SuccinCT: Succinct Confidential Transaction for Miner Privacy

Ying-Teng Chen
Monash University
Clayton, Victoria, Australia
ying-teng.chen@monash.edu

Tsz Hon Yuen
Monash University
Clayton, Victoria, Australia
john.tszhonyuen@monash.edu

Dongkun Hou
Monash University
Clayton, Victoria, Australia
dongkun.hou@monash.edu

Jie Xu
Monash University
Clayton, Victoria, Australia
jie.xu@monash.edu

Joseph K. Liu
Monash University
Clayton, Victoria, Australia
joseph.liu@monash.edu

Wayne Yang
The University of Sydney
Sydney, NSW, Australia
xyan0559@sydney.edu.au

Jiangshan Yu
The University of Sydney
Sydney, NSW, Australia
jiangshan.yu@sydney.edu.au

Abstract

Confidential transaction (CT) protocols are widely used to protect the privacy of blockchain-based cryptocurrency transactions. However, existing CT schemes primarily focus on ordinary users' anonymity and amount confidentiality, while overlooking miner privacy as a native objective. In particular, the privacy of miners' coinbase-receiving addresses has not been systematically considered in security models. In this paper, we identify two novel attacks against miner privacy in existing CT protocols, called *miner privacy attacks* and *anonymity reduction attacks*. These attacks arise from the transparency of the mining coinbase amounts and transaction fees in the underlying blockchain systems. When miners' accounts are included in a CT, our general solutions compatible with all schemes can prevent adversaries from identifying miners as the real spenders or excluding miners from the possible spender sets.

We propose *SuccinCT*, the first CT protocol based on the DualRing structure to address the aforementioned attacks. Compared with existing schemes, SuccinCT achieves the smallest overall transaction size while protecting miner privacy. Specifically, the proof size of SuccinCT is about 20% smaller than the latest BulletCT (USENIX SEC'25). We implement SuccinCT and evaluate its performance for ring sizes from 128 to 1024. The results show that signature generation takes about 2–3 ms and verification takes about 1–2 ms. These results demonstrate that SuccinCT provides stronger privacy guarantees, novel construction, and practical efficiency for real-world deployment.

Keywords

confidential transaction, blockchain privacy, security model

1 Introduction

Initially, cryptocurrencies like Bitcoin offered pseudonymity but lacked full privacy as transaction amounts and addresses were publicly visible. Maxwell proposed Confidential Transactions (CT) in 2015 [14], introducing the idea of hiding transaction amounts through Pedersen commitments. This method allowed transaction amounts to be encrypted while still being verifiable by the network, laying the foundation for privacy-enhancing blockchain transactions.

Monero, a privacy-focused cryptocurrency, integrated an advanced form of confidential transactions called Ring Confidential Transactions (RingCT) [16]. Ring signature schemes [19] allow the signer to anonymously sign a message on behalf of a spontaneously chosen group. RingCT combined *ring signatures*, *stealth addresses* [22], and *Pedersen commitment* to hide both the transaction amount and the identities of senders and receivers, marking a significant advancement in blockchain privacy.

RingCT protocols have been widely studied in the literature. Some papers [11, 23, 26, 30, 31] studied the efficiency improvement over Monero's protocol [16]. Some papers proposed a post-quantum version of confidential transaction protocols [2, 3, 7, 8, 13, 24, 25]. In these works, the proposed protocols are proved secure in the formal security models, which include privacy, balance, and non-slanderability.

