduce a strong property called key knowledge soundness. Roughly speaking, it states that LRS is a signature of knowledge, i.e., the adversary can only come up with a valid signature if it knows a valid secret key $(sk, auxsk)$. Before we present the definition, let us discuss one subtlety. A natural way of defining this notion would be to allow the adversary to submit tuples (R, m, σ) to an oracle O , and let this oracle try to extract suitable secret keys via an extractor in the algebraic group

model. If this extraction fails, the adversary wins. While this is a good start, it is not exactly what we want. Namely, in our setting, the adversary also receives signatures from the outside, e.g., when we want to do a reduction breaking key onewayness. If the adversary simply submits these signatures to O , there is no hope that the adversary knew any secret keys. On a technical level, we would also encounter composition problems with the algebraic group model. This is because our definition defines the basis for algebraic representations that the algebraic adversary submits, and that are used by the extractor. If this basis is different when we want to apply key knowledge soundness (e.g., because the adversary received additional group elements as part of keys of honest users), then the extractor is useless. This motivates why we give additional oracles to the adversary in our notion.

We now sketch the final definition of key knowledge soundness. We require that there is an efficient extractor Ext_{LRS} , such that no efficient adversary \mathcal{A} wins the following game. Adversary \mathcal{A} gets access to the same oracles NEWIDENTITY , CORR , SIGN as in the non-slanderability game. Further, it gets access to an oracle O and wins, if for at least one of its queries to O , a certain winning condition is triggered. When \mathcal{A} calls O , it has to submit a triple (R, m, σ) such that it never submitted R, m together to the signing oracle SIGN and obtained a signature for it, and also it has to hold that $\text{Ver}(R, m, \sigma) = 1$, i.e., σ is a valid signature for R, m . If these conditions hold, parse R as $R = (\text{pk}_i, \text{auxpk}_i)_i$. Then, the extractor Ext_{LRS} is run and outputs $(i^*, \text{sk}_{i^*}, \text{auxsk}_{i^*})$, which intuitively should mean that \mathcal{A} used secret keys $\text{sk}_{i^*}, \text{auxsk}_{i^*}$ for $\text{pk}_{i^*}, \text{auxpk}_{i^*}$ to compute σ . The adversary wins if this extraction failed, i.e., $(\text{auxpk}_{i^*}, \text{auxsk}_{i^*})$ or $(\text{pk}_{i^*}, \text{sk}_{i^*})$ are not a valid key pair.

Knowledge Linkability. Typically, linkable ring signatures should satisfy linkability. Informally, this notion states that if one uses the same secret key to compute two signatures, then these will link. The formalization of this intuition is non-trivial. In particular, we observe that the standard formalization (sometimes called pigeonhole linkability) is not enough for our purposes (cf. Section 1.2). Instead, we need a notion that is compatible with the extractor we defined for key knowledge soundness. This is because, in some sense, the extractor already tells us which key was used to compute a signature. Motivated by this, we define knowledge linkability, which roughly rules out that the extractor extracted the same sk twice from two signatures σ, σ' that do not link. In other words, it guarantees that if the extractor extracts the same key twice, then the corresponding signatures must link. More concretely, the knowledge linkability game is similar to the key knowledge soundness game that we introduced before. The only change is the winning condition in oracle O . To describe this new winning condition, we use the notation that we used to describe the key knowledge soundness game. With this notation, \mathcal{A} wins, if it submits a triple (R, m, σ) to O , subject to the same restrictions as in key knowledge soundness, and σ does not link to any signature σ' output by SIGN or submitted to O before, but a

secret key associated to pk_{i^*} has been used to sign before. This includes the case where the extractor identified pk_{i^*} as the signing public key before.

4.2 System Level Analysis

We show that any private transaction scheme is secure, given that its subcomponents satisfy the notions introduced in the previous section. Informally, we prove the following statement.

Theorem 1 (Informal). *Let PTS be a private transaction scheme. Assume that the subcomponents of PTS satisfy all security notions introduced in Sect. 4.1. Then, PTS is secure.*

For the formal statement, we refer to our full version [12]. We now present the main ideas used to prove this theorem. For both this informal overview and the formal analysis, we consider the two winning conditions separately.

