A Holistic Security Analysis of Monero Transactions

Cas Cremers[1], Julian Loss[1], and Benedikt Wagner[1,2]

[1]CISPA Helmholtz Center for Information Security
[2]Saarland University
{cremers,loss,benedikt.wagner}@cispa.de

February 27, 2024, v1.01

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, Algebraic Group Model, Network Flows, Transaction Scheme Security.

Version History. v1.0: initial version; v1.01: editorial improvements, introduction revisited, more explanations.

Publication and Acknowledgments. ©IACR 2024. This is the full version of an article that will be published in the proceedings of EUROCRYPT 2024. 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.

Contents

1 Introduction

In the rapidly growing zoo of cryptocurrencies, Monero[1] [VS13, KAN20] is among the largest and most well-known systems, with a market capitalization of about 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 [lfS, Nicb, Nica] 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) [FKL18] 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 Section 1.2).

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,

[1] See `https://www.getmonero.org`.

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) [FKL18]. 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 [FKL18]. 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 [MBKM19, CHM+20]. 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 $\mathsf{R} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, 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 $\mathsf{Link}(\sigma, \sigma')$ 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 [GNB19], 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 [SALY17, YSL+20, LRR+19, EZS+19, ESZ22]. Notably, all of them analyze new constructions and not RingCT as it is. Further, some of these models [SALY17, YSL+20, EZS+19, ESZ22] omit important non-trivial aspects of RingCT, e.g., adversarial key derivation. Some of them [SALY17, YSL+20, LRR+19] fail to give a single security definition and 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 Section 9.

History of Monero. Monero's transaction scheme RingCT originates in the CryptoNote protocol [VS13], which is based on a linkable ring signature presented in [FS07]. Noether [Noe15] introduced a way to hide transaction amounts using Pedersen commitments [Ped92] and range proofs, and also presented a compatible new ring signature component, called MLSAG. The construction of MLSAG is mostly based on [LWW04]. Later, MLSAG was replaced by a more concise ring signature component, called CLSAG [GNB19], and Bulletproofs/Bulletproofs+ [BBB+18, GT21, CHJ+22] are being used as range proofs. Bulletproofs++ [Eag22] are investigated for potential use [Pro]. Overviews of Monero and its transaction system can be found in [AJ18, KAN20]. Prior work has studied the security of some of RingCTs's building blocks in isolation [Ped92, GNB19, BBB+18, GT21], 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 [SALY17, YSL+20, LRR+19, JF21] to the use of post-quantum assumptions [EZS+19, ESZ22]. Also, some works modify RingCT with the motivation to increase compatibility with other protocols, e.g., second-layer protocols [MBL+20] or proof of stake consensus [MCdS20]. A variety of protocols has been designed add new functionality to the Monero ecosystem. Examples include proofs of reserve [DV19, DBV21], payment channels [TMSS20, MBL+20, SLYQ22], and protocols atomic swaps [Gug20, TMSS20].

Attacks on Monero. Researchers have also studied attacks against Monero and their mitigations. These target privacy [KFTS17, WLSL18, MSH+18, YAV19, Vij21, REL+21, DRR22, ELR+22, Fro, Kle], centralization [CYD+20] and security aspects [lfS, Nicb, Nica]. In terms of privacy, attacks reach from passive attacks [MSH+18, Vij21] to active attacks [YAV19, WLSL18], and temporal attacks [KFTS17] that make users traceable. These attacks are purely combinatorial in nature. The works [REL+21, ELR+22] study how to mitigate such combinatorial attacks.

Related Currencies and Their Analysis. ZCash [HBHW] is one of the most prominent privacy-focused cryptocurrencies. It is based on the Zerocash protocol [BCG+14], which comes with a

cryptographic security analysis. The current protocol specification of ZCash [HBHW] suggests that ZCash deviates from Zerocash in multiple ways. Mimblewimble [Jed] is a currency prototype that uses homomorphic commitments for efficiency reasons and to hide transaction amounts. In contrast to Monero's transaction scheme, Mimblewimble does not rely on ring signatures or stealth addresses. A security model and analysis of Mimblewimble has been given in [FOS19, FO22].

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 $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$. 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 $\mathsf{isk} = k_s \in \mathbb{Z}_p$, and $\mathsf{ivk} = k_v \in \mathbb{Z}_p$, i.e., we have $\mathsf{ipk} = (g^{\mathsf{ivk}}, g^{\mathsf{isk}})$.

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 Figure 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.

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) $\mathsf{pseed} = g^r$ in the transaction. The public seed will be used for the entire transaction, and not just for one output. Then, she derives $\mathsf{ok} = (K_v)^r$, where K_v is the view key part of Bob's identity public key $\mathsf{ipk} = (K_v, K_s)$, i.e.,

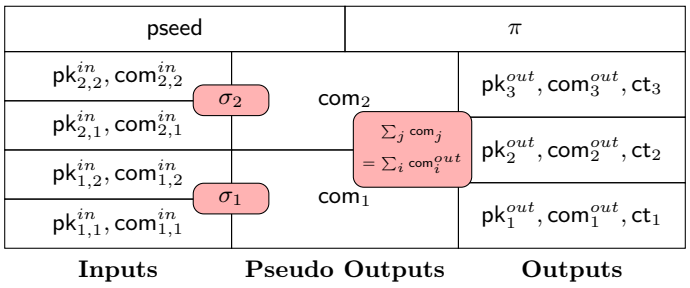

Figure 1: Schematic overview of an example transaction in Monero with two inputs and three outputs. Inputs are actually references to previous outputs. Signatures σ_i connect inputs and pseudo outputs. The homomorphic property of commitments ties pseudo outputs to outputs. In addition to inputs, outputs,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

$\mathsf{ok} = g^{\mathsf{ivk} \cdot r}$. 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 $\mathsf{pk} = K_s \cdot g^{\tau}$, 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 $\mathsf{com} = \mathsf{Com}(\mathsf{amt}, \mathsf{cr})$, 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 $\mathsf{ivk} = k_v$ and the public seed pseed. Then, he can also derive τ and cr from ok, decrypt ct, and check whether the equations $\mathsf{pk} = K_s \cdot g^{\tau}$ and $\mathsf{com} = \mathsf{Com}(\mathsf{amt}, \mathsf{cr})$ hold. If so, Bob knows that he just received amt coins. This is possible even if isk is unknown. If $\mathsf{isk} = k_s$ is known, then Bob can recover a secret key $\mathsf{sk} = k_s + \tau$ 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 $o^* = (\mathsf{pk}^*, \mathsf{com}^*, \mathsf{ct}^*)$ 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 $o' = (\mathsf{pk}', \cdot, \cdot)$ 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 $\mathsf{R} = \{\mathsf{pk}^*, \mathsf{pk}'\}$. 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 $\sum_j \mathsf{amt}^{in}_j = \sum_i \mathsf{amt}^{out}_i$, where amt^{in}_j and amt^{out}_i 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 $\sum_j \mathsf{com}_j^{in} = \sum_i \mathsf{com}_i^{out}$. 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 $\sum_j \mathsf{com}_j^{in} = \sum_i \mathsf{com}_i^{out}$ holds? Namely, for the Pedersen commitment, this not only requires $\sum_j \mathsf{amt}_j^{in} = \sum_i \mathsf{amt}_i^{out}$, but also $\sum_j \mathsf{cr}_j^{in} = \sum_i \mathsf{cr}_i^{out}$, where $\mathsf{com}_*^* = \mathsf{Com}(\mathsf{amt}_*^*, \mathsf{cr}_*^*)$. Given the structure of outputs, Alice has no way to ensure this.
- If we insist on the equation $\sum_j \mathsf{com}_j^{in} = \sum_i \mathsf{com}_i^{out}$, 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 $\mathsf{com}_j = \mathsf{Com}(\mathsf{amt}_j^{in}, \mathsf{cr}_j)$, with freshly sampled randomness cr_j, and such that $\sum_j \mathsf{cr}_j = \sum_i \mathsf{cr}_i^{out}$. Then, instead of homomorphically checking equality between inputs and outputs, we now check equality between pseudo outputs and outputs using the equation $\sum_j \mathsf{com}_j = \sum_i \mathsf{com}_i^{out}$. 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 $(\mathsf{pk}^*, \mathsf{com}^*, \mathsf{ct}^*)$. Recall that in our running example, this input is grouped with a decoy $(\mathsf{pk}', \mathsf{com}', \mathsf{ct}')$. We can not just insist on $\mathsf{com}^* = \mathsf{com}_j$, because this reintroduces the two problems from above. Instead, Alice could prove that $\mathsf{com}^* - \mathsf{com}_j$ or $\mathsf{com}' - \mathsf{com}_j$ is a commitment to 0. For Pedersen commitments with basis g, h, this is equivalent to proving that Alice knows some r such that $\mathsf{com}^* - \mathsf{com}_j = g^r$ or $\mathsf{com}' - \mathsf{com}_j = g^r$. Interestingly, this proof can 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 $\mathsf{R} = \{(\mathsf{pk}^*, \mathsf{com}^* - \mathsf{com}_j), (\mathsf{pk}', \mathsf{com}' - \mathsf{com}_j)\}$ 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 $\mathsf{Link}(\sigma, \sigma')$ 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 $\mathsf{pseed} = g^{\mathsf{sseed}}$ and includes it in the transaction.
2. Alice computes outputs. That is, for each recipient Bob with identity public key $\mathsf{ipk} = (K_v, K_s)$ 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 $(\mathsf{pk}, \mathsf{com}, \mathsf{ct})$.
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 $\sum_j \mathsf{com}_j = \sum_i \mathsf{com}_i^{out}$, where com_j are the pseudo outputs, and com_i^{out} are the output commitments.
5. The range proof for the commitments verifies.

3 Preliminaries and Notation

We introduce our notation, conventions, and basic preliminaries.

Monero Documentation. The version of Monero that we analyze is based on [GNB19, Noe15, KAN20], as well as the source code[2].

Sets and Probability. We define $[K] := \{1, \ldots, K\} \subseteq \mathbb{N}$ to be the set of the first K natural numbers. Fix a finite set S and some probability distribution $\mathcal{D}$. Then, the notation $x \leftarrow\!\!\$\, S$ means that x is sampled uniformly at random from S, and the notation $x \leftarrow \mathcal{D}$ indicates that x is sampled according to $\mathcal{D}$. For an event E, and a probabilistic experiment $\mathfrak{G}$, we write $\Pr_{\mathfrak{G}}[E]$ to denote the probability that the event E occurs if we run the experiment $\mathfrak{G}$. If $\mathfrak{G}$ is clear from the context, we omit it. A function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is said to be negligble in its input λ, if it vanishes faster than the inverse of any polynomial, i.e $f \in \lambda^{-\omega(1)}$.

Algorithms. Throughout, we make the convention that every algorithm gets the security parameter λ (encoded in unary as 1^λ) at least implicitly as input. Let A be a probabilistic algorithm. We denote the (upper bound of) the running time of A by $\mathbf{T}(\mathsf{A})$. We say that A is PPT, if $\mathbf{T}(\mathsf{A})$ is polynomial in A's input. We treat probabilistic algorithms similar to distributions and write $y \leftarrow \mathsf{A}(x)$ to indicate that A is run on input x with uniform coins, and the result is y. If A is deterministic, we write $y := \mathsf{A}(x)$. To make the random coins ρ of a probabilistic A explicit, we write $y := \mathsf{A}(x; \rho)$. The notation $y \in \mathsf{A}(x)$ states that there is some ρ such that $y = \mathsf{A}(x; \rho)$. To distinguish algorithms with the same name of different schemes, we may sometimes use the notation $\mathsf{Sc.A}$ to make explicit that we are refering to algorithm A of scheme Sc.

Games. Let $\mathbf{G}$ be a security game, i.e., a probabilistic experiment used to define security notions. We write $\mathbf{G} \rightarrow b$ to indicate that $\mathbf{G}$ outputs b. Unless explicitly stated otherwise, lists, maps, and sets are initialized empty by default, and numerical variables are initialized with 0. When we describe algorithms, oracles, and games via pseudocode, we use the notation **parse** $a := b$ to denote that variable b is parsed according to the form of a. For example, we may write **parse** $(\mathsf{pk}, \mathsf{com}) := b$ to denote that variable b is parsed as a tuple $(\mathsf{pk}, \mathsf{com})$. This implicitly means that the algorithm, oracle, or game aborts if b does not have this form. We may group multiple of these parsing statements together, e.g. **parse** $a := b, c := d$ means parsing b into a and d into c.

Idealized Models. Most of our proofs require to model certain hash functions as a random oracle. In this case, the adversary gets access to an oracle computing a random function in a lazy fashion. For some proofs, we make use of the algebraic group model [FKL18] over cyclic groups $\mathbb{G}$. In this model, we only consider algebraic algorithms. That is, whenever an algorithm outputs a group element $T \in \mathbb{G}$, it also outputs a representation $(a_1, \ldots, a_k) \in \mathbb{Z}_p^k$ such that $T = \prod_{i=1}^{k} h_i^{a_i}$, where $h_1, \ldots, h_k$ are all group elements that the algorithm received so far (either as input or as oracle responses).

Hardness Assumptions. We make use of the standard discrete logarithm (DLOG) assumption. By the description of a prime order group $\mathbb{G}$ of order $p \geq 2^\lambda$ with generator g, we mean g, p, and the group operation.

Definition 1 (DLOG Assumption)**.** *Let GGen be an algorithm that on input 1^λ outputs the description of a prime order group $\mathbb{G}$ of order $p \geq 2^\lambda$ with generator g. We say that the DLOG assumption holds relative to GGen, if for all PPT algorithms $\mathcal{A}$, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{A},\mathsf{GGen}}(\lambda) := \Pr_{\mathfrak{G}}[z = x],$$

where $\mathfrak{G}$ is given as $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$, $x \leftarrow\!\!\$\, \mathbb{Z}_p$, $z \leftarrow \mathcal{A}(\mathbb{G}, p, g, g^x)$.

[2] See https://github.com/monero-project/monero, release-v0.18.1.2.

4 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.

4.1 Syntax

Throughout, we assume that some system parameters $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$ 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. 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 $\mathsf{GenID}(\mathsf{par})$ that outputs a triple $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$. As explained in Section 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 $\mathsf{Encaps}(\mathsf{par})$. 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 $\mathsf{SendDecaps}$ and $\mathsf{RecDecaps}$ are used, where Alice runs $\mathsf{SendDecaps}(\mathsf{ipk}, \mathsf{sseed})$ and Bob runs $\mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed})$. As a result, both obtain a shared secret ok, called the output key. Alice now uses this shared secret in algorithm $\mathsf{DerPK}(\mathsf{ipk}, \mathsf{ok}, \mathsf{tag})$ to derive the public key pk, which is the stealth address. Bob uses algorithm $\mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}, \mathsf{tag})$ 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 $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) = 1$, then this indicates that pk is derived for him. Here, the tag $\mathsf{tag} \in \mathbb{N}$ is used for domain separation and ordering. Formally, a key derivation scheme (KDS) is a tuple $\mathsf{KDS} = (\mathsf{GenID}, \mathsf{Encaps}, \mathsf{SendDecaps}, \mathsf{RecDecaps}$ $\mathsf{DerPK}, \mathsf{DerSK}, \mathsf{Track})$ of PPT algorithms with the following syntax, where the sets $\mathcal{PSE}$, $\mathcal{SSE}$, $\mathcal{PK}$, and $\mathcal{SK}$ are specified by par:

- $\mathsf{GenID}(\mathsf{par}) \to (\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$ takes as input system parameters par and outputs an identity public key ipk, an identity view key ivk, and an identity signing key isk.
- $\mathsf{Encaps}(\mathsf{par}) \to (\mathsf{pseed}, \mathsf{sseed})$ takes as input system parameters par and outputs a secret seed $\mathsf{sseed} \in \mathcal{SSE}$ and a public seed $\mathsf{pseed} \in \mathcal{PSE}$.
- $\mathsf{SendDecaps}(\mathsf{ipk}, \mathsf{sseed}) \to \mathsf{ok}$ is deterministic, takes as input an identity public key and a secret seed, and outputs an output key ok.
- $\mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}) \to \mathsf{ok}$ is deterministic, takes as input an identity view key ivk, and a public seed $\mathsf{pseed} \in \mathcal{PSE}$, and outputs an output key ok.
- $\mathsf{DerPK}(\mathsf{ipk}, \mathsf{ok}, \mathsf{tag}) \to \mathsf{pk}$ is deterministic, takes as input an identity public key ipk, an output key ok, tag $\mathsf{tag} \in \mathbb{N}$, and outputs a public key $\mathsf{pk} \in \mathcal{PK}$.
- $\mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}, \mathsf{tag}) \to \mathsf{sk}$ is deterministic, takes as input an identity signing key isk, an output key ok, tag $\mathsf{tag} \in \mathbb{N}$, and outputs a secret key $\mathsf{sk} \in \mathcal{SK}$.

- $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) \to b$ is deterministic, takes as input an identity public key ipk, an output key ok, a public key $\mathsf{pk} \in \mathcal{PK}$, tag $\mathsf{tag} \in \mathbb{N}$, and outputs a bit $b \in \{0,1\}$.

Further, we require that par specifies an efficiently decidable binary relation $\mathcal{KR}$, the relation of valid key pairs, and that these algorithms and the relation satisfy the following completeness property: For all $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \in \mathsf{GenID}(\mathsf{par})$ and all $\mathsf{tag} \in \mathbb{N}$, we have

$$\Pr_{\mathfrak{G}}\left[\mathsf{ok} = \mathsf{ok}' \wedge b = 1 \wedge (\mathsf{pk}, \mathsf{sk}) \in \mathcal{KR}\right] = 1,$$

where $\mathfrak{G}$ is given as

$$\begin{aligned}
(\mathsf{pseed}, \mathsf{sseed}) &\leftarrow \mathsf{Encaps}(\mathsf{par}), & \mathsf{ok} &:= \mathsf{SendDecaps}(\mathsf{ipk}, \mathsf{sseed}),\\
\mathsf{ok}' &:= \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}), & \mathsf{pk} &:= \mathsf{DerPK}(\mathsf{ipk}, \mathsf{ok}, \mathsf{tag}),\\
b &:= \mathsf{Track}(\mathsf{ipk}, \mathsf{ok}', \mathsf{pk}, \mathsf{tag}), & \mathsf{sk} &:= \mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}', \mathsf{tag}).
\end{aligned}$$

Consider the experiment $\mathfrak{G}$ that is given by sampling $(\mathsf{pseed}, \mathsf{sseed}) \leftarrow \mathsf{Encaps}(\mathsf{par})$. We say that KDS has h bits of entropy of public seeds, if for all $\mathsf{pseed}_0 \in \mathcal{PSE}$ we have $\Pr_{\mathfrak{G}}[\mathsf{pseed} = \mathsf{pseed}_0] \leq 2^{-h}$.

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. 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 $\mathsf{DerRand}(\mathsf{ok}, \mathsf{tag})$ that outputs cr. Given such random coins and an amount $\mathsf{amt} \in \mathcal{D} \subseteq \mathbb{N}_0$, one can commit to amt using $\mathsf{com} := \mathsf{Com}(\mathsf{amt}, \mathsf{cr})$. 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 $\mathsf{PProve}(\mathsf{stmt}, \mathsf{witn})$ 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 $\mathsf{PVer}(\mathsf{stmt}, \pi)$. Formally, a verifiable homomorphic commitment scheme (VHC) is a tuple $\mathsf{VHC} = (\mathsf{DerRand}, \mathsf{Com}, \mathsf{PProve}, \mathsf{PVer})$ of PPT algorithms with the following syntax, where $\mathcal{CR} \subseteq \mathbb{N}_0$, $\mathcal{D} \subseteq \mathbb{N}_0$, and a group $\mathcal{C}$ is specified by par:

- $\mathsf{DerRand}(\mathsf{ok}, \mathsf{tag}) \to \mathsf{cr}$ is deterministic, takes as input an output key ok, and a tag $\mathsf{tag} \in \mathbb{N}$, and outputs a commitment randomness $\mathsf{cr} \in \mathcal{CR}$.
- $\mathsf{Com}(\mathsf{amt}, \mathsf{cr}) \to \mathsf{com}$ is deterministic, takes as input data $\mathsf{amt} \in \mathbb{N}_0$, and a commitment randomness $\mathsf{cr} \in \mathcal{CR}$, and outputs a commitment $\mathsf{com} \in \mathcal{C}$.
- $\mathsf{PProve}(\mathsf{stmt}, \mathsf{witn}) \to \pi$ takes as input a statement $\mathsf{stmt} = (\mathsf{com}_i)_{i=1}^K$, and a witness $\mathsf{witn} = (\mathsf{amt}_i, \mathsf{cr}_i)_{i=1}^K$, and outputs a proof π.
- $\mathsf{PVer}(\mathsf{stmt}, \pi) \to b$ is deterministic, takes as input a statement $\mathsf{stmt} = (\mathsf{com}_i)_{i=1}^K$, and a prove π, and outputs a bit $b \in \{0,1\}$.

Further, we require that these algorithms satisfy the following completeness properties:

1. For all $\mathsf{amt}, \mathsf{amt}' \in \mathcal{D}$ and all $\mathsf{cr}, \mathsf{cr}' \in \mathcal{CR}$ we have[3]

$$\mathsf{Com}(\mathsf{amt}, \mathsf{cr}) + \mathsf{Com}(\mathsf{amt}', \mathsf{cr}') = \mathsf{Com}(\mathsf{amt} + \mathsf{amt}', \mathsf{cr} + \mathsf{cr}').$$

[3]Observe that the addition $\mathsf{amt} + \mathsf{amt}'$ is over the integers, and $\mathcal{D}$ models the range of valid amounts such that no overflow occurs.

2. For all (polynomially bounded) integers $K \in \mathbb{N}$, and all $\mathsf{amt}_i \in \mathcal{D}, \mathsf{cr}_i \in \mathcal{CR}$ for $i \in [K]$, we have

$$\Pr_{\mathfrak{G}}[\mathsf{PVer}(\mathsf{stmt}, \pi) = 1] = 1,$$

where $\mathfrak{G}$ is given as $\mathsf{com}_i := \mathsf{Com}(\mathsf{amt}_i, \mathsf{cr}_i)$ for all $i \in [K]$, $\mathsf{stmt} := (\mathsf{com}_i)_{i=1}^K$, $\mathsf{witn} := (\mathsf{amt}_i, \mathsf{cr}_i)_{i=1}^K$, and $\pi \leftarrow \mathsf{PProve}(\mathsf{stmt}, \mathsf{witn})$.

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 is used. It makes use of the shared secret ok as a key. Formally, a data encryption scheme (DE) is a tuple $\mathsf{DE} = (\mathsf{Enc}, \mathsf{Dec})$ of PPT algorithms with the following syntax:

- $\mathsf{Enc}(\mathsf{ok}, \mathsf{tag}, \mathsf{amt}) \to \mathsf{ct}$ is deterministic, takes as input an output key ok, tag $\mathsf{tag} \in \mathbb{N}$, data $\mathsf{amt} \in \mathcal{D}$, and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{ok}, \mathsf{tag}, \mathsf{ct}) \to \mathsf{amt}$ is deterministic, takes as input an output key ok, tag $\mathsf{tag} \in \mathbb{N}$, and a ciphertext ct, and outputs data $\mathsf{amt} \in \mathcal{D}$.

We require that these algorithms satisfy the following completeness property: For all output keys ok, all $\mathsf{amt} \in \mathcal{D}$, and all $\mathsf{tag} \in \mathbb{N}$, we have

$$\mathsf{Dec}(\mathsf{ok}, \mathsf{tag}, \mathsf{Enc}(\mathsf{ok}, \mathsf{tag}, \mathsf{amt})) = \mathsf{amt}.$$

Key Conversion Scheme. Recall from our overview in Section 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. Namely, Alice runs an algorithm $\mathsf{auxpk} := \mathsf{ConvertPublic}(\mathsf{com}, \mathsf{com}')$ 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 $\mathsf{auxsk} := \mathsf{ConvertSecret}(\mathsf{cr}, \mathsf{cr}')$ for the associated randomness $\mathsf{cr}, \mathsf{cr}'$ to get an auxiliary secret key auxsk. The guarantee is that if com and com' commit to the same amount with randomness $\mathsf{cr}, \mathsf{cr}'$, respectively, then auxsk is a valid secret key for auxpk, and can then be used within the linkable ring signature component. Formally, we define a key conversion scheme (KCS) as a tuple $\mathsf{KCS} = (\mathsf{ConvertPublic}, \mathsf{ConvertSecret})$ of PPT algorithms with the following syntax, where the sets $\mathcal{APK}, \mathcal{ASK}$ are specified by par:

- $\mathsf{ConvertPublic}(\mathsf{com}, \mathsf{com}') \to \mathsf{auxpk}$ is deterministic, takes as input commitments $\mathsf{com}, \mathsf{com}' \in \mathcal{C}$, and outputs an auxiliary public key $\mathsf{auxpk} \in \mathcal{APK}$.
- $\mathsf{ConvertSecret}(\mathsf{cr}, \mathsf{cr}') \to \mathsf{auxsk}$ is deterministic, takes as input commitment randomness $\mathsf{cr}, \mathsf{cr}' \in \mathcal{CR}$, and outputs an auxiliary secret key $\mathsf{auxsk} \in \mathcal{ASK}$.

Further, we require that par specifies an efficiently decidable binary relation $\mathcal{AKR}$ and that these algorithms and the relation satisfy the following completeness property: For every $\mathsf{amt} \in \mathcal{D}$, $\mathsf{cr}, \mathsf{cr}' \in \mathcal{CR}$, we have $(\mathsf{auxpk}, \mathsf{auxsk}) \in \mathcal{AKR}$, where

$$\begin{aligned} \mathsf{com} &:= \mathsf{Com}(\mathsf{amt}, \mathsf{cr}), & \mathsf{com}' &:= \mathsf{Com}(\mathsf{amt}, \mathsf{cr}'), \\ \mathsf{auxpk} &:= \mathsf{ConvertPublic}(\mathsf{com}, \mathsf{com}'), & \mathsf{auxsk} &:= \mathsf{ConvertSecret}(\mathsf{cr}, \mathsf{cr}'). \end{aligned}$$

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 Section 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, which is given by three algorithms $\mathsf{Sig}, \mathsf{Ver}$, and Link. To sign a message m, e.g., a transaction, with respect to some key ring $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$, Alice has to know a valid pair of secret keys $\mathsf{sk}, \mathsf{auxsk}$ for one of the $\mathsf{pk}_i, \mathsf{auxpk}_i$. Then, she can compute a signature $\sigma \leftarrow \mathsf{Sig}(\mathsf{R}, \mathsf{sk}, \mathsf{auxsk}, \mathsf{m})$. This signature can be verified with respect to R and m by running $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma)$. Also, one can check whether two signatures σ, σ' were computed using the same key by running $\mathsf{Link}(\sigma, \sigma')$. Formally, a two-dimensional linkable ring signature scheme (LRS) is specified by a tuple $\mathsf{LRS} = (\mathsf{Sig}, \mathsf{Ver}, \mathsf{Link})$ of PPT algorithms with the following syntax, where the sets $\mathsf{LRS}.\mathcal{M}$, $\mathsf{LRS}.\Sigma$ are specified by par:

- $\mathsf{Sig}(\mathsf{R}, \mathsf{sk}, \mathsf{auxsk}, \mathsf{m}) \to \sigma$ takes as input a list $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ of public keys $\mathsf{pk}_i \in \mathcal{PK}$ and auxiliary public keys $\mathsf{auxpk}_i \in \mathcal{APK}$, a secret key $\mathsf{sk} \in \mathcal{SK}$, an auxiliary secret key $\mathsf{auxsk} \in \mathcal{ASK}$, and a message $\mathsf{m} \in \mathsf{LRS}.\mathcal{M}$, and outputs a signature $\sigma \in \mathsf{LRS}.\Sigma$. We assume that σ implicitly contains R.
- $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) \to b$ is deterministic, takes as input a list $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ of public keys $\mathsf{pk}_i \in \mathcal{PK}$ and auxiliary public keys $\mathsf{auxpk}_i \in \mathcal{APK}$, a message $\mathsf{m} \in \mathsf{LRS}.\mathcal{M}$, and a signature $\sigma \in \mathsf{LRS}.\Sigma$, and outputs a bit $b \in \{0, 1\}$.
- $\mathsf{Link}(\sigma, \sigma') \to b$ is deterministic, takes as input signatures $\sigma, \sigma' \in \mathsf{LRS}.\Sigma$ and outputs a bit $b \in \{0, 1\}$. We assume that Link is symmetric and outputs 0 whenever the lists R, R' implicitly contained in the signatures are disjoint.

We require that these algorithms satisfy the following completeness properties:

1. For all (polynomially bounded) integers $N \in \mathbb{N}$, all $i^* \in [N]$, all $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$, all auxsk_{i^*} and sk_{i^*} with $(\mathsf{auxpk}_{i^*}, \mathsf{auxsk}_{i^*}) \in \mathcal{AKR}$ and $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \in \mathcal{KR}$, and every $\mathsf{m} \in \mathsf{LRS}.\mathcal{M}$, we have

$$\Pr_{\mathfrak{G}}[\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1] = 1,$$

   where $\mathfrak{G}$ is given as $\sigma \leftarrow \mathsf{Sig}(\mathsf{R}, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*}, \mathsf{m})$.

2. For all (polynomially bounded) integers $N, N' \in \mathbb{N}$, all $i^* \in [N], j^* \in [N']$, all $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$, $\mathsf{R}' = (\mathsf{pk}'_j, \mathsf{auxpk}'_j)_{j=1}^{N'}$, every $\mathsf{auxsk}_{i^*}, \mathsf{auxsk}'_{j^*}, \mathsf{sk}_{i^*}, \mathsf{sk}'_{j^*}$ such that $(\mathsf{auxpk}_{i^*}, \mathsf{auxsk}_{i^*}) \in \mathcal{AKR}$ and $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \in \mathcal{KR}$ and $(\mathsf{auxpk}'_{j^*}, \mathsf{auxsk}'_{j^*}) \in \mathcal{AKR}, (\mathsf{pk}'_{j^*}, \mathsf{sk}'_{j^*}) \in \mathcal{KR}$, and every $\mathsf{m}, \mathsf{m}' \in \mathsf{LRS}.\mathcal{M}$, we have

$$\Pr_{\mathfrak{G}}\left[\mathsf{sk}_{i^*} = \mathsf{sk}'_{j^*} \iff \mathsf{Link}(\sigma, \sigma') = 1\right] = 1,$$

   where $\mathfrak{G}$ is given as

$$\sigma \leftarrow \mathsf{Sig}(\mathsf{R}, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*}, \mathsf{m}), \quad \sigma' \leftarrow \mathsf{Sig}(\mathsf{R}', \mathsf{sk}'_{j^*}, \mathsf{auxpk}'_{j^*}, \mathsf{m}').$$

Private Transaction Scheme. Given the components we introduced so far, we define a private transaction scheme. Namely, a private transaction scheme (PTS) is a tuple $\mathsf{PTS} = (\mathsf{LRS}, \mathsf{KDS}, \mathsf{VHC}, \mathsf{DE}, \mathsf{KCS})$ with the following properties:

- $\mathsf{KDS} = (\mathsf{GenID}, \mathsf{Encaps}, \mathsf{SendDecaps}, \mathsf{RecDecapsDerPK}, \mathsf{DerSK}, \mathsf{Track})$ is a key derivation scheme.
- $\mathsf{DE} = (\mathsf{Enc}, \mathsf{Dec})$ is a data encryption scheme.
- $\mathsf{VHC} = (\mathsf{DerRand}, \mathsf{Com}, \mathsf{PProve}, \mathsf{PVer})$ is a verifiable homomorphic commitment scheme.
- $\mathsf{KCS} = (\mathsf{ConvertPublic}, \mathsf{ConvertSecret})$ is a key conversion scheme.
- $\mathsf{LRS} = (\mathsf{Sig}, \mathsf{Ver}, \mathsf{Link})$ is a two-dimensional linkable ring signature scheme.

Transactions. We define a transaction to be a tuple $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$, where

- $\mathsf{In} = (\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_{j=1}^{L}$ is a list, where $\mathsf{Ref}_j = (\mathsf{pk}_{j,r}^{in}, \mathsf{com}_{j,r}^{in})_r$ is a list of public keys $\mathsf{pk}_{j,r}^{in} \in \mathsf{KDS}.\mathcal{PK}$ and commitments $\mathsf{com}_{j,r}^{in} \in \mathcal{C}$, $\mathsf{com}_j \in \mathcal{C}$ is a commitment, and $\sigma_j \in \Sigma$ is a signature. We refer to an entry in this list as an input of the transaction.
- $\mathsf{Out} = \left(\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i := i\right)_{i=1}^{K}$ is a list of quadruples of public keys $\mathsf{pk}_i^{out} \in \mathsf{KDS}.\mathcal{PK}$, commitments $\mathsf{com}_i^{out} \in \mathcal{C}$, ciphertexts ct_i, and tags. We refer to the pairs $(\mathsf{pk}_i^{out}, \mathsf{com}_i^{out})$ for $i \in [K]$ as outputs of the transaction, keeping in mind that these outputs are always associated with a ciphertext and a tag.
- $\mathsf{pseed} \in \mathcal{PSE}$ is a public seed and π is a proof.

Generating Transactions. Suppose Alice wants to spend amt_i^{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 $(\mathsf{pk}_j^{in}, \mathsf{com}_j^{in})$ (for $j \in [L]$) of previous transactions that she owns. Because she owns them, she knows the associated amount and commitment randomness $\mathsf{amt}_j^{in}, \mathsf{cr}_j^{in}$, and the corresponding secret key sk_j. We write $\mathsf{Use}_j = (\mathsf{pk}_j^{in}, \mathsf{com}_j^{in}, \mathsf{amt}_j^{in}, \mathsf{cr}_j^{in}, \mathsf{sk}_j)$ 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 $\mathsf{pk}_j^{in}, \mathsf{com}_j^{in}$. Now, we specify how Alice generates a transaction by defining an algorithm GenTx, which takes as inputs $(\mathsf{Use}_j)_{j=1}^{L}$, $(\mathsf{Ref}_j)_{j=1}^{L}$, and $(\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i=1}^{K}$. We present this algorithm in Figure 2.

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 $\mathsf{Outputs}$, and all signatures contained in previous transactions by a list $\mathsf{Signatures}$. Then, algorithm VerTx takes as input the lists $\mathsf{PSeeds}, \mathsf{Outputs}, \mathsf{Signatures}$ and a transaction Tx. We define this algorithm in Figure 3. We make the additional assumption that public seeds are not repeated. For a discussion, we refer to Section 8.

Receiving Outputs. When a transaction Tx is published, users should be able to identify outputs that they receive. For that, we define an algorithm $\mathsf{Receive}$ in Figure 3. Concretely, write $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$ and let its outputs be $\mathsf{Out} = \left(\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i\right)_{i=1}^{K}$. Then, each user with keys $\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}$ runs $\mathsf{Receive}$ on inputs $\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i$ for each $i \in [K]$. Additionally, the user inputs pseed and its keys $\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}$. 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.

Alg $\mathsf{GenTx}\left((\mathsf{Use}_j)_{j=1}^{L}, (\mathsf{Ref}_j)_{j=1}^{L}, (\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i=1}^{K}\right)$

01 **for** $j \in [L]$: **parse** $(\mathsf{pk}_j^{in}, \mathsf{com}_j^{in}, \mathsf{amt}_j^{in}, \mathsf{cr}_j^{in}, \mathsf{sk}_j) := \mathsf{Use}_j$
02 **for** $j \in [L]$: **parse** $(\mathsf{pk}_{j,r}^{in}, \mathsf{com}_{j,r}^{in})_{r=1}^{|\mathsf{Ref}_j|} := \mathsf{Ref}_j$

// Compute output public keys and ciphertexts.
03 $(\mathsf{pseed}, \mathsf{sseed}) \leftarrow \mathsf{Encaps}(\mathsf{par})$
04 **for** $i \in [K]$:
05 $\quad \mathsf{tag}_i := i, \mathsf{ok}_i := \mathsf{SendDecaps}(\mathsf{ipk}_i, \mathsf{sseed})$
06 $\quad \mathsf{ct}_i := \mathsf{Enc}(\mathsf{ok}_i, \mathsf{tag}_i, \mathsf{amt}_i^{out})$
07 $\quad \mathsf{pk}_i^{out} := \mathsf{DerPK}(\mathsf{ipk}_i, \mathsf{ok}_i, \mathsf{tag}_i)$

// Compute output commitments with proof and input commitments.
08 **for** $i \in [K]$:
09 $\quad \mathsf{cr}_i^{out} := \mathsf{DerRand}(\mathsf{ok}_i, \mathsf{tag}_i)$
10 $\quad \mathsf{com}_i^{out} := \mathsf{Com}(\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out})$
11 **for** $j \in [L-1]$: $\mathsf{cr}_j \leftarrow\!\!\$\, \mathcal{CR}$
12 $\mathsf{cr}_L := \sum_{i=1}^{K} \mathsf{cr}_i^{out} - \sum_{j=1}^{L-1} \mathsf{cr}_j$
13 **for** $j \in [L]$: $\mathsf{com}_j := \mathsf{Com}(\mathsf{amt}_j^{in}, \mathsf{cr}_j)$
14 $\mathsf{stmt} := \left((\mathsf{com}_j)_{j=1}^{L}, (\mathsf{com}_i^{out})_{i=1}^{K}\right)$
15 $\mathsf{witn} := \left((\mathsf{amt}_j^{in}, \mathsf{cr}_j)_{j=1}^{L}, (\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out})_{i=1}^{K}\right)$
16 $\pi \leftarrow \mathsf{PProve}(\mathsf{stmt}, \mathsf{witn})$

// Compute signatures for the inputs.
17 $\mathsf{m} := \mathsf{H}_{trm}\left(\mathsf{pseed}, (\mathsf{Ref}_j, \mathsf{com}_j)_{j=1}^{L}, (\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i)_{i=1}^{K}\right)$
18 **for** $j \in [L]$:
19 $\quad$ **for** $r \in [|\mathsf{Ref}_j|]$: $\mathsf{auxpk}_r := \mathsf{ConvertPublic}(\mathsf{com}_{j,r}^{in}, \mathsf{com}_j)$
20 $\quad \mathsf{R}_j := (\mathsf{pk}_{j,r}^{in}, \mathsf{auxpk}_r)_{r=1}^{|\mathsf{Ref}_j|}$
21 $\quad \mathsf{auxsk}_j := \mathsf{ConvertSecret}(\mathsf{cr}_j^{in}, \mathsf{cr}_j)$
22 $\quad \sigma_j \leftarrow \mathsf{LRS.Sig}(\mathsf{R}_j, \mathsf{sk}_j, \mathsf{auxsk}_j, \mathsf{m})$

// Define transaction.
23 $\mathsf{In} := (\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_{j=1}^{L}$
24 $\mathsf{Out} := (\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i)_{i=1}^{K}$
25 **return** $\mathsf{Tx} := (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$

Figure 2: The transaction generation algorithm GenTx, where $\mathsf{H}_{trm} : \{0,1\}^* \to \{0,1\}^\ell$ is a random oracle. The algorithm generates a transaction spending amt_i^{out} coins to identity ipk_i, while using inputs Use_j. For each input j, the list Ref_j contains the actual input and the decoy inputs.

Alg $\mathsf{VerTx}(\mathsf{PSeeds}, \mathsf{Outputs}, \mathsf{Signatures}, \mathsf{Tx})$

$(\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi) := \mathsf{Tx}$

$(\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_{j\in[L]} := \mathsf{In}$

for $j \in [L]$: **parse** $(\mathsf{pk}^{in}_{j,r}, \mathsf{com}^{in}_{j,r})_{r=1}^{|\mathsf{Ref}_j|} := \mathsf{Ref}_j$

parse $\left(\mathsf{pk}^{out}_i, \mathsf{com}^{out}_i, \mathsf{ct}_i, \mathsf{tag}_i\right)_{i=1}^{K} := \mathsf{Out}$

// The public seed should be fresh.

if $\mathsf{pseed} \in \mathsf{PSeeds}$: **return** 0

// Inputs should be previous outputs.

for $j \in [L]$:

  for $r \in [|\mathsf{Ref}_j|]$:

    if $(\mathsf{pk}^{in}_{j,r}, \mathsf{com}^{in}_{j,r}) \notin \mathsf{Outputs}$:

      return 0

// Signatures should be valid.