Although confidential transaction is well-studied as a standalone protocol, the integration with the cryptocurrency ecosystem is not fully considered. In this paper, we identify three issues of the existing CT protocols when integrated with Monero: (1) the potential attack on privacy related to the miner account, (2) the lack of comprehensive analysis of overall transaction size, and (3) the gap of applying DualRing [29] structure for efficient CT construction using the concept of sign first, compress later. These gaps in the state-of-the-art lead us to consider the following question: *How to construct a new Confidential Transaction succinctly to capture miner security with the shortest overall transaction size?*

1.1 Potential attack on privacy related to miners

The security models of previous works only consider the privacy of normal users (non-miner) in transactions. However, we observe that when miners' accounts are included in CTs, the anonymity of miners may be compromised. Specifically, the transaction fees and the mining reward received by the miner of each block are shown in plain text in most cryptocurrency, including Monero. This is because (1) the miner's identity cannot be confirmed when the

transaction is generated, (2) the transaction fee should be transparent to all miners so that they can maximize their profit when building a block of transactions, and (3) the mining reward should be a value determined by the system. To the best of the authors' knowledge, there is no formal analysis on the privacy for the miners' accounts in Monero.

The user can generate a *view key* for his accounts and delegate it to a third party, so that he can receive coins sent to his one-time address. By sharing a view key, the third party can access all one-time addresses of the user, the amount received, and the key of the coin. However, they cannot spend the coin without the user's long-term secret key.

Due to the transparency of the amount owned by the miner's account, the adversary possessing the output view key may detect whether the miner is the real spender in CT by comparing the mining rewards (as input amounts) with the output amounts plus transaction fees. For a transaction which includes the miners' accounts as mixin in the ring, the adversary can also deduce that the miner is not the real spender, resulting in an *anonymity reduction*. These attacks will be discussed in Section 3.

1.2 Analysis on overall transaction size

Many confidential transaction protocols [11, 23, 26, 30, 31] tried to reduce the proof size over Monero's MLSAG [16] and CLSAG [9] schemes. A transaction contains inputs (pointers referencing to some previous transactions on the blockchain), key images, a proof (generated by the confidential transaction protocol), and transaction outputs (the output one-time addresses and the output coins). However, existing works compared only the proof size instead of the whole transaction.

We give a comprehensive analysis of the whole transaction size. Denote $\mathfrak{R}$, $\mathfrak{S}$, $\mathfrak{T}$ as the number of the (input) ring accounts, the source accounts and the (output) target accounts, respectively. Denote β as the bit-length of the transaction amount and $|\mathcal{I}|$ as the data size to reference a previous transaction. For all schemes, the total transaction input size is $\mathfrak{R}|\mathcal{I}|$. Assume that each group element in $\mathbb{G}$ and each field element in $\mathbb{Z}_p$ are both represented by $|\mathbb{G}| = 32$ bytes. We compare the existing schemes in Table 1. We do not compare with [23] because we only compare schemes without a trusted setup. We show the transaction size for a 2-input 2-output transaction with a ring size of 256 in Figure 1.

RingCT 3.0 [30] (FC '20) improved CLSAG by combining $\mathfrak{S}$ linkable ring signatures and public key sets into a $\mathfrak{S}\times\frac{\mathfrak{R}}{\mathfrak{S}}$ matrix to reduce the proof size from linear to logarithmic to $\mathfrak{R}$ using Bulletproofs [4]. They have the same input size, key image size ($\mathfrak{S}$ group elements) and output size (one group element for each output address, one group element for each output coin, β bits for each encrypted output amount, and one DH ephemeral key for decrypting all amounts and coins). Omniring [11] (CCS '19) has a slightly smaller proof size by integrating all secret values into the Bulletproof witness to form a unified ring. However, they have a larger output size. For each output, they also use two group elements for each output address and output coin. Their β-bit output amount and the coin key r of size $|\mathbb{G}|$ is encrypted by the labeled public-key encryption ECIES, which has the ciphertext size of $3|\mathbb{G}| + 2\frac{\beta}{8}$. There is also an extra ciphertext of an ephemeral key encrypted by ECIES with