Honest User Can not Spend. We start with winning condition `win-steal`. Informally, the adversary wins via winning condition `win-steal`, if there is a transaction with an output o that an honest user receives, and later the honest user can not spend this output. More concretely, the adversary instructs the user to compute a transaction Tx using this output via algorithm GenTx , and then Tx is invalid, i.e., VerTx outputs 0. To bound the probability of this event, we consider the different conditions that make algorithm VerTx output 0. Write $\text{Tx} = (\text{In}, \text{Out}, \text{pseed}, \pi)$, $\text{In} = (\text{Ref}_j, \text{com}_j, \sigma_j)_{j=1}^L$, and $\text{Out} = (\text{pk}_i^{\text{out}}, \text{com}_i^{\text{out}}, \text{ct}_i, \text{tag}_i)_{i=1}^K$. The cases are as follows.

- VerTx may output 0 because the public seed pseed contained in Tx is not fresh, i.e., there is a previous transaction that has the same public seed. As pseed is generated freshly by the honest user during generation of Tx , we can rely on the entropy of pseed to rule this case out.
- VerTx may output 0 because some input contained in the transaction Tx is not a previous output. However, an honest user would never include such an input in a transaction. Thus, this case can not occur.
- VerTx may output 0 because commitments included in the transaction Tx are not valid, i.e., the proof π does not verify, or $\sum_{j=1}^L \text{com}_j \neq \sum_{i=1}^K \text{com}_i^{\text{out}}$. Note that all involved commitments and the proof π are computed honestly in GenTx , and it follows from the completeness of VHC that this case never happens.
- VerTx may output 0 because one of the signatures σ_j is not valid.
- VerTx may output 0 because of double spending detection. That is, it may reject the transaction because one of the signatures σ_j links to a previous signature.

For the last two cases, we observe that the secret key that is used to compute the signatures is derived from the output o , which is provided by the adversary.

Therefore, we can not use completeness properties immediately and require additional arguments. Namely, for the case of invalid signatures, we first apply the tracking soundness notion. This notion tells us that for any output $o = (\text{pk}, \text{com})$ that an honest user receives from an adversary, it derives a valid secret key sk such that $(\text{pk}, \text{sk}) \in \mathcal{KR}$. Now, we can apply completeness of LRS to argue that the signature is always valid. The case of linking signatures is a bit more challenging. Namely, we consider two sub-cases. If the signature links to a maliciously generated signature, i.e., a signature that is contained in a transaction that the adversary submitted, then the adversary breaks non-slanderability. On the other hand, if the signature links to a signature that is also generated by an honest user, then we want to use the completeness property of LRS again. Specifically, it states that signatures computed honestly using different secret keys do not link. Now, it remains to argue that an honest party does not use the same secret key twice. For that, we make use of the key spreadness notion, and the fact that public seeds are not reused.

Adversary Creates Money. Consider the second winning condition win-create. Roughly, our main strategy is to define a directed graph G with weighted edges modeling the state of the system during the security game. Then, we use the security notions for building blocks to argue that this satisfies the conditions of a flow network. Recall that in such a flow network, a flow value $f(e) \geq 0$ is assigned to each edge e in the graph, such that for each vertex (except a dedicated source and sink) the incoming flow equals the outgoing flow. Then, we use the theory of network flows to conclude. We will now make this rough idea more explicit. Namely, our proof proceeds in four main steps, which are as follows:

1. We consider each transaction and extract all hidden amounts and used secret keys. More precisely, for each output and pseudo output of the transaction, we extract the hidden amount and random coins for the commitments using commitment knowledge soundness. For each signature contained in the transaction, we extract the secret key and auxiliary secret that have been used to generate the signature. This is done using key knowledge soundness. Especially, we are now able to distinguish real inputs from decoys.
2. Using the knowledge we gained in the first step, we define a directed graph $G = (V, E)$, and an assignment $f(e) \geq 0$ to each edge $e \in E$. In this graph, for each output and each transaction in the system, there is an associated vertex. Whenever an output is used in a transaction as an input, there is an edge e from the output vertex to the transaction vertex. Further, there is an edge from each transaction to all of its outputs. In addition, there are dedicated vertices s and t , where $\{s, t\} \subseteq V$. For each source output, we add an edge from s to the vertex of this output. Finally, we add an edge from each output vertex that does not have an outgoing edge yet to t . In other words, outgoing edges of s model the initial money supply of the system, while ingoing edges of t model unused outputs. In terms of edge weights $f(e)$, notice that each

edge e is incident to one² output vertex. We set $f(e)$ to be the amount that we extracted from this output in the first step.