$\mathsf{m} := \mathsf{H}_{trm}\left(\mathsf{pseed}, (\mathsf{Ref}_j, \mathsf{com}_j)_{j=1}^{L}, \left(\mathsf{pk}^{out}_i, \mathsf{com}^{out}_i, \mathsf{ct}_i, \mathsf{tag}_i\right)_{i=1}^{K}\right)$

for $j \in [L]$:

  for $r \in [|\mathsf{Ref}_j|]$: $\mathsf{auxpk}_r := \mathsf{ConvertPublic}(\mathsf{com}^{in}_{j,r}, \mathsf{com}_j)$

  $\mathsf{R}_j := \left(\mathsf{pk}^{in}_{j,r}, \mathsf{auxpk}_r\right)_{r=1}^{|\mathsf{Ref}_j|}$

  if $\mathsf{LRS.Ver}(\mathsf{R}_j, \mathsf{m}, \sigma_j) = 0$: **return** 0

// Commitments should be valid.

$\mathsf{stmt} := \left((\mathsf{com}_j)_j, (\mathsf{com}^{out}_i)_i\right)$

if $\mathsf{PVer}(\mathsf{stmt}, \pi) = 0 \vee \sum_{j=1}^{L} \mathsf{com}_j \neq \sum_{i=1}^{K} \mathsf{com}^{out}_i$: **return** 0

// Double spending detection.

if $\exists j, j' \in [L]$ s.t. $j \neq j' \wedge \mathsf{Link}(\sigma_j, \sigma_{j'}) = 1$: **return** 0

if $\exists j \in [L], \sigma \in \mathsf{Signatures}$ s.t. $\mathsf{Link}(\sigma_j, \sigma) = 1$: **return** 0

return 1

Alg $\mathsf{Receive}(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}, \mathsf{pseed}, \mathsf{pk}, \mathsf{com}, \mathsf{tag}, \mathsf{ct})$

// Recover output key and check if output is owned.

$\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed})$

if $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) = 0$: **return** $(0, \perp, \perp)$

// Recover secrets.

$\mathsf{sk} := \mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}, \mathsf{tag})$

$\mathsf{amt} := \mathsf{Dec}(\mathsf{ok}, \mathsf{tag}, \mathsf{ct})$

$\mathsf{cr} := \mathsf{DerRand}(\mathsf{ok}, \mathsf{tag})$

// Commitment randomness should be correct.

if $\mathsf{Com}(\mathsf{amt}, \mathsf{cr}) \neq \mathsf{com}$: **return return** $(0, \perp, \perp)$

return $(\mathsf{amt}, \mathsf{cr}, \mathsf{sk})$

Figure 3: The transaction verification algorithm VerTx and the algorithm Receive that specifies how to receive an output. Here, $\mathsf{H}_{trm}\colon \{0,1\}^* \to \{0,1\}^{\ell}$ is a random oracle.

4.2 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 $\mathsf{PTS} = (\mathsf{LRS}, \mathsf{KDS}, \mathsf{VHC}, \mathsf{DE}, \mathsf{KCS})$ and a PPT 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.

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

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.
- $\mathsf{Outputs}$: This list contains all outputs $(\mathsf{pk}, \mathsf{com})$ that are currently in the system. These may, for example, be part of previous transactions.
- $\mathsf{Signatures}$: 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.

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

- $\mathsf{spent} \in \mathbb{N}_0$: 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 Figures 4 and 6.

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.

- NewIdentity() : This oracle generates a new honest user. For that, it generates keys $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \leftarrow \mathsf{GenID}(\mathsf{par})$. Then, it inserts ipk into the list $\mathsf{Identities}$, and sets $\mathsf{ivk}[\mathsf{ipk}] := \mathsf{ivk}$, $\mathsf{isk}[\mathsf{ipk}] := \emptyset$, $\mathsf{corr}[\mathsf{ipk}] := 0$. It returns ipk to $\mathcal{A}$.

- PartCorr(ipk) : This oracle allows $\mathcal{A}$ to learn the identity view key of an honest user. Precisely, the oracle returns $\perp$ if $\mathsf{ipk} \notin \mathsf{Identities}$ or $\mathsf{corr}[\mathsf{ipk}] \neq 0$. Otherwise, it sets $\mathsf{corr}[\mathsf{ipk}] := 1$ and returns $\mathsf{ivk}[\mathsf{ipk}]$ to $\mathcal{A}$.

- FullCorr(ipk) : This oracle allows $\mathcal{A}$ to learn the identity signing key of an honest user. Precisely, the oracle returns $\perp$ if $\mathsf{ipk} \notin \mathsf{Identities}$ or $\mathsf{corr}[\mathsf{ipk}] \neq 1$. Otherwise, it sets $\mathsf{corr}[\mathsf{ipk}] := 2$. Then, it updates $\mathsf{received}$ accordingly, i.e.,

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

  It returns $\mathsf{isk}[\mathsf{ipk}]$ 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.

- NewHonSrc($\mathsf{pk}, \mathsf{pseed}, \mathsf{com}, \mathsf{tag}, \mathsf{ct}$) : This oracle tries find an honest user to receive the given output. For that, it runs $\mathsf{Receive}$ for each user $\mathsf{ipk} \in \mathsf{Identities}$ with $\mathsf{corr}[\mathsf{ipk}] < 2$. If for some user, the received amount is non-zero, it inserts $(\mathsf{pk}, \mathsf{com})$ into $\mathsf{Outputs}$ and stores the output together with the secrets necessary to spend it in the list $\mathsf{Owned}[\mathsf{ipk}]$.

- NewSrc($\mathsf{pk}, \mathsf{amt}, \mathsf{cr}$) : This oracle inserts $(\mathsf{pk}, \mathsf{com})$ into $\mathsf{Outputs}$, where $\mathsf{com} := \mathsf{Com}(\mathsf{amt}, \mathsf{cr})$. It also updates $\mathsf{received}$ accordingly, i.e., $\mathsf{received} := \mathsf{received} + \mathsf{amt}$.

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.

- AddAdvTrans(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 spent := spent + spentnow, where spentnow is the total amount that honest users received from Tx.

- AddHonTrans(ipk, $(\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i=1}^{K}$, ISamp, RSamp) : By calling this oracle, $\mathcal{A}$ instructs an honest user with identity public key ipk to pay amt_i^{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 ipk $\notin$ Identities or corr[ipk] = 2. Otherwise, it generates a transaction as follows.

  1. Sample inputs to use by running $(\mathsf{Use}_j)_{j=1}^{L} \leftarrow \mathsf{ISamp}(\mathsf{Owned}[\mathsf{ipk}])$ and $(\mathsf{Ref}_j)_{j=1}^{L} \leftarrow \mathsf{RSamp}(\mathsf{Owned}[\mathsf{ipk}], (\mathsf{Use}_j)_{j=1}^{L})$.
  2. Check validity of the inputs. Namely, each $\mathsf{Use}_j = (\mathsf{pk}_j, \mathsf{com}_j, \mathsf{amt}_j, \mathsf{cr}_j, \mathsf{sk}_j)$ should be in Owned[ipk], the output $(\mathsf{pk}_j, \mathsf{com}_j)$ contained in Use_j should be in Ref_j, and each $(\mathsf{pk}, \mathsf{com}) \in \mathsf{Ref}_j$ should be in Outputs. Also, $\sum_{j=1}^{L} \mathsf{amt}_j = \sum_{i=1}^{K} \mathsf{amt}_i^{out}$ 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 $\mathsf{Owned}[\mathsf{ipk}] := \mathsf{Owned}[\mathsf{ipk}] \setminus \{\mathsf{Use}_j\}_{j\in[L]}$. It also updates Owned, TXs, PSeeds, Outputs as in oracle AddAdvTrans. Then, it updates received accordingly, i.e., $\mathsf{received} := \mathsf{received} + \mathsf{amt}_i^{out}$ for each $i \in [K]$ with $\mathsf{ipk}_i \notin$ Identities or corr[ipk] = 2. Finally, it returns Tx to $\mathcal{A}$.

Security Notion. Next, we define the security notion for a private transaction scheme PTS = (LRS, KDS, VHC, DE, KCS). To this end, we introduce a security game $\mathbf{UNF}_{\mathsf{PTS}}^{\mathcal{A}}(\lambda)$ 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 spent > 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., VerTx(PSeeds, Outputs, Signatures, Tx) = 0.

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

1. Consider oracles O_{id} := (NewIdentity, PartCorr, FullCorr), O_{src} = (NewHonSrc, NewSrc), and O_{tx} = (AddAdvTrans, AddHonTrans) described above.
2. Run $\mathcal{A}$ with access to oracles $\mathrm{O}_{id}, \mathrm{O}_{src}, \mathrm{O}_{tx}$ on input 1^{λ}.

3. Output 1, if win-create $= 1$ or win-steal $= 1$. Otherwise, output 0.

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

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

Oracle NewIdentity()
01 $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \leftarrow \mathsf{GenID}(\mathsf{par})$
02 $\mathsf{Identities} := \mathsf{Identities} \cup \{\mathsf{ipk}\}$
03 $\mathsf{ivk}[\mathsf{ipk}] := \mathsf{ivk},\ \mathsf{isk}[\mathsf{ipk}] := \mathsf{isk}$
04 $\mathsf{corr}[\mathsf{ipk}] := 0,\ \mathsf{Owned}[\mathsf{ipk}] := \emptyset$
05 **return** ipk

Oracle PartCorr(ipk)
06 **if** $\mathsf{ipk} \notin \mathsf{Identities} \vee \mathsf{corr}[\mathsf{ipk}] \neq 0$: **return** $\perp$
07 $\mathsf{corr}[\mathsf{ipk}] := 1$
08 **return** $\mathsf{ivk}[\mathsf{ipk}]$

Oracle FullCorr(ipk)
09 **if** $\mathsf{ipk} \notin \mathsf{Identities} \vee \mathsf{corr}[\mathsf{ipk}] \neq 1$: **return** $\perp$
10 $\mathsf{corr}[\mathsf{ipk}] := 2$

// *User is corrupted. Adversary can control its coins.*
11 $\mathsf{received} := \mathsf{received} + \sum_{(\mathsf{pk},\mathsf{com},\mathsf{amt},\mathsf{cr},\mathsf{sk}) \in \mathsf{Owned}[\mathsf{ipk}]} \mathsf{amt}$
12 **return** $\mathsf{isk}[\mathsf{ipk}]$

Figure 4: The oracles $\mathrm{O}_{id} := (\textsc{NewIdentity}, \textsc{PartCorr}, \textsc{FullCorr})$ that are used in the security definition in Section 4.2.

Alg UpdateState(Tx)
// *Update seeds, outputs, and signatures.*
01 **parse** $(\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi) := \mathsf{Tx}$
02 $\mathsf{TXs} := \mathsf{TXs} \cup \{\mathsf{Tx}\}$
03 $\mathsf{PSeeds} := \mathsf{PSeeds} \cup \{\mathsf{pseed}\}$
04 **for** $(\mathsf{pk}, \mathsf{com}, \mathsf{ct}, \mathsf{tag}) \in \mathsf{Out}$:
05 $\quad \mathsf{Outputs} := \mathsf{Outputs} \cup \{(\mathsf{pk}, \mathsf{com})\}$
06 **for** $(\mathsf{Ref}, \mathsf{com}, \sigma) \in \mathsf{In}$:
07 $\quad \mathsf{Signatures} := \mathsf{Signatures} \cup \{\sigma\}$

Alg UpdateBalances(Tx)
// *Update balances of honest users.*
08 **parse** $(\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi) := \mathsf{Tx}$
09 $\mathsf{totalrcvd} := 0$
10 **for** $(\mathsf{pk}, \mathsf{com}, \mathsf{ct}, \mathsf{tag}) \in \mathsf{Out}$:
11 $\quad$ **for** $\mathsf{ipk} \in \mathsf{Identities}$ s.t. $\mathsf{corr}[\mathsf{ipk}] \neq 2$:
12 $\quad\quad \mathsf{rcvd} := \mathsf{StoreReceive}(\mathsf{ipk}, \mathsf{pseed}, \mathsf{pk}, \mathsf{com}, \mathsf{tag}, \mathsf{ct})$
13 $\quad\quad \mathsf{totalrcvd} := \mathsf{totalrcvd} + \mathsf{rcvd}$
14 **return** totalrcvd

Alg StoreReceive(ipk, pseed, pk, com, tag, ct)
// *An honest user tries to receive an output and store it.*
15 $(\mathsf{amt}, \mathsf{cr}, \mathsf{sk}) := \mathsf{Receive}(\mathsf{ipk}, \mathsf{ivk}[\mathsf{ipk}], \mathsf{isk}[\mathsf{ipk}], \mathsf{pseed}, \mathsf{pk}, \mathsf{com}, \mathsf{tag}, \mathsf{ct})$
16 **if** $\mathsf{amt} > 0$: $\mathsf{Owned}[\mathsf{ipk}] := \mathsf{Owned}[\mathsf{ipk}] \cup \{(\mathsf{pk}, \mathsf{com}, \mathsf{amt}, \mathsf{cr}, \mathsf{sk})\}$
17 **return** amt

Figure 5: The algorithms UpdateState, UpdateBalances, and StoreReceive used in the security definition in Section 4.2. Intuitively, these algorithms are called when outputs or transactions are added to the system, see Figure 6.

Oracle NewHonSrc(pk, pseed, com, tag, ct)

// *Check if non-corrupted user can receive the output.*
01 **for** ipk ∈ Identities s.t. corr[ipk] < 2 :
02 rcvd := StoreReceive(ipk, pseed, pk, com, tag, ct)
03 **if** rcvd > 0 : Outputs := Outputs ∪ {(pk, com)}

Oracle NewSrc(pk, amt, cr)
04 com := Com(amt, cr)
05 Outputs := Outputs ∪ {(pk, com)}

// *Adversary can control this output.*
06 received := received + amt

Oracle AddAdvTrans(Tx)
07 **if** VerTx(PSeeds, Outputs, Signatures, Tx) = 0 : **return**
08 UpdateState(Tx)
09 spentnow ← UpdateBalances(Tx)
10 spent := spent + spentnow
11 **if** spent > received : win-create := 1

Oracle AddHonTrans(ipk, $(\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i\in[K]}$, ISamp, RSamp)
12 **if** ipk ∉ Identities ∨ corr[ipk] = 2 : **return** ⊥

// *Sample parameters for the transaction.*
13 $(\mathsf{Use}_j)_{j=1}^{L}$ ← ISamp(Owned[ipk])
14 $(\mathsf{Ref}_j)_{j=1}^{L}$ ← RSamp(Owned[ipk], $(\mathsf{Use}_j)_{j=1}^{L}$)

// *Check that parameters can lead to valid transaction.*
15 inputsum := 0
16 **for** $j \in [L]$:
17 **if** $\mathsf{Use}_j \notin$ Owned[ipk] : **return** ⊥
18 **parse** $(\mathsf{pk}_j, \mathsf{com}_j, \mathsf{amt}_j, \mathsf{cr}_j, \mathsf{sk}_j) := \mathsf{Use}_j$
19 inputsum := inputsum + amt_j
20 **if** $(\mathsf{pk}_j, \mathsf{com}_j) \notin \mathsf{Ref}_j$: **return** ⊥
21 **if** ∃(pk, com) ∈ Ref_j s.t. (pk, com) ∉ Outputs : **return** ⊥
22 **if** inputsum ≠ $\sum_{i=1}^{K} \mathsf{amt}_i^{out}$: **return** ⊥

// *Generate transaction, update state, check winning condition.*
23 Tx ← GenTx $\left((\mathsf{Use}_j)_{j=1}^{L}, (\mathsf{Ref}_j)_{j=1}^{L}, (\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i=1}^{K}\right)$
24 **if** VerTx(PSeeds, Outputs, Signatures, Tx) = 0 :
25 win-steal := 1, **return** ⊥
26 Owned[ipk] := Owned[ipk] \ $\{\mathsf{Use}_j\}_{j\in[L]}$
27 UpdateState(Tx)
28 UpdateBalances(Tx)
29 **for** $i \in [K]$ s.t. $\mathsf{ipk}_i \notin$ Identities ∨ corr[ipk] = 2 :
30 received := received + amt_i^{out}
31 **return** Tx

Figure 6: The oracles O_{src} = (NewHonSrc, NewSrc) and O_{tx} = (AddAdvTrans, AddHonTrans) that are used in the security definition in Section 4.2. Algorithms UpdateState, UpdateBalances, and StoreReceive are given in Figure 5.

5 Security Notions for Components

In this section, we fix a private transaction scheme $\mathsf{PTS} = (\mathsf{LRS}, \mathsf{KDS}, \mathsf{VHC}, \mathsf{DE}, \mathsf{KCS})$. We define security notions for different components of the scheme. Looking ahead, we will show that if all components satisfy these notions, then PTS is secure. For each of the notions, we give some intuition assuming that the reader carefully followed Section 2. For a quick overview, we refer to Table 1.

5.1 Notions for Key Derivation

We start with security notions regarding the key derivation scheme KDS. Later, we will see that KDS also impacts the notions for the linkable ring signature scheme.

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.

Definition 2 (Tracking Soundness)**.** *Let $\mathcal{A}$ be an algorithm. We define the tracking soundness game $\mathbf{TR\text{-}SOUND}^{\mathcal{A}}_{\mathsf{KDS}}(\lambda)$ as follows:*

1. *Generate* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \leftarrow \mathsf{GenID}(\mathsf{par})$ *and run* $\mathcal{A}$ *on input* $\mathsf{par}, \mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}$.
2. *Obtain* $\mathsf{pseed}, \mathsf{pk}, \mathsf{tag}$ *from* $\mathcal{A}$ *and compute* $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed})$ *and* $\mathsf{sk} := \mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}, \mathsf{tag})$.
3. *Output* 1 *if* $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) = 1$ *and* $(\mathsf{pk}, \mathsf{sk}) \notin \mathsf{KDS}.\mathcal{KR}$. *Otherwise, output* 0.

We say that KDS *satisfies tracking soundness, if for every PPT algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{tr\text{-}sound}}_{\mathcal{A},\mathsf{KDS}}(\lambda) := \Pr\left[\mathbf{TR\text{-}SOUND}^{\mathcal{A}}_{\mathsf{KDS}}(\lambda) \rightarrow 1\right].$$

Key Spreadness. We introduce a notion that we call key spreadness. Roughly, it states that different public seeds $\mathsf{pseed}, \mathsf{pseed}'$ or different tags $\mathsf{tag}, \mathsf{tag}'$ lead to different derived keys $\mathsf{sk}, \mathsf{sk}'$. Looking ahead, this ensures that no two signatures generated by honest users link.

Definition 3 (Key Spreadness)**.** *Let $\mathcal{A}$ be an algorithm. We define the key spreadness game $\mathbf{SPREAD}^{\mathcal{A}}_{\mathsf{KDS}}(\lambda)$ as follows:*

1. *Initialize empty lists $\mathcal{L}_{id}$ and run $\mathcal{A}$ on input* par *with access to an oracle* NewIdentity *that does not take any input, generates* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \leftarrow \mathsf{GenID}(\mathsf{par})$*, insert* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$ *into $\mathcal{L}_{id}$, and returns* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$.
2. *Obtain an output* $\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}, \mathsf{pseed}, \mathsf{pseed}' \in \mathcal{PSE}, \mathsf{tag}, \mathsf{tag}' \in \mathbb{N}$ *from* $\mathcal{A}$.
3. *Compute*

$$\begin{aligned} \mathsf{sk} &:= \mathsf{DerSK}(\mathsf{isk}, \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}), \mathsf{tag}), \\ \mathsf{sk}' &:= \mathsf{DerSK}(\mathsf{isk}, \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}'), \mathsf{tag}'). \end{aligned}$$

4. *Output* 1 *if* $(\mathsf{pseed}, \mathsf{tag}) \neq (\mathsf{pseed}', \mathsf{tag}')$*,* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \in \mathcal{L}_{id}$*, and* $\mathsf{sk} = \mathsf{sk}'$. *Otherwise, output* 0.

We say that KDS *satisfies key spreadness, if for every PPT algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{spread}}_{\mathcal{A},\mathsf{KDS}}(\lambda) := \Pr\left[\mathbf{SPREAD}^{\mathcal{A}}_{\mathsf{KDS}}(\lambda) \rightarrow 1\right].$$

5.2 Notions for the Verifiable Homomorphic Commitment

Next, we define the security notions related to the verifiable homomorphic commitment scheme VHC, and the key conversion scheme KCS.

Conversion Soundness. Recall that the key conversion scheme KCS allows to transform pairs of commitment randomness $\mathsf{cr}, \mathsf{cr}'$ for commitments $\mathsf{com}, \mathsf{com}'$ 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'.

Definition 4 (Conversion Soundness)**.** *Let $\mathcal{A}$ and $\mathsf{Translate}$ be algorithms. We define the conversion soundness game $\mathbf{CONV\text{-}SOUND}^{\mathcal{A}}_{\mathsf{KCS},\mathsf{VHC},\mathsf{Translate}}(\lambda)$ as follows:*

1. *Run $\mathcal{A}$ on input par and obtain $\mathsf{amt}, \mathsf{cr}', \mathsf{com}, \mathsf{com}', \mathsf{auxpk}, \mathsf{auxsk}$ in return.*
2. *Run $\mathsf{cr} \leftarrow \mathsf{Translate}(\mathsf{amt}, \mathsf{cr}', \mathsf{com}, \mathsf{com}', \mathsf{auxpk}, \mathsf{auxsk})$.*
3. *Output 1 if both of the following two conditions hold, otherwise output 0:*
   (a) $\mathsf{Com}(\mathsf{amt}, \mathsf{cr}') = \mathsf{com}'$, $(\mathsf{auxpk}, \mathsf{auxsk}) \in \mathcal{AKR}$, *and* $\mathsf{ConvertPublic}(\mathsf{com}, \mathsf{com}') = \mathsf{auxpk}$.
   (b) $\mathsf{Com}(\mathsf{amt}, \mathsf{cr}) \neq \mathsf{com}$ *or* $\mathsf{ConvertSecret}(\mathsf{cr}, \mathsf{cr}') \neq \mathsf{auxsk}$.

We say that $(\mathsf{KCS}, \mathsf{VHC})$ satisfies conversion soundness, if there is a PPT algorithm $\mathsf{Translate}$, such that for every PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{conv\text{-}sound}}_{\mathcal{A},\mathsf{Translate},\mathsf{KCS},\mathsf{VHC}}(\lambda) := \Pr\left[\mathbf{CONV\text{-}SOUND}^{\mathcal{A}}_{\mathsf{KCS},\mathsf{VHC},\mathsf{Translate}}(\lambda) \rightarrow 1\right].$$

In this case, we say that $\mathsf{Translate}$ is the conversion translator of $(\mathsf{KCS}, \mathsf{VHC})$.

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.

Definition 5 (Binding)**.** *We say that VHC is binding, if for every PPT algorithm, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathcal{A},\mathsf{VHC}}(\lambda) := \Pr_{\mathfrak{G}}\left[\begin{array}{rl} (\mathsf{amt}, \mathsf{cr}) & \neq (\mathsf{amt}', \mathsf{cr}') \\ \wedge\ \mathsf{Com}(\mathsf{amt}, \mathsf{cr}) & = \mathsf{Com}(\mathsf{amt}', \mathsf{cr}') \end{array}\right],$$

where $\mathfrak{G}$ is given as $(\mathsf{amt}, \mathsf{cr}, \mathsf{amt}', \mathsf{cr}') \leftarrow \mathcal{A}(\mathsf{par})$.

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.

Definition 6 (Commitment Knowledge Soundness)**.** *Let $\mathcal{A}$ and $\mathsf{Ext}_{\mathsf{VHC}}$ be algorithms. We define the commitment knowledge soundness game* $\mathbf{C\text{-}KN\text{-}SOUND}^{\mathcal{A}}_{\mathsf{VHC},\mathsf{Ext}_{\mathsf{VHC}}}(\lambda)$ *as follows:*

1. *Set* $\mathsf{bad} := 0$. *Let* O *be an oracle which does the following on input* (stmt, π)*:*
   (a) *Run* $\mathsf{witn} \leftarrow \mathsf{Ext}_{\mathsf{VHC}}(\mathsf{stmt}, \pi)$*, where* $\mathsf{Ext}_{\mathsf{VHC}}$ *may inspect the random oracle queries (if any) of* $\mathcal{A}$*.*
   (b) *Write* $\mathsf{stmt} = (\mathsf{com}_i)_i$ *and* $\mathsf{witn} = (\mathsf{amt}_i, \mathsf{cr}_i)_i$*.*
   (c) *If* $\mathsf{PVer}(\mathsf{stmt}, \pi) = 1$ *and there is some* i *such that* $\mathsf{Com}(\mathsf{amt}_i, \mathsf{cr}_i) \neq \mathsf{com}_i$*, then set* $\mathsf{bad} := 1$*.*
2. *Run* $\mathcal{A}$ *on input* par *with access to oracle* O*. Then, output* bad*.*

We say that VHC *satisfies commitment knowledge soundness, if there is a PPT algorithm* $\mathsf{Ext}_{\mathsf{VHC}}$*, such that for every PPT algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{c\text{-}kn\text{-}sound}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{VHC}},\mathsf{VHC}}(\lambda) := \Pr\left[\mathbf{C\text{-}KN\text{-}SOUND}^{\mathcal{A}}_{\mathsf{VHC},\mathsf{Ext}_{\mathsf{VHC}}}(\lambda) \rightarrow 1\right].$$

In this case, we refer to algorithm $\mathsf{Ext}_{\mathsf{VHC}}$ *as the commitment knowledge extractor of* VHC*.*

5.3 Notions for the Ring Signature

In this section, we define security notions for the two-dimensional linkable ring signature scheme LRS. Notably, these deviate from standard notions for linkable ring signatures, and are related to the key derivation scheme KDS. Intuitively, this is because key derivation may introduce additional relations between keys that are not captured by the standard notions.

Non-Slanderability. A well-established notion for linkable ring signatures is non-slanderability [TW05, GNB19] (sometimes called non-frameability [BDH+19, BKP20]). 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.

Definition 7 (Non-Slanderability)**.** *Let $\mathcal{A}$ be an algorithm. We define the non-slanderability game* $\mathbf{NON\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda)$ *as follows:*

1. *Initialize empty lists* $\mathcal{L}_{id}, \mathcal{L}_s, \mathcal{L}_c, \mathsf{Signatures}$*, and* PKs*, and define the following oracles:*
   - NewIdentity() : *Generate* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \leftarrow \mathsf{GenID}(\mathsf{par})$ *and insert the triple* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$ *into* $\mathcal{L}_{id}$*. Return* $(\mathsf{ipk}, \mathsf{ivk})$*.*
   - Corr(ipk) : *If there is an entry of the form* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$ *in* $\mathcal{L}_{id}$*, then insert* ipk *into* $\mathcal{L}_c$ *and return* isk*. Otherwise, return* $\perp$*.*
   - Sign(pk, pseed, tag, R, auxsk, m) :
     (a) *Find an entry* $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \in \mathcal{L}_{id}$ *such that* $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) = 1$ *for* $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed})$*. If no such entry is found, return* $\perp$*.*

(b) *Run* $\mathsf{sk} := \mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}, \mathsf{tag})$.

(c) *Compute* $\sigma \leftarrow \mathsf{Sig}(\mathsf{R}, \mathsf{sk}, \mathsf{auxsk}, \mathsf{m})$. *Then, insert* $(\mathsf{ipk}, \mathsf{pk}, \mathsf{R}, \mathsf{m}, \sigma)$ *into* $\mathcal{L}_s$, σ *into* Signatures, *and* pk *into* PKs.

(d) *Return* σ.

2. *Run* $\mathcal{A}$ *on input* par *with access to oracles* NewIdentity, Corr, Sign. *Obtain an output* $(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$ *from* $\mathcal{A}$. *Output* 1 *if all of the following conditions hold. Otherwise output* 0*:*

(a) *We have* $\mathsf{Ver}(\mathsf{R}^*, \mathsf{m}^*, \sigma^*) = 1$.

(b) *There does not exist* $\mathsf{ipk}, \mathsf{pk}, \sigma$ *such that* $(\mathsf{ipk}, \mathsf{pk}, \mathsf{R}^*, \mathsf{m}^*, \sigma) \in \mathcal{L}_s$.

(c) *There exists an* $(\mathsf{ipk}, \mathsf{pk}, \mathsf{R}, \mathsf{m}, \sigma) \in \mathcal{L}_s$ *such that* $\mathsf{Link}(\sigma, \sigma^*) = 1$ *and* $\mathsf{ipk} \notin \mathcal{L}_c$.

We say that $(\mathsf{LRS}, \mathsf{KDS})$ *satisfies non-slanderability, if for every PPT algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{non\text{-}sland}}_{\mathcal{A},\mathsf{LRS},\mathsf{KDS}}(\lambda) := \Pr\left[\mathbf{NON\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda) \rightarrow 1\right].$$

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.

Definition 8 (Key Onewayness)**.** *Let* $\mathcal{A}$ *be an algorithm. We define the key onewayness game* $\mathbf{K\text{-}OW}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda)$ *as follows:*

1. *Let the oracles* NewIdentity, Corr, Sign *be as in game* $\mathbf{NON\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda)$.

2. *Run* $\mathcal{A}$ *on input* par *with access to oracles* NewIdentity, Corr, Sign. *Obtain an output* $(\mathsf{ipk}^*, \mathsf{pk}^*, \mathsf{pseed}^*, \mathsf{tag}^*, \mathsf{sk}^*)$ *from* $\mathcal{A}$. *Output* 1 *if both of the following conditions hold. Otherwise output* 0*:*

(a) *We have* $(\mathsf{pk}^*, \mathsf{sk}^*) \in \mathcal{KR}$.

(b) *There exists an entry* $(\mathsf{ipk}^*, \mathsf{ivk}, \mathsf{isk}) \in \mathcal{L}_{id}$ *such that* $\mathsf{ipk}^* \notin \mathcal{L}_c$, *and* $\mathsf{Track}(\mathsf{ipk}^*, \mathsf{ok}, \mathsf{pk}^*, \mathsf{tag}^*) = 1$ *for* $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}^*)$.

We say that $(\mathsf{LRS}, \mathsf{KDS})$ *satisfies key onewayness, if for every PPT algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{k\text{-}ow}}_{\mathcal{A},\mathsf{LRS},\mathsf{KDS}}(\lambda) := \Pr\left[\mathbf{K\text{-}OW}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda) \rightarrow 1\right].$$

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 $(\mathsf{sk}, \mathsf{auxsk})$. 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 $(\mathsf{R}, \mathsf{m}, \sigma)$ 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.

Definition 9 (Key Knowledge Soundness)**.** *Let $\mathcal{A}$ be an algebraic algorithm, and $\mathsf{Ext}_{\mathsf{LRS}}$ be an algorithm. We define the key knowledge soundness game $\mathbf{K\text{-}KN\text{-}SOUND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda)$ as follows:*

1. *Let the oracles* NewIdentity, Corr, Sign *be as in game $\mathbf{NON\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda)$.*
2. *Let* O *be an oracle which does the following on input $(\mathsf{R},\mathsf{m},\sigma)$:*
   (a) *If there are some $\mathsf{ipk},\mathsf{pk},\sigma'$ such that an entry $(\mathsf{ipk},\mathsf{pk},\mathsf{R},\mathsf{m},\sigma')$ is in $\mathcal{L}_s$, then return.*
   (b) *If $\mathsf{Ver}(\mathsf{R},\mathsf{m},\sigma)=0$, then return.*
   (c) *Run $(i^*,\mathsf{sk}_{i^*},\mathsf{auxsk}_{i^*}) \leftarrow \mathsf{Ext}_{\mathsf{LRS}}(\mathsf{R},\mathsf{m},\sigma)$, where $\mathsf{Ext}_{\mathsf{LRS}}$ may inspect the random oracle queries (if any) of $\mathcal{A}$, but $\mathsf{Ext}_{\mathsf{LRS}}$ is not allowed to inspect the state or random oracle queries of other oracles or the game. Further, $\mathsf{Ext}_{\mathsf{LRS}}$ is allowed to inspect all interfaces (i.e., inputs and outputs) of oracles* NewIdentity, Corr, Sign*.*
   (d) *Write $\mathsf{R}=(\mathsf{pk}_i,\mathsf{auxpk}_i)_i$. If $(\mathsf{auxpk}_{i^*},\mathsf{auxsk}_{i^*}) \notin \mathcal{AKR}$ or $(\mathsf{pk}_{i^*},\mathsf{sk}_{i^*}) \notin \mathcal{KR}$, then set $\mathsf{bad} := 1$.*
3. *Run $\mathcal{A}$ on input* par *with access to oracles* NewIdentity, Corr, Sign, O*. Then, output* bad*.*

We say that (LRS, KDS) *satisfies key knowledge soundness, if there is a PPT algorithm $\mathsf{Ext}_{\mathsf{LRS}}$, such that for every PPT algorithm $\mathcal{A}$ the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{k\text{-}kn\text{-}sound}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) := \Pr\left[\mathbf{K\text{-}KN\text{-}SOUND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda) \rightarrow 1\right].$$

In this case, we refer to algorithm $\mathsf{Ext}_{\mathsf{LRS}}$ as the knowledge extractor of (LRS, KDS)*.*

Remark 1. *For the proof of key knowledge soundness (cf. Lemma 16), we encourage the interested reader to identify the complexity that stems from the existence of oracles* NewIdentity, Corr, Sign*. In a nutshell, the proof would be significantly less cumbersome if the adversary was not able to use group elements output by these oracles as basis elements for the representation in the algebraic group model.*

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.

Definition 10 (Knowledge Linkability)**.** *Let $\mathcal{A}$ be an algebraic algorithm, and $\mathsf{Ext}_{\mathsf{LRS}}$ be an algorithm. We define the knowledge linkability game $\mathbf{KN\text{-}LINK}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda)$ as follows:*

1. *Let the oracles* NewIdentity, Corr, Sign *be as in game $\mathbf{NON\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda)$.*
2. *Let* O *be an oracle which does the following on input $(\mathsf{R},\mathsf{m},\sigma)$:*
   (a) *If there are some $\mathsf{ipk},\mathsf{pk},\sigma'$ such that an entry $(\mathsf{ipk},\mathsf{pk},\mathsf{R},\mathsf{m},\sigma')$ is in $\mathcal{L}_s$, then return.*
   (b) *If $\mathsf{Ver}(\mathsf{R},\mathsf{m},\sigma)=0$, return. Otherwise, parse $\mathsf{R}=(\mathsf{pk}_i,\mathsf{auxpk}_i)_i$.*
   (c) *Run $(i^*,\mathsf{sk}_{i^*},\mathsf{auxsk}_{i^*}) \leftarrow \mathsf{Ext}_{\mathsf{LRS}}(\mathsf{R},\mathsf{m},\sigma)$, where $\mathsf{Ext}_{\mathsf{LRS}}$ may inspect the random oracle queries (if any) of $\mathcal{A}$, but $\mathsf{Ext}_{\mathsf{LRS}}$ is not allowed to inspect the state or random oracle queries of other oracles or the game. Further, $\mathsf{Ext}_{\mathsf{LRS}}$ is allowed to inspect all interfaces (i.e., inputs and outputs) of oracles* NewIdentity, Corr, Sign*.*
   (d) *If $(i^*,\mathsf{sk}_{i^*},\mathsf{auxsk}_{i^*}) \neq \bot$, and all of the following conditions hold, then set $\mathsf{bad} := 1$:*
      i. *For all $\sigma' \in \mathsf{Signatures}$, we have $\mathsf{Link}(\sigma,\sigma')=0$.*
      ii. *The key pk_{i^*} is in PKs (i.e., a corresponding secret key has been used to generate a signature in $\mathsf{Signatures}$).*
   (e) *Insert pk_{i^*} into PKs and σ into $\mathsf{Signatures}$.*
3. *Run $\mathcal{A}$ on input par with access to oracles* NewIdentity, Corr, Sign, O*. Then, output bad.*

Assume that $(\mathsf{LRS},\mathsf{KDS})$ satisfies key knowledge soundness with knowledge extractor $\mathsf{Ext}_{\mathsf{LRS}}$. We say that $(\mathsf{LRS},\mathsf{KDS})$ satisfies knowledge linkability, if for every PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{kn\text{-}link}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) := \Pr\left[\mathbf{KN\text{-}LINK}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda) \rightarrow 1\right].$$

Knowledge Non-Slanderability. While we already defined non-slanderability, it will be convenient to show non-slanderability by using the knowledge extractor. We capture the requirements that the extractor needs to satisfy to be useful for this in the following notion. However, we highlight that this notion is not used directly in the proof of the system. Instead, we will show that this notion, in combination with key onewayness, implies non-slanderability, see Lemma 10. Intuitively, this notion, which we call knowledge non-slanderability, serves as a counterpart to knowledge linkability: Knowledge linkability states that if we extract the same key, then signatures will link. Knowledge non-slanderability states that if two signatures link, then we extract the same key.

Definition 11 (Knowledge Non-Slanderability)**.** *Let $\mathcal{A}$ be an algebraic algorithm, and $\mathsf{Ext}_{\mathsf{LRS}}$ be an algorithm. We define the knowledge non-slanderability game $\mathbf{KN\text{-}N\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda)$ as follows:*

1. *Let the oracles* NewIdentity, Corr, Sign *be as in game $\mathbf{NON\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS}}(\lambda)$.*
2. *Let* O *be an oracle which does the following on input $(\mathsf{R},\mathsf{m},\sigma)$:*
   (a) *If there are some $\mathsf{ipk},\mathsf{pk},\sigma'$ such that an entry $(\mathsf{ipk},\mathsf{pk},\mathsf{R},\mathsf{m},\sigma')$ is in $\mathcal{L}_s$, then return.*
   (b) *If $\mathsf{Ver}(\mathsf{R},\mathsf{m},\sigma)=0$, return. Otherwise, parse $\mathsf{R}=(\mathsf{pk}_i,\mathsf{auxpk}_i)_i$.*
   (c) *Run $(i^*,\mathsf{sk}_{i^*},\mathsf{auxsk}_{i^*}) \leftarrow \mathsf{Ext}_{\mathsf{LRS}}(\mathsf{R},\mathsf{m},\sigma)$, where $\mathsf{Ext}_{\mathsf{LRS}}$ may inspect the random oracle queries (if any) of $\mathcal{A}$, but $\mathsf{Ext}_{\mathsf{LRS}}$ is not allowed to inspect the state or random oracle queries of other oracles or the game. Further, $\mathsf{Ext}_{\mathsf{LRS}}$ is allowed to inspect all interfaces (i.e., inputs and outputs) of oracles* NewIdentity, Corr, Sign*.*

(d) If $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*}) \neq \perp$*, and there is an entry* $(\mathsf{ipk}', \mathsf{pk}', \mathsf{R}', \mathsf{m}', \sigma')$ *in* $\mathcal{L}_s$ *such that following conditions hold, then set* $\mathsf{bad} := 1$*:*

i. We have $\mathsf{Link}(\sigma, \sigma') = 1$.

ii. We have $\mathsf{pk}_{i^*} \neq \mathsf{pk}'$.

(e) Insert pk_{i^*} *into* PKs *and* σ *into* Signatures.