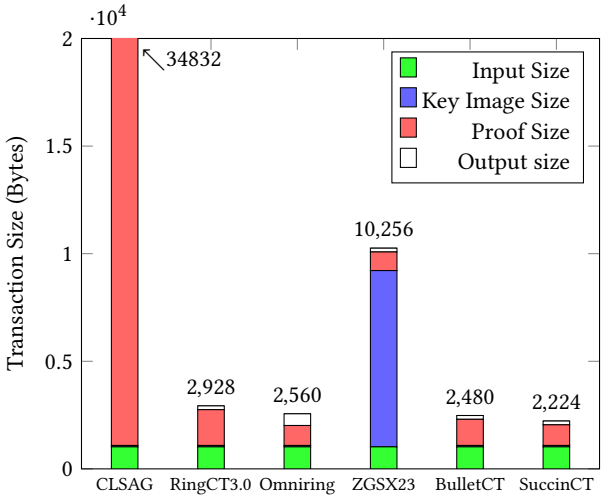

Figure 1: Comparison of confidential transaction in bytes, considering $|\mathbb{G}| = 32$ bytes, $\mathfrak{S} = \mathfrak{T} = 2$ and fixed $\mathfrak{R} = 256$.

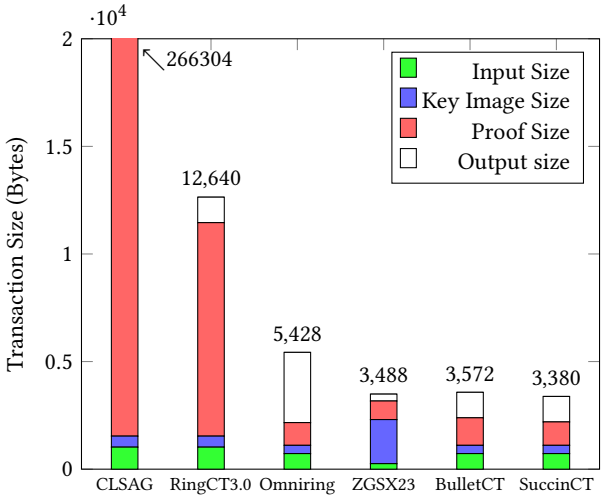

Figure 2: Comparison of confidential transaction in bytes, considering $|\mathbb{G}| = 32$ bytes and the same anonymity space with $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$ as in [31].

$3|\mathbb{G}|$ size. ZGSX23 [31] (S&P '23) used the any-out-of-many proof to hide the number of source accounts $\mathfrak{S}$ and remove $\mathfrak{S}$ from the proof size. However, the set size of the tag is increased to $\mathfrak{R}$ group elements. Their paper did not specify how to integrate with the one-time address and share the output amount and coin key with the recipient. BulletCT [26] (USENIX '25) also applied "K-Weight"-based technique from ZGSX23 to develop a K-out-of-N proof, which can be further integrated with their new tag proofs. Their overall transaction size is still slightly larger than our SuccinCT because of the additional constant factors in proof size.

Compare by Anonymity Space. Based on the above analysis, Omniring improved the proof size but has a larger output size.

Table 1: Comparison of the input size, key image size, proof size and output size in a confidential transaction, in terms of the input set size $\mathfrak{R}$, the number of transaction input $\mathfrak{S}$ and the number of transaction output $\mathfrak{T}$. Here $\mathbb{G}$ is the elliptic curve group used, $|\mathcal{I}|$ is the size to reference an input address, and β is the bit-size of the transaction amount.