3. We show that this graph G and the assignment f define a flow network. To do so, we need to prove that for each vertex v (except s and t) the incoming flow, i.e., $\sum_{e=(u,v) \in E} f(e)$, is equal to the outgoing flow, i.e., $\sum_{e=(v,w) \in E} f(e)$. For that, we distinguish transaction and output vertices:
 - (a) For transaction vertices, we first show that the sum of amounts is preserved between pseudo outputs and outputs. To do that, we use the homomorphic property and the binding property of VHC. Then, we show that for each input, the amount is preserved between the input (which is the output of a previous transaction) and the associated pseudo output. For that, we first leverage conversion soundness, and then apply binding of VHC once more. Note that we can only reduce from binding because we extracted amounts and random coins for each commitment before.
 - (b) Each output vertex has in-degree one by definition. Thus, as long as we can show it also has out-degree one, the flow preservation follows. The main tool to show this is knowledge linkability.
4. Now that we showed that we have a flow network, we leverage the theory of flow networks to conclude. Omitting some details, this works as follows. Recall that an st -cut in G is a partition of V into two disjoint sets of vertices V_s, V_t with $s \in V_s$ and $t \in V_t$. The value of any such st -cut is the net flow from V_s to V_t , i.e., the flow from V_s to V_t minus the flow from V_t to V_s . In our proof, we are now interested in the following st -cut. We let V_s contain s and all vertices that are controlled by honest parties, i.e., transactions that honest parties created and outputs that are owned by honest parties. We let V_t contain all other vertices, i.e., t , all transactions created by the adversary, and all outputs not owned by honest parties. For this specific cut, we can argue that its value is at most $L + \text{received} - \text{spent}$, where L is the flow from V_s to vertex t . To see that, note that the flow from V_s to V_t is at most $L + \text{received}$, because each edge with weight f from V_s to V_t which is not going into vertex t increases the value of received by f . Further, the flow from V_t to V_s is at least spent , because whenever spent is increased by f , a new edge with weight f from V_t to V_s is added to the graph.

Recall that it is our goal to argue that $\text{received} - \text{spent} \geq 0$, i.e., the adversary spent at most as much as it received. Now, our central idea is to rely on the fact that in any flow network, the value of any cut is equal to the incoming flow T of the sink vertex t . In combination with the observation above, this shows that $L + \text{received} - \text{spent} \geq T$. By definition, we have $T \geq L$, and thus $L + \text{received} - \text{spent} \geq L$. Subtracting L from both sides, we get $\text{received} - \text{spent} \geq 0$, i.e., $\text{received} \geq \text{spent}$, which means the adversary can not create coins.

² Special care needs to be taken for corruptions, but we ignore them in this informal overview.

4.3 Component Level Analysis

To conclude that Monero’s transaction scheme RingCT is secure, it remains to show security of its subcomponents with respect to the notions introduced in Sect. 4.1.

Theorem 2 (Informal). *Let PTS be the RingCT private transaction scheme. The subcomponents of PTS satisfy the security notions introduced in Sect. 4.1.*

We provide the formal theorem in our full version [12].

We prove all notions based on the discrete logarithm assumption in the random oracle model. While for some of the notions (e.g., binding) standard techniques suffice, the analysis of the linkable ring signature component turns out to be the most challenging part. Here, we rely on the algebraic group model to prove key knowledge soundness, which is natural for a knowledge-based security notion. We emphasize that we do not prove the notion of commitment knowledge soundness. This notion is defined for algorithms PProve and PVer of VHC, which are implemented using Bulletproofs/Bulletproofs+ [7, 11] in Monero. A detailed analysis of this would not fit the scope of this work, and we leave it as a conjecture that the schemes satisfy commitment knowledge soundness. For an analysis of Bulletproofs in a similar model the reader may consult [26].

5 Other Models for RingCT-Like Systems

While no previous work analyzes Monero’s transaction scheme RingCT as it is, some previous works [18, 19, 35, 49, 56] introduce models for protocols similar to RingCT. In this section, we elaborate on the shortcomings of these models. We also encourage the reader to consult the discussion on different models in [19, 35]. As our work is only about transaction security and not about privacy, we omit discussing the privacy aspects of these previous models. We assume that the reader is familiar with our overview in Sect. 2.