3. Run $\mathcal{A}$ *on input* par *with access to oracles* NewIdentity, Corr, Sign, O. *Then, output* bad.

Assume that (LRS, KDS) *satisfies key knowledge soundness with knowledge extractor* $\mathsf{Ext}_{\mathsf{LRS}}$. *We say that* (LRS, KDS) *satisfies knowledge non-slanderability, if for every PPT algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{kn\text{-}n\text{-}sland}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) := \Pr\left[\mathbf{KN\text{-}N\text{-}SLAND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda) \rightarrow 1\right].$$

Notion	Components	Definition	Proof	Used to Bound win-steal	Used to Bound win-create
Tracking Soundness	KDS	Definition 2	Lemma 5	✓	✓
Key Spreadness	KDS	Definition 3	Lemma 6	✓	·
Conversion Soundness	KCS, VHC	Definition 4	Lemma 8	·	✓
Binding	VHC	Definition 5	Lemma 7	·	✓
Comm. Knowledge Soundness	VHC	Definition 6	-	·	✓
Non-Slanderability	LRS, KDS	Definition 7	Lemma 10	✓	·
Key Onewayness	LRS, KDS	Definition 8	Lemma 9	·	✓
Key Knowledge Soundn.	LRS, KDS	Definition 9	Lemma 16	·	✓
Knowledge Linkability	LRS, KDS	Definition 10	Lemma 17	·	✓

Table 1: Overview of the security notions we define for components. We list for which components the notions are defined, where they are defined, and where they are proven for the components used in Monero.

6 System Level Security Analysis

In this section, we prove the security of a private transaction scheme assuming the security of building blocks. Before we give the formal statement and proof, we present the main ideas in an informal overview. For both the informal overview and the formal analysis, we consider the two winning conditions separately. We assume that the reader is familiar with the syntax and security definition introduced in Section 4, the security notions for building blocks introduced in Section 5, and our informal overview of Monero in Section 2.

Honest User Can not Spend. We start with winning condition win-steal. Formally, we bound the probability that the adversary wins by triggering this condition in Lemma 1. 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, see Figure 3. Write $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$, $\mathsf{In} = (\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_{j=1}^{L}$, and $\mathsf{Out} = \left(\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i\right)_{i=1}^{K}$. 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 (see algorithm GenTx in Figure 2), 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 $\sum_{j=1}^{L} \mathsf{com}_j \neq \sum_{i=1}^{K} \mathsf{com}_i^{out}$. 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 $o = (\mathsf{pk}, \mathsf{com})$ that an honest user receives from an adversary, it derives a valid secret key sk such that $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{KR}$. 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 $\mathsf{win\text{-}create}$. Formally, the probability that the adversary wins by triggering this condition is bounded in Lemma 3. 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 one[4] 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:

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

(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 $L + \mathsf{received} - \mathsf{spent}$, 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 $L + \mathsf{received}$, because each edge with weight f from V_s to V_t which is not going into vertex t increases the value of $\mathsf{received}$ 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 $\mathsf{received} - \mathsf{spent} \geq 0$, 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 $L + \mathsf{received} - \mathsf{spent} \geq T$. By definition, we have $T \geq L$, and thus $L + \mathsf{received} - \mathsf{spent} \geq L$. Subtracting L from both sides, we get $\mathsf{received} - \mathsf{spent} \geq 0$, i.e., $\mathsf{received} \geq \mathsf{spent}$, which means the adversary can not create coins.

Formal Security Result. We obtain the following security theorem, which follows directly from Lemmas 1 and 3.

Theorem 1. *Let* $\mathsf{PTS} = (\mathsf{LRS}, \mathsf{KDS}, \mathsf{VHC}, \mathsf{DE}, \mathsf{KCS})$ *be a private transaction scheme, and assume that the following properties hold:*

- KDS *satisfies tracking soundness and key spreadness, and has* $\omega(\log \lambda)$ *bits of entropy of public seeds.*
- $(\mathsf{KCS}, \mathsf{VHC})$ *satisfies conversion soundness, and* VHC *is binding and satisfies commitment knowledge soundness.*
- $(\mathsf{LRS}, \mathsf{KDS})$ *satisfies non-slanderability, key onewayness, key knowledge soundness, and knowledge linkability.*
- *The relation* $\mathcal{KR}$ *is unique, i.e., for each* $\mathsf{pk} \in \mathcal{PK}$ *there is at most one* $\mathsf{sk} \in \mathcal{SK}$ *such that* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{KR}$.

Then, PTS *is secure.*

6.1 Bounding Winning Condition win-steal

Lemma 1. *Assume* KDS *has h bits of entropy of public seeds. Let $\mathcal{A}$ be a PPT algorithm making at most $Q_{aht}, Q_{at}, Q_{ni}, Q_{\mathsf{H}_{trm}}$ queries to oracles* AddHonTrans, AddAdvTrans, NewIdentity, H_{trm}*, respectively. Then, there are PPT algorithms $\mathcal{B}, \mathcal{B}', \mathcal{B}''$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A}), \mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{A}), \mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{A})$ and*

$$\Pr[\mathsf{win\text{-}steal} = 1] \leq \frac{Q_{aht}(Q_{at} + Q_{aht})}{2^h} + \frac{Q_{\mathsf{H}_{trm}}^2}{2^\ell} + 2Q_{ni} \cdot \mathsf{Adv}^{\mathsf{tr\text{-}sound}}_{\mathcal{B},\mathsf{KDS}}(\lambda) + \mathsf{Adv}^{\mathsf{spread}}_{\mathcal{B}',\mathsf{KDS}}(\lambda) + \mathsf{Adv}^{\mathsf{non\text{-}sland}}_{\mathcal{B}'',\mathsf{LRS},\mathsf{KDS}}(\lambda).$$

Proof. Consider an adversary $\mathcal{A}$ and the game $\mathbf{UNF}^{\mathcal{A}}_{\mathsf{PTS}}(\lambda)$. Assume that win-steal occurs. By definition, this means that during an oracle query AddHonTrans$(\mathsf{ipk}, (\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i=1}^K, \mathsf{ISamp}, \mathsf{RSamp})$ a transaction $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$ is generated using algorithm GenTx, and then it holds that $\mathsf{VerTx}(\mathsf{PSeeds}, \mathsf{Outputs}, \mathsf{Signatures}, \mathsf{Tx}) = 0$. Recall that algorithm VerTx outputs 0 due to one of five reasons. For each of these, we define a corresponding event. Formally, write $\mathsf{In} = (\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_{j=1}^L$ and $\mathsf{Ref}_j = (\mathsf{pk}_{j,r}^{in}, \mathsf{com}_{j,r}^{in})_{r=1}^{|\mathsf{Ref}_j|}$ for each $j \in [L]$. Also, define the variables $\mathsf{m}, \mathsf{auxpk}_r, \mathsf{R}_j$ as in the definition of algorithm VerTx. We define the following events, covering the four reasons that lead to algorithm VerTx outputting 0. When we say that such an event occurs, we mean that it occurs in any oracle call.

1. Event PSeedColl: This event occurs, if $\mathsf{pseed} \in \mathsf{PSeeds}$.
2. Event NoPrevious: This event occurs, if there is a $j \in [L]$ and an $r \in [|\mathsf{Ref}_j|]$ such that $(\mathsf{pk}_{j,r}^{in}, \mathsf{com}_{j,r}^{in}) \notin \mathsf{Outputs}$. This means that there is an input that is not a previous output.
3. Event InvalidSig: This event occurs, if there is a $j \in [L]$ such that $\mathsf{LRS.Ver}(\mathsf{R}_j, \mathsf{m}, \sigma_j) = 0$. This means that there is a signature that is not valid.
4. Event InvalidCom: This event occurs, if for $\mathsf{stmt} = \left((\mathsf{com}_j)_{j=1}^L, (\mathsf{com}_i^{out})_{i=1}^K\right)$ we have $\mathsf{PVer}(\mathsf{stmt}, \pi) = 0$, or if we have $\sum_{j=1}^L \mathsf{com}_j \neq \sum_{i=1}^K \mathsf{com}_i^{out}$. This means that the commitments are not valid.
5. Event Linking: This event occurs if there are distinct $j, j' \in [L]$ such that $\mathsf{Link}(\sigma_j, \sigma_{j'}) = 1$ or $\mathsf{Link}(\sigma_j, \sigma) = 1$ for some $\sigma \in \mathsf{Signatures}$. In such a case, we call σ_j and $\sigma_{j'}$ *new linking signatures*, and σ an *old linking signature*.

Clearly, we have

$$\Pr[\mathsf{win\text{-}steal} = 1] \leq \Pr[\mathsf{PSeedColl}] + \Pr[\mathsf{NoPrevious}] + \Pr[\mathsf{InvalidSig}] + \Pr[\mathsf{InvalidCom}] + \Pr[\mathsf{Linking} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}].$$

We bound the probability of these events separately.

Claim 1. $\Pr[\mathsf{PSeedColl}] \leq Q_{aht}(Q_{at} + Q_{aht}) \cdot 2^{-h}$.

To show the claim, consider a fixed call of the Q_{aht} calls to oracle AddHonTrans. As the size of list PSeeds is at most $Q_{at} + Q_{aht}$, the probability that in this call event PSeedColl occurs is at most $(Q_{at} + Q_{aht}) \cdot 2^{-h}$, by a union bound and the entropy of public seeds. Another union bound over the Q_{aht} calls to oracle AddHonTrans shows the claim.

Claim 2. $\Pr[\mathsf{NoPrevious}] = 0$.

Recall that event NoPrevious can only occur in calls to oracle AddHonTrans. Thus, consider such a call. Recall that oracle AddHonTrans checks for each $j \in [L]$ if there is some entry $(\mathsf{pk}, \mathsf{com})$ in Ref_j such that $(\mathsf{pk}, \mathsf{com}) \notin \mathsf{Outputs}$. If this is the case for any such j, oracle AddHonTrans does not generate a transaction Tx at all. Therefore, we know that $\Pr[\mathsf{NoPrevious}] = 0$.

Claim 3. $\Pr[\mathsf{InvalidCom}] = 0$.

Recall the definition of $(\mathsf{Use}_j)_{j=1}^{L}$ from oracle AddHonTrans. For each $j \in [L]$ we can write $\mathsf{Use}_j = (\mathsf{pk}_j, \mathsf{com}_j, \mathsf{amt}_j, \mathsf{cr}_j, \mathsf{sk}_j)$. We can assume that for each $j \in [L]$, it holds that $\mathsf{Use}_j \in \mathsf{Owned}[\mathsf{ipk}]$, as otherwise the oracle would return $\perp$ and not generate Tx. Therefore, we know that $\mathsf{com}_j = \mathsf{Com}(\mathsf{amt}_j, \mathsf{cr}_j)$, as otherwise this entry would not have been added to list $\mathsf{Owned}[\mathsf{ipk}]$. Also, oracle AddHonTrans would return $\perp$ and not generate Tx if $\sum_{j=1}^{L} \mathsf{amt}_j \neq \sum_{i=1}^{K} \mathsf{amt}_i^{out}$. By definition of algorithm GenTx, we have

$$\mathsf{com}_i^{out} = \mathsf{Com}(\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out}) \text{ for all } i \in [K], \text{ and } \sum_{j=1}^{L} \mathsf{cr}_j = \sum_{i=1}^{K} \mathsf{cr}_i^{out}.$$

The first property, in combination with the completeness of VHC, implies that $\mathsf{PVer}(\mathsf{stmt}, \pi) = 1$. The second property, in combination with $\mathsf{com}_j = \mathsf{Com}(\mathsf{amt}_j, \mathsf{cr}_j)$ for all $j \in [L]$, and the homomorphic property of VHC implies that

$$\begin{aligned}\sum_{j=1}^{L} \mathsf{com}_j &= \sum_{j=1}^{L} \mathsf{Com}\left(\mathsf{amt}_j, \mathsf{cr}_j\right) = \mathsf{Com}\left(\sum_{j=1}^{L} \mathsf{amt}_j, \sum_{j=1}^{L} \mathsf{cr}_j\right) \\ &= \mathsf{Com}\left(\sum_{i=1}^{K} \mathsf{amt}_i^{out}, \sum_{i=1}^{K} \mathsf{cr}_i^{out}\right) = \sum_{i=1}^{K} \mathsf{Com}\left(\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out}\right) = \sum_{i=1}^{K} \mathsf{com}_i^{out}.\end{aligned}$$

Therefore, event InvalidCom does not occur.

To analyze event InvalidSig, we define the following event:

- Event InvalidKey: This event occurs whenever an entry $(\mathsf{pk}, \mathsf{com}, \mathsf{amt}, \mathsf{cr}, \mathsf{sk})$ is added to $\mathsf{Owned}[\mathsf{ipk}]$ for some $\mathsf{ipk} \in \mathsf{Identities}$ and $(\mathsf{pk}, \mathsf{sk}) \notin \mathcal{KR}$.

Claim 4. $\Pr[\mathsf{InvalidSig}] \leq \Pr[\mathsf{InvalidKey}]$.

To show this claim, we first note that

$$\Pr[\mathsf{InvalidSig}] \leq \Pr[\mathsf{InvalidKey}] + \Pr[\mathsf{InvalidSig} \mid \neg\mathsf{InvalidKey}].$$

Therefore, it is sufficient to show that InvalidSig can not occur, assuming that InvalidKey does not occur. To see this, fix some $j \in [L]$ and note that oracle AddHonTrans would return $\perp$ and not generate Tx, if $(\mathsf{pk}_j, \mathsf{com}_j) \notin \mathsf{Ref}_j$. Also, recall that σ_j is computed via $\sigma_j \leftarrow \mathsf{LRS.Sig}(\mathsf{R}_j, \mathsf{sk}_j, \mathsf{auxsk}_j, \mathsf{m})$. Let $\hat{r}$ be the index of $(\mathsf{pk}_j, \mathsf{com}_j)$ in Ref_j. By definition of GenTx, we see that the $\hat{r}$th entry in R_j is $(\mathsf{pk}_j, \mathsf{auxpk}_j)$ for $\mathsf{auxpk}_j = \mathsf{ConvertPublic}(\mathsf{com}_j, \mathsf{Com}(\mathsf{amt}_j, \mathsf{cr}'_j))$ for some cr'_j. Further, we have $\mathsf{com}_j = \mathsf{Com}(\mathsf{amt}_j, \mathsf{cr}_j)$ by definition of algorithm StoreReceive. Also, we have $\mathsf{auxsk}_j = \mathsf{ConvertSecret}(\mathsf{cr}_j, \mathsf{cr}'_j)$. By completeness of KCS, this implies that $(\mathsf{auxpk}_j, \mathsf{auxsk}_j) \in \mathcal{AKR}$. Also, as InvalidKey does not occur, and oracle AddHonTrans checks that $\mathsf{Use}_j \in \mathsf{Owned}[\mathsf{ipk}]$, we know that $(\mathsf{pk}_j, \mathsf{sk}_j) \in \mathcal{KR}$. Thus, it follows by completeness of LRS that σ_j is a valid signature. This shows the claim.

Claim 5. *For every PPT algorithm $\mathcal{A}$, there exists a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\Pr[\mathsf{InvalidKey}] \leq Q_{ni} \cdot \mathsf{Adv}^{\mathsf{tr\text{-}sound}}_{\mathcal{B},\mathsf{KDS}}(\lambda).$$

To prove the claim, we describe algorithm $\mathcal{B}$ against the tracking soundness of KDS.

- $\mathcal{B}$ gets as input $(\mathsf{par}, \mathsf{ipk}^*, \mathsf{ivk}^*, \mathsf{isk}^*)$. It first samples $i^* \leftarrow\!\!\$\, [Q_{ni}]$.
- $\mathcal{B}$ simulates game $\mathbf{UNF}_{\mathsf{PTS}}$ honestly for $\mathcal{A}$, except for the i^*th query to oracle NewIdentity. In this query, it inserts ipk^* into Identities, and sets $\mathsf{ivk}[\mathsf{ipk}^*] := \mathsf{ivk}^*, \mathsf{isk}[\mathsf{ipk}^*] := \mathsf{isk}^*$ and returns ipk^*.
- Whenever $\mathcal{B}$ runs StoreReceive$(\mathsf{ipk}, \mathsf{pseed}, \mathsf{pk}, \mathsf{com}, \mathsf{tag}, \mathsf{ct})$ during this simulation for $\mathsf{ipk} = \mathsf{ipk}^*$, $\mathcal{B}$ checks if InvalidKey occurs in this execution of StoreReceive. Note that $\mathcal{B}$ can efficiently check that, as $\mathcal{KR}$ is efficiently decidable. If InvalidKey occurs, $\mathcal{B}$ outputs $\mathsf{pseed}, \mathsf{pk}, \mathsf{tag}$ and terminates.

Clearly, the running time of $\mathcal{B}$ dominated by the running time of $\mathcal{A}$, and $\mathcal{B}$ simulates game $\mathbf{UNF}_{\mathsf{PTS}}$ perfectly until it terminates, and $\mathcal{B}$ terminates exactly when InvalidKey occurs. Further, if $\mathcal{B}$ terminates, then it breaks tracking soundness of KDS. As $\mathcal{A}$ does not obtain any information about i^*, the claim follows.

Claim 6. *For every PPT algorithm $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}'$ and $\mathcal{B}''$ with $\mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}'')$ and*

$$\begin{aligned}\Pr[\mathsf{Linking} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}] \leq\ & \mathsf{Adv}^{\mathsf{spread}}_{\mathcal{B}',\mathsf{KDS}}(\lambda) + \Pr[\mathsf{InvalidKey}] \\ & + \mathsf{Adv}^{\mathsf{non\text{-}sland}}_{\mathcal{B}'',\mathsf{LRS},\mathsf{KDS}}(\lambda) + \frac{Q_{\mathsf{H}_{trm}}^2}{2^\ell}.\end{aligned}$$

To prove the claim, we define two more events.

- Event LinkingHon: This event occurs, if Linking occurs, and the linking signatures σ, σ' were both computed by the game during queries to AddHonTrans.
- Event LinkingMal: This event occurs, if Linking occurs, and the old signature σ is part of a transaction Tx submitted by $\mathcal{A}$ via oracle AddAdvTrans.

Also, LinkingHon and LinkingMal cover event Linking, and thus we have

$$\begin{aligned}\Pr[\mathsf{Linking} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}] \leq\ & \Pr[\mathsf{LinkingHon} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}] \\ & + \Pr[\mathsf{LinkingMal} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}].\end{aligned}$$

We bound these probability separately, starting with the former one. First, recall the event InvalidKey from before. Assuming $\neg\mathsf{InvalidSig} \wedge \neg\mathsf{InvalidKey}$, completeness of LRS implies that LinkingHon can only occur, if the two secret keys sk and sk' that are used to compute the linking signatures are the same. This leads to a straight-forward reduction $\mathcal{B}'$ that breaks the key spreadness of KDS, assuming $\neg\mathsf{InvalidSig} \wedge \neg\mathsf{InvalidKey} \wedge \neg\mathsf{PSeedColl} \wedge \mathsf{LinkingHon}$. We get

$$\Pr[\mathsf{LinkingHon} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}] \leq \mathsf{Adv}^{\mathsf{spread}}_{\mathcal{B}',\mathsf{KDS}}(\lambda) + \Pr[\mathsf{InvalidKey}].$$

To bound the probability of $\mathsf{LinkingMal} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}$ and finish the proof of the claim, we sketch algorithm $\mathcal{B}''$ against the non-slanderability of $(\mathsf{LRS}, \mathsf{KDS})$, that is successful whenever $\mathsf{LinkingMal} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}$ occurs.

- $\mathcal{B}''$ gets as input parameters par, and oracle access to oracles NewIdentity$'$, Corr, Sign. It simulates game $\mathbf{UNF}_{\mathsf{PTS}}$ for $\mathcal{A}$, except with the following changes.
- When $\mathcal{A}$ queries oracle NewIdentity, $\mathcal{B}''$ queries NewIdentity$'$ and obtains $(\mathsf{ipk}, \mathsf{ivk})$. It inserts ipk into $\mathsf{Identities}$ and sets $\mathsf{ivk}[\mathsf{ipk}] := \mathsf{ivk}$ as the real game would do. Then, it returns ipk.
- When $\mathcal{A}$ queries oracle FullCorr(ipk), $\mathcal{B}''$ queries Corr(ipk) to get $\mathsf{isk}[\mathsf{ipk}]$. It continues the simulation of FullCorr as the real game does.
- Whenever $\mathcal{B}''$ needs to compute a key sk via $\mathsf{DerSK}(\mathsf{isk}[\mathsf{ipk}], \mathsf{ok}, \mathsf{tag})$ during execution of algorithm $\mathsf{StoreReceive}$, and it holds that $\mathsf{corr}[\mathsf{ipk}] < 2$, i.e., $\mathcal{B}''$ does not have $\mathsf{isk}[\mathsf{ipk}]$, it postpones the computation of sk. However, note that it can still compute $\mathsf{pk} := \mathsf{DerPK}(\mathsf{ipk}, \mathsf{ok}, \mathsf{tag})$. Later, when $\mathcal{A}$ queries oracle AddHonTrans($\mathsf{ipk}, \cdot$) such that this sk is needed to sign in algorithm GenTx, $\mathcal{B}''$ uses oracle Sign to compute the signature. Concretely, it calls Sign($\mathsf{pk}, \mathsf{pseed}, \mathsf{tag}, \mathsf{R}^+, \mathsf{auxsk}, \mathsf{m}^+$), where $\mathsf{pseed}, \mathsf{tag}, \mathsf{R}^+, \mathsf{auxsk}, \mathsf{m}^+$ are chosen appropriately as in algorithm GenTx. If at this point event $\mathsf{LinkingMal} \wedge \neg\mathsf{InvalidSig}$ occurs (which can be efficiently verified), then we know that the resulting signature σ^+ is a new linking signature and there is an old linking signature σ^*, using the terminology introduced in event $\mathsf{Linking}$. We have $\mathsf{Link}(\sigma^+, \sigma^*) = 1$. Let $\mathsf{R}^*, \mathsf{m}^*$ be such that $\mathsf{LRS.Ver}(\mathsf{R}^*, \mathsf{m}^*, \sigma^*) = 1$ during the verification of the transaction that contained σ^*. In this case, $\mathcal{B}''$ terminates and outputs $(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$ as a forgery in the non-slanderability game, where σ^* is the old linking signature (with the terminology introduced in event $\mathsf{Linking}$), and $\mathsf{R}^*, \mathsf{m}^*$ are such that $\mathsf{LRS.Ver}(\mathsf{R}^*, \mathsf{m}^*, \sigma^*) = 1$

Clearly, the running time of $\mathcal{B}''$ dominated by the running time of $\mathcal{A}$, and $\mathcal{B}''$ simulates game $\mathbf{UNF}_{\mathsf{PTS}}$ perfectly until it terminates, and $\mathcal{B}''$ terminates exactly when $\mathsf{LinkingMal}$ occurs. Further, we claim that if $\mathcal{B}''$ terminates, then it breaks non-slanderability of $(\mathsf{LRS}, \mathsf{KDS})$ with overwhelming probability. As AddHonTrans only signs transactions for users that are not fully corrupted, we know that $\mathcal{B}''$ never queried Corr(ipk), where ipk is the first input of AddHonTrans that leads to termination of $\mathcal{B}''$. By definition of $\mathsf{LinkingMal} \wedge \neg\mathsf{InvalidSig}$, the only winning condition that is left to check is the freshness condition, i.e., R^* and m^* were never submitted as part of an input of a query to oracle Sign. Assume towards contradiction that this does not hold. Then especially m^* has been submitted to Sign, which is only called by $\mathcal{B}''$ when it generates and signs honest transactions. Let that transaction be $\mathsf{Tx}_{\mathrm{hon}}$. As we assume $\mathsf{LinkingMal}$ occurs, we know that $\mathsf{R}^*, \mathsf{m}^*, \sigma^*$ is associated to a maliciously generated transaction $\mathsf{Tx}_{\mathrm{mal}}$. As we assume $\neg\mathsf{PSeedColl}$, we know that the transactions $\mathsf{Tx}_{\mathrm{hon}}$ and $\mathsf{Tx}_{\mathrm{mal}}$ have different public seeds, which are part of the random oracle input H_{trm} that generated m^* (cf. Figure 2). This implies that there is a collision for random oracle H_{trm}. A union bound over all $Q^2_{\mathsf{H}_{trm}}$ pairs of queries to H_{trm} shows that the probability of such a collision for random oracle H_{trm} is at most $Q^2_{\mathsf{H}_{trm}}/2^\ell$. Therefore, we have

$$\Pr\left[\mathsf{LinkingMal} \wedge \neg\mathsf{InvalidSig} \wedge \neg\mathsf{PSeedColl}\right] \leq \mathsf{Adv}^{\mathsf{non\text{-}sland}}_{\mathcal{B}'',\mathsf{LRS},\mathsf{KDS}}(\lambda) + \frac{Q^2_{\mathsf{H}_{trm}}}{2^\ell}.$$

The claim follows, finishing the proof of the Lemma.

□

6.2 Bounding Winning Condition win-create

Before we can bound win-create, we need to define flow networks and prove a simple and well-known fact about these. We start with some definitions.

Definition 12 (Flow Network)**.** *A flow network is a tuple (G, s, t, f) with the following properties:*

- *$G = (V, E)$ be a connected directed graph, which has exactly one vertex $s \in V$ with in-degree zero and exactly one vertex $t \in V \setminus \{s\}$ with out-degree zero.*
- *$f \colon E \to \mathbb{R}_{\geq 0}$ is a function, such that*

$$\forall v \in V \setminus \{s, t\} : \sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e).$$

Definition 13 (*st*-Cut)**.** *Let $(G = (V, E), s, t, f)$ be a flow network. An st-cut (V_s, V_t) is a partition $V = V_s \cup V_t$ into disjoint sets V_s, V_t such that $s \in V_s$ and $t \in V_t$. Further, the value of such an st-cut with respect to f and G is defined as*

$$\mathsf{val}(V_s, V_t) := \sum_{e\in V_s \times V_t \cap E} f(e) - \sum_{e\in V_t \times V_s \cap E} f(e).$$

Next, we show that any *st*-cut has the same value.

Lemma 2. *Let $(G = (V, E), s, t, f)$ be a flow network. Then all st-cuts have the same value. In particular, for any st-cut (V_s, V_t), we have*

$$\mathsf{val}(V_s, V_t) = \mathsf{val}(V \setminus \{t\}, \{t\}) = \sum_{e=(v,t)\in E} f(e).$$

Proof. Let G, s, t, and f be as in the statement. We show that all *st*-cuts have the same value. To do that, it is sufficient to show the following claim: For any *st*-cut (V_s, V_t) with $|V_s| \geq 2$, i.e., there is some vertex $x \in V_s \setminus \{s\}$, we have

$$\mathsf{val}(V_s', V_t') = \mathsf{val}(V_s, V_t) \text{ for } V_s' := V_s \setminus \{x\}, V_t' := V_t \cup \{x\}.$$

To simplify notation, we define $[X, Y] := X \times Y \cap E$ for any vertex sets $X, Y \subset V$. To show this claim, fix such an *st*-cut (V_s, V_t) and such a vertex x. We have to compute

$$\mathsf{val}(V_s', V_t') = \sum_{e\in[V_s', V_t']} f(e) - \sum_{e\in[V_t', V_s']} f(e).$$

We can write the sets $[V_s', V_t']$ and $[V_t', V's]$ as follows.

$$[V_s', V_t'] = [V_s, V_t] \cup [V_s \setminus \{x\}, \{x\}] \setminus [\{x\}, V_t],$$
$$[V_t', V_s'] = [V_t, V_s] \cup [\{x\}, V_s \setminus \{x\}] \setminus [V_t, \{x\}],$$

where the unions are disjoint. Therefore, we can write

$$\sum_{e\in[V_s', V_t']} f(e) = \sum_{e\in[V_s, V_t]} f(e) + \sum_{e\in[V_s\setminus\{x\}, \{x\}]} f(e) - \sum_{e\in[\{x\}, V_t]} f(e),$$
$$\sum_{e\in[V_t', V's]} f(e) = \sum_{e\in[V_t, V_s]} f(e) + \sum_{e\in[\{x\}, V_s\setminus\{x\}]} f(e) - \sum_{e\in[V_t, \{x\}]} f(e).$$

In combination, we get that

$$\mathsf{val}(V_s', V_t') = \mathsf{val}(V_s, V_t) + \left(\sum_{e \in [V_s \setminus \{x\}, \{x\}]} f(e) + \sum_{e \in [V_t, \{x\}]} f(e) \right) - \left(\sum_{e \in [\{x\}, V_t]} f(e) + \sum_{e \in [\{x\}, V_s \setminus \{x\}]} f(e) \right).$$

Here, the second term is equal to $\sum_{e=(u,x) \in E} f(e)$, and the third term is equal to $\sum_{e=(x,w) \in E} f(e)$. By the assumption about f, we know that these are equal. Therefore, the second and third term cancel out, and we have $\mathsf{val}(V_s', V_t') = \mathsf{val}(V_s, V_t)$. □

Lemma 3. *Assume that the relation $\mathcal{KR}$ is unique, i.e., for each $\mathsf{pk} \in \mathcal{PK}$ there is at most one $\mathsf{sk} \in \mathcal{SK}$ such that $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{KR}$. For any PPT algorithm $\mathcal{A}$ there are PPT algorithms $\mathcal{B}_i$ with $\mathbf{T}(\mathcal{B}_i) \approx \mathbf{T}(\mathcal{A})$ for $i \in [7]$ and*

$$\begin{aligned} \Pr[\mathsf{win\text{-}create} = 1] \leq \quad & \frac{Q^2_{\mathsf{H}_{trm}}}{2^\ell} + \mathsf{Adv}^{\mathsf{c\text{-}kn\text{-}sound}}_{\mathcal{B}_1, \mathsf{Ext}_{\mathsf{VHC}}, \mathsf{VHC}}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{bind}}_{\mathcal{B}_2, \mathsf{VHC}}(\lambda) \\ & + \mathsf{Adv}^{\mathsf{k\text{-}kn\text{-}sound}}_{\mathcal{B}_3, \mathsf{Ext}_{\mathsf{LRS}}, \mathsf{LRS}, \mathsf{KDS}}(\lambda) + Q_{ni} \cdot \mathsf{Adv}^{\mathsf{tr\text{-}sound}}_{\mathcal{B}_4, \mathsf{KDS}}(\lambda) + \mathsf{Adv}^{\mathsf{kn\text{-}link}}_{\mathcal{B}_5, \mathsf{Ext}_{\mathsf{LRS}}, \mathsf{LRS}, \mathsf{KDS}}(\lambda) \\ & + \mathsf{Adv}^{\mathsf{conv\text{-}sound}}_{\mathcal{B}_6, \mathsf{Translate}, \mathsf{KCS}, \mathsf{VHC}}(\lambda) + \mathsf{Adv}^{\mathsf{k\text{-}ow}}_{\mathcal{B}_7, \mathsf{LRS}, \mathsf{KDS}}(\lambda). \end{aligned}$$

Proof. Let $\mathcal{A}$ be a PPT algorithm as in the statement. To prove the lemma, we give a sequence of games $\mathbf{G}_i$.

Game $\mathbf{G}_0$: We start with game $\mathbf{G}_0$, which is defined exactly as $\mathbf{UNF}^{\mathcal{A}}_{\mathsf{PTS}}$, but only outputs 1 if $\mathsf{win\text{-}create} = 1$ holds, and introduces an additional abort conditions. Namely, it aborts if two different transaction sign the same message. We can easily bound the probability of this event. To this end, we first note that no two transactions share the same public seed pseed. The message that is signed by a transaction is the hash of a string that contains these public seeds. Therefore, two transactions only sign the same message if a collision for random oracle H_{trm} occurs. A union bound over all $Q^2_{\mathsf{H}_{trm}}$ pairs of queries to H_{trm} shows that the probability of such a collision for random oracle H_{trm} is at most $Q^2_{\mathsf{H}_{trm}}/2^\ell$. As a consequence, we can assume that each transaction signs a different message. We have

$$\Pr[\mathsf{win\text{-}create} = 1] \leq \frac{Q^2_{\mathsf{H}_{trm}}}{2^\ell} + \Pr[\mathbf{G}_0 \to 1].$$

Game $\mathbf{G}_1$: Game $\mathbf{G}_1$ is as $\mathbf{G}_0$, but with the following change. The game holds maps $d[\cdot], r[\cdot]$, and $\mathsf{stmt}[\cdot], \mathsf{witn}[\cdot]$. At any time, the maps $d[\cdot], r[\cdot]$ map all outputs $o = (\mathsf{pk}, \mathsf{com}) \in \mathsf{Outputs}$ in the system to data $\mathsf{amt} = d[o] \in \mathcal{D}$ and commitment randomness $\mathsf{cr} = r[o] \in \mathcal{CR}$. Further, the maps $\mathsf{stmt}[\cdot], \mathsf{witn}[\cdot]$ keep track of the statement and witness used to compute the proof π for any transaction $\mathsf{Tx} \in \mathsf{TXs}$ in the system. Recall that outputs are added to the list $\mathsf{Outputs}$ as a consequence of oracles calls to NewHonSrc, NewSrc, AddHonTrans, AddAdvTrans. When this happens, the game fills the maps $d[\cdot], r[\cdot]$ and $\mathsf{stmt}[\cdot], \mathsf{witn}[\cdot]$ as follows:

- Queries NewHonSrc$(\mathsf{pk}, \mathsf{pseed}, \mathsf{com}, \mathsf{tag}, \mathsf{ct})$: Recall that the output $o = (\mathsf{pk}, \mathsf{com})$ is only added to $\mathsf{Outputs}$ if algorithm $\mathsf{StoreReceive}(\mathsf{ipk}, \mathsf{pseed}, \mathsf{pk}, \mathsf{com}, \mathsf{tag}, \mathsf{ct})$ returned a non-zero value for some user ipk with $\mathsf{ipk} \in \mathsf{Identities}$ with $\mathsf{corr}[\mathsf{ipk}] < 2$. Note that in this case, algorithm $\mathsf{StoreReceive}$ internally recovered amt and cr such that $\mathsf{com} = \mathsf{Com}(\mathsf{amt}, \mathsf{cr})$. The game sets $d[o] := \mathsf{amt}$ and $r[o] := \mathsf{cr}$.

- Queries $\textsc{NewSrc}(\mathsf{pk}, \mathsf{amt}, \mathsf{cr})$: Recall that here, the output $o = (\mathsf{pk}, \mathsf{com})$ for $\mathsf{com} = \mathsf{Com}(\mathsf{amt}, \mathsf{cr})$ is added to $\mathsf{Outputs}$. The game sets $d[o] := \mathsf{amt}$ and $r[o] := \mathsf{cr}$.

- Queries $\textsc{AddHonTrans}(\mathsf{ipk}, (\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i=1}^K, \mathsf{ISamp}, \mathsf{RSamp})$: Here, recall that new outputs are added to $\mathsf{Outputs}$ during the execution of algorithm $\mathsf{UpdateState}(\mathsf{Tx})$, where $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$ is the transaction that the oracle generated before using algorithm $\mathsf{GenTx}((\mathsf{Use}_j)_{j=1}^L, (\mathsf{Ref}_j)_{j=1}^L, (\mathsf{ipk}_i, \mathsf{amt}_i^{out})_{i=1}^K)$. Concretely, recall that this algorithm derives public keys pk_i^{out}, commitment randomness cr_i^{out} and commitments $\mathsf{com}_i^{out} = \mathsf{Com}(\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out})$ for each $i \in [K]$. Then it includes $o_i := (\mathsf{pk}_i^{out}, \mathsf{com}_i^{out})$ for each $i \in [K]$ in Out. Exactly these outputs are then added to $\mathsf{Outputs}$ by algorithm $\mathsf{UpdateState}(\mathsf{Tx})$. Therefore, the game can simply set $d[o_i] := \mathsf{amt}_i^{out}$ and $r[o_i] := \mathsf{cr}_i^{out}$ for all $i \in [K]$. Further, recall that algorithm GenTx as above computes a proof $\pi \leftarrow \mathsf{PProve}(\mathsf{stmt}, \mathsf{witn})$ for

$$\mathsf{stmt} := \left((\mathsf{com}_j)_{j=1}^L, (\mathsf{com}_i^{out})_{i=1}^K\right), \ \mathsf{witn} := \left((\mathsf{amt}_j^{in}, \mathsf{cr}_j)_{j=1}^L, (\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out})_{i=1}^K\right).$$

  The game sets $\mathsf{stmt}[\mathsf{Tx}] := \mathsf{stmt}$ and $\mathsf{witn}[\mathsf{Tx}] := \mathsf{witn}$.

- Queries $\textsc{AddAdvTrans}(\mathsf{Tx})$: As in the previous case, recall that outputs are added to list $\mathsf{Outputs}$ by algorithm $\mathsf{UpdateState}(\mathsf{Tx})$. Concretely, the transaction is parsed as $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$ and $\mathsf{In} = (\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_{j=1}^L$, and $\mathsf{Out} = \left(\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i\right)_{i=1}^K$. Then, for each $i \in [K]$, the output $o_i = (\mathsf{pk}_i^{out}, \mathsf{com}_i^{out})$ is added to $\mathsf{Outputs}$. Also, recall that algorithm $\mathsf{UpdateBalances}$ is called, which calls $\mathsf{StoreReceive}(\mathsf{ipk}, \cdot)$ for each such output o_i and each $\mathsf{ipk} \in \mathsf{Identities}$ with $\mathsf{corr}[\mathsf{ipk}] \neq 2$. If $\mathsf{StoreReceive}$ returns a non-zero value (i.e., output o is received by an honest user), then during its execution it computed $\overline{\mathsf{amt}}_i^{out}$ and $\overline{\mathsf{cr}}_i^{out}$ such that $\mathsf{Com}(\overline{\mathsf{amt}}_i^{out}, \overline{\mathsf{cr}}_i^{out}) = \mathsf{com}_i^{out}$. For these i, the game now sets $d[o_i] := \overline{\mathsf{amt}}_i^{out}$ and $r[o_i] := \overline{\mathsf{cr}}_i^{out}$. For the remaining i, we let the game extract $d[o_i] := \mathsf{amt}_i^{out}$ and $r[o_i] := \mathsf{cr}_i^{out}$ by setting

$$\mathsf{stmt} := \left((\mathsf{com}_j)_{j=1}^L, (\mathsf{com}_i^{out})_{i=1}^K\right)$$

  and running

$$\left((\mathsf{amt}_j^{in}, \mathsf{cr}_j)_{j=1}^L, (\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out})_{i=1}^K\right) = \mathsf{witn} \leftarrow \mathsf{Ext}_{\mathsf{VHC}}(\mathsf{stmt}, \pi),$$

  where $\mathsf{Ext}_{\mathsf{VHC}}$ is the commitment knowledge extractor of VHC. The game further sets $\mathsf{stmt}[\mathsf{Tx}] := \mathsf{stmt}$ and $\mathsf{witn}[\mathsf{Tx}] := \mathsf{witn}$.

We introduce the bad event $\mathsf{BadComExtr} = \bigvee_{\mathsf{Tx} \in \mathsf{TXs}} \mathsf{BadComExtr}_{\mathsf{Tx}}$ that occurs if the above extraction fails. More precisely:

- Event $\mathsf{BadComExtr}_{\mathsf{Tx}}$: This event occurs if $\mathsf{Com}(\mathsf{amt}_j^{in}, \mathsf{cr}_j) \neq \mathsf{com}_j$ for some j or $\mathsf{Com}(\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out}) \neq \mathsf{com}_i^{out}$ for some i, where we use the notation $\mathsf{stmt}[\mathsf{Tx}] = \left((\mathsf{com}_j)_j, (\mathsf{com}_i^{out})_i\right)$ and $\mathsf{witn}[\mathsf{Tx}] = \left((\mathsf{amt}_j^{in}, \mathsf{cr}_j)_j, (\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out})_i\right)$.