Scheme	Input Size	Key Image Size	Proof Size	Output Size
CLSAG [9]	$\mathfrak{R}\|\mathcal{I}\|$	$\mathfrak{S}\|\mathbb{G}\|$	$((2\mathfrak{R}+1)\mathfrak{S}+2\lceil\log_2(\beta\mathfrak{T})\rceil+9)\|\mathbb{G}\|$	$(2\mathfrak{T}+1)\|\mathbb{G}\|+\mathfrak{T}\cdot\frac{\beta}{8}$
RingCT 3.0 [30]	$\mathfrak{R}\|\mathcal{I}\|$	$\mathfrak{S}\|\mathbb{G}\|$	$(2\mathfrak{S}\lceil\log_2(\mathfrak{R}+17)\rceil+2\lceil\log_2(\beta\mathfrak{T})\rceil+2)\|\mathbb{G}\|$	$(2\mathfrak{T}+1)\|\mathbb{G}\|+\mathfrak{T}\cdot\frac{\beta}{8}$
Omniring [11]	$\mathfrak{R}\|\mathcal{I}\|$	$\mathfrak{S}\|\mathbb{G}\|$	$(2\lceil\log_2(3+\mathfrak{R}+\mathfrak{R}\mathfrak{S}+\beta\mathfrak{T}+3\mathfrak{S})\rceil+9)\|\mathbb{G}\|$	$8\mathfrak{T}\|\mathbb{G}\|+2\mathfrak{T}\cdot\frac{\beta}{8}$
ZGSX23 [31]	$\mathfrak{R}\|\mathcal{I}\|$	$\mathfrak{R}\|\mathbb{G}\|$	$(2\lceil\log_2(\mathfrak{R}+\beta\mathfrak{T})\rceil+9)\|\mathbb{G}\|$	?
BulletCT [26]	$\mathfrak{R}\|\mathcal{I}\|$	$\mathfrak{S}\|\mathbb{G}\|$	$(2\lceil\log_2(\mathfrak{R}+\mathfrak{S}+\beta\mathfrak{T})\rceil+20)\|\mathbb{G}\|$	$(2\mathfrak{T}+1)\|\mathbb{G}\|+\mathfrak{T}\cdot\frac{\beta}{8}$
SuccinCT	$\mathfrak{R}\|\mathcal{I}\|$	$\mathfrak{S}\|\mathbb{G}\|$	$(2\lceil\log_2(\mathfrak{R}(\mathfrak{S}+1)+\beta\mathfrak{T})\rceil+10)\|\mathbb{G}\|$	$(2\mathfrak{T}+1)\|\mathbb{G}\|+\mathfrak{T}\cdot\frac{\beta}{8}$

ZGSX23 has the smallest proof size but has a significant increase in key image size (from $\mathfrak{S}$ to $\mathfrak{R}$). Next, we compare the concrete size of different schemes based on the *anonymity space* defined in [31]. Roughly speaking, the anonymity space is the total number of combinations that the sender set can be. ZGSX23 hid the number of signers $\mathfrak{S}$ to maximize its anonymity space to $2^{\mathfrak{R}}$. In Omniring with a ring size $\mathfrak{R}$ and $\mathfrak{S}$ signers, the total number of signer combination is $\binom{\mathfrak{R}}{\mathfrak{S}}$. In RingCT 3.0 and MLSAG, the anonymity space is $(\frac{\mathfrak{R}}{\mathfrak{S}})^{\mathfrak{S}}$. ZGSX23 suggested that by using the same empirical anonymity space, it only requires a 64-sized ring in ZGSX23, while Ominring requires a ring of size 181 and RingCT 3.0 requires a ring of size 256. They set $\mathfrak{S}=\mathfrak{T}=\mathfrak{R}/16$ for all settings. We give the comprehensive analysis of the total transaction size in Figure 2 using these parameters. Each transaction input is represented by a small set $\mathcal{I}$ that is at most 4 bytes (see Appendix E). For ZGSX23, we conjecture their output size by using the smaller CLSAG/RingCT 3.0 setting. However, BulletCT identified several limitations of utilizing anonymity space in any-out-of-many proofs. Detailed analysis are given in Appendix D and I.

In summary, the construction of our SuccinCT is the most efficient in both scenarios by comparing the overall transaction size. The proof size of SuccinCT is much smaller than CLSAG and RingCT3.0, while the sum of key image and output size is also smaller than Omniring and ZGSX23.

1.3 Contributions

The main contributions are summarized as follows:

(1) We identify potential attacks on the miner's account and the transactions including miners' accounts in the ring, which may deteriorate the security of general RingCT protocols. Attacks and solutions are given in Section 3.
(2) Based on the attacks, we propose a stronger security model for CTs in Section 4, capturing the transaction fee, block reward, the miner's account, and the mining process.
(3) We design a new confidential transaction protocol *SuccinCT* in Section 6, which outperforms existing schemes in the overall transaction size, as shown in Figure 1 and 2. The high-level idea of general SuccinCT construction will be introduced in Section 5.
(4) We give a comprehensive analysis of the confidential transaction protocols in Appendix I.
(5) We implement our scheme in Section 7.