Fragmented Security Notions. In our work, we provide a single experiment defining security for the transaction scheme as a whole. Informally, security means that an adversary can only spend what it owns, and not steal users coins. Unfortunately, most previous models [35, 49, 56] do not give a single security model for that. Instead, they provide a set of notions for components. Mostly, these mimic the standard notions of a linkable ring signature scheme, e.g., non-slanderability, linkability, unforgeability, and the notions of a commitment scheme, e.g., binding. We call such a model *fragmented*. The problem of such a model is that it is not clear how the notions relate, whether they compose, and how they imply security for the entire transaction scheme. For example, in [35], it is not obvious how and why the notions of binding, balance, and non-slanderability imply security of the entire transaction scheme when combined. Comparing to our work, fragmented models are somewhat similar to the set of security notions we define for our components. For example, we also have a

binding and a non-slanderability notion for the components. Arguing that such a set of notions implies the security of the entire transaction scheme is highly non-trivial, as our analysis shows.

Adversarial Outputs. Recall that in Monero, each output of a transaction corresponds to a public key pk and a commitment com . If an adversary creates a transaction spending coins to an honest user, it derives this public key pk and the commitment com based on the public seed pseed of a transaction, and the recipients identity public key ipk . As a consequence, the adversary may know relations between different outputs of the same honest user, possibly leading to related key attacks. This means that any reasonable security model has to give the adversary the ability to derive outputs for honest users. We observe that several security models in previous works [18, 19, 49, 56] do not have this feature.

Sun et al.’s RingCT 2.0. Sun et al. introduce [49] a model for protocols similar to RingCT and give a new construction based on pairings. Their model has several shortcomings. First, by defining security via two notions called balance and non-slanderability, they obtain a fragmented model in the above sense. Second, in terms of adversarial capabilities, their model is restricted. For example, as already noted in [35], their notions do not model adversarially generated outputs (i.e., stealth addresses). Instead, they only consider honestly generated outputs, which can not be assumed in the case of Monero. Moreover, the adversary does not have the ability to submit an arbitrary transaction to the chain. Instead, it can only add transactions by calling an oracle that honestly creates the transaction. Overall, these aspects limit the expressiveness of the model significantly. Third, the authors of [49] informally claim that linkability follows from their non-slanderability notion. As explained in [35], this is not true in general. In the context of Monero, this means that there can be counterexamples in which the given non-slanderability notion holds but double spending is possible.

Yuen et al.’s RingCT 3.0. Yuen et al. [56] also provide a model for protocols similar to RingCT and give a construction based on a new ring signature scheme. In terms of security, Yuen et al. provide three notions, called unforgeability, equivalence, linkability, and non-slanderability, which is fragmented in the above sense. Similar to the model by Sun et al. [49], the adversary can only add transactions via an oracle that generates these transactions honestly, and all outputs of honest parties are derived honestly.

Lai et al.’s Omniring. Lai et al. [35] introduce a model for transaction schemes and propose a new scheme that is more efficient than Monero’s current transaction scheme. Then, they give an analysis of this new scheme with respect to their notions. In their model, Lai et al. first introduce two security properties, called balance and binding. Binding is defined in a natural way, and balance

is formalized via an extractor that can extract all witnesses from an adversarially generated transaction. Moreover, non-slanderability is defined as a separate notion. This leads to a fragmented model and it is not clear how these three notions relate to each other and what they mean in combination. For example, while the non-slanderability notion gives the adversary access to oracles that allow to add transactions to the system arbitrarily, this is not the case for the balance and binding notions. Also, while having an extractor seems to be close to one of the security notions we introduce for components, the extractor in [35] only has to work for a single transaction. It is not clear what happens if we run such an extractor for multiple transactions. For example, the extractor is allowed to use rewinding, leading to an exponential blowup in running time when done naively on multiple transactions. Finally, the model of Lai et al. does not capture that honest users reuse randomness within one transaction for creating the outputs.