Clearly, if event $\mathsf{BadComExtr}$ does not occur, we have

$$\forall o = (\mathsf{pk}, \mathsf{com}) \in \mathsf{Outputs} : \mathsf{Com}(d[o], r[o]) = \mathsf{com}.$$

We let game $\mathbf{G}_1$ abort if event $\mathsf{BadComExtr}$ occurs. As long as this event does not occur, the view of $\mathcal{A}$ in $\mathbf{G}_1$ is identical to its view in $\mathbf{G}_0$. Therefore, we have

$$|\Pr[\mathbf{G}_0 \rightarrow 1] - \Pr[\mathbf{G}_1 \rightarrow 1]| \leq \Pr[\mathsf{BadComExtr}].$$

It is easy to see that the probability of event $\mathsf{BadComExtr}$ can be upper bounded using a reduction that breaks commitment knowledge soundness of VHC. Intuitively, the event can only occur for transactions added during a query AddAdvTrans(Tx). For this query, if it occurs, this means that algorithm $\mathsf{Ext}_{\mathsf{VHC}}$ did not extract a valid witness for the statement stmt. More formally, we can construct a reduction $\mathcal{B}_1$ against the ommitment knowledge soundness of VHC that gets access to an oracle O. The reduction simulates game $\mathbf{G}_0$ for $\mathcal{A}$. For any query AddAdvTrans(Tx), the reduction first verifies Tx. Note that this includes verifying the proof π that is included. If verification is successful, the reduction passes (stmt, π) to its oracle O, where stmt is as above. Then, it continues the simulation as in $\mathbf{G}_0$. Note that O sets the flag bad if and only if event $\mathsf{BadComExtr}$ in game $\mathbf{G}_1$ would occur. Further, the running time of $\mathcal{B}_1$ is dominated by running $\mathcal{A}$. Thus, we have

$$\Pr\left[\mathsf{BadComExtr}\right] \leq \mathsf{Adv}^{\mathsf{c\text{-}kn\text{-}sound}}_{\mathcal{B}_1,\mathsf{Ext}_{\mathsf{VHC}},\mathsf{VHC}}(\lambda).$$

Game $\mathbf{G}_2$: We introduce another bad event and let the game abort if it occurs. Intuitively, we want to ensure that for every transaction Tx, with pseudo outputs $(\mathsf{com}_j)_j$ and outputs $(\mathsf{com}^{out}_i)_i$, the sum of data is preserved from pseudo outputs to outptus. More formally, we define the bad event $\mathsf{BadSum} = \bigvee_{\mathsf{Tx}\in\mathsf{TXs}} \mathsf{BadSum}_{\mathsf{Tx}}$, where

- Event $\mathsf{BadSum}_{\mathsf{Tx}}$: This event occurs if for $\mathsf{witn}[\mathsf{Tx}] = \left(\left(\mathsf{amt}^{in}_j, \mathsf{cr}_j\right)_j, \left(\mathsf{amt}^{out}_i, \mathsf{cr}^{out}_i\right)_i\right)$ we have $\sum_j \mathsf{amt}^{in}_j \neq \sum_i \mathsf{amt}^{out}_i$.

We bound the probability of event BadSum using a reduction $\mathcal{B}_2$ that breaks binding of VHC. The reduction gets as input system parameters, and simulates game $\mathbf{G}_1$ for $\mathcal{A}$. Notice that if the event $\mathsf{BadSum}_{\mathsf{Tx}}$ occurs for some transaction Tx, then Tx has been verified by algorithm VerTx before. Especially, we have

$$\sum_j \mathsf{com}_j = \sum_i \mathsf{com}^{out}_i$$

for $\mathsf{stmt}[\mathsf{Tx}] = \left(\left(\mathsf{com}_j\right)_j, \left(\mathsf{com}^{out}_i\right)_i\right)$. By the homomorphic property of VHC, we therefore have (assuming $\mathsf{BadComExtr}_{\mathsf{Tx}}$ does not occur)

$$\mathsf{Com}\left(\sum_j \mathsf{amt}^{in}_j - \sum_i \mathsf{amt}^{out}_i, \sum_j \mathsf{cr}_j - \sum_i \mathsf{cr}^{out}_i\right) = \sum_j \mathsf{com}_j - \sum_i \mathsf{com}^{out}_i = 0.$$

by the homomorphic property of VHC. This means that if $\mathsf{BadSum}_{\mathsf{Tx}}$ occurs, the reduction $\mathcal{B}_2$ can compute a non-zero preimage

$$(\mathsf{amt}^*, \mathsf{cr}^*) := \left(\sum_j \mathsf{amt}^{in}_j - \sum_i \mathsf{amt}^{out}_i, \sum_j \mathsf{cr}_j - \sum_i \mathsf{cr}^{out}_i\right)$$

of $\mathsf{com}^* = 0$. It easily follows from the homomorphic properties of VHC that $(0,0)$ is also a preimage of $\mathsf{com}^* = 0$. Thus, reduction $\mathcal{B}_2$ can identify the transaction Tx for which $\mathsf{BadSum}_{\mathsf{Tx}}$ occurs, and then return $(\mathsf{amt}^*, \mathsf{cr}^*, 0, 0)$ as above to break binding. The running time of $\mathcal{B}_2$ is dominated by the running time of $\mathcal{A}$. It follows that we have

$$\left|\Pr\left[\mathbf{G}_1 \rightarrow 1\right] - \Pr\left[\mathbf{G}_2 \rightarrow 1\right]\right| \leq \Pr\left[\mathsf{BadSum}\right] \leq \mathsf{Adv}^{\mathsf{bind}}_{\mathcal{B}_2,\mathsf{VHC}}(\lambda).$$

Game $\mathbf{G}_3$: In this game, we introduce another map $I[\cdot]$, and a bad event related to it. We let the game abort if this event occurs. Concretely, the map $I[\cdot]$ maps transactions $\mathsf{Tx} \in \mathsf{TXs}$ to lists of

quintuples. Informally, for each input of a transaction, it holds the index of the secret key that is used to sign, the associated public key and commitment, and the auxiliary secret key that is used. Recall that transactions are added to list TXs during oracle queries AddHonTrans and AddAdvTrans. For these queries, the map $I[\cdot]$ is populated as follows:

- For queries to AddHonTrans: Let $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$ be a transaction that is added to TXs in AddHonTrans. Let $\mathsf{In} = (\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_j$. Recall that the oracle generates this transaction honestly using algorithm GenTx (cf. Figure 2). This algorithm takes as input lists $(\mathsf{Use}_j)_j$ and $(\mathsf{Ref}_j)_j$. Let j be such an input index. Then, we know that oracle AddHonTrans ensures that for $\mathsf{Use}_j = (\mathsf{pk}_j^{in}, \mathsf{com}_j^{in}, \mathsf{amt}_j^{in}, \mathsf{cr}_j^{in}, \mathsf{sk}_j)$, there is some index r_j^* such that $(\mathsf{pk}_j^{in}, \mathsf{com}_j^{in})$ is the r_j^*th entry in the list Ref_j. Further, recall that algorithm GenTx defines some cr_j and sets $\mathsf{auxsk}_j := \mathsf{ConvertSecret}(\mathsf{cr}_j^{in}, \mathsf{cr}_j)$. Then, it computes a signature σ_j using sk_j and auxsk_j. With this notation, the game now sets $I[\mathsf{Tx}] = ((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}_j^{in}, \mathsf{com}_j^{in}))_j$.
- For queries to AddAdvTrans: Let $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi)$ be a transaction that is added to TXs in AddAdvTrans. Let $\mathsf{In} = (\mathsf{Ref}_j, \mathsf{com}_j, \sigma_j)_j$. Recall that Tx is added to TXs only if it verifies. In this case, the game now first parses $\mathsf{Ref}_j = (\mathsf{pk}_{j,r}^{in}, \mathsf{com}_{j,r}^{in})_{r=1}^{|\mathsf{Ref}_j|}$ and sets m and $\mathsf{R}_j = \left(\mathsf{pk}_{j,r}^{in}, \mathsf{auxpk}_{j,r}\right)_{r=1}^{|\mathsf{Ref}_j|}$ for all j as in the verification of Tx (algorithm VerTx). To recall, $\mathsf{auxpk}_{j,r}$ is defined as $\mathsf{auxpk}_{j,r} = \mathsf{ConvertPublic}(\mathsf{com}_{j,r}^{in}, \mathsf{com}_j)$. Then, it runs $(r_j^*, \mathsf{sk}_j, \mathsf{auxsk}_j) \leftarrow \mathsf{Ext}_{\mathsf{LRS}}(\mathsf{R}_j, \mathsf{m}, \sigma_j)$, where $\mathsf{Ext}_{\mathsf{LRS}}$ is the knowledge extractor of LRS. The game now sets $I[\mathsf{Tx}] = ((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}_{j,r_j^*}^{in}, \mathsf{com}_{j,r_j^*}^{in}))_j$.

Further, we let the game abort if the following bad event occurs (with the notation as above):

- Event $\mathsf{BadSigExtr}$: This event occurs if we have $(\mathsf{auxpk}_{j,r_j^*}, \mathsf{auxsk}_j) \notin \mathcal{AKR}$ or $(\mathsf{pk}_{j,r_j^*}^{in}, \mathsf{sk}_j) \notin \mathcal{KR}$ for some transaction added during AddAdvTrans and some j, with the notation as above.

We can easily bound the probability of $\mathsf{BadSigExtr}$ using a natural reduction $\mathcal{B}_3$ against the key knowledge soundness of $(\mathsf{LRS}, \mathsf{KDS})$. Namely, the reduction gets as input system parameters and access to oracles NewIdentity, Corr, Sign, O. It simulates game $\mathbf{G}_3$ for $\mathcal{A}$, using oracle NewIdentity to generate new identities ipk and the corresponding ivk, oracle Corr to simulate oracle FullCorr, and oracle Sign to simulate signatures needed in AddHonTrans. Whenever a transaction is added to TXs in AddAdvTrans, it calls oracle O for each j instead of running $\mathsf{Ext}_{\mathsf{LRS}}$ as above. Note that event $\mathsf{BadSigExtr}$ occurs if and only if the oracle O sets the flag bad. Here, we use that each transaction signs a different message (cf. $\mathbf{G}_0$). The running time of $\mathcal{B}_3$ is dominated by the running time of $\mathcal{A}$. Thus, we have

$$|\Pr[\mathbf{G}_2 \rightarrow 1] - \Pr[\mathbf{G}_3 \rightarrow 1]| \leq \Pr[\mathsf{BadSigExtr}] \leq \mathsf{Adv}^{\mathsf{k\text{-}kn\text{-}sound}}_{\mathcal{B}_3, \mathsf{Ext}_{\mathsf{LRS}}, \mathsf{LRS}, \mathsf{KDS}}(\lambda).$$

Game $\mathbf{G}_4$: We define another bad event related to the knowledge extractor $\mathsf{Ext}_{\mathsf{LRS}}$, and let the game abort if it occurs. Namely, we let the game abort if the following event occurs:

- Event $\mathsf{DoubleSpend}$: This event occurs if some public key pk occurs twice as the fourth component of a quintuple contained in $I[\cdot]$. That is, either there are distinct $\mathsf{Tx}, \mathsf{Tx}' \in \mathsf{TXs}$ such that pk is the fourth component of a quintuple in the list $I[\mathsf{Tx}]$ and a quintuple in the list $I[\mathsf{Tx}']$, or there is a transaction $\mathsf{Tx} \in \mathsf{TXs}$ such that pk is the fourth component of two different entries in the list $I[\mathsf{Tx}]$.

We further partition the event $\mathsf{DoubleSpend}$ into events $\mathsf{DoubleSpendPureHon}$ and $\mathsf{DoubleSpendMal}$ with $\mathsf{DoubleSpend} = \mathsf{DoubleSpendPureHon} \vee \mathsf{DoubleSpendMal}$. Namely, we define

- Event $\mathsf{DoubleSpendPureHon}$: This event occurs if $\mathsf{DoubleSpend}$ and both of the two involved transactions (which could be the same) are generated by oracle AddHonTrans.
- Event $\mathsf{DoubleSpendMal}$: This event occurs if $\mathsf{DoubleSpend}$ occurs and $\mathsf{DoubleSpendPureHon}$ does not, i.e., at least one of the two involved transactions is added via AddAdvTrans.

We informally sketch how to bound event $\mathsf{DoubleSpendPureHon}$. First, we can use tracking soundness of KDS (as in Lemma 1) to argue that with high probability, all honest users derive valid secret keys. By uniqueness of the key relation $\mathcal{KR}$, if $\mathsf{DoubleSpendPureHon}$ occurs, then the same secret key is used to compute two different signatures. By completeness of LRS these link, and thus one of the transactions would have been rejected. A more formal argument similar to this can be found in the proof of Lemma 1. We have a reduction $\mathcal{B}_4$ with

$$\Pr\left[\mathsf{DoubleSpendPureHon}\right] \leq Q_{ni} \cdot \mathsf{Adv}^{\mathsf{tr\text{-}sound}}_{\mathcal{B}_4,\mathsf{KDS}}(\lambda).$$

The probability of $\mathsf{DoubleSpendMal}$ can be upper bounded using knowledge linkability of LRS. Namely, a reduction $\mathcal{B}_5$ gets as input system parameters and access to oracles NewIdentity, Corr, Sign, O, and is as reduction $\mathcal{B}_3$ above. The key insight is that if event $\mathsf{DoubleSpendMal}$ occurs, then the associated signatures are valid and do not link, because the involved transactions are valid. As at least one of the involved transactions is added via AddAdvTrans, oracle O is called. Therefore, oracle O sets the flag bad. We have

$$\Pr\left[\mathsf{DoubleSpendMal}\right] \leq \mathsf{Adv}^{\mathsf{kn\text{-}link}}_{\mathcal{B}_5,\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda)$$

and

$$\left|\Pr\left[\mathbf{G}_3 \rightarrow 1\right] - \Pr\left[\mathbf{G}_4 \rightarrow 1\right]\right| \leq Q_{ni} \cdot \mathsf{Adv}^{\mathsf{tr\text{-}sound}}_{\mathcal{B}_4,\mathsf{KDS}}(\lambda) + \mathsf{Adv}^{\mathsf{kn\text{-}link}}_{\mathcal{B}_5,\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda).$$

Game $\mathbf{G}_5$: In this game, we extend the map $I[\cdot]$ and let the game abort if some bad event related to this occurs. Let $\mathsf{Translate}$ be the conversion translator of $(\mathsf{KCS}, \mathsf{VHC})$. Recall that in game $\mathbf{G}_4$, we inserted lists of quintuples into the map $I[\cdot]$. That is, for a transaction Tx game $\mathbf{G}_4$ defined $I[\mathsf{Tx}] := ((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}^{in}_{j,r_j^*}, \mathsf{com}^{in}_{j,r_j^*}))_j$ at some point during the game. Now, in game $\mathbf{G}_5$, we instead insert lists of 7-tuples. Namely, whenever $\mathbf{G}_4$ would insert a list of quintuples $((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}^{in}_{j,r_j^*}, \mathsf{com}^{in}_{j,r_j^*}))_j$, the game $\mathbf{G}_5$ first retrieves the entries

$$\left((\mathsf{com}_j)_j, \left(\mathsf{com}_i^{out}\right)_i\right) := \mathsf{stmt}[\mathsf{Tx}], \quad \left(\left(\mathsf{amt}_j^{in}, \mathsf{cr}_j\right)_j, \left(\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out}\right)_i\right) := \mathsf{witn}[\mathsf{Tx}].$$

Then, for each j, it computes

$$\mathsf{cr}_j^* \leftarrow \mathsf{Translate}(\mathsf{amt}_j^{in}, \mathsf{cr}_j, \mathsf{com}^{in}_{j,r_j^*}, \mathsf{com}_j, \mathsf{auxpk}_{j,r_j^*}, \mathsf{auxsk}_j).$$

Finally, it defines $I[\mathsf{Tx}] := ((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}^{in}_{j,r_j^*}, \mathsf{com}^{in}_{j,r_j^*}, \mathsf{amt}_j^{in}, \mathsf{cr}_j^*))_j$. Further, it aborts if at any point, the following bad event occurs:

- Event $\mathsf{BadConv}$: This event occurs if we have $\mathsf{Com}(\mathsf{amt}_j^{in}, \mathsf{cr}_j^*) \neq \mathsf{com}^{in}_{j,r_j^*}$ for some transaction and some j, with the notation as above.

We can easily bound the probability of event $\mathsf{BadConv}$ using a straight-forward reduction $\mathcal{B}_6$ from conversion soundness of $(\mathsf{KCS}, \mathsf{VHC})$. The reduction gets as input system parameters, simulates game $\mathbf{G}_5$ for $\mathcal{A}$, and outputs $(\mathsf{amt}_j^{in}, \mathsf{cr}_j, \mathsf{com}^{in}_{j,r_j^*}, \mathsf{com}_j, \mathsf{auxpk}_{j,r_j^*}, \mathsf{auxsk}_j)$ as soon as event $\mathsf{BadConv}$ occurs. The running time of $\mathcal{B}_6$ is dominated by the running time of $\mathcal{A}$, and we have

$$\left|\Pr\left[\mathbf{G}_4 \rightarrow 1\right] - \Pr\left[\mathbf{G}_5 \rightarrow 1\right]\right| \leq \Pr\left[\mathsf{BadConv}\right] \leq \mathsf{Adv}^{\mathsf{conv\text{-}sound}}_{\mathcal{B}_6,\mathsf{Translate},\mathsf{KCS},\mathsf{VHC}}(\lambda).$$

Game $\mathbf{G}_6$: We introduce another bad event Steal and let the game abort if it occurs. Intuitively, it occurs if the adversary spends (according to map I) an output that is owned by an honest user. Formally, the event is defined as follows:

- Event Steal: This event occurs if a transaction Tx is added to TXs during a query to oracle AddAdvTrans, such that $I[\mathsf{Tx}] := ((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}^{in}_{j,r_j^*}, \mathsf{com}^{in}_{j,r_j^*}, \mathsf{amt}^{in}_j, \mathsf{cr}^*_j))_j$ and at that time there is an $\mathsf{ipk}^* \in \mathsf{Identities}$ with $\mathsf{corr}[\mathsf{ipk}^*] \neq 2$ such that an entry of the form $(\mathsf{pk}^{in}_{j,r_j^*}, \mathsf{com}^{in}_{j,r_j^*}, \cdot, \cdot, \cdot)$ is in $\mathsf{Owned}[\mathsf{ipk}^*]$.

We bound the probability of Steal by sketching a reduction $\mathcal{B}_7$ agains the key onewayness of $(\mathsf{LRS}, \mathsf{KDS})$. The reduction is as follows:

- $\mathcal{B}_7$ gets as input parameters par, and oracle access to oracles NewIdentity$'$, Corr, Sign. It simulates game $\mathbf{G}_5$ for $\mathcal{A}$, except with the following changes.
- When $\mathcal{A}$ queries oracle NewIdentity, $\mathcal{B}_7$ queries NewIdentity$'$ and obtains $(\mathsf{ipk}, \mathsf{ivk})$. It inserts ipk into $\mathsf{Identities}$ and sets $\mathsf{ivk}[\mathsf{ipk}] := \mathsf{ivk}$ as the real game would do. Then, it returns ipk.
- When $\mathcal{A}$ queries oracle FullCorr(ipk), $\mathcal{B}_7$ queries Corr(ipk) to get $\mathsf{isk}[\mathsf{ipk}]$. It continues the simulation of FullCorr as game $\mathbf{G}_5$ does.
- Whenever $\mathcal{B}_7$ needs to compute a key sk via $\mathsf{DerSK}(\mathsf{isk}[\mathsf{ipk}], \mathsf{ok}, \mathsf{tag})$ during execution of algorithm $\mathsf{StoreReceive}$, and it holds that $\mathsf{corr}[\mathsf{ipk}] < 2$, i.e., $\mathcal{B}_6$ does not have $\mathsf{isk}[\mathsf{ipk}]$, it postpones the computation of sk. However, note that it can still compute $\mathsf{pk} := \mathsf{DerPK}(\mathsf{ipk}, \mathsf{ok}, \mathsf{tag})$. Later, when $\mathcal{A}$ queries oracle AddHonTrans($\mathsf{ipk}, \cdot$) such that this sk is needed to sign in algorithm GenTx, the reduction uses oracle Sign to compute the signature. Concretely, it calls Sign($\mathsf{pk}, \mathsf{pseed}, \mathsf{tag}, \mathsf{R}, \mathsf{auxsk}, \mathsf{m}$), where $\mathsf{pseed}, \mathsf{tag}, \mathsf{R}^+, \mathsf{m}^+$ are chosen appropriately as in algorithm GenTx. As soon as event Steal occurs for some ipk^*, $\mathcal{B}_7$ terminates and outputs $\mathsf{ipk}^*, \mathsf{pk}^*, \mathsf{pseed}^*, \mathsf{tag}^*, \mathsf{sk}^*$, where $\mathsf{pk}^* := \mathsf{pk}^{in}_{j,r_j^*}$ as in the definition of Steal, pseed^* and tag^* were part of the transaction in which ipk^* received the output o^* containing pk^*, and $\mathsf{sk}^* := \mathsf{sk}_j$ as in the definition of Steal.

Clearly, the running time of $\mathcal{B}_7$ is dominated by the running time of $\mathcal{A}$. Further, $\mathcal{B}_7$ simulates $\mathbf{G}_5$ perfectly[5] Further, as we can assume that event $\mathsf{BadSigExtr}$ does not occur, we have $(\mathsf{pk}^*, \mathsf{sk}^*) \in \mathcal{KR}$, and we get

$$|\Pr[\mathbf{G}_5 \to 1] - \Pr[\mathbf{G}_6 \to 1]| \leq \Pr[\mathsf{Steal}] \leq \mathsf{Adv}^{\mathsf{k\text{-}ow}}_{\mathcal{B}_7,\mathsf{LRS},\mathsf{KDS}}(\lambda).$$

Game $\mathbf{G}_7$: We introduce a final bad event $\mathsf{BadBind} = \bigvee_{\mathsf{Tx} \in \mathsf{TXs}} \mathsf{BadBind}_{\mathsf{Tx}}$ and let the game abort if it occurs. The event is defined as follows:

- Event $\mathsf{BadBind}_{\mathsf{Tx}}$: This event occurs if we have $(\mathsf{amt}^{in}_j, \mathsf{cr}^*_j) \neq (d[o_j], r[o_j])$ for $I[\mathsf{Tx}] = ((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}^{in}_{j,r_j^*}, \mathsf{com}^{in}_{j,r_j^*}, \mathsf{amt}^{in}_j, \mathsf{cr}^*_j))_j$, some j and $o_j := (\mathsf{pk}^{in}_{j,r_j^*}, \mathsf{com}^{in}_{j,r_j^*})$.

We bound the probability of event $\mathsf{BadBind}$ using a reduction $\mathcal{B}_8$ against the binding property of VHC. Namely, assume this event occurs in $\mathbf{G}_6$ for some transaction Tx. Then, we have

$$\mathsf{Com}(\mathsf{amt}^{in}_j, \mathsf{cr}^*_j) = \mathsf{com}^{in}_{j,r_j^*},$$

[5]Note that in the AGM, it is important that $\mathsf{Ext}_{\mathsf{LRS}}$ receives the correct format of representations. One can observe that this holds, because the key knowledge soundness game provided exactly the same group elements to $\mathcal{A}$ as $\mathbf{G}_5$ does. This is the reason why the key knowledge soundness game provides more oracles than just oracle O.

as we can assume that event BadConv does not occur. Further, as we can assume that event BadComExtr does not occur, we have

$$\mathsf{Com}(d[o_j], r[o_j]) = \mathsf{com}^{in}_{j,r^*_j} \text{ for } o_j = (\mathsf{pk}^{in}_{j,r^*_j}, \mathsf{com}^{in}_{j,r^*_j}).$$

Thus, the reduction can simply output $\mathsf{amt}^{in}_j, \mathsf{cr}^*_j, d[o_j], r[o_j]$ as soon as event BadBind occurs. The running time is dominated by the running time of $\mathcal{A}$, and we have

$$|\Pr[\mathbf{G}_6 \to 1] - \Pr[\mathbf{G}_7 \to 1]| \leq \Pr[\mathsf{BadBind}] \leq \mathsf{Adv}^{\mathsf{bind}}_{\mathcal{B}_8,\mathsf{VHC}}(\lambda).$$

We finish the proof using a purely combinatorial argument. Namely, we show that $\mathbf{G}_7$ never outputs 1. To do that, we define a graph, show that we obtain a flow network from it, and use the Lemma about st-cuts to finish the proof. Concretely, we start with an empty graph G and a cut $H = \emptyset, M = \emptyset$. Then, throughout the game, we add vertices and edges to G, and define the function $f\colon E \to \mathbb{R}_{\geq 0}$. Roughly, the resulting flow network and the cut has the following structure:

- *Vertices.* The graph contains dedicated vertices s, t. For each transaction $\mathsf{Tx} \in \mathsf{TXs}$ in the system, it contains a vertex v_{Tx}. For each output $o = (\mathsf{pk}, \mathsf{com}) \in \mathsf{Outputs}$ in the system, it contains a vertex v_o.
- *Edges.* If $o \in \mathsf{Outputs}$ is an output of a transaction TXs, then there is an edge e from v_{Tx} to v_o. If $o \in \mathsf{Outputs}$ is used as an input of a transaction TXs, then there is an edge e from v_o to v_{Tx}. In both cases, we set $f(e)$ to be $d[o]$. There are edges from s to all source outputs (i.e., outputs created by oracles NewHonSrc, NewSrc), and edges from all unused outputs to t. These edges are also labeled according to $d[\cdot]$.

Precisely, $(G = (V, E), s, t, f)$ is constructed during the game as follows:

- *Initialization.* We set $G = (V, E)$ with $V = \{s, t\}$ and $E = \emptyset$. Here, s, t are the dedicated source and sink of the flow network. Also, we initiate an st-cut (H, M), with $H = \{s\}$ and $M = \{t\}$.
- *Invariant.* For each output $o \in \mathsf{Outputs}$ with vertex v_o, the following invariant is ensured by appropriately adding and removing edges: There is an edge $e = (v_o, t)$ if and only if there is no edge $e = (v_o, v_{\mathsf{Tx}})$ to a transaction vertex for some $\mathsf{Tx} \in \mathsf{TXs}$. In both cases, $f(e) := d[o]$.
- *New Sources.* For queries of the form NewHonSrc($\mathsf{pk}, \mathsf{pseed}, \mathsf{com}, \mathsf{tag}, \mathsf{ct}$) or NewSrc($\mathsf{pk}$, $\mathsf{amt}, \mathsf{cr}$), if an output $o = (\mathsf{pk}, \mathsf{com})$ is inserted into $\mathsf{Outputs}$ in such a query, then a vertex v_o is added to V. Further, an edge $e = (s, v_o)$ with $f(e) := d[o]$ is added. In the special case that o already was in $\mathsf{Outputs}$ and thus the vertex v_o already existed, a new dummy vertex is inserted (with a similar edge, also satisfying above invariant). In both cases, the new vertex is inserted into partition H (resp. M) in oracle NewHonSrc (resp. NewSrc).
- *New Transactions.* For queries AddAdvTrans(Tx) and AddHonTrans($\mathsf{ipk}, (\mathsf{ipk}_i, \mathsf{amt}^{out}_i)_{i\in[K]}$, $\mathsf{ISamp}, \mathsf{RSamp}$), recall that a transaction Tx may be added to list TXs by algorithm UpdateState. In this case, a vertex v_{Tx} is added to V. For queries of oracle AddHonTrans (resp. AddAdvTrans), this vertex is inserted into partition H (resp. M). Next, we consider outputs and inputs. First, for each output $o = (\mathsf{pk}, \mathsf{com})$ that is inserted into list $\mathsf{Outputs}$ (also by algorithm UpdateState), a vertex v_o is added to V, in combination with an edge $e = (v_{\mathsf{Tx}}, v_o)$ with $f(e) := d[o]$. Also, recall that algorithm UpdateBalances is called, which calls StoreReceive($\mathsf{ipk}, \cdot$) for each such output o and each $\mathsf{ipk} \in \mathsf{Identities}$ with $\mathsf{corr}[\mathsf{ipk}] \neq 2$. If StoreReceive returns a non-zero value (i.e., output o is received by an honest user), then

we insert v_o into partition H. Otherwise, it is inserted into partition M. In the special case that o already was in $\mathsf{Outputs}$ and thus the vertex v_o already existed, a new dummy vertex is inserted instead of v_o (with a similar edge, in the same partition, and also satisfying above invariant). Second, write $I[\mathsf{Tx}] = ((r_j^*, \mathsf{auxsk}_j, \mathsf{sk}_j, \mathsf{pk}_{j,r_j^*}^{in}, \mathsf{com}_{j,r_j^*}^{in}, \mathsf{amt}_j^{in}, \mathsf{cr}_j^*))_j$. Then, we set $o_j = (\mathsf{pk}_{j,r_j^*}^{in}, \mathsf{com}_{j,r_j^*}^{in})$ and add edges $e_j = (v_{o_j}, v_{\mathsf{Tx}})$ with $f(e_j) := d[o_j]$ for each j. Note that the validity of Tx ensures that v_{o_j} exists.

- *Corruptions.* For queries FullCorr(ipk) for which the oracle does not return $\perp$, we consider all outputs $o = (\mathsf{pk}, \mathsf{com}) \in \mathsf{Outputs}$ with $(\mathsf{pk}, \mathsf{com}, \cdot, \cdot, \cdot) \in \mathsf{Owned}[\mathsf{ipk}]$. For each such output, we do the following: A new vertex v_o' is added to V and inserted into partition H. The vertex v_o is inserted into partition M. All ingoing edges $e = (w, v_o)$ for $w \in V$ are replaced by edges $e' = (w, v_o')$ (while setting $f(e')$ to the value of $f(e)$ before). Further, an edge $\bar{e} = (v_o', v_o)$ is added to E, with $f(\bar{e}) := d[o]$.

We show that (G, s, t, f) indeed is a flow network in Lemma 4. Next, we show how the value of the st-cut (H, M) is related to the winning condition of $\mathbf{G}_7$. This will allow us to finish the proof.

Claim 7. *As long as $\boldsymbol{G}_7$ does not abort, we have*

$$\sum_{e \in H \times M \cap E} f(e) \leq \mathsf{received} + \sum_{e \in H \times \{t\} \cap E} f(e).$$

To show the claim, it is sufficient to observe that for each edge $e \in H \times M \cap E$ which is not of the form $e = (\cdot, t)$ that is added to G during the game, the game also increases the value $\mathsf{received}$ by $f(e)$. Indeed, edges from H to M belong to one of the following classes:

- Edges $e = (v_o', v_o)$ for some output $o \in \mathsf{Outputs}$: Such edges are added when oracle FullCorr(ipk) is called. By definition of this oracle, for each such edge the value $\mathsf{received}$ is increased by $d[o] = f(e)$.
- Edges $e = (s, v_o)$ for some output $o \in \mathsf{Outputs}$ (or edges e from s to a dummy): Such edges are added on an oracle query NewSrc($\mathsf{pk}, \mathsf{amt}, \mathsf{cr}$). By definition of this oracle, variable $\mathsf{received}$ is increased by $\mathsf{amt} = d[o] = f(e)$.
- Edges $e = (v_{\mathsf{Tx}}, v_o)$ for some transaction $\mathsf{Tx} \in \mathsf{TXs}$ and some output $o \in \mathsf{Outputs}$. For such edges, we know that Tx was added to TXs during a query of oracle AddHonTrans (because $v_{\mathsf{Tx}} \in H$), and o was added to $\mathsf{Outputs}$ in the same query. One can observe that because $v_o \in M$, the output o was not received by an honest user, in which case the value $\mathsf{received}$ is increased by $d[o]$ in oracle AddHonTrans.
- Edges $e = (v_o, v_{\mathsf{Tx}})$ for some transaction $\mathsf{Tx} \in \mathsf{TXs}$ and some output $o \in \mathsf{Outputs}$: Such edges can not exists for $v_o \in H$ and $v_{\mathsf{Tx}} \in M$, as we assume that event Steal does not occur.

Claim 8. *As long as $\boldsymbol{G}_7$ does not abort, we have*

$$\sum_{e \in M \times H \cap E} f(e) \geq \mathsf{spent}.$$

To prove this claim, it is sufficient to notice that whenever the variable spent is encreased by some amount in the game, then new edges e_i from M to H are added such that $\sum_i f(e_i)$ equals that amount. Indeed, the variable spent is only increased in oracle AddAdvTrans(Tx). It is increased by

the amount that honest users (formally, $\mathsf{ipk} \in \mathsf{Identities}$ with $\mathsf{corr}[\mathsf{ipk}] \neq 2$) received. At the same time, for each such received output o_i, an edge e_i from v_{Tx} to $v_{o_i} \in H$ is added, with $f(e_i) = d[o_i] = \overline{\mathsf{amt}}_i^{out}$ (with the notation of $\mathbf{G}_1$). By definition of oracle AddAdvTrans, the variable $\overline{\mathsf{amt}}_i^{out}$ is increased by the sum of these $\overline{\mathsf{amt}}_i^{out}$, which proves the claim.

Claim 9. $\Pr[\boldsymbol{G}_7 \to 1] = 0$.

Assume that $\mathbf{G}_7$ does not abort. Then, using the previous claims, and Lemmas 2 and 4, we get

$$\begin{aligned}\sum_{e \in H \times \{t\} \cap E} f(e) + \mathsf{received} - \mathsf{spent} &\geq \sum_{e \in H \times M \cap E} f(e) - \sum_{e \in M \times H \cap E} f(e) \\ &= \mathsf{val}(H, M) = \mathsf{val}(V \setminus \{t\}, \{t\}) \\ &= \sum_{e=(v,t) \in E} f(e) \geq \sum_{e \in H \times \{t\} \cap E} f(e).\end{aligned}$$

Rearranging, this leads to $\mathsf{received} \geq \mathsf{spent}$. Therefore, the winning condition of $\mathbf{G}_7$ (i.e., $\mathsf{spent} > \mathsf{received}$) can never hold. □

Lemma 4. *With the notation from the proof of Lemma 3, as long as $\boldsymbol{G}_7$ does not abort, the tuple (G, s, t, f) is a flow network as defined in Definition 12.*

Proof. We show the properties of a flow network separately. For each of these, we assume that $\mathbf{G}_7$ does not abort.

Claim 10. *Vertex s is the only vertex with in-degree zero, and t is the only vertex with out-degree zero.*

Each transaction has both at least one output and at least one input. For each output $o \in \mathsf{Outputs}$, v_o has an ingoing edge (either from s, from some v_{Tx}, or from v'_o), and v_o has an outgoing edge (by the invariant, either to t or to some v_{Tx}). The same holds for dummy vertices. This shows the claim.

Claim 11. *For each $\mathsf{Tx} \in \mathsf{TXs}$, we have*

$$\sum_{e=(u,v_{\mathsf{Tx}}) \in E} f(e) = \sum_{e=(v_{\mathsf{Tx}},u) \in E} f(e).$$

Let $\mathsf{Tx} = (\mathsf{In}, \mathsf{Out}, \mathsf{pseed}, \pi) \in \mathsf{TXs}$ and $v_{\mathsf{Tx}} \in V$. Write $I[\mathsf{Tx}] = ((r_j^*, \mathsf{auxsk}_j, , \mathsf{sk}_j, \mathsf{pk}_{j,r_j^*}^{in}, \mathsf{com}_{j,r^*}^{in}, \mathsf{amt}_j^{in}, \mathsf{cr}_j^*))_j$, $o_j = (\mathsf{pk}_{j,r_j^*}^{in}, \mathsf{com}_{j,r_j^*}^{in})$ for each j. Further, write $\mathsf{witn}[\mathsf{Tx}] = \left(\left(\mathsf{amt}_j^{in}, \mathsf{cr}_j\right)_j, \left(\mathsf{amt}_i^{out}, \mathsf{cr}_i^{out}\right)_i\right)$ and $\mathsf{Out} = \left(\mathsf{pk}_i^{out}, \mathsf{com}_i^{out}, \mathsf{ct}_i, \mathsf{tag}_i\right)_{i \in [K]}$. Set $o'_i := (\mathsf{pk}_i^{out}, \mathsf{com}_i^{out})$ for all i. Then, by definition of f, we have

$$\sum_{e=(u,v_{\mathsf{Tx}}) \in E} f(e) = \sum_j d[o_j].$$

As we can assume that events $\mathsf{BadBind}_{\mathsf{Tx}}$ and $\mathsf{BadSum}_{\mathsf{Tx}}$ do not occur, we have

$$\sum_j d[o_j] = \sum_j \mathsf{amt}_j^{in} = \sum_i \mathsf{amt}_i^{out}.$$

By definition of $d[\cdot]$ and f, we have

$$\sum_i \mathsf{amt}_i^{out} = \sum_i d[o'_i] = \sum_{e=(v_{\mathsf{Tx}},u)\in E} f(e).$$

In combination, we showed the claim.

Claim 12. *For each* $o = (\mathsf{pk}, \mathsf{com}) \in \mathsf{Outputs}$*, we have*

$$\sum_{e=(u,v_o)\in E} f(e) = \sum_{e=(v_o,u)\in E} f(e)$$

The same holds true for each dummy vertex.

By definition, v_o has exactly one ingoing edge e. For this edge, it holds that $f(e) = d[o]$. Then, by the invariant, there are two cases to consider. Either, v_o has an outgoing edge to at least one transaction vertex v_{Tx}, or it has an outgoing edge e' to t and no other outgoing edge. In the latter case, we have $f(e') = d[o]$ by definition, and we are done. In the former case, by definition, all outgoing edges e' satisfy $f(e') = d[o]$. This means that we have to argue that there is at most one outgoing edge. Indeed, this follows directly from the assumption that event $\mathsf{DoubleSpend}$ does not occur.

Claim 13. *For each* $o = (\mathsf{pk}, \mathsf{com}) \in \mathsf{Outputs}$ *for which* v'_o *exists in* V*, we have*

$$\sum_{e=(u,v'_o)\in E} f(e) = \sum_{e=(v'_o,u)\in E} f(e)$$

By definition, vertex v'_o has one ingoing edge e (either from s or from some v_{Tx}), and one outgoing edge $\bar{e}$ (to v_o). They satisfy $f(e) = d[o] = f(\bar{e})$. □

7 Component Level Security Analysis

In this section, we present the components used in Monero, and their analysis. To find a specific part of the analysis, the reader may consult Table 1. We do not cover the Bulletproof+ [BBB+18, GT21, CHJ+22] component, as a detailed analysis of it in the algebraic group model would not fit the scope of this work.

7.1 The Components used in Monero

Let GGen be an algorithm that on input 1^λ outputs the description of a prime order group $\mathbb{G}$ of order $p \geq 2^\lambda$ with generator g. We assume that the description of $\mathbb{G}, g$, and p are given to all algorithms as system parameters par. Further, we assume random oracles $\mathsf{H}_p, \mathsf{H}_h \colon \{0,1\}^* \to \mathbb{G}, , \mathsf{H}_c \colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_a, \mathsf{H} \colon \{0,1\}^* \to \mathbb{Z}_p^2$, and $\mathsf{H}_{enc} \colon \{0,1\}^* \to \{0,1\}^\lambda$. We define the key relations $\mathcal{KR}, \mathcal{AKR}$ as

$$\mathcal{KR} := \{(\mathsf{pk}, \mathsf{sk}) \in \mathbb{G}^2 \mid \mathsf{pk} = g^{\mathsf{sk}}\}, \quad \mathcal{AKR} := \{(\mathsf{auxpk}, \mathsf{auxsk}) \in \mathbb{G}^2 \mid \mathsf{auxpk} = g^{\mathsf{auxsk}}\}.$$

Finally, the instantiations are given as follows.