To the best of the authors' knowledge, this paper is the first to demonstrate potential attacks on miners, a stronger security model to address these attacks, and the efficiency improvement in terms of overall transaction size.

2 Background

In this section, we introduce the notations used in this work, and illustrate a brief overview of CT schemes. The security of our building blocks relies on the DL, CDH, DDH and ODH assumptions provided in Appendix G.

2.1 Notations

We use $\lambda \in \mathbb{N}$ as the security parameter. We write $\mathsf{negl}(\lambda)$ for a negligible function and $\mathsf{poly}(\lambda)$ for a function polynomial in λ. PPT means probabilistic polynomial time. Given a set S, $x \leftarrow_s S$ means sampling an element x from S uniformly at random. For an algorithm A with input x and output z, we write $z \leftarrow A(x)$. Unless specified otherwise, all sets are implicitly ordered. All algorithms may output $\perp$ upon failure. For a binary relation R, we use the syntax $\{(\mathsf{wit}) : \mathsf{stmt}\}$ to represent the witness wit and the statement stmt in the relation. We use a cyclic group $\mathbb{G}$ of prime order p. For the ease of understanding, we use uppercase letters for group elements in the cyclic group $\mathbb{G}$, and lowercase letters for elements in $\mathbb{Z}_p$. Denote $\mathfrak{R}$, $\mathfrak{S}$, $\mathfrak{T}$ as the number of the (input) ring accounts, the source accounts, and the (output) target accounts, respectively. Denote β as the bit-length of the transaction amount and $|\mathcal{I}|$ as the data size to reference a previous transaction. Table 2 defines symbols used in this paper.

2.2 Confidential Transaction

In Monero, the *source account* $\mathcal{S}$ represents the account owned by the real spender (signer) who initiates the transaction. A set of *decoy accounts* is also called mixin, which is a group of randomly selected accounts used to hide the source accounts. In each transaction, the sender mixes up $\mathfrak{S}$ number of source accounts with $\mathfrak{R}-\mathfrak{S}$ number of decoy accounts to form $\mathfrak{R}$ number of input ring accounts $\mathcal{R}$.

We use RingCT3.0 to demonstrate the accounts arrangement $\mathcal{R}$ in a confidential transaction as Figure 3. The $\mathfrak{S}\times(\mathfrak{R}/\mathfrak{S})$ matrix $\mathbb{A}_{\mathsf{in}}$ contains $\mathfrak{R}$ input accounts in total, so we can denote the number of ring accounts in each row $(\mathfrak{R}/\mathfrak{S})$ as n, and the number of source

Table 2: Definition of symbols.

Symbols	Definitions
$\mathcal{R}, \mathcal{S}, \mathcal{T}$	The set of input ring, real signer and output.
$\mathfrak{R}, \mathfrak{S}, \mathfrak{T}$	The set size of $\mathcal{R}, \mathcal{S}, \mathcal{T}$.
i, j, k	The index within $\mathcal{R}, \mathcal{S}, \mathcal{T}$.
G, U, V, W	Generators of cyclic group $\mathbb{G}$.
sk, pk	Secret key and public key.
ind	Index of the signer in the ring.
H	Hash function.
$\vec{b}$, $\vec{1}^{\mathfrak{R}}$, $\vec{0}^{\mathfrak{R}}$	A binary vector, a vector of ones and a zero vector of length $\mathfrak{R}$.
tag	Key image.
co, a, ck	A coin co which consists of an amount a and a coin key ck.
π, σ	NIZK proof π and signature σ.
β	Bit size of the transaction amount.
$\langle \vec{a}, \vec{b} \rangle$	Inner product of vectors $\vec{a}$ and $\vec{b}$.
acc	Account containing pk and co.
tx	Transaction containing $\mathcal{R}, \mathcal{S}, \mathcal{T}$.