MatRiCT and MatRiCT⁺. In [18, 19], constructions of transaction schemes based on lattice assumptions are presented. Contrary to previous works, both works provide a single experiment for security instead of giving fragmented security models. On the downside, both works [18, 19] do not model adversarially generated outputs (i.e., stealth addresses). It is mentioned in Appendix C.A of [18] that stealth addresses can be added to their lattice-based scheme in an easy way. However, it is clear that not modeling stealth addresses formally completely removes the challenge of dealing with related key attacks as discussed before. Finally, both works [18, 19] do not model the reuse of randomness for output generation of honest users.

6 Limitations and Future Work

In our work, we only deal with standard Monero addresses and do not consider the case of subaddresses or integrated addresses. We also do not cover multi-signatures and multi-signature addresses. This work focuses on the security of Monero’s transaction scheme. In particular, we do not consider the consensus layer, and we do not model privacy of the transaction scheme. We plan to elaborate a model and analysis for privacy in future work. As it is standard in the literature, we use the abstraction of a prime order group to analyze the components of Monero, while it is actually implemented over curve Ed25519 [4]. Due to the modularity of our framework, one could extend our results to the setting of Ed25519 (in the spirit of, e.g., [6]) without the need of redoing the entire analysis. We assume that transaction public keys are never reused, yet we observe the consequences for transaction proofs (see the full version [12]). Finally, we do not show that the Bulletproof/Bulletproof+ component [7, 8, 11, 25] of the system satisfies the security notion we define for it. It has been shown in [26] that Bulletproofs satisfy a related notion. After discussion with the authors of [26], we conjecture that their proof can be extended to show that Bulletproofs satisfy our notion as well. We leave investigating all of these directions as future work.

Acknowledgments. Julian Loss and Benedikt Wagner are funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 507237585, and by the European Union, ERC-2023-STG, Project ID: 101116713. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

1. Alonso, K.M., Joancomartí, J.H.: Monero - privacy in the blockchain. *Cryptology ePrint Archive, Report 2018/535* (2018). <https://eprint.iacr.org/2018/535>
2. Backes, M., Döttling, N., Hanzlik, L., Kluczniak, K., Schneider, J.: Ring signatures: logarithmic-size, no setup—from standard assumptions. In: Ishai, Y., Rijmen, V. (eds.) *EUROCRYPT 2019*. LNCS, vol. 11478, pp. 281–311. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_10
3. Ben-Sasson, E., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: *2014 IEEE Symposium on Security and Privacy*, pp. 459–474. IEEE Computer Society Press (2014). <https://doi.org/10.1109/SP.2014.36>
4. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) *CHES 2011*. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23951-9_9
5. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falafel: logarithmic (linkable) ring signatures from isogenies and lattices. In: Moriai, S., Wang, H. (eds.) *ASIACRYPT 2020*. LNCS, vol. 12492, pp. 464–492. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_16
6. Brendel, J., Cremers, C., Jackson, D., Zhao, M.: The provable security of Ed25519: theory and practice. In: *2021 IEEE Symposium on Security and Privacy*, pp. 1659–1676. IEEE Computer Society Press (2021). <https://doi.org/10.1109/SP40001.2021.00042>
7. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: *2018 IEEE Symposium on Security and Privacy*, pp. 315–334. IEEE Computer Society Press (2018). <https://doi.org/10.1109/SP.2018.00020>
8. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: Tibouchi, M., Wang, H. (eds.) *ASIACRYPT 2021*. LNCS, vol. 13092, pp. 65–97. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92078-4_3
9. Cao, T., Yu, J., Decouchant, J., Luo, X., Verissimo, P.: Exploring the Monero peer-to-peer network. In: Bonneau, J., Heninger, N. (eds.) *FC 2020*. LNCS, vol. 12059, pp. 578–594. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_31
10. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) *EUROCRYPT 2020*. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_26
11. Chung, H., Han, K., Ju, C., Kim, M., Seo, J.H.: Bulletproofs+: shorter proofs for a privacy-enhanced distributed ledger. *IEEE Access* **10**, 42067–42082 (2022). <https://doi.org/10.1109/ACCESS.2022.3167806>