- The key derivation scheme $\mathsf{KDS} = (\mathsf{GenID}, \mathsf{Encaps}, \mathsf{SendDecaps}, \mathsf{RecDecapsDerPK}, \mathsf{DerSK}, \mathsf{Track})$ is presented in Figure 7.
- The data encryption scheme $\mathsf{DE} = (\mathsf{Enc}, \mathsf{Dec})$ is given in Figure 8.
- The verifiable homomorphic commitment scheme $\mathsf{VHC} = (\mathsf{DerRand}, \mathsf{Com}, \mathsf{PProve}, \mathsf{PVer})$ is given in Figure 8, without specifying algorithms PProve and PVer. Algorithms PProve and PVer implement Bulletproofs+ [CHJ+22], and we do not give an analysis for them.
- The key conversion scheme $\mathsf{KCS} = (\mathsf{ConvertPublic}, \mathsf{ConvertSecret})$ and the linkable ring signature scheme $\mathsf{LRS} = (\mathsf{Sig}, \mathsf{Ver}, \mathsf{Link})$ are given in Figure 9.

7.2 Analysis of Key Derivation

Lemma 5. *The scheme* KDS *(given in Figure 7) satisfies tracking soundness. Concretely, for any algorithm* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{\mathsf{tr\text{-}sound}}_{\mathcal{A},\mathsf{KDS}}(\lambda) = 0.$$

Proof. Let $\mathcal{A}$ be an adversary against the tracking soundness of KDS. We first recall the tracking soundness game for the specific scheme at hand. First, keys $(\mathsf{ipk} = (K_v, K_s), \mathsf{ivk} = k_v, \mathsf{isk} = k_s)$ are generated, where $K_v = g^{k_v}, K_s = g^{k_s}$. Then, $\mathcal{A}$ gets $\mathsf{par}, \mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}$, and outputs $\mathsf{pseed} = R$,$\mathsf{pk} = K_o$, and tag. The game now derives $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed})$ and $\mathsf{sk} := \mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}, \mathsf{tag})$. Concretely, it computes $\mathsf{sk} = k_o = \mathsf{H}(\mathsf{ok}, \mathsf{tag}) + k_s$. The game outputs 1 if $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) = 1$, but $(\mathsf{pk}, \mathsf{sk}) \notin \mathcal{KR}$, i.e., $K_o \neq g^{k_o}$. By definition, $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag})$ outputs 1 if and only if $K_o = K_s \cdot g^{\mathsf{H}(\mathsf{ok},\mathsf{tag})}$. We claim that the game never outputs 1. To this end, assume that Track outputs 1. Then, note that

$$g^{k_o} = g^{k_s + \mathsf{H}(\mathsf{ok},\mathsf{tag})} = K_s \cdot g^{\mathsf{H}(\mathsf{ok},\mathsf{tag})} = K_o,$$

where the first equation follows from the definition of k_o, the second from the definition of K_s, and the third one is given assuming that Track outputs 1. Thus, we always have $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{KR}$. □

Alg GenID(par)
$k_s \leftarrow\!\!\$\ \mathbb{Z}_p$
$k_v \leftarrow\!\!\$\ \mathbb{Z}_p$
$\mathsf{ivk} := k_v, \mathsf{isk} := k_s$
$\mathsf{ipk} := (K_v, K_s) := (g^{k_v}, g^{k_s})$
return $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$

Alg DerPK(ipk, ok, tag)
parse $(K_v, K_s) := \mathsf{ipk}$
$K_o := g^{\mathsf{H}(\mathsf{ok},\mathsf{tag})} \cdot K_s$
return $\mathsf{pk} := K_o$

Alg Encaps(par)
$\mathsf{sseed} := r \leftarrow\!\!\$\ \mathbb{Z}_p$
$\mathsf{pseed} := R := g^r$
return $(\mathsf{pseed}, \mathsf{sseed})$

Alg RecDecaps(ivk, pseed)
parse $k_v := \mathsf{ivk}, R := \mathsf{pseed}$
return $\mathsf{ok} := R^{k_v}$

Alg SendDecaps(ipk, sseed)
parse $(K_v, K_s) := \mathsf{ipk}, r := \mathsf{sseed}$
return $\mathsf{ok} := K_v^r$

Alg Track(ipk, ok, pk, tag)
parse $K_o := \mathsf{pk}, (K_v, K_s) := \mathsf{ipk}$
if $K_o = K_s \cdot g^{\mathsf{H}(\mathsf{ok},\mathsf{tag})}$: **return** 1
return 0

Alg DerSK(isk, ok, tag)
parse $k_s := \mathsf{isk}$
$k_o := \mathsf{H}(\mathsf{ok}, \mathsf{tag}) + k_s$
return $\mathsf{sk} := k_o$

Figure 7: The key derivation scheme KDS = (GenID, Encaps, SendDecaps, RecDecapsDerPK, DerSK, Track) used in Monero.

Alg Enc(ok, tag, amt)
return $\mathsf{ct} := \mathsf{amt} \oplus \mathsf{H}_{enc}(\mathsf{ok}, \mathsf{tag})$

Alg Com(amt, cr)
$h := \mathsf{H}_h(g)$
return $\mathsf{com} := g^{\mathsf{cr}} \cdot h^{\mathsf{amt}}$

Alg Dec(ok, tag, ct)
return $\mathsf{amt} := \mathsf{ct} \oplus \mathsf{H}_{enc}(\mathsf{ok}, \mathsf{tag})$

Alg DerRand(ok, tag)
return $\mathsf{cr} := \mathsf{H}_{com}(\mathsf{ok}, \mathsf{tag})$

Figure 8: The data encryption scheme DE = (Enc, Dec) and the verifiable homomorphic commitment scheme VHC = (DerRand, Com, PProve, PVer) used in Monero. Algorithms PProve and PVer are implemented using Bulletproofs+ [CHJ+22], and are omitted here.

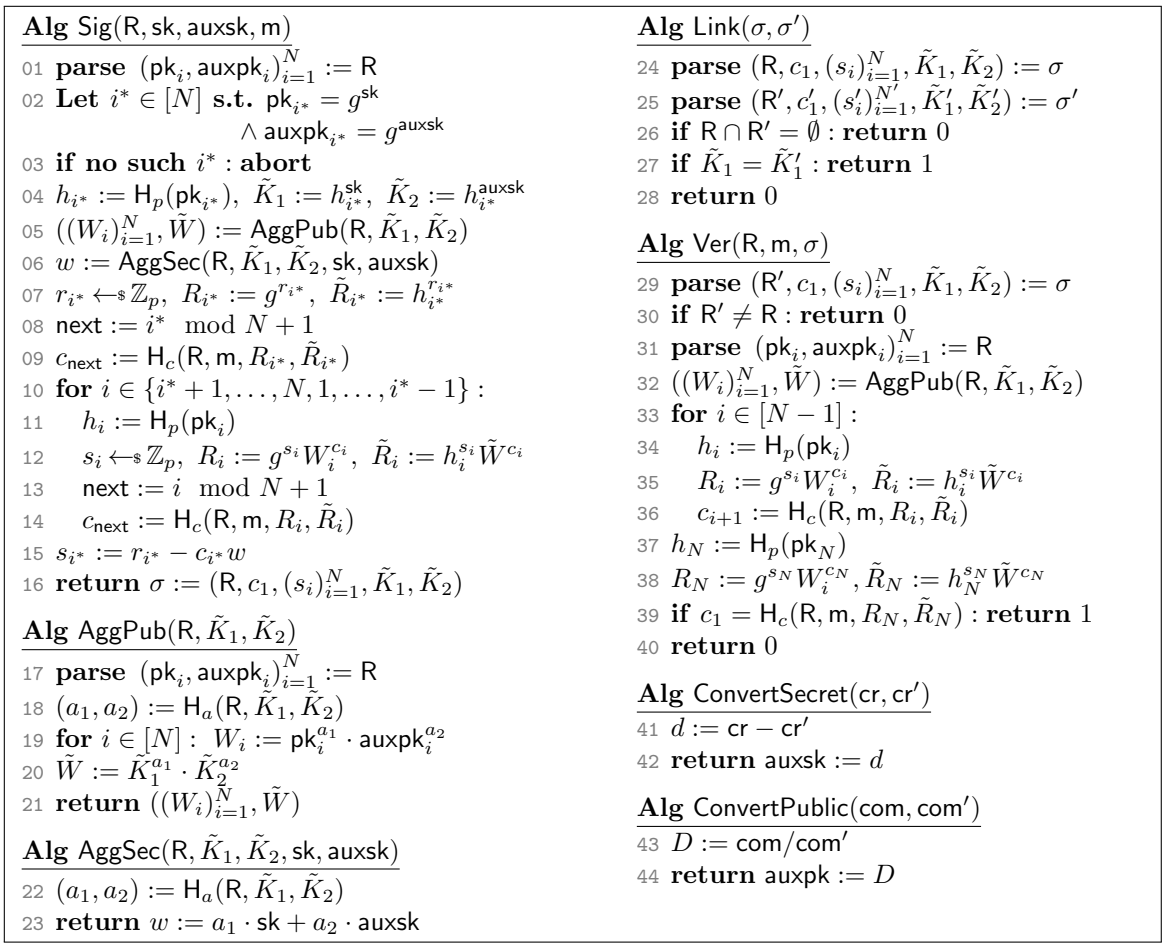

Figure 9: The two-dimensional linkable ring signature scheme $\mathsf{LRS} = (\mathsf{Sig}, \mathsf{Ver}, \mathsf{Link})$ (known as CLSAG) and the key conversion scheme $\mathsf{KCS} = (\mathsf{ConvertPublic}, \mathsf{ConvertSecret})$ used in Monero.

Lemma 6. *Let* $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$ *be a random oracle. Then,* KDS *(given in Figure 7) satisfies key spreadness. Concretely, for any algorithm* $\mathcal{A}$ *that makes at most* Q_{H}, Q_{ni} *queries to oracles* H, NewIdentity*, respectively, we have*

$$\mathsf{Adv}^{\mathsf{spread}}_{\mathcal{A},\mathsf{KDS}}(\lambda) \leq \frac{Q_{\mathsf{H}}^2}{p} + \frac{Q_{ni}}{p}.$$

Proof. Recall that in the key spreadness game, adversary $\mathcal{A}$ gets access to an oracle NewIdentity, that does not take any input, and outputs key triples $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}) \leftarrow \mathsf{GenID}(\mathsf{par})$. Then, $\mathcal{A}$ outputs a triple $\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}$, public seeds $\mathsf{pseed} = R \in \mathbb{G}$, $\mathsf{pseed}' = R' \in \mathbb{G}$ and tags $\mathsf{tag}, \mathsf{tag}' \in \mathbb{N}$. The adversary $\mathcal{A}$ wins, if $\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk}$ was output by the oracle NewIdentity before, $(R, \mathsf{tag}) \neq (R', \mathsf{tag}')$, and $\mathsf{sk} = \mathsf{sk}'$ for $\mathsf{sk} := \mathsf{DerSK}(\mathsf{isk}, \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}), \mathsf{tag})$ and $\mathsf{sk}' := \mathsf{DerSK}(\mathsf{isk}, \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}'), \mathsf{tag}')$. For the concrete scheme at hand, this means that $\mathsf{sk} = \mathsf{H}(R^{k_v}, \mathsf{tag}) + k_s$ and $\mathsf{sk}' = \mathsf{H}(R'^{k_v}, \mathsf{tag}') + k_s$, where $\mathsf{ivk} = k_v \in \mathbb{Z}_p$ and $\mathsf{isk} = k_s \in \mathbb{Z}_p$. Therefore, $\mathcal{A}$ can only win the game if $\mathsf{H}(R^{k_v}, \mathsf{tag}) = \mathsf{H}(R'^{k_v}, \mathsf{tag}')$. To rule this out, we define two bad events:

- Event Coll: This event occurs if there are two random oracle queries $x \neq x'$ during the game, such that $\mathsf{H}(x) = \mathsf{H}(x')$.
- Event Zero: This event occurs if for some query to NewIdentity that returns $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$, we have $\mathsf{ivk} = k_v = 0 \in \mathbb{Z}_p$.

Via a union bound over the (pairs of) hash queries, and the queries to NewIdentity, we get

$$\Pr[\mathsf{Coll}] \leq \frac{Q_{\mathsf{H}}^2}{p}, \quad \Pr[\mathsf{Zero}] \leq \frac{Q_{ni}}{p}.$$

Further, we define the event that the key spreadness game outputs 1 as Win. Then, have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{spread}}_{\mathcal{A},\mathsf{KDS}}(\lambda) &= \Pr[\mathsf{Win} \wedge \mathsf{Zero}] + \Pr[\mathsf{Win} \wedge \neg\mathsf{Zero}] \\ &\leq \Pr[\mathsf{Zero}] + \Pr[\mathsf{Win} \wedge \neg\mathsf{Zero}] \leq \Pr[\mathsf{Zero}] + \Pr[\mathsf{Coll}],\end{aligned}$$

where we used that the event $\mathsf{Win} \wedge \neg\mathsf{Zero}$ implies event Coll. This is because $k_v \neq 0$ and $R \neq R'$ implies $R^{k_v} \neq R'^{k_v}$. □

7.3 Analysis of the Verifiable Homomorphic Commitment

Lemma 7. *Let* $\mathsf{H}_h \colon \{0,1\}^* \to \mathbb{G}$ *be a random oracle. If the* DLOG *assumption holds relative to* GGen*, then* VHC *(given in Figure 8) is binding. Concretely, for any PPT algorithm* $\mathcal{A}$*, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *and*

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathcal{A},\mathsf{VHC}}(\lambda) \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda).$$

Proof. The proof is identical to the classical proof of the binding property of Pedersen commitment [Ped92]. Let $\mathcal{A}$ be an adversary against the binding property of VHC. We construct an algorithm $\mathcal{B}$ that solves DLOG in $\mathbb{G}$. Namely, $\mathcal{B}$ gets as input $g, h \in \mathbb{G}$. It uses g to define system parameters par,

and programs $\mathsf{H}_h(g) := h$. It runs $\mathcal{A}$ on input par. Assume that adversary $\mathcal{A}$ breaks binding, i.e., it outputs $(\mathsf{amt}, \mathsf{cr}, \mathsf{amt}', \mathsf{cr}')$ such that $\mathsf{amt} \neq \mathsf{amt}'$ but $\mathsf{Com}(\mathsf{amt}, \mathsf{cr}) = \mathsf{Com}(\mathsf{amt}', \mathsf{cr}')$. This means that

$$g^{\mathsf{cr}} \cdot h^{\mathsf{amt}} = \mathsf{Com}(\mathsf{amt}, \mathsf{cr}) = \mathsf{Com}(\mathsf{amt}', \mathsf{cr}') = g^{\mathsf{cr}'} \cdot h^{\mathsf{amt}'}.$$

Writing $h = g^x$, this implies

$$\mathsf{cr} + x \cdot \mathsf{amt} = \mathsf{cr}' + x \cdot \mathsf{amt}' \Longrightarrow x = \frac{\mathsf{cr} - \mathsf{cr}'}{\mathsf{amt}' - \mathsf{amt}},$$

where $\mathsf{amt}' - \mathsf{amt} \neq 0$. Algorithm $\mathcal{B}$ simply outputs $(\mathsf{cr} - \mathsf{cr}')/(\mathsf{amt}' - \mathsf{amt})$. This shows the claim. □

Lemma 8. *The pair* $(\mathsf{KCS}, \mathsf{VHC})$ *(given in Figures 8 and 9) satisfies conversion soundness. Concretely, there is a PPT algorithm* $\mathsf{Translate}$*, such that for every algorithm* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{\mathsf{conv\text{-}sound}}_{\mathcal{A},\mathsf{Translate},\mathsf{KCS},\mathsf{VHC}}(\lambda) = 0.$$

Proof. We define algorithm $\mathsf{Translate}$ as follows

$$\mathsf{Translate}(\mathsf{amt}, \mathsf{cr}', \mathsf{com}, \mathsf{com}', \mathsf{auxpk}, \mathsf{auxsk}) = \mathsf{auxsk} + \mathsf{cr}' \in \mathbb{Z}_p.$$

Clearly, $\mathsf{Translate}$ runs in polynomial time. Next, we show that for any PPT algorithm $\mathcal{A}$, the probability that the conversion soundness game (with respect to $\mathsf{Translate}$) outputs 1 is negligible. Recall that in this game, $\mathcal{A}$ gets parameters par, and outputs $\mathsf{amt}, \mathsf{cr}'$, commitments $\mathsf{com}, \mathsf{com}' \in \mathbb{G}$, and keys $\mathsf{auxpk} \in \mathbb{G}, \mathsf{auxsk} \in \mathbb{Z}_p$. By definition of the game and the concrete scheme at hand, if the game outputs 1, then the following must hold

$$\begin{aligned}
\mathsf{Com}(\mathsf{amt}, \mathsf{cr}') = \mathsf{com}', &\text{ i.e., } g^{\mathsf{cr}'} \cdot h^{\mathsf{amt}} = \mathsf{com}', \\
(\mathsf{auxpk}, \mathsf{auxsk}) \in \mathcal{AKR}, &\text{ i.e., } \mathsf{auxpk} = g^{\mathsf{auxsk}}, \\
\mathsf{ConvertPublic}(\mathsf{com}, \mathsf{com}') = \mathsf{auxpk}, &\text{ i.e., } \mathsf{auxpk} = \mathsf{com}/\mathsf{com}'.
\end{aligned}$$

Further, it must hold that $\mathsf{Com}(\mathsf{amt}, \mathsf{cr}) \neq \mathsf{com}$, or $\mathsf{ConvertSecret}(\mathsf{cr}, \mathsf{cr}') \neq \mathsf{auxsk}$ for cr output by $\mathsf{Translate}$, i.e., $\mathsf{cr} = \mathsf{auxsk} + \mathsf{cr}'$. The latter condition can not hold by definition of algorithm $\mathsf{ConvertSecret}$. Namely, we have

$$\mathsf{ConvertSecret}(\mathsf{cr}, \mathsf{cr}') = \mathsf{cr} - \mathsf{cr}' = \mathsf{auxsk} + \mathsf{cr}' - \mathsf{cr}' = \mathsf{auxsk}.$$

Next, we argue that the former can not hold. Namely, we show that $\mathsf{Com}(\mathsf{amt}, \mathsf{cr}) = \mathsf{com}$, i.e., $g^{\mathsf{cr}} \cdot h^{\mathsf{amt}} = \mathsf{com}$. Using the equations derived above, we have

$$g^{\mathsf{cr}} \cdot h^{\mathsf{amt}} = g^{\mathsf{auxsk}+\mathsf{cr}'} \cdot h^{\mathsf{amt}} = \mathsf{auxpk} \cdot g^{\mathsf{cr}'} \cdot h^{\mathsf{amt}} = \mathsf{auxpk} \cdot \mathsf{com}' = \mathsf{com}.$$

□

7.4 Analysis of the Ring Signature

Lemma 9. *Let* $\mathsf{H}_p \colon \{0,1\}^* \to \mathbb{G}, , \mathsf{H}_c \colon \{0,1\}^* \to \mathbb{Z}_p$ *be random oracles. If the* DLOG *assumption holds relative to* GGen*, then* $(\mathsf{LRS}, \mathsf{KDS})$ *(given in Figures 7 and 9) satisfies key onewayness. Concretely, for any PPT algorithm* $\mathcal{A}$ *that makes at most* $Q_S, Q_{ni}, Q_{\mathsf{H}_c}$ *queries to oracles* Sign, NewIdentity, H_c*, respectively, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *and*

$$\mathsf{Adv}^{\mathsf{k\text{-}ow}}_{\mathcal{A},\mathsf{LRS},\mathsf{KDS}}(\lambda) \leq \frac{Q_S Q_{\mathsf{H}_c}}{p} + Q_{ni} \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda).$$

Proof. We prove the statement using a sequence of games.

Game $\mathbf{G}_0$: This is the real key onewayness game. That is, $\mathcal{A}$ gets access to oracles NewIdentity, Corr, Sign, and is run on input par. In the end, it terminates with an output $(\mathsf{ipk}^*, \mathsf{pk}^*, \mathsf{pseed}^*, \mathsf{tag}^*, \mathsf{sk}^*)$. The game outputs 1 if $\mathsf{pk}^* = g^{\mathsf{sk}^*}$, and there is an identity $(\mathsf{ipk}^*, \mathsf{ivk})$ output by NewIdentity, such that ipk^* was never queried to Corr and $\mathsf{Track}(\mathsf{ipk}^*, \mathsf{ok}, \mathsf{pk}^*, \mathsf{tag}^*) = 1$ for $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}^*)$. In other words, $\mathcal{A}$ wins if it can find a secret key for a public key that is tracked by a non-corrupted identity.

Game $\mathbf{G}_1$: This is as $\mathbf{G}_0$, but we change the way random oracle H_p is simulated. Namely, for each undefined hash value, the game now first samples $\vartheta \leftarrow\!\!\$\, \mathbb{Z}_p$, then sets the hash value to $h := g^{\vartheta}$, and stores $\vartheta_h := \vartheta$ for later use. Clearly, the view of $\mathcal{A}$ does not change, and we have

$$\Pr\left[\mathbf{G}_0 \rightarrow 1\right] = \Pr\left[\mathbf{G}_1 \rightarrow 1\right].$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we change how oracle Sign is executed. Recall that in $\mathbf{G}_1$, oracle Sign works as follows on input $\mathsf{pk}, \mathsf{pseed}, \mathsf{tag}, \mathsf{R}, \mathsf{auxsk}, \mathsf{m}$: It first finds an identity $(\mathsf{ipk}, \mathsf{ivk}, \mathsf{isk})$ such that $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) = 1$ for $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed})$. Then, it derives $\mathsf{sk} := \mathsf{DerSK}(\mathsf{isk}, \mathsf{ok}, \mathsf{tag})$ and computes a signature $\sigma \leftarrow \mathsf{Sig}(\mathsf{R}, \mathsf{sk}, \mathsf{auxsk}, \mathsf{m})$. In $\mathbf{G}_1$, we instead compute σ without knowledge of sk and thus without knowledge of isk. This is done by replacing algorithm Sig with algorithm Sig'', which is presented in Figure 10. Note that in Line 06, this algorithm makes use of the change we introduced in $\mathbf{G}_1$. It follows from standard honest-verifier zero-knowledge properties that the view of $\mathcal{A}$ does not change from $\mathbf{G}_1$ to $\mathbf{G}_2$, unless the random oracle value that is programmed in Line 12 of Figure 10 is already defined. As the value R_N is distributed uniformly at random, this occurs with probability at most Q_{H_c}/p for every execution of Sig''. Therefore, a union bound of the number of signing queries shows

$$\left|\Pr\left[\mathbf{G}_1 \rightarrow 1\right] - \Pr\left[\mathbf{G}_2 \rightarrow 1\right]\right| \leq \frac{Q_S Q_{\mathsf{H}_c}}{p}.$$

Finally, we bound the probability that $\mathbf{G}_2$ outputs 1 using a reduction $\mathcal{B}$ breaking the DLOG assumption. Informally, the reduction guesses the non-corrupted identity ipk^*. Formally, the reduction is as follows

1. $\mathcal{B}$ gets as input a DLOG instance $g, y = g^x$. It samples an index $i_0 \leftarrow\!\!\$\, [Q_{ni}]$.
2. $\mathcal{B}$ simulates $\mathbf{G}_2$ for $\mathcal{A}$, except for the i_0th query to oracle NewIdentity, and related queries to Corr. Namely, on the i_0th query to NewIdentity, $\mathcal{B}$ defines $\mathsf{ipk} = (K_v, K_s)$ as it is done in $\mathbf{G}_2$, except that it sets $K_s := y$. Whenever $\mathcal{A}$ queries Corr on this ipk, the reduction $\mathcal{B}$ aborts its execution. Note that oracle Sign can be simulated without knowing the discrete logarithm of y, due to the changes we introduced.
3. When $\mathcal{A}$ terminates with output $(\mathsf{ipk}^*, \mathsf{pk}^*, \mathsf{pseed}^*, \mathsf{tag}^*, \mathsf{sk}^*)$ such that $\mathbf{G}_2$ would output 1, then we have $\mathsf{pk}^* = g^{\mathsf{sk}^*}$, and there is an identity $(\mathsf{ipk}^*, \mathsf{ivk})$ output by NewIdentity, such that ipk^* was never queried to Corr and $\mathsf{Track}(\mathsf{ipk}^*, \mathsf{ok}, \mathsf{pk}^*, \mathsf{tag}^*) = 1$ for $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}^*)$. Let $\mathsf{ipk}^* = (K_v, K_s)$. If $K_s \neq y$, i.e., the query was not guessed correctly, $\mathcal{B}$ aborts its execution. Otherwise, we know that (by definition of Track)

$$g^{\mathsf{sk}^*} = \mathsf{pk}^* = K_s \cdot g^{\mathsf{H}(\mathsf{ok},\mathsf{tag})} = y \cdot g^{\mathsf{H}(\mathsf{ok},\mathsf{tag})}.$$

The reduction outputs $x := \mathsf{sk}^* - \mathsf{H}(\mathsf{ok}, \mathsf{tag})$ as the discrete logarithm of y.

Clearly, the running time of $\mathcal{B}$ is dominated by the running time of $\mathcal{A}$. Further, reduction $\mathcal{B}$ simulates $\mathbf{G}_2$ perfectly for $\mathcal{A}$ until a potential abort, and the view of $\mathcal{A}$ is independent of index i_0. Further, if the index is guessed correctly, and $\mathbf{G}_2$ outputs 1, then $\mathcal{B}$ computes a correct discrete logarithm. We have

$$\Pr\left[\mathbf{G}_2 \rightarrow 1\right] \leq Q_{ni} \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda).$$

□

Lemma 10. *Assume that* $(\mathsf{LRS}, \mathsf{KDS})$ *satisfies key onewayness, key knowledge soundness, and knowledge non-slanderability. Then,* $(\mathsf{LRS}, \mathsf{KDS})$ *satisfies non-slanderability in the algebraic group model. Concretely, for any algebraic PPT algorithm* $\mathcal{A}$*, there is are PPT algorithms* $\mathcal{B}_1$*,* $\mathcal{B}_2$*,* $\mathcal{B}_3$ *with* $\mathbf{T}(\mathcal{B}_i) \approx \mathbf{T}(\mathcal{A})$ *for all* $i \in \{1,2,3\}$*, and*

$$\mathsf{Adv}^{\mathsf{non\text{-}sland}}_{\mathcal{A},\mathsf{LRS},\mathsf{KDS}}(\lambda) \leq \mathsf{Adv}^{\mathsf{k\text{-}kn\text{-}sound}}_{\mathcal{B}_1,\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) + \mathsf{Adv}^{\mathsf{kn\text{-}n\text{-}sland}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) + \mathsf{Adv}^{\mathsf{k\text{-}ow}}_{\mathcal{B}_3,\mathsf{LRS},\mathsf{KDS}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an algebraic algorithm against non-slanderability of $(\mathsf{LRS}, \mathsf{KDS})$. Let $\mathsf{Ext}_{\mathsf{LRS}}$ be the knowledge extractor of $(\mathsf{LRS}, \mathsf{KDS})$. We show the statement via a sequence of games $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2$, and a final reduction breaking key onewayness.

Game $\mathbf{G}_0$: We start with $\mathbf{G}_0$, which is the non-slanderability game. To recall, $\mathcal{A}$ gets as input system parameters par and access to oracles NewIdentity, Corr, Sign. Then, it outputs a forgery $(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$, and the game outputs 1 if the following three conditions hold: The signature verifies, i.e., $\mathsf{Ver}(\mathsf{R}^*, \mathsf{m}^*, \sigma^*) = 1$. The signature is fresh, i.e., $\mathcal{A}$ never submitted R^* and m^* as part of the input of a query to Sign. There is a signature σ returned by Sign such that σ^* links to σ, and the identity ipk that generated σ is not corrupted. More formally, this means that there is an entry $(\mathsf{ipk}, \mathsf{pk}, \mathsf{R}, \mathsf{m}, \sigma) \in \mathcal{L}_s$ with $\mathsf{Link}(\sigma, \sigma^*) = 1$ and $\mathsf{ipk} \notin \mathcal{L}_c$. By definition, we have

$$\mathsf{Adv}^{\mathsf{non\text{-}sland}}_{\mathcal{A},\mathsf{LRS},\mathsf{KDS}}(\lambda) = \Pr\left[\mathbf{G}_0 \rightarrow 1\right].$$

Game $\mathbf{G}_1$: We define $\mathbf{G}_1$ to be exactly as $\mathbf{G}_0$, but we introduce an additional step and potential abort after the game receives $\mathcal{A}$'s forgery $(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$. Namely, we run the knowledge extractor $\mathsf{Ext}_{\mathsf{LRS}}$ to extract secret keys and an index from the forgery. More precisely, before $\mathbf{G}_0$ would output 1, we run write $\mathsf{R}^* = (\mathsf{pk}_i, \mathsf{auxpk}_i)_i$ and run $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*}) \leftarrow \mathsf{Ext}_{\mathsf{LRS}}(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$. Here, $\mathsf{Ext}_{\mathsf{LRS}}$ can inspect all random oracle queries that $\mathcal{A}$ makes. Then, we abort if $(\mathsf{auxpk}_{i^*}, \mathsf{auxsk}_{i^*}) \notin \mathcal{AKR}$ or $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \notin \mathcal{KR}$. Otherwise, the game outputs 1, as $\mathbf{G}_0$ would do. Note that $\mathbf{G}_0$ and $\mathbf{G}_1$ only differ if the newly introduced abort is triggered.

We can bound the probability of this event easily using a reduction $\mathcal{B}_1$ that breaks key knowledge soundness. Namely, the reduction gets as input par, and access to the same oracles as in the non-slanderability game. Further, it gets access to an oracle O. The reduction passes par to $\mathcal{A}$ and simulates game $\mathbf{G}_0$ by forwarding oracle queries to its own oracles. Before the game would output 1, it submits the forgery $(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$ to oracle O. Clearly, the running time of $\mathcal{B}_1$ is dominated by the running time of $\mathcal{A}$, and $\mathcal{B}_1$ perfectly simulates game $\mathbf{G}_0$ for $\mathcal{A}$. Further, note that if $\mathbf{G}_0$ would output 1, then the oracle O does not return early, i.e., it runs $\mathsf{Ext}_{\mathsf{LRS}}$. This is because the winning condition states that the forgery verifies and is fresh. Once the oracle O runs $\mathsf{Ext}_{\mathsf{LRS}}$, it sets the flag bad exactly when $\mathbf{G}_1$ would abort. Thus, if the event separating $\mathbf{G}_0$ and $\mathbf{G}_1$ occurs, $\mathcal{B}_1$ breaks key knowledge soundness. We have

$$\left|\Pr\left[\mathbf{G}_0 \rightarrow 1\right] - \Pr\left[\mathbf{G}_1 \rightarrow 1\right]\right| \leq \mathsf{Adv}^{\mathsf{k\text{-}kn\text{-}sound}}_{\mathcal{B}_1,\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda).$$

Game $\mathbf{G}_2$: In game $\mathbf{G}_2$, we add an additional abort event related to the extracted secret keys sk_{i^*} that we introduced in $\mathbf{G}_1$. Then, we will bound the Assume that $\mathbf{G}_1$ would output 1. Then, let $(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$ be $\mathcal{A}$'s forgery with $\mathsf{R}^* = (\mathsf{pk}_i, \mathsf{auxpk}_i)_i$. Let $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*})$ be the output of $\mathsf{Ext}_{\mathsf{LRS}}$ as written in $\mathbf{G}_1$. Further, let $(\mathsf{ipk}, \mathsf{pk}, \mathsf{R}, \mathsf{m}, \sigma) \in \mathcal{L}_s$ be the entry in $\mathcal{L}_s$ such that $\mathsf{Link}(\sigma, \sigma^*) = 1$ and $\mathsf{ipk} \notin \mathcal{L}_c$. Then, we let $\mathbf{G}_2$ abort if $\mathsf{pk}_{i^*} \neq \mathsf{pk}$. It is easy to see that the probability of this event can be bounded using a reduction $\mathcal{B}_2$ that breaks knowledge non-slanderability. The reduction is basically identical to the reduction breaking key knowledge soundness we described before, and we just have to observe that the winning condition of the knowledge non-slanderability game is equivalent to the event that we want to bound here. We have

$$|\Pr[\mathbf{G}_1 \to 1] - \Pr[\mathbf{G}_2 \to 1]| \leq \mathsf{Adv}^{\mathsf{kn\text{-}n\text{-}sland}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda).$$

In the final step of the proof, we bound the probability that $\mathbf{G}_2$ outputs 1 using a reduction $\mathcal{B}_3$ that breaks key onewayness. The reduction is as follows:

1. Reduction $\mathcal{B}_3$ gets as input par, it gets access to oracles NewIdentity, Corr, Sign.
2. The reduction runs $\mathcal{A}$ on input par, and provides the oracles NewIdentity, Corr, Sign to $\mathcal{A}$ by forwarding. Note that $\mathcal{B}_3$ can keep the lists $\mathcal{L}_{id}, \mathcal{L}_s$, and $\mathcal{L}_c$ up-to-date during forwarding. It also provides all random oracles (if any) to $\mathcal{A}$ by forwarding.
3. When $\mathcal{A}$ outputs a forgery $(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$, the reduction first verifies the forgery as in the non-slanderability game, i.e., it checks if $\mathsf{Ver}(\mathsf{R}^*, \mathsf{m}^*, \sigma^*) = 1$, there does not exist $\mathsf{ipk}, \mathsf{pk}, \sigma$ such that $(\mathsf{ipk}, \mathsf{pk}, \mathsf{R}^*, \mathsf{m}^*, \sigma) \in \mathcal{L}_s$, and there there exists an entry $(\mathsf{ipk}^*, \mathsf{pk}^*, \mathsf{R}, \mathsf{m}, \sigma) \in \mathcal{L}_s$ such that $\mathsf{Link}(\sigma, \sigma^*) = 1$ and $\mathsf{ipk}^* \notin \mathcal{L}_c$.
4. If all of these checks pass, $\mathcal{B}_3$ runs $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*}) \leftarrow \mathsf{Ext}_{\mathsf{LRS}}(\mathsf{R}^*, \mathsf{m}^*, \sigma^*)$, where $\mathsf{Ext}_{\mathsf{LRS}}$ can inspect all random oracle queries that $\mathcal{A}$ makes. The reduction $\mathcal{B}_3$ then aborts if $(\mathsf{auxpk}_{i^*}, \mathsf{auxsk}_{i^*}) \notin \mathcal{AKR}$, $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \notin \mathcal{KR}$, or $\mathsf{pk}_{i^*} \neq \mathsf{pk}^*$. Otherwise, let $\textsc{Sign}(\mathsf{pk}^*, \mathsf{pseed}^*, \mathsf{tag}^*, \mathsf{R}, \mathsf{auxsk}, \mathsf{m})$ be the signing query that lead to inserting $(\mathsf{ipk}^*, \mathsf{pk}^*, \mathsf{R}, \mathsf{m}, \sigma)$ into $\mathcal{L}_s$. The reduction $\mathcal{B}_3$ outputs $(\mathsf{ipk}^*, \mathsf{pk}^*, \mathsf{pseed}^*, \mathsf{tag}^*, \mathsf{sk}_{i^*})$ to the key onewayness game.

Clearly, $\mathcal{B}_3$'s running time is dominated by the running time of $\mathcal{A}$. Further, $\mathcal{B}_3$ simulates game $\mathbf{G}_2$ perfectly for $\mathcal{A}$[6]. Finally, we claim that if $\mathbf{G}_2$ outputs 1, then $\mathcal{B}_3$ breaks key onewayness. First, it follows from the winning condition of non-slanderability (which is still present in $\mathbf{G}_2$) that $\mathsf{ipk}^* \notin \mathcal{L}_c$. Second, we have $\mathsf{Track}(\mathsf{ipk}^*, \mathsf{ok}, \mathsf{pk}^*, \mathsf{tag}^*) = 1$ for $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed}^*)$, because otherwise oracle Sign would not have returned a signature. Third, we know that (cf. $\mathbf{G}_1$) $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \in \mathcal{KR}$. As (cf. $\mathbf{G}_2$) $\mathsf{pk}^* = \mathsf{pk}_{i^*}$, we also have $(\mathsf{pk}^*, \mathsf{sk}_{i^*}) \in \mathcal{KR}$. Thus, we have

$$\Pr[\mathbf{G}_2 \to 1] \leq \mathsf{Adv}^{\mathsf{k\text{-}ow}}_{\mathcal{B}_3,\mathsf{LRS},\mathsf{KDS}}(\lambda).$$

□

Lemma 11. *Let* GGen *be an algorithm that on input* 1^λ *outputs the description of a prime order group* $\mathbb{G}$ *of order* $p \geq 2^\lambda$ *with generator* g*. Let* $\hat{\mathrm{O}}$ *be an oracle that does not take any input, and when queried the* i*th time it samples* $h_i \leftarrow_\$ \mathbb{G}$ *and returns it. Let* $\mathcal{A}$ *be any algorithm making at most* Q *queries to* $\hat{\mathrm{O}}$ *in the experiment* $\mathfrak{G}$*, which is given as*

$$(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda), \quad (\alpha, (\gamma_i)_{i=1}^Q, \alpha', (\gamma'_i)_{i=1}^Q) \leftarrow \mathcal{A}^{\hat{\mathrm{O}}}(g).$$

[6]The only subtlety here is that we need to observe that the algebraic representations that $\mathsf{Ext}_{\mathsf{LRS}}$ gets have the same structure as in $\mathbf{G}_2$. Otherwise, $\mathcal{B}_3$ could not provide the input for $\mathsf{Ext}_{\mathsf{LRS}}$ correctly. This is one reason why the definition of key knowledge soundness needs to contain the other oracles, and not just oracle O.

Then, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ such that

$$\Pr_{\mathfrak{G}}\left[\begin{array}{rc} & (\alpha, (\gamma_i)_{i=1}^Q) \neq (\alpha', (\gamma_i')_{i=1}^Q) \\ \wedge & g^\alpha \prod_{i=1}^Q h_i^{\gamma_i} = g^{\alpha'} \prod_{i=1}^Q h_i^{\gamma_i'} \end{array}\right] \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

Proof. Before we describe the reduction $\mathcal{B}$ against the DLOG assumption, we first make some observations. We can write every h_i returned by oracle $\hat{\mathrm{O}}$ as $h_i = h^{\nu_i} \cdot g^{\tau_i}$ for some fixed $h = g^\vartheta \in \mathbb{G}, \vartheta \in \mathbb{Z}_p$ and independent and uniformly random $\nu_i, \tau_i \in \mathbb{Z}_p$. Next, assume the event we want to bound holds. Define $\Delta_\alpha := \alpha - \alpha'$ and $\Delta_i := \gamma_i - \gamma_i'$ for all $i \in [Q]$. At least one of these is non-zero. We have

$$g^0 = g^{\Delta_\alpha} \cdot \prod_{i=1}^Q h_i^{\Delta_i} = g^{\Delta_\alpha + \sum_{i=1}^Q \Delta_i(\tau_i + \nu_i\vartheta)}$$
$$\implies -\Delta_\alpha - \sum_{i=1}^Q \Delta_i\tau_i = \vartheta \sum_{i=1}^Q \Delta_i\nu_i.$$

Therefore, as long as $\sum_{i=1}^Q \Delta_i\nu_i \neq 0$, the reduction can solve for ϑ. More precisely, the reduction works as follows.