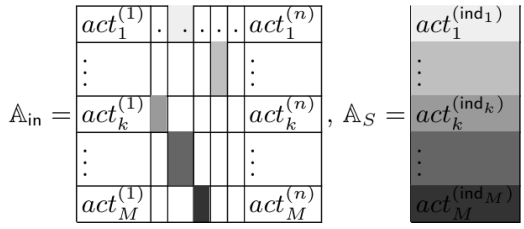

Figure 3: Accounts arrangement copied from RingCT 3.0 [30].

accounts $\mathfrak{S}$ as M. Each row in the matrix can be seen as a ring signature containing $n-1$ decoy accounts and one real source account $\text{acc}_j^{\mathcal{S}} \in \mathcal{S}$ (or $\text{act}_k^{(\text{ind}_k)} \in \mathbb{A}_S$ in Figure 3) in the dark color. Therefore, RingCT3.0 hides $\mathfrak{S}$ source accounts among the group of $\mathfrak{R}$ input accounts, which is used to anonymize the real sender in transactions.

RingCT generates range proofs for all output coins $\text{co}^{\mathcal{T}}$ from target accounts $\mathcal{T}$ to prove that every output amount $a^{\mathcal{T}}$ lies within a certain range. The balance proof shows that the sum of input amounts $a^{\mathcal{S}}$ from $\mathcal{S}$ equals to the sum of output amounts $a^{\mathcal{T}}$ from $\mathcal{T}$. The signer uses his sk corresponding to $\mathcal{S}$ to generate $\mathfrak{S}$ number of key images tag to prevent double spending attacks.

Suppose that $\{G, U, V, W\} \in \mathbb{G}$ are random group generators, Omniring defines the language for a RingCT signature as follow:

$$\begin{cases} \text{stmt} = (\{\text{pk}_i^{\mathcal{R}}, \text{co}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}, \{\text{tag}_j\}_{j=1}^{\mathfrak{S}}, \{\text{co}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}) \\ \exists \text{wit} = (\{\text{sk}_j, a_j^{\mathcal{S}}, \text{ck}_j^{\mathcal{S}}\}_{j=1}^{\mathfrak{S}}, \{a_k^{\mathcal{T}}, \text{ck}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}) \; s.t. : \\ \forall j \in \mathfrak{S} : \text{pk}_j = G^{\text{sk}_j}, \; \text{tag}_j = U^{\text{sk}_j}, \; \text{co}_j^{\mathcal{R}} = V^{a_j^{\mathcal{S}}} W^{\text{ck}_j^{\mathcal{S}}} \\ \forall k \in \mathfrak{T} : \text{co}_k^{\mathcal{T}} = V^{a_k^{\mathcal{T}}} W^{\text{ck}_k^{\mathcal{T}}}, \; a_k^{\mathcal{T}} \in [0, 2^{\beta} - 1] \\ \sum_{j=1}^{\mathfrak{S}} a_j^{\mathcal{S}} = \sum_{k=1}^{\mathfrak{T}} a_k^{\mathcal{T}} \end{cases}$$

3 Attacks on Privacy for the Miner's Accounts

In this section, we first characterize the attacks when miners' accounts are included in confidential transactions, called *miner privacy attack* for the real spender and *anonymity reduction attack* for mixin. We then present a mitigation for these attacks. Before detailing the attacks, we briefly review the categories of Monero miners.

(1) Centralized mining pools. The operators of the centralized mining pools get the coinbase outputs, and pay the miners in a normal transaction.
(2) Solo miners, who occasionally mine a block by themselves and receive the entire block reward.
(3) P2Pool miners. P2Pool is a decentralized mining pool working on a side chain. The coinbase transaction mined by P2Pool will have one output for each individual miner who earned a share, and hence will have many outputs. Usually, a single output is not valuable enough to be worth sending to another party. P2Pool miners usually send many of these outputs together in a "sweep" transaction.