12. Cremers, C., Loss, J., Wagner, B.: A holistic security analysis of Monero transactions. Cryptology ePrint Archive, Report 2023/321 (2023). <https://eprint.iacr.org/2023/321>
13. Deuber, D., Ronge, V., Rückert, C.: SoK: assumptions underlying cryptocurrency deanonymizations. PoPETs **2022**(3), 670–691 (2022). <https://doi.org/10.56553/popets-2022-0091>
14. Dutta, A., Bagad, S., Vijayakumaran, S.: MProve+: privacy enhancing proof of reserves protocol for Monero. IEEE Trans. Inf. Forensics Secur. **16**, 3900–3915 (2021). <https://doi.org/10.1109/TIFS.2021.3088035>
15. Dutta, A., Vijayakumaran, S.: MProve: a proof of reserves protocol for Monero exchanges. In: 2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, 17–19 June 2019, pp. 330–339. IEEE (2019). <https://doi.org/10.1109/EuroSPW.2019.00043>
16. Eagen, L.: Bulletproofs++. Cryptology ePrint Archive, Report 2022/510 (2022). <https://eprint.iacr.org/2022/510>
17. Egger, C., Lai, R.W.F., Ronge, V., Woo, I.K.Y., Yin, H.H.F.: On defeating graph analysis of anonymous transactions. PoPETs **2022**(3), 538–557 (2022). <https://doi.org/10.56553/popets-2022-0085>
18. Esgin, M.F., Steinfeld, R., Zhao, R.K.: MatRiCT⁺: more efficient post-quantum private blockchain payments. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, 22–26 May 2022, pp. 1281–1298. IEEE (2022). <https://doi.org/10.1109/SP46214.2022.9833655>
19. Esgin, M.F., Zhao, R.K., Steinfeld, R., Liu, J.K., Liu, D.: MatRiCT: efficient, scalable and post-quantum blockchain confidential transactions protocol. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 567–584. ACM Press (2019). <https://doi.org/10.1145/3319535.3354200>
20. Frost, L.: Monero developers disclose ‘significant’ bug in privacy algorithm. <https://decrypt.co/76938/monero-developers-disclose-significant-bug-privacy-algorithm>. Accessed 14 Feb 2023
21. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_2
22. Fuchsbauer, G., Orrù, M.: Non-interactive Mimblewimble transactions, revisited. Cryptology ePrint Archive, Report 2022/265 (2022). <https://eprint.iacr.org/2022/265>
23. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: a cryptographic investigation of Mimblewimble. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 657–689. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_22
24. Fujisaki, E., Suzuki, K.: Traceable ring signature. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 181–200. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_13
25. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07085-3_14
26. Ghoshal, A., Tessaro, S.: Tight state-restoration soundness in the algebraic group model. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 64–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_3

27. Goodell, B., Noether, S., Blue, A.: Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Paper 2019/654 (2019). <https://eprint.iacr.org/2019/654>, <https://eprint.iacr.org/2019/654>
28. Gugger, J.: Bitcoin-monero cross-chain atomic swap. Cryptology ePrint Archive, Report 2020/1126 (2020). <https://eprint.iacr.org/2020/1126>
29. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash Protocol Specification, Version 2022.3.8. <https://zips.z.cash/protocol/protocol.pdf>. Accessed 15 Feb 2023
30. Jedusor, T.E.: Mumblewimble. <https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt>. Accessed 15 Feb 2023
31. Jivanyan, A., Feickert, A.: Lelantus spark: Secure and flexible private transactions. Cryptology ePrint Archive, Report 2021/1173 (2021). <https://eprint.iacr.org/2021/1173>
32. Klee, C.: Monero XMR: “Signifikanter” Privacy Bug entdeckt. <https://www.btc-echo.de/schlagzeilen/monero-xmr-signifikanter-privacy-bug-entdeckt-123001/>. Accessed 14 Feb 2023
33. Koe, Alonso, K.M., Noether, S.: Zero to Monero v2.0.0. <https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf> (2020). Accessed 21 Nov 2022
34. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A traceability analysis of Monero’s blockchain. In: Foley, S.N., Gollmann, D., Sneekenes, E. (eds.) ESORICS 2017. LNCS, vol. 10493, pp. 153–173. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66399-9_9
35. Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: scaling private payments without trusted setup. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 31–48. ACM Press (2019). <https://doi.org/10.1145/3319535.3345655>
36. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for Ad Hoc groups. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 325–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_28
37. luigi1111, “fluffypony” Spagni, R.: Disclosure of a Major Bug in CryptoNote Based Currencies. <https://www.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html>. Accessed 14 Feb 2023
38. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2111–2128. ACM Press (2019). <https://doi.org/10.1145/3319535.3339817>
39. Morais, R., Crocker, P., de Sousa, S.M.: Delegated RingCT: faster anonymous transactions. Cryptology ePrint Archive, Report 2020/1521 (2020). <https://eprint.iacr.org/2020/1521>
40. Moreno-Sanchez, P., Blue, A., Le, D.V., Noether, S., Goodell, B., Kate, A.: DLSAG: non-interactive refund transactions for interoperable payment channels in Monero. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 325–345. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_18
41. Möser, M., et al.: An empirical analysis of traceability in the Monero blockchain. PoPETs **2018**(3), 143–163 (2018). <https://doi.org/10.1515/popets-2018-0025>
42. Nick, J.: A Problem With Monero’s RingCT. <https://jonasnick.github.io/blog/2016/12/17/a-problem-with-ringct/>. Accessed 14 Feb 2023
43. Nick, J.: Exploiting low order generators in one-time ring signatures. <https://jonasnick.github.io/blog/2017/05/23/exploiting-low-order-generators-in-one-time-ring-signatures/>. Accessed 14 Feb 2023