1. Reduction $\mathcal{B}$ gets as input $g, h \in \mathbb{G}$ and runs $\mathcal{A}$ on input g, while simulating oracle $\hat{\mathrm{O}}$ for $\mathcal{A}$ as follows: When $\mathcal{A}$ queries $\hat{\mathrm{O}}$ for the ith time, $\mathcal{B}$ samples $\nu_i, \tau_i \leftarrow_\$ \mathbb{Z}_p$, sets $h_i := h^{\nu_i} \cdot g^{\tau_i}$, and returns h_i.
2. When $\mathcal{A}$ terminates with output $(\alpha, (\gamma_i)_{i=1}^Q, \alpha', (\gamma_i')_{i=1}^Q)$, the reduction defines $\Delta_\alpha := \alpha - \alpha'$ and $\Delta_i := \gamma_i - \gamma_i'$. If $\sum_{i=1}^Q \Delta_i\nu_i = 0$, the reduction aborts. Otherwise, it outputs

$$\vartheta := \frac{-\Delta_\alpha - \sum_{i=1}^Q \Delta_i\tau_i}{\sum_{i=1}^Q \Delta_i\nu_i}.$$

Clearly, the running time of $\mathcal{B}$ is dominated by the running time of $\mathcal{A}$. Further, $\mathcal{B}$ perfectly simulates the game for $\mathcal{A}$. By the above discussion, if the event we want to bound holds and $\sum_{i=1}^Q \Delta_i\nu_i \neq 0$, then $\mathcal{B}$ computes a correct discrete logarithm of h with respect to g. It remains to argue that $\sum_{i=1}^Q \Delta_i\nu_i \neq 0$ holds except with negligible probability. To see this, first note that $\mathcal{A}$'s view is independent of the ν_i, due to the uniformity of the τ_i. Further, the matrix $\mathbf{D} := (\Delta_1, \ldots, \Delta_Q) \in \mathbb{Z}_p^{1\times Q}$ is non-zero, because otherwise Δ_α also has to be zero, which contradicts the event that we want to bound. Therefore, the kernel of $\mathbf{D}$ has dimension at most $Q-1$. Thus, with probability at most $p^{Q-1}/p^Q = 1/p$, the random vector $(\nu_1, \ldots, \nu_Q)^\top$ is in the kernel of $\mathbf{D}$. □

Lemma 12. *Let GGen be an algorithm that on input 1^λ outputs the description of a prime order group $\mathbb{G}$ of order $p \geq 2^\lambda$ with generator g. Let $\hat{\mathsf{H}} : \{0,1\}^* \to \mathbb{Z}_p^2$ be a random oracle and $\mathcal{A}$ be any algorithm making at most $Q_{\hat{\mathsf{H}}}$ queries to $\hat{\mathsf{H}}$ in the experiment $\mathfrak{G}$, which is given as*

$$\begin{array}{rlrl} (\mathbb{G}, g, p) &\leftarrow \mathsf{GGen}(1^\lambda), & h &\leftarrow_\$ \mathbb{G} \setminus \{g^0\}, \\ (St, R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}) &\leftarrow \mathcal{A}^{\hat{\mathsf{H}}}(g, h), & c &\leftarrow_\$ \mathbb{Z}_p \setminus \{0\}, \\ (s, \tilde{K}_1, \tilde{K}_2, idx) &\leftarrow \mathcal{A}^{\hat{\mathsf{H}}}(St, c), & (a_1, a_2) &:= \hat{\mathsf{H}}(idx, \tilde{K}_1, \tilde{K}_2), \\ W &:= \mathsf{pk}^{a_1}\mathsf{auxpk}^{a_2}, & \tilde{W} &:= \tilde{K}_1^{a_1}\tilde{K}_2^{a_2}. \end{array}$$

Then, we have

$$\Pr_{\mathfrak{G}}\left[\ R = g^s W^c \wedge \tilde{R} = h^s \tilde{W}^c \wedge \forall w \in \mathbb{Z}_p : \neg(W_{i^*} = g^w \wedge \tilde{W} = h^w)\ \right] \leq \frac{2Q_{\hat{\mathsf{H}}}}{p}.$$

Proof. To prove the lemma, we first introduce notation. Namely, write $W = g^w$, $\tilde{W} = h^{\tilde{w}}$ and $R = g^r$, $\tilde{R} = h^{\tilde{r}}$ for uniquely determined exponents $w, \tilde{w}, r, \tilde{r} \in \mathbb{Z}_p$. We have to bound the probability of the following event:

- Event Win: This event occurs, if $r = s + cw$, $\tilde{r} = s + c\tilde{w}$, and $w \neq \tilde{w}$.

Let $i^* \in [Q_{\hat{\mathsf{H}}}]$ be the index of the first query of the form $\hat{\mathsf{H}}(idx, \tilde{K}_1, \tilde{K}_2)$, where $idx, \tilde{K}_1, \tilde{K}_2$ are part of $\mathcal{A}$'s final responses. Then, we can partition event Win into $Q_{\hat{\mathsf{H}}}$ events:

- Event Win_i: This event occurs, if Win occurs and $i^* = i$.

By a union bound, we have

$$\Pr[\mathsf{Win}] \leq \sum_{i=1}^{Q_{\hat{\mathsf{H}}}} \Pr[\mathsf{Win}_i].$$

To bound Win_i, we distinguish two cases, depending on whether $\mathcal{A}$ first queries $\hat{\mathsf{H}}(idx, \tilde{K}_1, \tilde{K}_2)$ or returns $R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}$. Namely, define the following events:

- Event $\mathsf{HashFirst}$: This event occurs, if $\mathcal{A}$ queries $\hat{\mathsf{H}}(idx, \tilde{K}_1, \tilde{K}_2)$ for the first time before returning $R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}$, i.e., the i^*th query occurs before $\mathcal{A}$ returns $R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}$.
- Event $\mathsf{RsFirst}$: This event occurs, if $\mathcal{A}$ queries $\hat{\mathsf{H}}(idx, \tilde{K}_1, \tilde{K}_2)$ for the first time after returning $R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}$, i.e., the i^*th query occurs after $\mathcal{A}$ returns $R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}$.

Clearly, these events are complementary, and for all $i \in [Q_{\hat{\mathsf{H}}}]$ we have

$$\Pr[\mathsf{Win}_i] \leq \Pr[\mathsf{Win}_i \mid \mathsf{HashFirst}] + \Pr[\mathsf{Win}_i \mid \mathsf{RsFirst}].$$

We first bound the probability of the former term. Assuming that the query $i^* = i$ occurs before $\mathcal{A}$ returns $R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}$, the values w and $\tilde{w}$ are fixed before c is sampled and given to $\mathcal{A}$. We argue that under these assumptions there can be at most one c, for which a response s exists, such that Win_i occurs. Namely, suppose there are two distinct $c, c' \in \mathbb{Z}_p$, and corresponding $s, s' \in \mathbb{Z}_p$ such that Win_i occurs. Then, we have $r = s + cw$ and $r = s' + c'w$, implying $w = (s - s')/(c' - c)$. Similarly, we get $\tilde{w} = (s - s')/(c' - c) = w$, and thus event Win_i can not occur. Therefore, we know that the former term is at most $1/p$. Next, we bound the probability of the latter term, i.e., the probability that Win_i occurs given that $\mathcal{A}$ first outputs $R, \tilde{R}, \mathsf{pk}, \mathsf{auxpk}$, and then makes the i^*th query $\hat{\mathsf{H}}(idx, \tilde{K}_1, \tilde{K}_2)$. Let $z_1, z_2, \mathsf{sk}, \mathsf{auxsk} \in \mathbb{Z}_p$ be the uniquely determined exponents satisfying $\tilde{K}_1 = h^{z_1}$, $\tilde{K}_2 = g^{z_2}$, and $\mathsf{pk} = g^{\mathsf{sk}}$, $\mathsf{auxpk} = g^{\mathsf{auxsk}}$. Note that if Win_i occurs, it must hold that $s = r - cw$ and $s = \tilde{r} - c\tilde{w}$. Rearranging and substituting the definition of w and $\tilde{w}$, this implies

$$a_1(z_1 - \mathsf{sk}) + a_2(z_2 - \mathsf{auxsk}) = \frac{r - \tilde{r}}{c}.$$

Assuming event $\mathsf{RsFirst}$ occurs, we know that the right-hand side of this equation, as well as $(z_1 - \mathsf{sk}, z_2 - \mathsf{auxsk})$ are fixed before a_1 and a_2 are sampled uniformly at random by $\hat{\mathsf{H}}$. If Win_i occurs, it can not happen that $(z_1 - \mathsf{sk}, z_2 - \mathsf{auxsk})$ is equal to $(0, 0)$ (as otherwise $w = \tilde{w}$), and therefore the left-hand side is uniformly distributed over $\mathbb{Z}_p$. Thus, the probability that the equation holds is at most $1/p$, finishing the proof. □

Lemma 13. *Let* GGen *be an algorithm that on input* 1^λ *outputs the description of a prime order group* $\mathbb{G}$ *of order* $p \geq 2^\lambda$ *with generator* g*. Let* $\hat{\mathrm{O}}$ *be an oracle that does not take any input, and when queried the* i*th time it samples* $h_i \leftarrow_s \mathbb{G}$ *and returns it. Let* $\mathcal{A}$ *be any algorithm making at most* Q *queries to* $\hat{\mathrm{O}}$ *in the experiment* $\mathfrak{G}$*, which is given as*

$$(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda), \quad h \leftarrow_s \mathbb{G} \setminus \{g^0\},$$
$$(W, \tilde{W}, \alpha, \beta, (\gamma_i)_{i=1}^Q, \tilde{\alpha}, \tilde{\beta}, (\tilde{\gamma}_i)_{i=1}^Q, c, s) \leftarrow \mathcal{A}^{\hat{\mathrm{O}}}(g, h)$$

Then, there is a PPT algorithm $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *such that*

$$\Pr_{\mathfrak{G}}\left[\begin{array}{cc} & g^\alpha h^\beta \prod_{i=1}^Q h_i^{\gamma_i} = g^s W^c \wedge g^{\tilde{\alpha}} h^{\tilde{\beta}} \prod_{i=1}^Q h_i^{\tilde{\gamma}_i} = h^s \tilde{W}^c \\ \wedge & \exists w \in \mathbb{Z}_p : W = g^w \wedge \tilde{W} = h^w \\ \wedge & (\beta \neq 0 \vee \tilde{\alpha} \neq 0 \vee \alpha \neq \tilde{\beta} \vee \bigvee_{i=1}^Q \gamma_i \neq 0 \vee \tilde{\gamma}_i \neq 0) \end{array}\right] \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

Proof. Before giving the reduction $\mathcal{B}$ as stated in the lemma, we make some observations. For that, write $h = g^\vartheta$. The goal of the reduction will be to find this $\vartheta \in \mathbb{Z}_p$. Assume that the reduction defines each h_i returned by $\hat{\mathrm{O}}$ as $h_i := h^{x_i} g^{y_i}$ for uniformly random and independent $x_i, y_i \in \mathbb{Z}_p$, $i \in [Q]$. Write $\mathbf{x} := (x_1, \ldots, x_Q)^\top \in \mathbb{Z}_p^Q$ and $\mathbf{y} := (y_1, \ldots, y_Q)^\top \in \mathbb{Z}_p^Q$. Assuming that the event that we want to bound occurs, we can look at the equations in the exponent of g, and get the equation

$$s\begin{pmatrix}1\\ \vartheta\end{pmatrix} + cw\begin{pmatrix}1\\ \vartheta\end{pmatrix} = \underbrace{\begin{pmatrix}\alpha & \beta\\ \tilde{\alpha} & \tilde{\beta}\end{pmatrix}}_{=:\mathbf{M}}\begin{pmatrix}1\\ \vartheta\end{pmatrix} + \underbrace{\begin{pmatrix}\gamma_1 & \cdots & \gamma_Q\\ \tilde{\gamma}_1 & \cdots & \tilde{\gamma}_Q\end{pmatrix}}_{=:\mathbf{L}}[\mathbf{y} \mid \mathbf{x}]\begin{pmatrix}1\\ \vartheta\end{pmatrix}.$$

In other words, we observe that the vector $\mathbf{v} := (1, \vartheta)^\top \in \mathbb{Z}_p^2$ is an eigenvector of the matrix $\mathbf{M} + \mathbf{L}[\mathbf{y} \mid \mathbf{x}]$ with eigenvalue $v = s + cw$. We claim that the corresponding eigenspace has dimension 1 with overwhelming probability. Clearly, it has dimension $0, 1$, or 2. As $\mathbf{v} \neq \mathbf{0}$ is in this eigenspace, it does not have dimension 0. Further, this eigenspace is exactly the kernel of the matrix

$$\mathbf{E} := \mathbf{M} + \mathbf{L}[\mathbf{y} \mid \mathbf{x}] - v\mathbf{I}_2,$$

where $\mathbf{I}_2$ is the 2×2 identity matrix. The only way this kernel could have dimension 2 is if the matrix $\mathbf{E}$ was the zero matrix. Now, consider two cases. In the first case, assume $\mathbf{L} = \mathbf{0}$. Then, $\mathbf{E} = \mathbf{0}$ can only happen if $\beta = 0$, $\tilde{\alpha} = 0$, and $\alpha = \tilde{\beta}$, which is ruled out by the event we want to bound. In the second case, assume $\mathbf{L} \neq \mathbf{0}$. Then, $\mathbf{E} = \mathbf{0}$ implies that

$$\mathbf{L}[\mathbf{y} \mid \mathbf{x}] = v\mathbf{I}_2 - \mathbf{M}.$$

In particular, the second columns of these matrices are equal. By the way we defined the h_i, no information about $\mathbf{x}$ is ever leaked to $\mathcal{A}$, due to uniformity of $\mathbf{y}$. Thus, from $\mathcal{A}$'s view, $\mathbf{x}$ remains uniformly random for the entire experiment. Further, if $\mathbf{L}$ is non-zero, it has a kernel of dimension at most $Q - 1$, meaning that the probability that above equation holds (for the uniform $\mathbf{x}$) is at most $p^{Q-1}/p^Q = 1/p$. In summary, we obtain that with overwhelming probability the eigenspace has dimension 1, i.e., it is generated by some non-zero vector $\mathbf{a} \in \mathbb{Z}_p$. Note that given $\mathbf{a}$, one can efficiently find a scalar $\nu \in \mathbb{Z}_p$ such that $\nu\mathbf{a} = \mathbf{v}$, and therefore one can efficiently find ϑ.

Thus, the reduction $\mathcal{B}$ works as follows: It gets as input a DLOG challenge $\mathbb{G}, p, g, h = g^\vartheta$, and runs $\mathcal{A}^{\hat{\mathrm{O}}}(g, h)$ while simulating the ith query to $\hat{\mathrm{O}}$ by sampling $x_i, y_i \leftarrow_s \mathbb{Z}_p$ and returning $h_i := h^{x_i} g^{y_i}$. When $\mathcal{A}$ terminates, $\mathcal{B}$ obtains $(W, \tilde{W}, \alpha, \beta, (\gamma_i)_{i=1}^Q, \tilde{\alpha}, \tilde{\beta}, (\tilde{\gamma}_i)_{i=1}^Q, c, s)$. It forms matrices $\mathbf{M}, \mathbf{L}$ as

above, and determines the eigenvalues and basis vectors for the eigenspaces of $\mathbf{M} + \mathbf{L}[\mathbf{y} \mid \mathbf{x}]$. If the event in doubt occurs, at least one of the eigenspaces has dimension 1 with overwhelming probability and contains $\mathbf{v}$ as above. The reduction can now find ϑ by appropriately scaling the basis vector of this eigenspace. □

Lemma 14. *Let* GGen *be an algorithm that on input* 1^λ *outputs the description of a prime order group* $\mathbb{G}$ *of order* $p \geq 2^\lambda$ *with generator* g*. Let* $\hat{\mathrm{O}}$ *be an oracle that does not take any input, and when queried the* i*th time it samples* $\vartheta_i \leftarrow_s \mathbb{Z}_p$ *and returns* $h_i := g^{\vartheta_i}$*. Let* $\mathcal{A}$ *be any algorithm making at most* Q *queries to* $\hat{\mathrm{O}}$ *in the experiment* $\mathfrak{G}$*, which is given as*

$$(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda), \quad \big((\delta_j)_{j\in[L]}, (\eta_{i,j})_{i\in[Q],j\in[L]}\big) \leftarrow \mathcal{A}^{\hat{\mathrm{O}}}(g).$$

Then, there is a PPT algorithm $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *such that*

$$\Pr_{\mathfrak{G}}\left[\begin{array}{cc} & (\exists j \in [L] : \delta_j \neq 0 \vee \exists (i,j) \in [Q] \times [L] : \eta_{i,j} \neq 0) \\ \wedge & \Big(\forall j \in [L] : \delta_j + \sum_{i=1}^{Q} \vartheta_i \eta_{i,j} = 0\Big) \end{array}\right] \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

Proof. The lemma follows immediately from Lemma 11, which rules out that one can efficiently find two distinct representations of the same group element with respect to basis g, h_i. Note that if the event we want to bound here occurs, then we have

$$g^0 = g^{\delta_j} \prod_i h_i^{\eta_{i,j}},$$

which are two distinct representations of the element g^0. The formal reduction is trivial. □

Lemma 15. *Let* GGen *be an algorithm that on input* 1^λ *outputs the description of a prime order group* $\mathbb{G}$ *of order* $p \geq 2^\lambda$ *with generator* g*. Consider the following oracles, where* $\mathcal{L}_C$ *is an initially empty set:*

- *An oracle* O_I *that does not take any input. On the* j*th query, it samples* $k_j \leftarrow_s \mathbb{Z}_p$ *and returns* $K_j := g^{k_j}$*.*
- *An oracle* O_C *that takes as input* $j \in \mathbb{N}$*. If* k_j *is not yet defined, it returns* $\perp$*. Otherwise, it inserts* j *into* $\mathcal{L}_C$ *and returns* k_j*.*
- *A random oracle* $\hat{\mathsf{H}} : \{0,1\}^* \to \mathbb{Z}_p^2$*.*

Let $\mathcal{A}$ *be an algebraic PPT algorithm making at most* $Q_{\hat{\mathsf{H}}}, Q, Q_C$ *queries to oracles* $\hat{\mathsf{H}}, \mathrm{O}, \mathrm{O}_C$*, respectively, in the experiment* $\mathfrak{G}$*, which is given as follows:*

1. *Generate* $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$ *and run* $(St, R, \mathsf{pk}, \mathsf{auxpk}) \leftarrow \mathcal{A}^{\hat{\mathsf{H}}, \mathrm{O}_I, \mathrm{O}_C}(g)$*.*
2. *Let* $\alpha^{(R)}, \alpha^{(\mathsf{pk})}, \alpha^{(\mathsf{auxpk})}, (\delta_j^{(R)}, \delta_j^{(\mathsf{pk})}, \delta_j^{(\mathsf{auxpk})})_{j=1}^{Q}$ *be the algebraic representation that* $\mathcal{A}$ *output along with* $R, \mathsf{pk}, \mathsf{auxpk}$*, i.e.,*

$$R = g^{\alpha^{(R)}} \prod_{j=1}^{Q} K_j^{\delta_j^{(R)}}, \quad \mathsf{pk} = g^{\alpha^{(\mathsf{pk})}} \prod_{j=1}^{Q} K_j^{\delta_j^{(\mathsf{pk})}}, \quad \mathsf{auxpk} = g^{\alpha^{(\mathsf{auxpk})}} \prod_{j=1}^{Q} K_j^{\delta_j^{(\mathsf{auxpk})}}.$$

3. *Sample* $c \leftarrow_s \mathbb{Z}_p \setminus \{0\}$ *and run* $(s, \mathsf{R}, idx) \leftarrow \mathcal{A}^{\hat{\mathsf{H}}, \mathrm{O}_I, \mathrm{O}_C}(St, c)$*.*

4. *Set* $(a_1, a_2) := \hat{\mathsf{H}}(\mathsf{R}, idx)$ *and define* $W := \mathsf{pk}^{a_1}\,\mathsf{auxpk}^{a_2}$.

5. *Set* $\mathsf{Win} := 1$ *if* $R = g^s W^c$, $(\mathsf{pk}, \mathsf{auxpk}) \in \mathsf{R}$, *and there is a* $j \in [Q] \setminus \mathcal{L}_C$ *such that* $(\delta_j^{(R)} \neq 0 \vee \delta_j^{(\mathsf{pk})} \neq 0 \vee \delta_j^{(\mathsf{auxpk})} \neq 0)$. *Otherwise, set it to* $\mathsf{Win} := 0$.

Then, there is a PPT algorithm $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$, *and*

$$\Pr_{\mathfrak{G}}[\mathsf{Win}] \leq 2Q \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{(N_{max}+1)QQ_{\hat{\mathsf{H}}} + Q}{p},$$

where N_{max} *is the maximum size of key rings* R *that are submitted to random oracle* $\hat{\mathsf{H}}$.

Proof. We first introduce some notation. Consider all variables as in the definition of the game. We define the vector $\mathbf{k} \in \mathbb{Z}_p^Q$ to be $\mathbf{k} := (k_1, \ldots, k_{Q_\mathrm{O}})^\top$. Further, we define the vector $\mathbf{d} \in \mathbb{Z}_p^Q$ to be $\mathbf{d} := (\delta_1^{(R)}, \ldots, \delta_Q^{(R)})^\top$, the vector $\mathbf{p} \in \mathbb{Z}_p^2$ to be $\mathbf{p} := (\alpha^{(\mathsf{pk})}, \alpha^{(\mathsf{auxpk})})^\top$, the vector $\mathbf{a} \in \mathbb{Z}_p^2$ to be $\mathbf{a} := (a_1, a_2)^\top$, and the matrix $\mathbf{D} \in \mathbb{Z}_p^{Q\times 2}$ as

$$\mathbf{D} := \begin{pmatrix} \delta_1^{(\mathsf{pk})} & \delta_1^{(\mathsf{auxpk})} \\ \vdots & \vdots \\ \delta_Q^{(\mathsf{pk})} & \delta_Q^{(\mathsf{auxpk})} \end{pmatrix} = \begin{pmatrix} \mathbf{D}_1^\top \\ \vdots \\ \mathbf{D}_Q^\top \end{pmatrix},$$

where $\mathbf{D}_j^\top \in \mathbb{Z}_p^{1\times 2}$ denote the rows of $\mathbf{D}$. Further, let $\mathsf{sk}, \mathsf{auxsk} \in \mathbb{Z}_p$ be exponents satisfying $\mathsf{pk} = g^{\mathsf{sk}}, \mathsf{auxpk} = g^{\mathsf{auxsk}}$. By definition of pk and auxpk, we have

$$\begin{pmatrix} \mathsf{sk} \\ \mathsf{auxsk} \end{pmatrix} = \mathbf{p} + \mathbf{D}^\top \mathbf{k}.$$

Thus, the equation $R = g^s W^c$ becomes

$$g^{\alpha^{(R)} + \mathbf{k}^\top \mathbf{d}} = R = g^s W^c = g^s \left(g^{\mathsf{sk}\cdot a_1 + \mathsf{auxsk}\cdot a_2}\right)^c = g^s \left(g^{(\mathbf{p}+\mathbf{D}^\top\mathbf{k})^\top \mathbf{a}}\right)^c = g^{s + c(\mathbf{p}+\mathbf{D}^\top\mathbf{k})^\top \mathbf{a}}.$$

Looking at exponents and rearranging terms, this is equivalent to the equation

$$\alpha^{(R)} - s - c \cdot \mathbf{p}^\top \mathbf{a} = c \cdot \mathbf{k}^\top \mathbf{D}\mathbf{a} - \mathbf{k}^\top \mathbf{d} = \mathbf{k}^\top (c \cdot \mathbf{D}\mathbf{a} - \mathbf{d}).$$

If we consider this equation, our intuition is that we can compute a non-corrupted coordinate k_j of $\mathbf{k}$, if the jth coordinate of $\mathbf{z} := c \cdot \mathbf{D}\mathbf{a} - \mathbf{d}$ is non-zero. More formally, we define an event Zero. Recall that event Win is the event that we want to bound in the lemma. We will bound the probability of Win using the DLOG assumption, as long as Zero does not occur. Later, we bound the probability of Zero. Let us rephrase the definition of event Win using above notation, and introduce event Zero.

- Event Win: This event occurs, if $\alpha^{(R)} - s - c \cdot \mathbf{p}^\top \mathbf{a} = \mathbf{k}^\top (c \cdot \mathbf{D}\mathbf{a} - \mathbf{d})$, $(\mathsf{pk}, \mathsf{auxpk}) \in \mathsf{R}$, and there is some $j \in [Q] \setminus \mathcal{L}_C$ such that the jth coordinate of $\mathbf{d}$ is non-zero, or the jth row of $\mathbf{D}$ is non-zero.
- Event Zero: This event occurs, if there is some $j \in [Q] \setminus \mathcal{L}_C$ such that the jth coordinate of $\mathbf{d}$ is non-zero, or the jth row of $\mathbf{D}$ is non-zero, but coordinate $\mathbf{z}_j$ of $\mathbf{z} := c \cdot \mathbf{D}\mathbf{a} - \mathbf{d}$ is zero, i.e., $\mathbf{z}_j = 0$. For each j, we denote by Zero_j the event that Zero occurs for this j.

We have to bound the probability of event Win. We have

$$\Pr[\mathsf{Win}] = \Pr[\mathsf{Win} \wedge \neg\mathsf{Zero}] + \Pr[\mathsf{Zero}].$$

We first bound the probability of $\mathsf{Win} \wedge \neg\mathsf{Zero}$ using a reduction $\mathcal{B}$ that breaks the DLOG assumption. Note that if this event occurs, then by definition we know that for some non-corrupted $j \in [Q] \setminus \mathcal{L}_C$, the coordinate $\mathbf{z}_j$ is non-zero. The reduction now guesses this j, and finds k_j. Formally, reduction $\mathcal{B}$ is as follows:

1. $\mathcal{B}$ gets as input a DLOG challenge $\mathbb{G}, p, g, h = g^k$ and aims to find $k \in \mathbb{Z}_p$.
2. To do that, $\mathcal{B}$ first samples $j^* \leftarrow\!\!\$\, [Q]$, and simulates the game from the lemma honestly for $\mathcal{A}$, except the j^* query to O_I. For that query, $\mathcal{B}$ sets $K_{j^*} := h$, thereby implicitly setting $k_{j^*} = k$. If $\mathcal{A}$ ever queries O_C on input j^*, the reduction aborts its execution.
3. Once $\mathcal{A}$ terminated, reduction $\mathcal{B}$ checks if event $\mathsf{Win} \wedge \neg\mathsf{Zero}$ occurs. Note that this can be done efficiently. If the event does not occur, $\mathcal{B}$ aborts its execution. Otherwise it returns

$$\frac{\alpha^{(R)} - s - c \cdot \mathbf{p}^\top \mathbf{a} - \sum_{j \in [Q] \setminus \{j^*\}} \mathbf{z}_j k_j}{\mathbf{z}_{j^*}},$$

   which is equal to k and well defined by above observations.

Clearly, the running time of $\mathcal{B}$ is dominated by the running time of $\mathcal{A}$, and $\mathcal{B}$ simulates the game perfectly for $\mathcal{B}$, unless the guessed index j^* was not correct, and $\mathcal{B}$ has to abort. Until such an abort, the view of $\mathcal{B}$ does not depend on j^*. Further, if $\mathsf{Win} \wedge \neg\mathsf{Zero}$ occurs, then the probability that the guess is correct is at least $1/Q$, and we have

$$\Pr_{\mathfrak{G}}[\mathsf{Win} \wedge \neg\mathsf{Zero}] \leq Q \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda).$$

It remains to bound the probability of event Zero. For that, we use a sequence of games $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$.

Game $\mathbf{G}_0$: This game is as game $\mathfrak{G}$ in the lemma, but it outputs 1 if event Zero occurs. By definition, if Zero occurs, then there is some $j \in [Q] \setminus \mathcal{L}_C$ such that

$$(\mathbf{D}_j \neq \mathbf{0} \vee \mathbf{d}_j \neq 0) \wedge c \cdot \mathbf{D}_j^\top \mathbf{a} - \mathbf{d}_j = 0.$$

We have

$$\Pr[\mathbf{G}_0 \to 1] = \Pr_{\mathfrak{G}}[\mathsf{Zero}].$$

Game $\mathbf{G}_1$: Game $\mathbf{G}_1$ is as $\mathbf{G}_0$, with an additional abort. Namely, the game aborts if $\mathcal{A}$ submits two representations of the same group element that differ for some non-corrupted basis element. More formally, we define the event $\mathsf{RepAmbig}$ and let the game abort if it occurs. The event is as follows.

- Event $\mathsf{RepAmbig}$: This event occurs, if the algebraic algorithm $\mathcal{A}$ ever submits a group element $X \in \mathbb{G}$ (to oracle $\hat{\mathsf{H}}$, as part of $R, \mathsf{pk}, \mathsf{auxpk}$, or as part of R) twice with two representations $(\alpha^{(X)}, (\delta_j^{(X)})_{j=1}^Q)$ and $(\bar{\alpha}^{(X)}, (\bar{\delta}_j^{(X)})_{j=1}^Q)$ such that $g^{\alpha^{(X)}} \prod_{j=1}^Q K_j^{\delta_j^{(X)}} = X = g^{\bar{\alpha}^{(X)}} \prod_{j=1}^Q K_j^{\bar{\delta}_j^{(X)}}$, and there is some $j \in [Q] \setminus \mathcal{L}_C$ such that $\delta_j^{(X)} \neq \bar{\delta}_j^{(X)}$.

We can easily bound the probability of $\mathsf{RepAmbig}$ using the DLOG assumption. The reduction $\mathcal{B}'$ is very similar to the reduction $\mathcal{B}$ described above, and we only sketch it. It gets as input a DLOG challenge $\mathbb{G}, p, g, h = g^k$ and aims to find $k \in \mathbb{Z}_p$. Then, it embeds h into a randomly selected oracle

query j^* for oracle O_I as $\mathcal{B}$ described above does. If $\mathsf{RepAmbig}$ occurs and $\delta_{j^*}^{(X)} \neq \bar{\delta}_{j^*}^{(X)}$, the reduction can compute k as $k = (\alpha^{(X)} - \bar{\alpha}^{(X)} + \sum_{j \in [Q] \setminus \{j^*\}} k_j(\delta_j^{(X)} - \bar{\delta}_j^{(X)}))/(\bar{\delta}_{j^*}^{(X)} - \delta_{j^*}^{(X)})$. We have

$$|\Pr[\mathbf{G}_0 \rightarrow 1] - \Pr[\mathbf{G}_1 \rightarrow 1]| \leq \Pr[\mathsf{RepAmbig}] \leq Q \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B}',\mathsf{GGen}}(\lambda).$$

Game $\mathbf{G}_2$: Game $\mathbf{G}_2$ is as $\mathbf{G}_1$, with an additional abort. Before we introduce the event that triggers the abort, we introduce more notation. Namely, consider a random oracle query of the form $\hat{\mathsf{H}}(\mathsf{R}, idx)$ submitted by $\mathcal{A}$, where $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ is a list of pairs of group elements. As $\mathcal{A}$ is algebraic, it also submits representations $(\alpha^{(\mathsf{pk}_i)}, (\delta_j^{(\mathsf{pk}_i)})_{j \in [Q]})$ and $(\alpha^{(\mathsf{auxpk}_i)}, (\delta_j^{(\mathsf{auxpk}_i)})_{j \in [Q]})$ for each $i \in [N]$ such that

$$\mathsf{pk}_i = g^{\alpha^{(\mathsf{pk}_i)}} \prod_{j=1}^{Q} K_j^{\delta_j^{(\mathsf{pk}_i)}}, \quad \mathsf{auxpk}_i = g^{\alpha^{(\mathsf{auxpk}_i)}} \prod_{j=1}^{Q} K_j^{\delta_j^{(\mathsf{auxpk}_i)}}.$$

Now, given a fixed query as above, say the lth, and fixed i, j as above, we define $\hat{\mathbf{d}}_{l,i,j} := (\delta_j^{(\mathsf{pk}_i)}, \delta_j^{(\mathsf{auxpk}_i)})^\top \in \mathbb{Z}_p^2$. Game $\mathbf{G}_2$ aborts if the following event occurs.

- Event $\mathsf{ROProdZero}$: This event occurs, if there is a query $l \in [Q_{\hat{\mathsf{H}}}]$ of the form $\hat{\mathsf{H}}(\mathsf{R}, idx)$ for $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$, that query outputs some $\mathbf{a}_l \in \mathbb{Z}_p^2$, and there are $i \in [N]$ and $j \in [Q] \setminus \mathcal{L}_C$ with
$$\hat{\mathbf{d}}_{l,i,j} \neq \mathbf{0} \wedge \hat{\mathbf{d}}_{l,i,j}^\top \mathbf{a}_l = 0.$$

We bound the probability of $\mathsf{ROProdZero}$ as follows. First, fix some l, i, j as in the definition of the event. Then, we can consider two cases. In the first case, the hash value on the lth query is not yet defined. Then this means that vector $\hat{\mathbf{d}}_{l,i,j} \in \mathbb{Z}_p^2 \setminus \{\mathbf{0}\}$ is fixed before $\mathbf{a}_l$ is sampled. As $\hat{\mathbf{d}}_{l,i,j}$ is non-zero, its kernel has dimension at most 1. Thus, the probability that $\hat{\mathbf{d}}_{l,i,j}^\top \mathbf{a}_l = 0$ is at most $p/p^2 = 1/p$. In the second case, the hash value on the lth query is already defined. Say it has been defined in query $l' < l$, and so $\mathbf{a}_l = \mathbf{a}_{l'}$. Then we know that $\hat{\mathbf{d}}_{l,i,j} = \hat{\mathbf{d}}_{l',i,j}$ as otherwise $\mathbf{G}_1$ would have aborted. Thus, if $\hat{\mathbf{d}}_{l,i,j}^\top \mathbf{a}_l = 0$, then $\hat{\mathbf{d}}_{l',i,j}^\top \mathbf{a}_{l'} = 0$ and we are in the first case. Finally, we can use a union bound over all l, i, j as above, and get

$$|\Pr[\mathbf{G}_1 \rightarrow 1] - \Pr[\mathbf{G}_2 \rightarrow 1]| \leq \Pr[\mathsf{ROProdZero}] \leq \frac{Q Q_{\hat{\mathsf{H}}} N_{\max}}{p}.$$

Next, we bound the probability that $\mathbf{G}_2$ outputs 1 using a statistical argument and by distinguishing two cases. For that let (s, R, idx) be the final output of $\mathcal{A}$, and define $l^* \in [Q_{\hat{\mathsf{H}}}]$ to be the index of the first query $\hat{\mathsf{H}}(\mathsf{R}, idx)$. Note that if $\mathcal{A}$ never makes that query, the game does. We introduce the following events.

- Event RFirst: This event occurs, if query l^* occurred after $\mathcal{A}$ received c.
- Event $\mathsf{HashFirst}$: This event occurs, if query l^* occurred before $\mathcal{A}$ received c.

Fix a $j \in [Q] \setminus \mathcal{L}_C$ for the rest of the proof. If Zero_j occurs, then by definition

$$(\mathbf{D}_j \neq \mathbf{0} \vee \mathbf{d}_j \neq 0) \wedge c \cdot \mathbf{D}_j^\top \mathbf{a} - \mathbf{d}_j = 0.$$

If $\mathbf{D}_j = \mathbf{0}$ and $c \cdot \mathbf{D}_j^\top \mathbf{a} - \mathbf{d}_j = 0$, we must have $\mathbf{d}_j = 0$. Therefore, event Zero_j implies that

$$\mathbf{D}_j \neq \mathbf{0} \wedge c \cdot \mathbf{D}_j^\top \mathbf{a} = \mathbf{d}_j.$$

Now, if the game outputs 1 and event RFirst occurs, then this means $c \cdot \mathbf{D}_j \neq \mathbf{0}$ and $\mathbf{d}_j$ were fixed before $\mathbf{a}$ was sampled uniformly at random. Thus the probability that this happens is at most $1/p$ for each fixed hash query of $\mathcal{A}$ that may have returned $\mathbf{a}$. This is because if $c \cdot \mathbf{D}_j \neq \mathbf{0}$, then the kernel of $c \cdot \mathbf{D}_j$ has dimension at most 1, and therefore at most p of the p^2 possible $\mathbf{a}$'s lead to $c \cdot \mathbf{D}_j^\top \mathbf{a} = \mathbf{d}_j$. Therefore, we get that the probability that $\mathbf{G}_2$ outputs 1 and RFirst is at most $Q_{\hat{\mathsf{H}}}/p$. On the other hand, if the game outputs 1 and event $\mathsf{HashFirst}$ occurs, then this means that $\mathbf{D}_j^\top \mathbf{a}$ and $\mathbf{d}_j$ were fixed before c was sampled uniformly at random. Also, note that due to the abort introduced in $\mathbf{G}_1$, we can assume that $\mathbf{D}_j = \hat{\mathbf{d}}_{l^*,i,j}$ and $\mathbf{a} = \mathbf{a}_{l^*}$, for some i and our fixed $j \in [Q] \setminus \mathcal{L}_C$. Thus, due to the abort introduced in $\mathbf{G}_2$ and the fact that $\mathbf{D}_j \neq \mathbf{0}$, we know that $0 \neq \hat{\mathbf{d}}_{l^*,i,j}^\top \mathbf{a}_{l^*} = \mathbf{D}_j^\top \mathbf{a}$. Thus, $\mathbf{D}_j^\top \mathbf{a} \neq 0$ and $\mathbf{d}_j$ were fixed before c was sampled uniformly at random, which means that the game outputs 1 with probability at most $1/p$. In summary, we bounded the probability that $\mathbf{G}_2$ outputs 1 by

$$\Pr\left[\mathbf{G}_2 \rightarrow 1\right] \leq \Pr\left[\mathbf{G}_2 \rightarrow 1 \wedge \mathsf{RFirst}\right] + \Pr\left[\mathbf{G}_2 \rightarrow 1 \wedge \mathsf{HashFirst}\right] \leq \frac{Q Q_{\hat{\mathsf{H}}} + Q}{p}.$$

□

Lemma 16. *Let* $\mathsf{H}_p, \mathsf{H}_h \colon \{0,1\}^* \rightarrow \mathbb{G}, , \mathsf{H}_c \colon \{0,1\}^* \rightarrow \mathbb{Z}_p, \mathsf{H}_a \colon \{0,1\}^* \rightarrow \mathbb{Z}_p^2$ *be random oracles. If the* DLOG *assumption holds relative to* GGen*, then* $(\mathsf{LRS}, \mathsf{KDS})$ *(given in Figures 7 and 9) satisfies key knowledge soundness in the algebraic group model. Concretely, there is a PPT algorithm* $\mathsf{Ext}_{\mathsf{LRS}}$*, such that for any algebraic PPT algorithm* $\mathcal{A}$ *that makes at most* $Q_{ni}, Q_S, Q_{\mathsf{H}_p}, Q_{\mathsf{H}_h}, Q_{\mathsf{H}_c}, Q_{\mathsf{H}_a}, Q_{\mathrm{O}}$ *queries to oracles* NewIdentity, Sign, $\mathsf{H}_p, \mathsf{H}_h, \mathsf{H}_c, \mathsf{H}_a$, O*, respectively, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{k\text{-}kn\text{-}sound}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) \leq\ & \frac{(2+Q_{\mathsf{H}_a})Q_{ni} + Q_{\mathsf{H}_p} + Q_{\mathsf{H}_h} + (Q_S+1)Q_{\mathsf{H}_c}}{p} \\ &+ \frac{(2Q_{\mathsf{H}_p}Q_{\mathsf{H}_c}^2 + Q_{ni} + 1)Q_{\mathsf{H}_a}N_{max} + 3}{p} \\ &+ (3 + 2Q_{ni}) \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda).\end{aligned}$$

Here, N_{max} *is the maximum size of key rings* R *that are submitted to random oracles* $\mathsf{H}_a, \mathsf{H}_c$*.*

Proof. We first introduce terminology, then describe the extractor $\mathsf{Ext}_{\mathsf{LRS}}$, and then analyze the game $\mathbf{K\text{-}KN\text{-}SOUND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda)$ for any algebraic PPT algorithm $\mathcal{A}$. Recall that in this game, such an adversary $\mathcal{A}$ gets as input system parameters par. It gets access to random oracles $\mathsf{H}_a, \mathsf{H}_c, \mathsf{H}_h$, H_p, oracles NewIdentity, Corr, Sign, and an oracle O. Throughout the game, $\mathcal{A}$ receives the following group elements:

- Group elements contained in the system parameters par. This is only the generator $g \in \mathbb{G}$.
- Group elements returned by random oracles H_p and H_h. Throughout this proof, we let $\mathcal{L}_{\mathrm{H}}$ be the set of group elements $h \in \mathbb{G}$ that $\mathcal{A}$ received from these oracles.
- Pairs of group elements $\mathsf{ipk} = (K_v, K_s) \in \mathbb{G}^2$ returned by oracle NewIdentity. We denote the set of all such group elements K_s by $\mathcal{L}_{\mathrm{I}}$.
- Pairs of group elements $(\tilde{K}_1, \tilde{K}_2)$ that are contained in signatures $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$ returned by oracle Sign.