Some examples given in this section involves the view key. It is described in [10] that the receiver "may give a third party his view key. Such a third party could be a trusted custodian, an auditor, a tax authority, etc. Somebody who could be allowed *read access to the user's transaction history, without any further rights*. This third party would also be able to decrypt the output's amount".

3.1 Miner Privacy Attack

Consider the case that the miner's account acc_M receives an amount $\$a_M$ as the *coinbase output* (block reward) for mining a block, and later acc_M spends $\$a_M$ in a one-input one-output transaction tx_B to Bob. Suppose that Bob delegates his *view key* to a third party Eve (e.g. a light-wallet server or a compliance auditor), so Eve can learn that Bob received $\$a^{\text{output}}$ satisfying $\$a^{\text{output}} + \$a^{\text{fee}} = \$a_M$, where $\$a^{\text{fee}}$ is the transaction fee, as the Attack 1 in Figure 4. Since the input ring of the transaction tx_B includes acc_M, and Eve knows from the publicly visible coinbase transaction that acc_M controls an output of $\$a_M$, Eve can identify that acc_M is the real spender, thereby compromising the anonymity of miner.

Monero has enforced that there should be at least two outputs in each transaction since fork version 12 in 2019. However, this attack is still effective for the miners' accounts even with the rule. For example, Eve who holds Bob's view key, sees that Bob receives an amount $\$a^{\text{output}}$ in a transaction tx_B. Eve then observes that the input ring of tx_B contains a miner's coinbase output of $\$a_M$ with $\$a^{\text{output}} + \$a^{\text{fee}} = \$a_M$. Eve can deduce that the second output in tx_B is actually a zero amount, therefore the miner's coinbase output must be the real input. The anonymity of miners is still compromised even if tx_B satisfies the two-output requirement. Hence, this attack could happen if the view keys of all output accounts (receiving non-zero amount) in the miner's spending transaction are known by the adversary.

Furthermore, this attack may also be launched from centralised exchange services, which have access to the private key/view key of their clients on Monero, or government agencies who have access to viewing keys (potentially provided by centralised services for meeting the Know Your Customer regulation). This would lead to