44. Noether, S.: Ring signature confidential transactions for Monero. Cryptology ePrint Archive, Report 2015/1098 (2015). <https://eprint.iacr.org/2015/1098>
45. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
46. Project, M.: Monero-Project/Meta: List of Issues. <https://github.com/monero-project/meta/issues>. Accessed 11 Apr 2023
47. Ronge, V., Egger, C., Lai, R.W.F., Schröder, D., Yin, H.H.F.: Foundations of ring sampling. PoPETs **2021**(3), 265–288 (2021). <https://doi.org/10.2478/popets-2021-0047>
48. Sui, Z., Liu, J.K., Yu, J., Qin, X.: MoNet: a fast payment channel network for scriptless cryptocurrency Monero. In: 42nd IEEE International Conference on Distributed Computing Systems, ICDCS 2022, Bologna, Italy, July 10–13, 2022, pp. 280–290. IEEE (2022). <https://doi.org/10.1109/ICDCS54860.2022.00035>
49. Sun, S.-F., Au, M.H., Liu, J.K., Yuen, T.H.: RingCT 2.0: a compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency Monero. In: Foley, S.N., Gollmann, D., Snekenes, E. (eds.) ESORICS 2017. LNCS, vol. 10493, pp. 456–474. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66399-9_25
50. Thyagarajan, S.A.K., Malavolta, G., Schmidt, F., Schröder, D.: PayMo: Payment channels for Monero. Cryptology ePrint Archive, Report 2020/1441 (2020). <https://eprint.iacr.org/2020/1441>
51. Tsang, P.P., Wei, V.K.: Short linkable ring signatures for E-voting, E-cash and attestation. In: Deng, R.H., Bao, F., Pang, H.H., Zhou, J. (eds.) ISPEC 2005. LNCS, vol. 3439, pp. 48–60. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31979-5_5
52. Van Saberhagen, N.: CryptoNote v2.0. <https://www.bytecoin.org/old/whitepaper.pdf> (2013). Accessed 21 Nov 2022
53. Vijayakumaran, S.: Analysis of CryptoNote transaction graphs using the Dulmage-Mendelsohn decomposition. Cryptology ePrint Archive, Report 2021/760 (2021). <https://eprint.iacr.org/2021/760>
54. Wijaya, D.A., Liu, J.K., Steinfeld, R., Liu, D.: Monero ring attack: recreating zero Mixin transaction effect. In: 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1–3, 2018, pp. 1196–1201. IEEE (2018). <https://doi.org/10.1109/TrustCom/BigDataSE.2018.00165>
55. Yu, J., Au, M.H.A., Veríssimo, P.J.E.: Re-thinking untraceability in the CryptoNote-style blockchain. In: Delaune, S., Jia, L. (eds.) CSF 2019 Computer Security Foundations Symposium, pp. 94–107. IEEE Computer Society Press (2019). <https://doi.org/10.1109/CSF.2019.00014>
56. Yuen, T.H., et al.: RingCT 3.0 for blockchain confidential transaction: shorter size and stronger security. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 464–483. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_25