This means whenever an algebraic algorithm $\mathcal{A}$ outputs a group element $X \in \mathbb{G}$, it also outputs exponents in $\mathbb{Z}_p$ to represent X in terms of these basis elements.

Preprocessed Representation. Before we continue, we introduce an initial preprocessing of this representation that the extractor will do. The idea is to simplify the representation by removing certain parts, and introducing artificial parts to have a unified format. This is done using the following rules:

- Note that oracle NewIdentity returns $\mathsf{ipk} = (K_v, K_s) \in \mathbb{G}^2$ but also $\mathsf{ivk} = k_v$ such that $K_v = g^{k_v}$. Therefore, whenever a representation contains an exponent $x \in \mathbb{Z}_p$ for K_v, the extractor (knowing k_v by inspecting oracle interfaces) can remove this part of the representation and instead add $x \cdot k_v$ to the exponent of g.
- When the adversary receives group elements $\tilde{K}_1, \tilde{K}_2$ from Sign, then by definition we have $\tilde{K}_2 = h^{\mathsf{auxsk}}$ for some $h \in \mathcal{L}_{\mathrm{H}}$. Note that the extractor can identify this h, and also knows auxsk, as it is part of the input of Sign. Therefore, the extractor can remove any such part of the representation as well.
- When the adversary receives group elements $\tilde{K}_1, \tilde{K}_2$ from a query $\textsc{Sign}(\mathsf{pk}, \mathsf{pseed}, \mathsf{tag}, \mathsf{R}, \mathsf{auxsk}, \mathsf{m})$, then recall that Sign first found some $(\mathsf{ipk} = (K_v, K_s), \mathsf{ivk})$ returned by NewIdentity such that $\mathsf{Track}(\mathsf{ipk}, \mathsf{ok}, \mathsf{pk}, \mathsf{tag}) = 1$ for $\mathsf{ok} := \mathsf{RecDecaps}(\mathsf{ivk}, \mathsf{pseed})$. Note that the extractor has all information to do this as well, because it can observe ivk as NewIdentity outputs it. Then, the oracle computes $\mathsf{sk} := \mathsf{H}(\mathsf{ok}, \mathsf{tag}) + k_s$, where $k_s \in \mathbb{Z}_p$ is such that $K_s = g^{k_s}$. Finally, $\tilde{K}_1$ is defined as $h^{\mathsf{sk}} = h^{\mathsf{H}(\mathsf{ok},\mathsf{tag})} \cdot h^{k_s}$ for some $h \in \mathcal{L}_{\mathrm{H}}$ that can be identified by the extractor. As the extractor can compute $\mathsf{H}(\mathsf{ok}, \mathsf{tag})$, it can translate the representation containing $\tilde{K}_1$ into a representation containing h^{k_s} as a new basis.
- Note that due to the previous rule, the representation will now contain h^{k_s} for some $h \in \mathcal{L}_{\mathrm{H}}$ and some $K_s = g^{k_s} \in \mathcal{L}_{\mathrm{I}}$. To simplify notation, we assume that the representation contains such a basis element for every such pair. This is justified by setting the appropriate exponents to zero. We define the notation

$$\forall h \in \mathcal{L}_{\mathrm{H}} : \ \forall K = g^k \in \mathcal{L}_{\mathrm{I}} : \ (h \diamond K) := h^k.$$

As a result of these rule, whenever an algebraic algorithm $\mathcal{A}$ outputs a group element $X \in \mathbb{G}$, and the extractor already preprocessed the representation, the extractor now knows exponents $\alpha^{(X)}, \gamma_h^{(X)}$ (for each $h \in \mathcal{L}_{\mathrm{H}}$), $\delta_K^{(X)}$ (for each $K \in \mathcal{L}_{\mathrm{I}}$), and $\eta_{h,K}^{(X)}$ (for each $h \in \mathcal{L}_{\mathrm{H}}, K \in \mathcal{L}_{\mathrm{I}}$) such that

$$X = g^{\alpha^{(X)}} \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(X)}} \prod_{K \in \mathcal{L}_{\mathrm{I}}} K^{\delta_K^{(X)}} \prod_{h \in \mathcal{L}_{\mathrm{H}}, K \in \mathcal{L}_{\mathrm{I}}} (h \diamond K)^{\eta_{h,K}^{(X)}}.$$

From now on for the rest of the proof, we will only talk about preprocessed representations. Without loss of generality, we can assume that this representation is unique. This is because the extractor can just ignore a second representation and only consider the first representation that the adversary submitted.

Terminology. Before we describe $\mathsf{Ext}_{\mathsf{LRS}}$, we introduce some terminology. For that, we consider a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$, where $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ and $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$. As in the verification algorithm, define $((W_i)_{i=1}^N, \tilde{W}) := \mathsf{AggPub}(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$.

- For any element group element $X \in \mathbb{G}$ output by $\mathcal{A}$, we say that the representation of X is *pure*, if for all $K \in \mathcal{L}_{\mathrm{I}}$ and all $h \in \mathcal{L}_{\mathrm{H}}$ we have $\delta_K^{(X)} = 0$ and $\eta_{h,K}^{(X)} = 0$. That is, an element X has a pure representation, if its representation has the form

$$X = g^{\alpha^{(X)}} \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(X)}}.$$

- We define the *dominant index* of $(\mathsf{R}, \mathsf{m}, \sigma)$ to be the unique $i \in [N]$ for which the query $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R_i, \tilde{R}_i)$ for $h_i := \mathsf{H}_p(\mathsf{pk}_i)$ and $R_i := g^{s_i} W_i^{c_i}$, $\tilde{R}_i := h_i^{s_i} \tilde{W}^{c_i}$ occurred first among all such queries made by the adversary. To emphasize, this does not consider queries made by oracle Sign (as these can not be observed by the extractor anyways). Recall that if $\mathcal{A}$ made this query, then it also submitted a representation of R_i and $\tilde{R}_i$. Also, if $\mathcal{A}$ never made such a query, then the extractor can run Ver, and gets a representation of R_i and $\tilde{R}_i$ using the representations of $\mathsf{pk}_i, \mathsf{auxpk}_i$ and $\tilde{K}_1, \tilde{K}_2$, and the definition of W_i and $\tilde{W}$. Note that the extractor can always identify the dominant index by identifying the query $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R_i, \tilde{R}_i)$ as above. We highlight that the extractor is only executed if (R, m) have never been signed by oracle Sign. Therefore, Sign never makes the query $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R_i, \tilde{R}_i)$, and therefore the first such query is made by $\mathcal{A}$, or the extractor itself while running Ver.

- Let i^* be the dominant index of a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ that $\mathcal{A}$ submitted to O. Let $h_{i^*} := \mathsf{H}_p(\mathsf{pk}_{i^*})$. We say that the dominant index of $(\mathsf{R}, \mathsf{m}, \sigma)$ has *admissible R's*, if the representations of $R_{i^*}, \tilde{R}_{i^*}$ are pure, and for these representations

$$R_{i^*} = g^{\alpha^{(R_{i^*})}} \cdot \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(R_{i^*})}}, \quad \tilde{R}_{i^*} = g^{\alpha^{(\tilde{R}_{i^*})}} \cdot h_{i^*}^{\gamma_{h_{i^*}}^{(\tilde{R}_{i^*})}} \prod_{h \in \mathcal{L}_{\mathrm{H}} \setminus \{h_{i^*}\}} h^{\gamma_h^{(R_{i^*})}},$$

  it holds that

$$\left(\forall h \in \mathcal{L}_{\mathrm{H}} : \gamma_h^{(R_{i^*})} = 0\right) \wedge \alpha^{(\tilde{R}_{i^*})} = 0 \wedge \left(\forall h \in \mathcal{L}_{\mathrm{H}} \setminus \{h_{i^*}\} : \gamma_h^{(\tilde{R}_{i^*})} = 0\right) \wedge \alpha^{(R_{i^*})} = \gamma^{(\tilde{R}_{i^*})}.$$

  In other words, the dominant index of $(\mathsf{R}, \mathsf{m}, \sigma)$ has admissible R's, if R_{i^*} and $\tilde{R}_{i^*}$ are represented as $R_{i^*} = g^{r_{i^*}}$, $\tilde{R}_{i^*} = h_{i^*}^{r_{i^*}}$ for some $r_{i^*} \in \mathbb{Z}_p$.

- Let i^* be the dominant index of a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ that $\mathcal{A}$ submitted to O. Assume pure representations

$$\mathsf{pk}_{i^*} = g^{\alpha^{(\mathsf{pk}_{i^*})}} \cdot \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(\mathsf{pk}_{i^*})}}, \quad \mathsf{auxpk}_{i^*} = g^{\alpha^{(\mathsf{auxpk}_{i^*})}} \cdot \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(\mathsf{auxpk}_{i^*})}}$$

  that $\mathcal{A}$ submitted along with $\mathsf{pk}_{i^*}, \mathsf{auxpk}_{i^*}$. If the representations are not pure, or there is an $h \in \mathcal{L}_{\mathrm{H}}$ such that $\gamma_h^{(\mathsf{pk}_{i^*})} \neq 0$ or $\gamma_h^{(\mathsf{auxpk}_{i^*})} \neq 0$, we say that the dominant index of $(\mathsf{R}, \mathsf{m}, \sigma)$ is *badly represented*. In other words, the dominant index of $(\mathsf{R}, \mathsf{m}, \sigma)$ is badly represented, if pk_{i^*} and auxpk_{i^*} are represented differently than $\mathsf{pk}_{i^*} = g^{\alpha^{(\mathsf{pk}_{i^*})}}$ and $\mathsf{auxpk}_{i^*} = g^{\alpha^{(\mathsf{auxpk}_{i^*})}}$.

Extractor. The extractor simply checks whether the dominant index of a submitted tuple is not badly represented and has admissible R's. If so, it can extract trivially from the representation. More formally, whenever $\mathcal{A}$ submits a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ with $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ to oracle O, extractor $\mathsf{Ext}_{\mathsf{LRS}}$ works as follows:

1. If $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 0$, return $\perp$.
2. If for some query $x \in \{0,1\}^*$ made by $\mathcal{A}$ or in Ver, we have $\mathsf{H}_c(x) = 0$, or $\mathsf{H}_h(x) = g^0$, or $\mathsf{H}_p(x) = g^0$, then return $\perp$.
3. Let i^* be the dominant index of $(\mathsf{R}, \mathsf{m}, \sigma)$. If the (preprocessed) representations of pk_{i^*} or auxpk_{i^*} are not pure, return $\perp$.
4. Otherwise, let

$$\mathsf{pk}_{i^*} = g^{\alpha^{(\mathsf{pk}_{i^*})}} \cdot \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(\mathsf{pk}_{i^*})}}, \quad \mathsf{auxpk}_{i^*} = g^{\alpha^{(\mathsf{auxpk}_{i^*})}} \cdot \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(\mathsf{auxpk}_{i^*})}}$$

   be the pure representations that $\mathcal{A}$ submitted along with $\mathsf{pk}_{i^*}, \mathsf{auxpk}_{i^*}$.
5. If the dominant index of $(\mathsf{R}, \mathsf{m}, \sigma)$ is badly represented or does not have admissible R's, then return $\perp$.
6. Otherwise, set $\mathsf{sk}_{i^*} := \alpha^{(\mathsf{pk}_{i^*})}$, $\mathsf{auxsk}_{i^*} := \alpha^{(\mathsf{auxpk}_{i^*})}$ and return $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*})$.

Analysis. Next, we analyze the key knowledge soundness game with respect to $\mathsf{Ext}_{\mathsf{LRS}}$ as defined above and an algebraic PPT algorithm $\mathcal{A}$. Informally, our strategy is as follows:

1. Consider a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ submitted to O.
2. We use the honest-verifier zero-knowledge property of the scheme, and patching of the random oracle, we simulate the signing oracle. As a result, we only need the secret keys for corruption queries and to compute the $\tilde{K}_1$-part of signatures.
3. Using a statistical argument, we show that there has to be some $w \in \mathbb{Z}_p$ such that $W_{i^*} = g^w$ and $\tilde{W} = \mathsf{H}_p(\mathsf{pk}_{i^*})^w$, where i^* is the dominant index of the tuple.
4. Using the DLOG assumption, we argue that involved $R_{i^*}, \tilde{R}_{i^*}$ and $\mathsf{pk}_{i^*}, \mathsf{auxpk}_{i^*}$ have pure representations. This will exploit the fact that random oracle queries of the signing oracle and submitted signatures are distinct.
5. Once this is established, we use another reduction to the DLOG assumption, to show that the values R_{i^*} and $\tilde{R}_{i^*}$ are represented as $R_{i^*} = g^{r_{i^*}}$ and $\tilde{R}_{i^*} = \mathsf{H}_p(\mathsf{pk}_{i^*})^{r_{i^*}}$, i.e., the dominant index has admissible R's.
6. Using this representation, we show how to efficiently compute the aforementioned w. Then, we argue that the dominant index can not be badly represented, as otherwise, we would have two different representations of W_{i^*}, violating the DLOG assumption.

We now make this strategy more formal using a sequence of games and bad events.

Game $\mathbf{G}_0$: We define game $\mathbf{G}_0$ to be the game $\textbf{K-KN-SOUND}^{\mathcal{A}}_{\mathsf{LRS},\mathsf{KDS},\mathsf{Ext}_{\mathsf{LRS}}}(\lambda)$ for algebraic adversary $\mathcal{A}$ and $\mathsf{Ext}_{\mathsf{LRS}}$ as above. The game outputs 1 if $\mathcal{A}$ submits a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ to oracle O, such that extraction fails. By definition, we have

$$\mathsf{Adv}^{\text{k-kn-sound}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) = \Pr\left[\mathbf{G}_0 \to 1\right].$$

Game $\mathbf{G}_1$: We define game $\mathbf{G}_1$ to be as $\mathbf{G}_0$, but additionally let the game abort if one of the following bad events occur:

- Event $\mathsf{KeyZero}$: This event occurs, if some $K \in \mathcal{L}_{\mathrm{I}}$ satisfies $K = g^0$.
- Event ROZero: This event occurs, if for some query $x \in \{0,1\}^*$, we have $\mathsf{H}_c(x) = 0$, or $\mathsf{H}_h(x) = g^0$, or $\mathsf{H}_p(x) = g^0$.
- Event BadA: This event occurs, if $\mathcal{A}$ queries $\mathsf{H}_a(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$ for $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$, this query returns $(a_1, a_2) \in \mathbb{Z}_p^2$, and there is some $i \in [N]$ such that

$$\boldsymbol{\Gamma}_i \neq \mathbf{0} \wedge \boldsymbol{\Gamma}_i \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \mathbf{0} \text{ for } \boldsymbol{\Gamma}_i := \begin{pmatrix} \gamma_{h_1}^{(\mathsf{pk}_i)} & \gamma_{h_1}^{(\mathsf{auxpk}_i)} \\ \vdots & \vdots \\ \gamma_{h_{|\mathcal{L}_{\mathrm{H}}|}}^{(\mathsf{pk}_i)} & \gamma_{h_{|\mathcal{L}_{\mathrm{H}}|}}^{(\mathsf{auxpk}_i)} \end{pmatrix},$$

  where $h_1, \ldots, h_{|\mathcal{L}_{\mathrm{H}}|}$ is an arbitrary ordering of $\mathcal{L}_{\mathrm{H}}$.

Next, we bound the probability of these events. Using a union bound over all queries to NewIdentity, we can bound the probability of $\mathsf{KeyZero}$ by Q_{ni}/p. Using a union bound over all random oracle queries, we can bound the probability of ROZero by $(Q_{\mathsf{H}_p} + Q_{\mathsf{H}_h} + Q_{\mathsf{H}_c})/p$. Further, note that whenever $\mathcal{A}$ makes query $\mathsf{H}_a(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$ as in event BadA for the first time, the matrices $\boldsymbol{\Gamma}_i$ are fixed by the representations that $\mathcal{A}$ provides. Fix such a query, fix some $i \in [N]$, and assume that $\boldsymbol{\Gamma}_i \neq \mathbf{0}$. Then the kernel of $\boldsymbol{\Gamma}_i$ has dimension at most 1. Therefore, the probability that the random vector $(a_1, a_2)^\top \in \mathbb{Z}_p$ is in this kernel is at most $p/p^2 = 1/p$. This holds because, as said, $\boldsymbol{\Gamma}_i$ is fixed before $(a_1, a_2)^\top$ is sampled. Taking a union bound over all i and all queries, we can then bound the probability of BadA. In summary, we have

$$\begin{aligned} |\Pr[\mathbf{G}_0 \to 1] - \Pr[\mathbf{G}_1 \to 1]| &\leq \Pr[\mathsf{KeyZero}] + \Pr[\mathsf{ROZero}] + \Pr[\mathsf{BadA}] \\ &\leq \frac{Q_{ni} + Q_{\mathsf{H}_p} + Q_{\mathsf{H}_h} + Q_{\mathsf{H}_c} + Q_{\mathsf{H}_a} N_{\max}}{p}. \end{aligned}$$

Game $\mathbf{G}_2$: Game $\mathbf{G}_2$ is as $\mathbf{G}_1$, but we change how the signing oracle Sign works. Namely, the oracle replaces the signing algorithm Sig by a signing algorithm Sig'. This algorithm is exactly as Sig, but computes the components $c_1, (s_i)_i$ of the signature by programming the random oracle H_c. Note that key image $\tilde{K}_1$ is still computed using the secret key. Formally, algorithm Sig' is given in Figure 10. It follows from standard honest-verifier zero-knowledge properties that the view of $\mathcal{A}$ does not change from $\mathbf{G}_1$ to $\mathbf{G}_2$, unless the random oracle value that is programmed in Line 12 of Figure 10 is already defined. As the value R_N is distributed uniformly at random, this occurs with probability at most Q_{H_c}/p for every execution of Sig'. Therefore, a union bound of the number of signing queries shows

$$|\Pr[\mathbf{G}_1 \to 1] - \Pr[\mathbf{G}_2 \to 1]| \leq \frac{Q_S Q_{\mathsf{H}_c}}{p}.$$

Game $\mathbf{G}_3$: Game $\mathbf{G}_3$ is as game $\mathbf{G}_2$, but it additionally aborts if the following event occurs:

- Event $\mathsf{BadTrans}$: This event occurs, if $\mathcal{A}$ ever outputs a group element X with representation $X = g^{\alpha^{(X)}} \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\gamma_h^{(X)}} \prod_{K \in \mathcal{L}_{\mathrm{I}}} K^{\delta_K^{(X)}} \prod_{h \in \mathcal{L}_{\mathrm{H}}, K \in \mathcal{L}_{\mathrm{I}}} (h \diamond K)^{\eta_{h,K}^{(X)}}$ such that

$$\left(\exists K \in \mathcal{L}_{\mathrm{I}}, h \in \mathcal{L}_{\mathrm{H}} : \delta_K^{(X)} \neq 0 \vee \eta_{h,K}^{(X)} \neq 0\right) \wedge g^{\delta_K^{(X)}} \prod_{h \in \mathcal{L}_{\mathrm{H}}} h^{\eta_{h,K}^{(X)}} = g^0.$$

We bound the probability of $\mathsf{BadTrans}$ using Lemma 14. Namely, a reduction in the game in Lemma 14 gets g as input, simulates $\mathbf{G}_2$, and outputs $(\delta_K^{(X)})_K, (\eta_{h,K}^{(X)})_{h,K}$ if event $\mathsf{BadTrans}$ occurs. Then, by Lemma 14, there exists a reduction $\mathcal{B}$ such that

$$|\Pr[\mathbf{G}_2 \to 1] - \Pr[\mathbf{G}_3 \to 1]| \leq \Pr[\mathsf{BadTrans}] \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

Game $\mathbf{G}_4$: Game $\mathbf{G}_4$ is as $\mathbf{G}_3$, but additionally aborts if at least one of the following two events occurs:

- Event $\mathsf{NotPure}$: This event occurs, if $\mathcal{A}$ submits a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ with $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_i$ and dominant index i^*, such that $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1$, and one of the representations of $R_{i^*}, \tilde{R}_{i^*}, \mathsf{pk}_{i^*}, \mathsf{auxpk}_{i^*}$ is not pure, where $R_{i^*}, \tilde{R}_{i^*}$ are as in the definition of admissible R's. We denote the event that $\mathsf{NotPure}$ occurs in the jth query to oracle O by $\mathsf{NotPure}_j$.
- Event $\mathsf{NonAdmRs}$: This event occurs, if $\mathcal{A}$ submits a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ with $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1$, and the dominant index of $(\mathsf{R}, \mathsf{m}, \sigma)$ does not have admissible R's. We denote the event that $\mathsf{NonAdmRs}$ occurs in the jth query to oracle O by $\mathsf{NonAdmRs}_j$.

Clearly, it holds that

$$|\Pr[\mathbf{G}_3 \to 1] - \Pr[\mathbf{G}_4 \to 1]| \leq \Pr[\mathsf{NotPure} \vee \mathsf{NonAdmRs}].$$

For the sake of analyzing this probability, we define another event as follows.

- Event $\mathsf{NoGoodW}$: This event occurs, if $\mathcal{A}$ submits a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ with dominant index i^*, and $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ and $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$ to oracle O, such that $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1$ and for $((W_i)_{i=1}^N, \tilde{W}) := \mathsf{AggPub}(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$, we have

  $$\forall w \in \mathbb{Z}_p : \neg(W_{i^*} = g^w \wedge \tilde{W} = \mathsf{H}_p(\mathsf{pk}_{i^*})^w).$$

  We denote the event that $\mathsf{NoGoodW}$ occurs in the jth query to oracle O by $\mathsf{NoGoodW}_j$.

Now, we can write

$$\begin{aligned}\Pr[\mathsf{NotPure} \vee \mathsf{NonAdmRs}] \leq \quad & \Pr[\mathsf{NoGoodW}] \\ & + \Pr[\mathsf{NotPure} \wedge \neg\mathsf{NoGoodW}] \\ & + \Pr[\mathsf{NonAdmRs} \wedge \neg\mathsf{NotPure} \wedge \neg\mathsf{NoGoodW}].\end{aligned}$$

We bound these terms individually in the three claims at the end of the proof, using Lemmas 12, 13 and 15, respectively.

It remains to bound the probability that $\mathbf{G}_4$ outputs 1. For that, assume that $\mathbf{G}_4$ outputs 1. This means that neither of the introduced aborts occurs and extraction fails. By definition of $\mathsf{Ext}_{\mathsf{LRS}}$, it is clear that extraction can only fail if the dominant index i^* of a submitted tuple is badly represented or does not have admissible R's. Due to the bad events ruled out in $\mathbf{G}_4$, the only way this can happen is that the adversary outputs a representation of $\mathsf{pk}_{i^*}, \mathsf{auxpk}_{i^*}$ with an exponent $\gamma_h^{(\mathsf{pk}_{i^*})} \neq 0$ or $\gamma_h^{(\mathsf{auxpk}_{i^*})} \neq 0$ for some $h \in \mathcal{L}_{\mathrm{H}}$. As we will see, this violates the DLOG assumption. More formally, we bound the probability that $\mathbf{G}_4$ outputs 1 using Lemma 11. For that, we give a reduction $\mathcal{B}$ that runs in the game described in Lemma 11, such that the event bounded in Lemma 11 occurs if $\mathbf{G}_4$ outputs 1. Namely, the reduction $\mathcal{B}$ is as follows:

1. The reduction $\mathcal{B}$ gets as input a generator $g \in \mathbb{G}$, and simulates game $\mathbf{G}_4$ for $\mathcal{A}$. Whenever a new element for oracles H_p and H_h has to be sampled, the reduction uses the oracle $\hat{\mathrm{O}}$ (as defined in Lemma 11).

2. When $\mathbf{G}_4$ would output 1 because extraction fails, this means that $\mathcal{A}$ submitted a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ with $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ to oracle O such that $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1$ and the dominant index i^* of $(\mathsf{R}, \mathsf{m}, \sigma)$ is badly represented. Further, it has admissible R's and the representation of $\mathsf{pk}_{i^*}, \mathsf{auxpk}_{i^*}$ is pure, as otherwise $\mathbf{G}_4$ would have aborted. Write $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$. By definition, this means that the adversary output a representation

$$\mathsf{pk}_{i^*} = g^{\alpha(\mathsf{pk}_{i^*})} \cdot \prod_{h \in \mathcal{L}_\mathrm{H}} h^{\gamma_h^{(\mathsf{pk}_{i^*})}}, \quad \mathsf{auxpk}_{i^*} = g^{\alpha(\mathsf{auxpk}_{i^*})} \cdot \prod_{h \in \mathcal{L}_\mathrm{H}} h^{\gamma_h^{(\mathsf{auxpk}_{i^*})}}$$

with $\gamma_h^{(\mathsf{pk}_{i^*})} \neq 0$ or $\gamma_h^{(\mathsf{auxpk}_{i^*})} \neq 0$ for some $h \in \mathcal{L}_\mathrm{H}$. As the dominant index i^* has admissible R's, the reduction knows exponent $r_{i^*} \in \mathbb{Z}_p$ such that $R_{i^*} = g^{r_{i^*}}$ and $\tilde{R}_{i^*} = \mathsf{H}_p(\mathsf{pk}_{i^*})^{r_{i^*}}$. The reduction computes $w := (r_{i^*} - s_{i^*})/c_{i^*}$. Because $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1$, we have $g^{r_{i^*}} = R_{i^*} = g^{s_{i^*}} W_{i^*}^{c_{i^*}}$ and therefore $g^w = W_{i^*}$. Further, the reduction sets $(a_1, a_2) := \mathsf{H}_a(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$ as in algorithm Ver. Then, we have

$$\begin{aligned} g^w = W_{i^*} &= \mathsf{pk}_{i^*}^{a_1} \cdot \mathsf{auxpk}_{i^*}^{a_2} \\ &= g^{a_1 \alpha(\mathsf{pk}_{i^*}) + a_2 \alpha(\mathsf{auxpk}_{i^*})} \cdot \prod_{h \in \mathcal{L}_\mathrm{H}} h^{a_1 \gamma_h^{(\mathsf{pk}_{i^*})} + a_2 \gamma_h^{(\mathsf{auxpk}_{i^*})}}. \end{aligned}$$

Since $\mathbf{G}_1$ we can assume that event BadA does not occur, and therefore for at least one $h \in \mathcal{L}_\mathrm{H}$ the exponent $a_1 \gamma_h^{(\mathsf{pk}_{i^*})} + a_2 \gamma_h^{(\mathsf{auxpk}_{i^*})}$ is non-zero. Thus, $\mathcal{B}$ can output two different representations of g^w to the game in Lemma 11.

Reduction $\mathcal{B}$ perfectly simulates $\mathbf{G}_4$ for $\mathcal{A}$. Further, if $\mathbf{G}_4$ outputs 1 (i.e., extraction fails), then $\mathcal{B}$ computed two different representations of pk_{i^*}. Thus, Lemma 11 implies that there is a reduction $\mathcal{B}_2$ with

$$\Pr[\mathbf{G}_4 \to 1] \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B}_2, \mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

Claim 14. *In game $\boldsymbol{G}_3$, we have*

$$\Pr[\mathsf{NoGoodW}] \leq Q_\mathrm{O} Q_{\mathsf{H}_p} Q_{\mathsf{H}_c}^2 N_{max} \cdot \frac{2 Q_{\mathsf{H}_a}}{p}.$$

To prove the claim, we first use a union bound to get

$$\Pr[\mathsf{NoGoodW}] \leq \sum_{j=1}^{Q_\mathrm{O}} \Pr[\mathsf{NoGoodW}_j].$$

Now, fix some $j \in [Q_\mathrm{O}]$. Then we bound $\Pr[\mathsf{NoGoodW}_j]$ using a reduction $\mathcal{B}$ running in the game of Lemma 12. The idea is to guess the involved random oracle queries. Precisely, the reduction is as follows.

1. Reduction $\mathcal{B}$ gets as input generators $g, h \in \mathbb{G}$, and access to a random oracle $\hat{\mathsf{H}}$.

2. Reduction $\mathcal{B}$ samples the following indices: $i_0 \leftarrow\!\!\$\, [Q_{\mathsf{H}_h}]$, $i_1 \leftarrow\!\!\$\, [Q_{\mathsf{H}_c}]$, $i_2 \leftarrow\!\!\$\, [Q_{\mathsf{H}_c}]$. If $i_1 > i_2$, it aborts.

3. Reduction $\mathcal{B}$ simulates game $\mathbf{G}_3$ for $\mathcal{A}$, while simulating oracles NewIdentity, Corr, Sign as in $\mathbf{G}_2$ and random oracle H_a via forwarding to $\hat{\mathsf{H}}$. It simulates the other random oracles honestly, except the i_0th query to H_h, and the i_1th and i_2th query to H_c. These are simulated as follows:

   - Let the i_0th query to H_h be $\mathsf{H}_h(\mathsf{pk})$. If the hash value is already defined, the reduction aborts. Otherwise, it aborts with probability $1/p$ (cf. event ROZero), and sets the hash value to be h with probability $1 - 1/p$.
   - Let the i_1th query to H_c be $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R, \tilde{R})$. The reduction parses $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$, and samples $i_3 \leftarrow\!\!\$\, [N]$. Then, it outputs its state and $R, \tilde{R}, \mathsf{pk}_{i_3}, \mathsf{auxpk}_{i_3}$ to the game in Lemma 12. It gets back a $c \in \mathbb{Z}_p$. Then, it simulates the answering of the query honestly. Later, when the i_2th query to H_c occurs, and the hash value is already defined, it aborts. If it is not yet defined, it aborts with probability $1/p$ (cf. event ROZero), and sets the hash value to c with probability $1 - 1/p$.

4. When $\mathcal{A}$ submits a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ with dominant index i^*, and $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ and $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$ to oracle O in the jth query, $\mathcal{B}$ first checks if all the guessed indices are correct, i.e.,(using notation as in Ver)

   - The first query $\mathsf{H}_p(\mathsf{pk}_{i^*})$ was the i_0th query to H_p.
   - The first query $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R_{i^*}, \tilde{R}_{i^*})$ was the i_1th query to H_c.
   - The first query of $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R_{i'}, \tilde{R}_{i'})$ was the i_2th query to H_c, where $i' \in [N]$ is such that $i' \mod N + 1 = i^*$.
   - We have $i_3 = i^*$.

   Note that if all guesses are correct in the sense above, then especially the programming that the reduction did for the i_1th and i_2th query to H_c does not conflict with the programming that oracle Sign does as introduced in game $\mathbf{G}_2$. If one of the indices is not guessed correct, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ outputs $s_{i^*}, \tilde{K}_1, \tilde{K}_2, idx := \mathsf{R}$ to the game in Lemma 12 and terminates. If $\mathsf{NoGoodW}_j$ does not occur, $\mathcal{B}$ aborts.

We see that until a potential abort occurs, $\mathcal{A}$'s view is independent of the indices i_0, i_1, i_2, i_3, and $\mathcal{B}$ perfectly simulates $\mathbf{G}_3$ for $\mathcal{A}$. Further, assume that $\mathcal{B}$ samples the correct indices and $\mathsf{NoGoodW}_j$ occurs. In this case, one can easily observe that, by definition of Ver and the event $\mathsf{NoGoodW}_j$, the event that is bounded in Lemma 12 occurs. Therefore, we have

$$\Pr\left[\mathsf{NoGoodW}_j\right] \leq Q_{\mathsf{H}_p} Q_{\mathsf{H}_c}^2 N_{\max} \cdot \frac{2Q_{\mathsf{H}_a}}{p},$$

and the claim is proven.

Claim 15. *In game $\boldsymbol{G}_3$, we have*

$$\Pr\left[\mathsf{NotPure} \wedge \neg\mathsf{NoGoodW}\right] \leq 2Q_{ni} \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B}_0,\mathsf{GGen}}(\lambda) + \frac{(N_{max}+1)Q_{ni}Q_{\mathsf{H}_a} + Q_{ni}}{p},$$

for some reduction $\mathcal{B}_0$ with $\mathbf{T}(\mathcal{B}_0) \approx \mathbf{T}(\mathcal{A})$.

To prove the claim, we first explain the main observations that allow us to use Lemma 15. Assume the representation

$$R_{i^*} = g^{\alpha^{(R_{i^*})}} \cdot \prod_{h \in \mathcal{L}_\mathrm{H}} h^{\gamma_h^{(R_{i^*})}} \cdot \prod_{K \in \mathcal{L}_\mathrm{I}} K^{\delta_K^{(R_{i^*})}} \cdot \prod_{h \in \mathcal{L}_\mathrm{H}, K \in \mathcal{L}_\mathrm{I}} (h \diamond K)^{\eta_{h,K}^{(R_{i^*})}}$$

is not pure. This means that some $\delta_K^{(R_{i^*})}$ or some $\eta_{h,K}^{(R_{i^*})}$ is non-zero. Our reduction to the game in Lemma 15 programs the random oracles such that it knows $\vartheta_h \in \mathbb{Z}_p$ for each $h = g^{\vartheta_h} \in \mathcal{L}_\mathrm{H}$. With this, it transforms above representation into a representation

$$R_{i^*} = g^{\alpha^{(R_{i^*})} + \sum_{h \in \mathcal{L}_\mathrm{H}} \vartheta_h \gamma_h^{(R_{i^*})}} \cdot \prod_{K \in \mathcal{L}_\mathrm{I}} K^{\delta_K^{(R_{i^*})} + \sum_{h \in \mathcal{L}_\mathrm{H}} \vartheta_h \eta_{h,K}^{(R_{i^*})}}.$$

Due to the abort we introduced in game $\mathbf{G}_3$, we know that there is some $K \in \mathcal{L}_\mathrm{I}$ for which the transformed exponent $\delta_K^{(R_{i^*})} + \sum_{h \in \mathcal{L}_\mathrm{H}} \vartheta_h \eta_{h,K}^{(R_{i^*})}$ is non-zero, which is what we need to apply Lemma 15. If the representations of pk_{i^*} or auxpk_{i^*} are not pure, we can do an analogous translation.

Next, assume the representation of $\tilde{R}_{i^*}$ is not pure. Then we need to do an additional translation, because the game in Lemma 15 does not consider such a value $\tilde{R}_{i^*}$. Namely, we show that the reduction can derive a non-pure representation of R_{i^*} in this case, using the assumption that $\mathsf{NoGoodW}$ does not occur. Once we showed this, Lemma 15 can be applied as before. If the event $\mathsf{NoGoodW}$ does not occur, then it is easy to see that

$$\tilde{W} = W_{i^*}^{\vartheta^*} \text{ for } \vartheta^* \in \mathbb{Z}_p \text{ s.t. } \mathsf{H}_p(\mathsf{pk}_{i^*}) = g^{\vartheta^*}.$$

Again, note that the reduction knows ϑ^*. As we assume that the submitted signature is valid, i.e., $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1$, we have (with notations as in Ver)

$$\tilde{R}_{i^*} = \mathsf{H}_p(\mathsf{pk}_{i^*})^{s_{i^*}} \tilde{W}^{c_{i^*}} = g^{s_{i^*}\vartheta^*} W_{i^*}^{c_{i^*}\vartheta^*} = R_{i^*}^{\vartheta^*}.$$

Therefore, the reduction can first transform a non-pure representation of $\tilde{R}_{i^*}$ into a non-pure representation

$$\tilde{R}_{i^*} = g^{\alpha} \cdot \prod_{K \in \mathcal{L}_\mathrm{I}} K^{\delta_K},$$

given by $\alpha, (\delta_K)_{K \in \mathcal{L}_\mathrm{I}}$ with some non-zero δ_K, as done for R_{i^*}. Then, the reduction transform this representation into a non-pure representation

$$R_{i^*} = \tilde{R}_{i^*}^{1/\vartheta^*} = g^{\alpha/\vartheta^*} \cdot \prod_{K \in \mathcal{L}_\mathrm{I}} K^{\delta_K/\vartheta^*}$$

of R_{i^*}. Here, we used that $\vartheta^* \neq 0$, see $\mathbf{G}_1$. We now formally describe the reduction $\mathcal{B}$ that runs in the game defined in Lemma 15, and uses above techniques.

1. $\mathcal{B}$ gets as input generator $g \in \mathbb{G}$, and access to oracles $\hat{\mathsf{H}}, \mathrm{O}_I, \mathrm{O}_C$. It first samples indices $i_1, i_2 \leftarrow\!\!\$\, [Q_{\mathsf{H}_c}]$. If $i_1 > i_2$, $\mathcal{B}$ aborts its execution. Then, $\mathcal{B}$ simulates game $\mathbf{G}_3$ for $\mathcal{A}$, while simulating the oracles NewIdentity, Corr, Sign, and the random oracles $\mathsf{H}_a, \mathsf{H}_p, \mathsf{H}_h, \mathsf{H}_c$ as follows:

   - Random oracles $\mathsf{H}_p, \mathsf{H}_h$ are simulated in the standard lazy way, but such that $\mathcal{B}$ knows for each returned group element $h \in \mathcal{L}_\mathrm{H}$ an exponent $\vartheta_h \in \mathbb{Z}_p$ with $h = g^{\vartheta_h}$. This can easily be achieved by sampling ϑ_h at random and defining $h := g^{\vartheta_h}$ whenever a new hash value is needed.

- Random oracle H_a is simulated by forwarding to random oracle $\hat{\mathsf{H}}$. Before forwarding group elements to $\hat{\mathsf{H}}$, the reduction first translates them into a suitable representation, exactly as it is explained for the values R_{i^*} above. That is, it removes all components with basis h or $(h \diamond K)$ for all $h \in \mathcal{L}_{\mathrm{H}}, K \in \mathcal{L}_{\mathrm{I}}$ using the values ϑ_h.
- Random oracle H_c is simulated honestly, except the i_1th and the i_2th query. These are simulated as follows: Let the i_1th query to H_c be $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R, \tilde{R})$. The reduction parses $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$, and samples $i_3 \leftarrow\!\!\$\, [N]$. As in the above discussion, it can translate the representations of $R, \tilde{R}, \mathsf{pk}_{i_3}, \mathsf{auxpk}_{i_3}$ into representations with basis g and $K, K \in \mathcal{L}_{\mathrm{I}}$. If all four representations are pure, it aborts the execution. If one of these representations for $R, \mathsf{pk}_{i_3}, \mathsf{auxpk}_{i_3}$ is not pure, it outputs its state, and these representations of $R, \mathsf{pk}_{i_3}, \mathsf{auxpk}_{i_3}$ to the game in Lemma 15. If only the representation of $\tilde{R}$ is not pure, then it first identifies ϑ^* such that $\mathsf{H}_p(\mathsf{pk}_{i_3}) = g^{\vartheta^*}$, and translates the non-pure representation of $\tilde{R}$ into a non-pure representation of R (as explained above), and then outputs this representation of R and the representations of $\mathsf{pk}_{i_3}, \mathsf{auxpk}_{i_3}$ to the game in Lemma 15. In either case, it gets back a $c \in \mathbb{Z}_p$. Then, it simulates the answering of the query honestly. Later, when the i_2th query to H_c occurs, and the hash value is already defined, it aborts. If it is not yet defined, it aborts with probability $1/p$ (cf. event ROZero), and sets the hash value to c with probability $1 - 1/p$.
- Oracle queries to NewIdentity are answered as in $\mathbf{G}_3$, but K_s is obtained from a query to oracle O_I. Corresponding oracle queries to Corr are answered by appropriately forwarding to oracle O_C.
- Oracle queries to Sign are answered as in $\mathbf{G}_3$, with the following change. Recall that $\mathbf{G}_3$, the oracle first computes a secret key $\mathsf{sk} = \mathsf{H}(\mathsf{ok}, \mathsf{tag}) + k$, where $\mathsf{ok}, \mathsf{tag}$ are part of the oracle's input, and k is an identity secret key for some identity ipk, i.e., there is some $K \in \mathcal{L}_{\mathrm{I}}$ such that $K = g^k$. The reduction $\mathcal{B}$ first identifies this K (by checking $\mathsf{sk} = \mathsf{H}(\mathsf{ok}, \mathsf{tag}) + k$ in the exponent of g). Further, recall that the oracle in $\mathbf{G}_3$ then uses algorithm Sig' given in Figure 10 to compute signatures, and that this algorithm makes use of sk only in Line 05 to compute the key image $\tilde{K}_1 = h_{i^*}^{\mathsf{sk}}$, where $h_{i^*} \in \mathcal{L}_{\mathrm{H}}$. The reduction $\mathcal{B}$ instead computes this value without using k. This is done using

$$\tilde{K}_1 = h_{i^*}^{\mathsf{sk}} = h_{i^*}^{\mathsf{H}(\mathsf{ok},\mathsf{tag})} \cdot h_{i^*}^{k} = h_{i^*}^{\mathsf{H}(\mathsf{ok},\mathsf{tag})} \cdot K^{\vartheta_{h_{i^*}}} .$$

2. Whenever $\mathcal{A}$ submits $(\mathsf{R}, \mathsf{m}, \sigma)$ with dominant index i^*, and $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ and $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$ to oracle O, $\mathcal{B}$ first checks if event $\mathsf{NotPure} \wedge \neg\mathsf{NoGoodW}$ occurs in this query. Note that it can efficiently check $\mathsf{NoGoodW}$ using knowledge of the discrete logarithm of $\mathsf{H}_p(\mathsf{pk}_{i^*})$. If this event does not occur, it continues as in $\mathbf{G}_3$ to simulate the oracle. If it occurs, $\mathcal{B}$ checks if all the guessed indices are correct, i.e., (using notation as in Ver)

   - The first query $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R_{i^*}, \tilde{R}_{i^*})$ was the i_1th query to H_c.
   - The first query of $\mathsf{H}_c(\mathsf{R}, \mathsf{m}, R_{i'}, \tilde{R}_{i'})$ was the i_2th query to H_c, where $i' \in [N]$ is such that $i' \mod N + 1 = i^*$.
   - We have $i_3 = i^*$.

   Note that if all guesses are correct in the sense above, then especially the programming that the reduction did for the i_1th and i_2th query to H_c does not conflict with the programming that oracle Sign does as introduced in game $\mathbf{G}_2$. If one of the indices is not guessed correct, $\mathcal{B}$ aborts. Otherwise, it outputs s_{i^*}, R, and $idx := (\tilde{K}_1, \tilde{K}_2)$ to the game in Lemma 15.

Until a potential abort occurs, $\mathcal{B}$ perfectly simulates $\mathbf{G}_3$ for $\mathcal{A}$, and the view of $\mathcal{A}$ is independent of i_1, i_2, i_3. If these indices are guessed correctly and the event $\mathsf{NotPure} \wedge \neg\mathsf{NoGoodW}$ occurs, then $\mathcal{B}$ triggers the event that is bounded in Lemma 15, which follows from above discussion. By Lemma 15, we therefore have a reduction $\mathcal{B}_0$ with

$$\Pr\left[\mathsf{NotPure} \wedge \neg\mathsf{NoGoodW}\right] \leq 2Q_{ni} \cdot \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B}_0,\mathsf{GGen}}(\lambda) + \frac{(N_{\max}+1)Q_{ni}Q_{\mathsf{H}_a} + Q_{ni}}{p}.$$

Claim 16. *In game $\boldsymbol{G}_3$, we have*

$$\Pr\left[\mathsf{NonAdmRs} \wedge \neg\mathsf{NotPure} \wedge \neg\mathsf{NoGoodW}\right] \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B}_1,\mathsf{GGen}}(\lambda) + \frac{1}{p},$$

for some reduction $\mathcal{B}_1$ with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A})$.

To prove the claim, we bound the probability using Lemma 13. Namely, we sketch a reduction $\mathcal{B}$ running in the game defined in Lemma 13.

1. Reduction $\mathcal{B}$ gets as input g, h, and access to an oracle $\hat{\mathrm{O}}$.
2. It samples a random $i_0 \leftarrow\!\!\$\, [Q_{\mathsf{H}_p}]$ and uses h as the hash value in the i_0th query to H_p (except with an abort probability of $1/p$, see event ROZero). To answer the remaining queries to oracles H_p and H_h, it uses $\hat{\mathrm{O}}$.
3. Whenever $\mathsf{NotPure}$ occurs, $\mathcal{B}$ aborts its execution.
4. Whenever $\mathcal{A}$ submits a tuple $(\mathsf{R}, \mathsf{m}, \sigma)$ to O, $\mathcal{B}$ checks if $\mathsf{NonAdmRs}$ occurs, i.e., $\mathsf{Ver}(\mathsf{R}, \mathsf{m}, \sigma) = 1$, and the dominant index i^* of $(\mathsf{R}, \mathsf{m}, \sigma)$ does not have admissible R's. If it does not occur on this query, $\mathcal{B}$ simulates the query as in $\mathbf{G}_3$. Otherwise, if it occurs, $\mathcal{B}$ checks if $h = \mathsf{H}_p(\mathsf{pk}_{i^*})$, i.e., the guessed query was correct. If it was, $\mathcal{B}$ outputs the representation of R_{i^*} and $\tilde{R}_{i^*}$ (as given in the definition of admissible R's), $W_{i^*}, \tilde{W}, c_{i^*}, s_{i^*}$ (as in Ver). Note that the representation of $R_{i^*}, \tilde{R}_{i^*}$ is pure (otherwise event $\mathsf{NotPure}$ would have occurred), and therefore the representations have the correct format for the game in Lemma 13.

Clearly, $\mathcal{B}$ perfectly simulates game $\mathbf{G}_3$ for $\mathcal{A}$, and if $\mathsf{NonAdmRs}$ occurs, and neither $\mathsf{NoGoodW}$ nor $\mathsf{NotPure}$ occur, then the event bounded in Lemma 13 occurs. Thus, we get a reduction $\mathcal{B}_1$ with the desired property. □

Lemma 17. *Let $\mathsf{H}_p, \mathsf{H}_h \colon \{0,1\}^* \to \mathbb{G}, , \mathsf{H}_c \colon \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}_a \colon \{0,1\}^* \to \mathbb{Z}_p^2$ be random oracles. Then, $(\mathsf{LRS}, \mathsf{KDS})$ (given in Figures 7 and 9) satisfies knowledge linkability in the algebraic group model. Concretely, with $\mathsf{Ext}_{\mathsf{LRS}}$ as in Lemma 16, we have that for any algebraic algorithm $\mathcal{A}$ that makes at most Q_{H_a} queries to oracle H_a, we have*

$$\mathsf{Adv}^{\mathsf{kn\text{-}link}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS}}(\lambda) \leq \frac{Q_{\mathsf{H}_a} N_{max}}{p}.$$

Here, N_{max} is the maximum size of key rings R that are submitted to random oracles $\mathsf{H}_a, \mathsf{H}_c$.

Proof. Before reading this proof, the reader is first encouraged to recall the terminology and the extractor $\mathsf{Ext}_{\mathsf{LRS}}$ from the proof of Lemma 16. Recall that if an adversary $\mathcal{A}$ wins the knowledge linkability game, then this means that one of the following two cases holds:

Alg $\mathsf{Sig}'(\mathsf{R},\mathsf{sk},\mathsf{auxsk},\mathsf{m}), \mathsf{Sig}''(\mathsf{R},\mathsf{auxsk},\mathsf{m})$

01 **parse** $(\mathsf{pk}_i,\mathsf{auxpk}_i)_{i=1}^N := \mathsf{R}$
02 **Let** $i^* \in [N]$ **s.t.** $\mathsf{pk}_{i^*} = g^{\mathsf{sk}} \wedge \mathsf{auxpk}_{i^*} = g^{\mathsf{auxsk}}$
03 **if no such** i^* : **abort**
04 $h_{i^*} := \mathsf{H}_p(\mathsf{pk}_{i^*}),\ \tilde{K}_2 := h_{i^*}^{\mathsf{auxsk}}$
05 $\tilde{K}_1 := h_{i^*}^{\mathsf{sk}}$ // Sig'
06 $\tilde{K}_1 := \mathsf{pk}_{i^*}^{\vartheta_{h_{i^*}}}$, where $h_{i^*} = g^{\vartheta_{h_{i^*}}}$ // Sig''
07 $((W_i)_{i=1}^N, \tilde{W}) := \mathsf{AggPub}(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$
08 $c_1 \leftarrow\!\!\$\ \mathbb{Z}_p,\ s_1 \leftarrow\!\!\$\ \mathbb{Z}_p,\ R_1 := g^{s_1} W_1^{c_1},\ \tilde{R}_1 := h_1^{s_1} \tilde{W}^{c_1}$
09 **for** $i \in \{2,\dots,N\}$:
10 $\quad c_i := \mathsf{H}_c(\mathsf{R},\mathsf{m},R_{i-1},\tilde{R}_{i-1}),\ h_i := \mathsf{H}_p(\mathsf{pk}_i)$
11 $\quad s_i \leftarrow\!\!\$\ \mathbb{Z}_p,\ R_i := g^{s_i} W_i^{c_i},\ \tilde{R}_i := h_i^{s_i} \tilde{W}^{c_i}$
12 Program $\mathsf{H}_c(\mathsf{R},\mathsf{m},R_N,\tilde{R}_N) := c_1$
13 **return** $\sigma := (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$

Figure 10: The modified signing algorithms Sig' and Sig'', used in the proof of Lemma 16 and Lemma 9, respectively.

1. $\mathcal{A}$ submitted two triples $(\mathsf{R},\mathsf{m},\sigma)$, $(\mathsf{R}',\mathsf{m}',\sigma')$, for $\mathsf{R} = (\mathsf{pk}_i,\mathsf{auxpk}_i)_i$, $\mathsf{R}' = \left(\mathsf{pk}'_i,\mathsf{auxpk}'_i\right)_i$ such that for both of them, $\mathsf{Ext}_{\mathsf{LRS}}$ extracts successfully, i.e., it returns $(i^*,\mathsf{sk}_{i^*},\mathsf{auxsk}_{i^*})$, $(i^{*\prime},\mathsf{sk}'_{i^*},\mathsf{auxsk}'_{i^*})$, respectively, and we have

$$\mathsf{pk}_{i^*} = \mathsf{pk}'_{i^{*\prime}} \wedge \mathsf{Link}(\sigma,\sigma') = 0.$$

2. $\mathcal{A}$ submitted a triple $(\mathsf{R},\mathsf{m},\sigma)$, for $\mathsf{R} = (\mathsf{pk}_i,\mathsf{auxpk}_i)_i$, such that $\mathsf{Ext}_{\mathsf{LRS}}$ extracts successfully, i.e., it returns $(i^*,\mathsf{sk}_{i^*},\mathsf{auxsk}_{i^*})$, and the oracle Sign returned a valid signature σ' on some input $\mathsf{pk}',\mathsf{pseed},\mathsf{tag},\mathsf{R}',\mathsf{auxsk},\mathsf{m}$ such that

$$\mathsf{pk}_{i^*} = \mathsf{pk}' \wedge \mathsf{Link}(\sigma,\sigma') = 0.$$

Now, we first make a simple observation about the extractor $\mathsf{Ext}_{\mathsf{LRS}}$. Then, we will define a bad event that occurs with negligible probability. Finally, we will argue that if this event does not occur, then $\mathcal{A}$ can not win the knowledge linkability game. We observe the following. Assume $\mathsf{Ext}_{\mathsf{LRS}}$ returns $(i^*,\mathsf{sk}_{i^*},\mathsf{auxsk}_{i^*})$ when $\mathcal{A}$ submits $(\mathsf{R},\mathsf{m},\sigma)$ with $\mathsf{R} = (\mathsf{pk}_i,\mathsf{auxpk}_i)_{i=1}^N$ and in a query. Write $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$ and define $((W_i)_{i=1}^N, \tilde{W}) := \mathsf{AggPub}(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$ as in algorithm Ver. Then, we have

$$\exists w \in \mathbb{Z}_p : W_{i^*} = g^w \wedge \tilde{W} = \mathsf{H}_p(\mathsf{pk}_{i^*})^w.$$

This can be seen as follows: If $\mathsf{Ext}_{\mathsf{LRS}}$ does not output $\perp$, then by definition of $\mathsf{Ext}_{\mathsf{LRS}}$, the dominant index i^* of $(\mathsf{R},\mathsf{m},\sigma)$ is not badly represented and has admissible R's. This means that (using notation as in Ver) we have $R_{i^*} = g^{r_{i^*}}$, $\tilde{R}_{i^*} = h_{i^*}^{r_{i^*}}$ for some $r_{i^*} \in \mathbb{Z}_p$. Further, if $\mathsf{Ext}_{\mathsf{LRS}}$ does not output $\perp$, then especially the verification equation holds and we have

$$\begin{aligned} & g^{r_{i^*}} = R_{i^*} = g^{s_{i^*}} W_{i^*}^{c_{i^*}},\ \ \mathsf{H}_p(\mathsf{pk}_{i^*})^{r_{i^*}} = \tilde{R}_{i^*} = \mathsf{H}_p(\mathsf{pk}_{i^*})^{s_{i^*}} \tilde{W}^{c_{i^*}} \\ \implies & W_{i^*} = g^w \wedge \tilde{W} = \mathsf{H}_p(\mathsf{pk}_{i^*})^w \text{ for } w = \frac{r_{i^*} - s_{i^*}}{c_{i^*}}, \end{aligned}$$

where we also used that $\mathsf{Ext}_{\mathsf{LRS}}$ would have output $\perp$ if $c_{i^*} = 0$. It is clear that such a w also exists for signatures that were computed honestly by oracle Sign.

Next, we define a bad event and bound its probability. Informally, we want to rule out that the aggregations $W_{i^*}, \tilde{W}$ form a Diffie-Hellman tuple, but $\mathsf{pk}_{i^*}, \tilde{K}_1$ do not.

- Event $\mathsf{SpanColl}$: This event occurs, if for a query $\mathsf{H}_a(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$ with $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ that returns (a_1, a_2), there is some $i \in [N]$ such that

$$\begin{aligned} &\forall z \in \mathbb{Z}_p : \neg \left(\mathsf{pk}_i = g^z \wedge \tilde{K}_1 = \mathsf{H}_p(\mathsf{pk}_i)^z \right) \\ \wedge \;\; &\exists w \in \mathbb{Z}_p : \left(\mathsf{pk}_i^{a_1} \cdot \mathsf{auxpk}_i^{a_2} = g^w \wedge \tilde{K}_1^{a_1} \cdot \tilde{K}_2^{a_2} = \mathsf{H}_p(\mathsf{pk}_i)^w \right). \end{aligned}$$

To bound the probability of $\mathsf{SpanColl}$, consider a fixed query $\mathsf{H}_a(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$ as in the definition of the event and a fixed $i \in [N]$. Now, let $\vartheta, \mathsf{sk}_i, \mathsf{auxsk}_i, c, d \in \mathbb{Z}_p$ be the uniquely determined exponents satisfying $g^\vartheta = \mathsf{H}_p(\mathsf{pk}_i), g^{\mathsf{sk}_i} = \mathsf{pk}_i, g^{\mathsf{auxsk}_i} = \mathsf{auxpk}_i, g^c = \tilde{K}_1, g^d = \tilde{K}_2$. Set $\mathbf{v} := (1, \vartheta)^\top$, $\mathbf{w}_1 := (\mathsf{sk}_i, c)^\top$, and $\mathbf{w}_2 := (\mathsf{auxsk}_i, d)^\top \in \mathbb{Z}_p^2$. Then, the event occurs for this query and this i if

$$\mathbf{w}_1 \notin \mathbb{Z}_p \mathbf{v} \wedge a_1 \mathbf{w}_1 + a_2 \mathbf{w}_2 \in \mathbb{Z}_p \mathbf{v}.$$

In particular, projecting these vectors to the one dimensional vector space $\mathbb{Z}_p^2 / \mathbb{Z}_p \mathbf{v} \cong \mathbb{Z}_p$, and denoting the respective elements by $w_1, w_2 \in \mathbb{Z}_p$, the event implies the equation

$$w_1 \neq 0 \wedge a_1 w_1 + a_2 w_2 = 0.$$

As w_1, w_2 are fixed before a_1, a_2 are sampled uniformly, and the linear map $(a_1, a_2) \mapsto a_1 w_1 + a_2 w_2$ is surjective given that $w_1 \neq 0$, this can only happen with probability $1/p$. In summary, for a fixed query and a fixed i, the event occurs with probability $1/p$. A union bound now implies

$$\Pr[\mathsf{SpanColl}] \leq \frac{Q_{\mathsf{H}_a} N_{\max}}{p}.$$

Finally, assume that $\mathsf{SpanColl}$ does not occur. We argue that conditioned on that, the knowledge linkability game never outputs 1. To see this, consider two triples $(\mathsf{R}, \mathsf{m}, \sigma)$, $(\mathsf{R}', \mathsf{m}', \sigma')$, for $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_i$, $\mathsf{R}' = \left(\mathsf{pk}'_i, \mathsf{auxpk}'_i\right)_i$ as in the two cases outlined in the beginning of the proof. That is, $\mathsf{Ext}_{\mathsf{LRS}}$ returns $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*})$ for $(\mathsf{R}, \mathsf{m}, \sigma)$, and for $(\mathsf{R}', \mathsf{m}', \sigma')$, either $\mathsf{Ext}_{\mathsf{LRS}}$ returns $(i^{*\prime}, \mathsf{sk}'_{i^*}, \mathsf{auxsk}'_{i^*})$ (Case 1), or there is such a tuple $(i^{*\prime}, \mathsf{sk}'_{i^*}, \mathsf{auxsk}'_{i^*})$ because σ' is computed honestly by Sign. Further, write $\sigma = (\mathsf{R}, c_1, (s_i)_i, \tilde{K}_1, \tilde{K}_2)$ and $\sigma' = (\mathsf{R}', c'_1, (s'_i)_i, \tilde{K}'_1, \tilde{K}'_2)$. As in algorithm Ver, define $((W_i)_i, \tilde{W}) := \mathsf{AggPub}(\mathsf{R}, \tilde{K}_1, \tilde{K}_2)$ and $((W'_i)_i, \tilde{W}') := \mathsf{AggPub}(\mathsf{R}', \tilde{K}'_1, \tilde{K}'_2)$. Using the observation we made above, we know that

$$\begin{aligned} &\exists w \in \mathbb{Z}_p : W_{i^*} = g^w \wedge \tilde{W} = \mathsf{H}_p(\mathsf{pk}_{i^*})^w \\ \wedge \;\; &\exists w' \in \mathbb{Z}_p : W'_{i^{*\prime}} = g^{w'} \wedge \tilde{W}' = \mathsf{H}_p(\mathsf{pk}'_{i^{*\prime}})^{w'}. \end{aligned}$$

Recall that AggPub internally uses random oracle H_a to compute its output. As $\mathsf{SpanColl}$ does not occur, this means that

$$\begin{aligned} &\exists z \in \mathbb{Z}_p : \mathsf{pk}_{i^*} = g^z \wedge \tilde{K}_1 = \mathsf{H}_p(\mathsf{pk}_{i^*})^z \\ \wedge \;\; &\exists z' \in \mathbb{Z}_p : \mathsf{pk}'_{i^{*\prime}} = g^{z'} \wedge \tilde{K}'_1 = \mathsf{H}_p(\mathsf{pk}'_{i^{*\prime}})^{z'}. \end{aligned}$$

Recall that algorithm Link just compares $\tilde{K}_1$ and $\tilde{K}'_1$. This means that we have to show that

$$\mathsf{pk}_{i^*} = \mathsf{pk}'_{i^{*\prime}} \wedge \tilde{K}_1 \neq \tilde{K}'_1$$

can not hold. To this end, assume that $\mathsf{pk}_{i^*} = \mathsf{pk}'_{i^{*\prime}}$. Then, we see that $z = z'$, and therefore

$$\tilde{K}_1 = \mathsf{H}_p(\mathsf{pk}_{i^*})^z = \mathsf{H}_p(\mathsf{pk}'_{i^{*\prime}})^z = \mathsf{H}_p(\mathsf{pk}_{i^*})^z = \tilde{K}'_1,$$

finishing the proof. □

Lemma 18. *Let* $\mathsf{H}_p, \mathsf{H}_h \colon \{0,1\}^* \to \mathbb{G}, , \mathsf{H}_c \colon \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}_a \colon \{0,1\}^* \to \mathbb{Z}_p^2$ *be random oracles. If the* DLOG *assumption holds relative to* GGen*, then* $(\mathsf{LRS}, \mathsf{KDS})$ *(given in Figures 7 and 9) satisfies knowledge non-slanderability in the algebraic group model. Concretely, with* $\mathsf{Ext}_{\mathsf{LRS}}$ *as in Lemma 16, we have that for any algebraic PPT algorithm* $\mathcal{A}$ *that makes at most* Q_{H_a} *queries to oracle* H_a *there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$*, such that*

$$\mathsf{Adv}^{\mathsf{kn\text{-}n\text{-}sland}}_{\mathcal{A},\mathsf{Ext}_{\mathsf{LRS}},\mathsf{LRS},\mathsf{KDS}}(\lambda) \leq \frac{Q_{\mathsf{H}_a} N_{max} + 1}{p} + \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda),$$

where N_{max} *is the maximum size of key rings* R *that are submitted to random oracles* $\mathsf{H}_a, \mathsf{H}_c$*.*

Proof. The reader is encouraged to recall the terminology and the extractor $\mathsf{Ext}_{\mathsf{LRS}}$ from the proof of Lemma 16, as well as the proof of Lemma 17. Consider the knowledge non-slanderability game for an adversary $\mathcal{A}$. By definition, the adversary wins this game, if it submits a triple $(\mathsf{R}, \mathsf{m}, \sigma)$, for $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ and $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$ to oracle O, such that $\mathsf{Ext}_{\mathsf{LRS}}$ extracts successfully, i.e., it returns $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*})$, and the oracle $\textsc{Sign}$ returned a valid signature $\sigma' = (\mathsf{R}', c'_1, (s'_i)_{i=1}^N, \tilde{K}'_1, \tilde{K}'_2)$ computed as $\sigma' \leftarrow \mathsf{Sig}(\mathsf{R}', \mathsf{sk}', \mathsf{auxsk}, \mathsf{m})$ on some input $\mathsf{pk}', \mathsf{pseed}, \mathsf{tag}, \mathsf{R}', \mathsf{auxsk}, \mathsf{m}$ such that

$$\mathsf{pk}_{i^*} \neq \mathsf{pk}' \wedge \mathsf{Link}(\sigma, \sigma') = 1.$$

Using notation as in algorithm Ver, we can first show that in this case, there exists some $w \in \mathbb{Z}_p$ such that $W_{i^*} = g^w$ and $\tilde{W} = \mathsf{H}_p(\mathsf{pk}_{i^*})^w$. This is done exactly as in the proof of Lemma 17. Then, we introduce and bound the event $\mathsf{SpanColl}$ exactly as in Lemma 17, and have

$$\Pr[\mathsf{SpanColl}] \leq \frac{Q_{\mathsf{H}_a} N_{\max}}{p}.$$

If $\mathsf{SpanColl}$ does not occur, we can argue as in the proof of Lemma 17 to show that there must be some $z \in \mathbb{Z}_p$ such that $\mathsf{pk}_{i^*} = g^z$ and $\tilde{K}_1 = \mathsf{H}_p(\mathsf{pk}_{i^*})^z$. By definition of $\mathsf{Ext}_{\mathsf{LRS}}$, this z is exactly equal to sk_{i^*}. Now, the winning condition $\mathsf{pk}_{i^*} \neq \mathsf{pk}' \wedge \mathsf{Link}(\sigma, \sigma') = 1$ translates to

$$\mathsf{pk}_{i^*} \neq \mathsf{pk}' \wedge \mathsf{H}_p(\mathsf{pk}_{i^*})^{\mathsf{sk}_{i^*}} = \tilde{K}_1 = \tilde{K}'_1 = \mathsf{H}_p(\mathsf{pk}')^{\mathsf{sk}'}.$$

This means, that $\mathcal{A}$ can only win the knowledge non-slanderability game, if the following event occurs.

- Event $\mathsf{KeyHashColl}$: This event occurs, if $\mathcal{A}$ submits triple $(\mathsf{R}, \mathsf{m}, \sigma)$ with $\mathsf{R} = (\mathsf{pk}_i, \mathsf{auxpk}_i)_{i=1}^N$ and $\sigma = (\mathsf{R}, c_1, (s_i)_{i=1}^N, \tilde{K}_1, \tilde{K}_2)$ to oracle O, such that $\mathsf{Ext}_{\mathsf{LRS}}$ extracts successfully, i.e., it returns $(i^*, \mathsf{sk}_{i^*}, \mathsf{auxsk}_{i^*})$, and before that, $\textsc{Sign}$ returned a valid signature $\sigma' = (\mathsf{R}', c'_1, (s'_i)_{i=1}^N, \tilde{K}'_1, \tilde{K}'_2)$ computed as $\sigma' \leftarrow \mathsf{Sig}(\mathsf{R}', \mathsf{sk}', \mathsf{auxsk}, \mathsf{m})$ on some input $\mathsf{pk}', \mathsf{pseed}, \mathsf{tag}, \mathsf{R}', \mathsf{auxsk}, \mathsf{m}$ such that

$$\mathsf{pk}_{i^*} \neq \mathsf{pk}' \wedge \mathsf{H}_p(\mathsf{pk}_{i^*})^{\mathsf{sk}_{i^*}} = \mathsf{H}_p(\mathsf{pk}')^{\mathsf{sk}'}.$$

To finish the proof, it remains to bound the probability of event $\mathsf{KeyHashColl}$. This is easily done using Lemma 11. Namely, a reduction running in the game defined in Lemma 11 gets as input generator $g \in \mathbb{G}$ and access to an oracle $\hat{\mathrm{O}}$ that outputs uniform group elements. It simulates the knowledge non-slanderability game for $\mathcal{A}$, while simulating random oracle H_p using oracle $\hat{\mathrm{O}}$. Then, once $\mathsf{KeyHashColl}$ occurs, it outputs the two distinct representations sk and sk_{i^*} such that $\mathsf{H}_p(\mathsf{pk}_{i^*})^{\mathsf{sk}_{i^*}} = \mathsf{H}_p(\mathsf{pk}')^{\mathsf{sk}'}$. Note that the reduction knows sk' as it simulates the signing oracle honestly, and $\mathsf{H}_p(\mathsf{pk}_{i^*})$ and $\mathsf{H}_p(\mathsf{pk}')$ are output by $\hat{\mathrm{O}}$ on distinct queries. Thus, if $\mathsf{KeyHashColl}$ occurs, the event bounded in Lemma 11 is triggered. Therefore, Lemma 11 tells us that there is a reduction $\mathcal{B}$ such that

$$\Pr\left[\mathsf{KeyHashColl}\right] \leq \mathsf{Adv}^{\mathsf{DLOG}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

□

8 The Burning Bug and Transaction Proofs

In 2018, Monero was found to be vulnerable to the so-called burning-bug [dEB]: if a malicious user spends coins to the same stealth address twice (which ought to be unique), the recipient would receive and see the amount of both payments in their wallet, but would not be able to spend the second one. This could be used to, e.g., defraud exchanges: send them X that they cannot spend, receive something of value X in return; the attacker would only spend transaction fees while the exchange would effectively lose money. The bug was fixed in 2018 by changing the way in which wallets present multiple incoming transactions with the same stealth address to users: roughly, the transaction with the highest amount is unchanged, and the others are shown as zero-value. This ensures that whatever the recipient sees in their wallet, can in fact be spent. In our analysis, we do not explicitly model this mitigation, as it is about what the wallet presents to the user, which we do not cover. Instead, we assume that transaction public keys, i.e., public seeds pseed, are unique, see Line 05 in Figure 3. This implies that stealth addresses are unique and thus prevents the burning-bug.

However, we identified that, due to the burning-bug, the current transaction proofs only prove *proof-of-payment*, but do not guarantee that the amount can actually be spent by the recipient. This holds for all types of transaction proofs that Monero's wallets currently support. While Monero's documentation does not claim that the amounts in transaction proofs can be spent, this seems like a natural assumption for normal users to make. We reported this to the Monero developers and suggested that the documentation (and perhaps the user interface) should be clarified in this respect.

9 Other Models for RingCT-Like Systems

While no previous work analyzes Monero's transaction scheme RingCT as it is, some previous works [SALY17, YSL+20, LRR+19, EZS+19, ESZ22] 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 [LRR+19, EZS+19]. 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 Section 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 [SALY17, YSL+20, LRR+19] 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 [LRR+19], 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 [SALY17, YSL+20, EZS+19, ESZ22] do not have this feature.

Sun et al.'s RingCT 2.0. Sun et al. introduce [SALY17] 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 [LRR+19], 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 [SALY17] informally claim that linkability follows from their non-slanderability notion. As explained in [LRR+19], 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. [YSL+20] 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. [SALY17], 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. [LRR+19] 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 [LRR+19] 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 [EZS+19, ESZ22], 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 [EZS+19, ESZ22] do not model adversarially generated outputs (i.e., stealth addresses). It is mentioned in Appendix C.A of [ESZ22] 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 [EZS+19, ESZ22] do not model the reuse of randomness for output generation of honest users.

10 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 [BDL+11]. Due to the modularity of our framework, one could extend our results to the setting of Ed25519 (in the spirit of, e.g., [BCJZ21]) 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 in Section 8. Finally, we do not show that the Bulletproof/Bulletproof+ component [BBB+18, BMM+21, GOP+22, CHJ+22] of the system satisfies the security notion we define for it. It has been shown in [GT21] that Bulletproofs satisfy a related notion. After discussion with the authors of [GT21], 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.

References

[AJ18] Kurt M. Alonso and Jordi Herrera Joancomartí. Monero - privacy in the blockchain. Cryptology ePrint Archive, Report 2018/535, 2018. https://eprint.iacr.org/2018/535. (Cited on page 6.)

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. (Cited on page 6, 50, 82.)

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. (Cited on page 6.)

[BCJZ21] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of Ed25519: Theory and practice. In *2021 IEEE Symposium on Security and Privacy*, pages 1659–1676. IEEE Computer Society Press, May 2021. (Cited on page 82.)

[BDH+19] Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Ring signatures: Logarithmic-size, no setup - from standard assumptions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 281–311. Springer, Heidelberg, May 2019. (Cited on page 27.)

[BDL+11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, Heidelberg, September / October 2011. (Cited on page 82.)

[BKP20] Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and Falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In Shiho Moriai and

Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 464–492. Springer, Heidelberg, December 2020. (Cited on page 27.)

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, Heidelberg, December 2021. (Cited on page 82.)

[CHJ+22] HeeWon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter Proofs for a Privacy-Enhanced Distributed Ledger. *IEEE Access*, 10:42067–42082, 2022. (Cited on page 6, 50, 51, 82.)

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020. (Cited on page 5.)

[CYD+20] Tong Cao, Jiangshan Yu, Jérémie Decouchant, Xiapu Luo, and Paulo Veríssimo. Exploring the monero peer-to-peer network. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 578–594. Springer, Heidelberg, February 2020. (Cited on page 6.)

[DBV21] Arijit Dutta, Suyash Bagad, and Saravanan Vijayakumaran. MProve+: Privacy Enhancing Proof of Reserves Protocol for Monero. *IEEE Trans. Inf. Forensics Secur.*, 16:3900–3915, 2021. (Cited on page 6.)

[dEB] dEBRUYNE. A Post Mortem of The Burning Bug. https://web.getmonero.org/2018/09/25/a-post-mortum-of-the-burning-bug.html. Accessed: 2023-04-11. (Cited on page 80.)

[DRR22] Dominic Deuber, Viktoria Ronge, and Christian Rückert. SoK: Assumptions underlying cryptocurrency deanonymizations. *PoPETs*, 2022(3):670–691, July 2022. (Cited on page 6.)

[DV19] Arijit Dutta and Saravanan Vijayakumaran. 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, June 17-19, 2019*, pages 330–339. IEEE, 2019. (Cited on page 6.)

[Eag22] Liam Eagen. Bulletproofs++. Cryptology ePrint Archive, Report 2022/510, 2022. https://eprint.iacr.org/2022/510. (Cited on page 6.)

[ELR+22] Christoph Egger, Russell W. F. Lai, Viktoria Ronge, Ivy K. Y. Woo, and Hoover H. F. Yin. On defeating graph analysis of anonymous transactions. *PoPETs*, 2022(3):538–557, July 2022. (Cited on page 6.)

[ESZ22] Muhammed F. Esgin, Ron Steinfeld, and Raymond K. Zhao. MatRiCT$^+$: More Efficient Post-Quantum Private Blockchain Payments. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1281–1298. IEEE, 2022. (Cited on page 6, 80, 81.)

[EZS+19] Muhammed F. Esgin, Raymond K. Zhao, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. MatRiCT: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 567–584. ACM Press, November 2019. (Cited on page 6, 80, 81.)

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. (Cited on page 3, 4, 5, 12.)

[FO22] Georg Fuchsbauer and Michele Orrù. Non-interactive mimblewimble transactions, revisited. Cryptology ePrint Archive, Report 2022/265, 2022. https://eprint.iacr.org/2022/265. (Cited on page 7.)

[FOS19] Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 657–689. Springer, Heidelberg, May 2019. (Cited on page 7.)

[Fro] Liam Frost. Monero developers disclose 'significant' bug in privacy algorithm. https://decrypt.co/76938/monero-developers-disclose-significant-bug-privacy-algorithm. Accessed: 2023-02-14. (Cited on page 6.)

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 181–200. Springer, Heidelberg, April 2007. (Cited on page 6.)

[GNB19] Brandon Goodell, Sarang Noether, and Arthur Blue. Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Paper 2019/654, 2019. https://eprint.iacr.org/2019/654. (Cited on page 5, 6, 12, 27.)

[GOP+22] Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 397–426. Springer, Heidelberg, May / June 2022. (Cited on page 82.)

[GT21] Ashrujit Ghoshal and Stefano Tessaro. Tight state-restoration soundness in the algebraic group model. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg. (Cited on page 6, 50, 82.)

[Gug20] Joël Gugger. Bitcoin-monero cross-chain atomic swap. Cryptology ePrint Archive, Report 2020/1126, 2020. https://eprint.iacr.org/2020/1126. (Cited on page 6.)

[HBHW] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash Protocol Specification, Version 2022.3.8. https://zips.z.cash/protocol/protocol.pdf. Accessed: 2023-02-15. (Cited on page 6, 7.)

[Jed] Tom Evlis Jedusor. Mimblewimble. https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt. Accessed: 2023-02-15. (Cited on page 7.)

[JF21] Aram Jivanyan and Aaron Feickert. Lelantus spark: Secure and flexible private transactions. Cryptology ePrint Archive, Report 2021/1173, 2021. https://eprint.iacr.org/2021/1173. (Cited on page 6.)

[KAN20] Koe, Kurt M. Alonso, and Sarang Noether. Zero to Monero v2.0.0. https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf, 2020. Accessed: 2022-11-21. (Cited on page 3, 6, 12.)

[KFTS17] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 153–173. Springer, Heidelberg, September 2017. (Cited on page 6.)

[Kle] Christopher Klee. Monero XMR: "Signifikanter" Privacy Bug entdeckt. https://www.btc-echo.de/schlagzeilen/monero-xmr-signifikanter-privacy-bug-entdeckt-123001/. Accessed: 2023-02-14. (Cited on page 6.)

[lfS] luigi1111 and Riccardo "fluffypony" Spagni. 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: 2023-02-14. (Cited on page 3, 6.)

[LRR+19] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, November 2019. (Cited on page 6, 80, 81.)

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 325–335. Springer, Heidelberg, July 2004. (Cited on page 6.)

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. (Cited on page 5.)

[MBL+20] Pedro Moreno-Sanchez, Arthur Blue, Duc Viet Le, Sarang Noether, Brandon Goodell, and Aniket Kate. DLSAG: Non-interactive refund transactions for interoperable payment channels in monero. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 325–345. Springer, Heidelberg, February 2020. (Cited on page 6.)

[MCdS20] Rui Morais, Paul Crocker, and Simao Melo de Sousa. Delegated RingCT: faster anonymous transactions. Cryptology ePrint Archive, Report 2020/1521, 2020. https://eprint.iacr.org/2020/1521. (Cited on page 6.)

[MSH+18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *PoPETs*, 2018(3):143–163, July 2018. (Cited on page 6.)

[Nica] Jonas Nick. A Problem With Monero's RingCT. https://jonasnick.github.io/blog/2016/12/17/a-problem-with-ringct/. Accessed: 2023-02-14. (Cited on page 3, 6.)

[Nicb] Jonas Nick. 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: 2023-02-14. (Cited on page 3, 6.)

[Noe15] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. https://eprint.iacr.org/2015/1098. (Cited on page 6, 12.)

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992. (Cited on page 6, 53.)

[Pro] Monero Project. Monero-Project/Meta: List of Issues. https://github.com/monero-project/meta/issues. Accessed: 2023-04-11. (Cited on page 6.)

[REL+21] Viktoria Ronge, Christoph Egger, Russell W. F. Lai, Dominique Schröder, and Hoover H. F. Yin. Foundations of ring sampling. *PoPETs*, 2021(3):265–288, July 2021. (Cited on page 6.)

[SALY17] Shi-Feng Sun, Man Ho Au, Joseph K. Liu, and Tsz Hon Yuen. RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 456–474. Springer, Heidelberg, September 2017. (Cited on page 6, 80, 81.)

[SLYQ22] Zhimei Sui, Joseph K. Liu, Jiangshan Yu, and Xianrui Qin. 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*, pages 280–290. IEEE, 2022. (Cited on page 6.)

[TMSS20] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmidt, and Dominique Schröder. PayMo: Payment channels for monero. Cryptology ePrint Archive, Report 2020/1441, 2020. https://eprint.iacr.org/2020/1441. (Cited on page 6.)

[TW05] Patrick P. Tsang and Victor K. Wei. Short Linkable Ring Signatures for E-Voting, E-Cash and Attestation. In Robert H. Deng, Feng Bao, HweeHwa Pang, and Jianying Zhou, editors, *Information Security Practice and Experience, First International Conference, ISPEC 2005, Singapore, April 11-14, 2005, Proceedings*, volume 3439 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2005. (Cited on page 27.)

[Vij21] Saravanan Vijayakumaran. Analysis of CryptoNote transaction graphs using the dulmage-mendelsohn decomposition. Cryptology ePrint Archive, Report 2021/760, 2021. https://eprint.iacr.org/2021/760. (Cited on page 6.)

[VS13] Nicolas Van Saberhagen. CryptoNote v2.0. https://www.bytecoin.org/old/whitepaper.pdf, 2013. Accessed: 2022-11-21. (Cited on page 3, 6.)

[WLSL18] Dimaz Ankaa Wijaya, Joseph K. Liu, Ron Steinfeld, and Dongxi Liu. 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, pages 1196–1201. IEEE, 2018. (Cited on page 6.)

[YAV19] Jiangshan Yu, Man Ho Allen Au, and Paulo Jorge Esteves Veríssimo. Re-thinking untraceability in the CryptoNote-style blockchain. In Stephanie Delaune and Limin Jia, editors, *CSF 2019 Computer Security Foundations Symposium*, pages 94–107. IEEE Computer Society Press, 2019. (Cited on page 6.)

[YSL+20] Tsz Hon Yuen, Shifeng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu. RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 464–483. Springer, Heidelberg, February 2020. (Cited on page 6, 80, 81.)