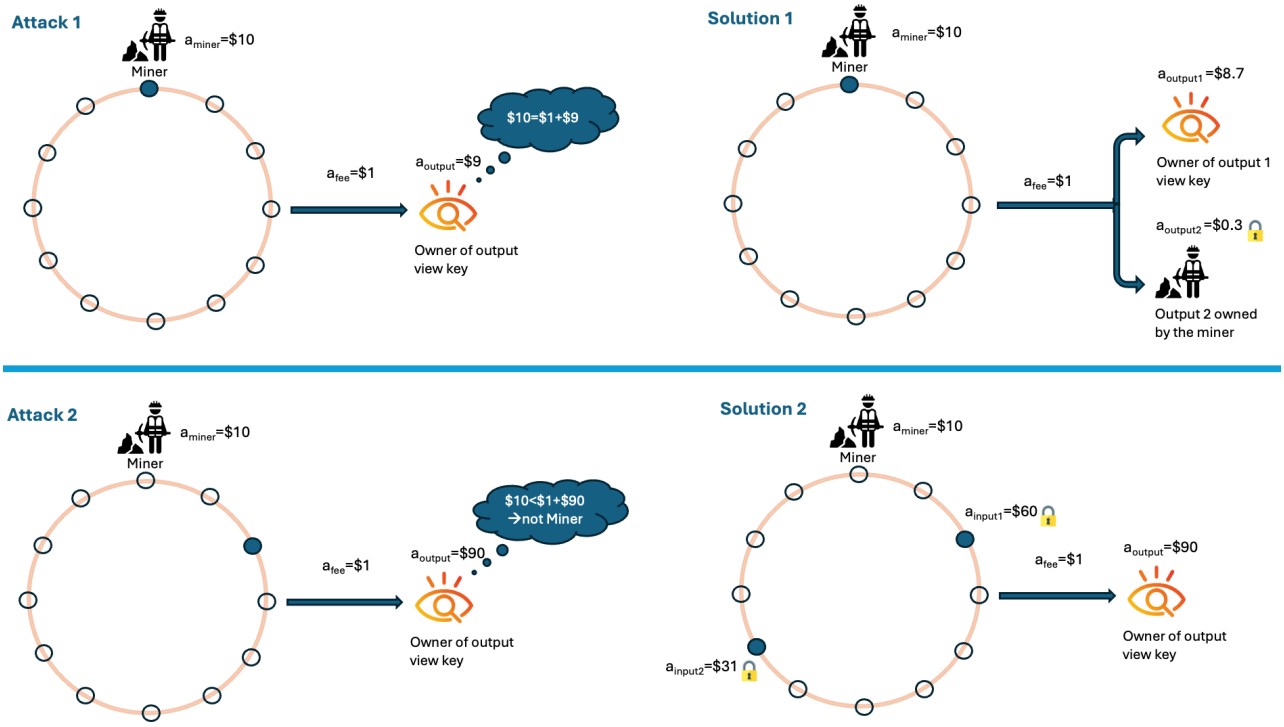

Figure 4: Two privacy attacks for the miner's account. A solid circle represents a real sender and an empty circle represents a mixin. The attacker has the output view key. The lock icon represents information that is hidden from the attacker.

reduced privacy for the entire system, due to the cascade effect. For instance, if an adversary knows that the miner is the real spender in tx_B, it can infer that this miner's address, when it appears in the rings of other transactions, is not the real sender. Reflecting the attack, it has a larger impact when an attacker has access to multiple (potentially a large number of) viewing keys. Consequently, rational users will try to not include any miner's account in their input rings. It forms a vicious cycle: A transaction with a miner's account in the ring will then be more likely to come from the miner. This attack affects (1) the pool operators of centralized mining pools (but not their users), (2) solo miner, and (3) P2Pool miners.

Monero plans to activate "Full Chain Membership Proofs" (FCMP) [18] in 2025, which will eliminate rings and their associated privacy downsides. Even with FCMP, a recipient (who knows the view key) of the full balance of coinbase outputs can still guess that they are receiving coinbase outputs from the miner. The recipient can search all coinbase outputs on the blockchain instead of the coinbase outputs solely in a single ring. When a transaction has k multiple inputs, the recipient can solve the associated subset sum problem by searching for the k coinbase outputs that is added up to the output amount plus the transaction fee. Hence, the miner privacy attack is still valid after FCMP is activated.

Attacks Intuition. The insight behind this attack is: (i) the coinbase output amount $\$a_M$, determined by the Monero protocol, is publicly known and; (ii) the transaction fee $\$a^{\text{fee}}$ is transparent to all users when a block of transactions is confirmed. Therefore, if the miner spends its coinbase output of amount $\$a_M$ to Bob in tx_B, any third party that holds Bob's view key can infer this miner is the real spender in tx_B.

A straightforward idea would be to hide the miner's coinbase amount $\$a_M$ by encrypting it and using a Pedersen commitment, similar to regular RingCT outputs. However, this method breaks the public verification of the consensus mechanism in Monero. Specifically, full nodes cannot check that the committed amount equals to the protocol-defined coinbase amount. Although the problem can be solved by attaching an additional zero-knowledge proof, it would significantly increase the complexity and the cost of consensus.

Our Approach. We propose a simple and efficient method called *self-transfer mechanism*, as the Solution 1 in Figure 4: a miner can spend its block reward only if it sends some non-zero amount back to another non-mining account owned by itself, where the view key of the non-mining account should not be disclosed, shared or under custodian. It ensures that any third party, who knows the view key of recipients, cannot identify the miner as the real spender.

However, we note that this approach may not be economically feasible for a P2Pool miner to spend only one coinbase output at a time. In a P2Pool, a single block reward may be split into a large number of small coinbase outputs, one for each contributing miner. As a result, an individual P2Pool miner receives many tiny coinbase outputs over time. If the miner spends a coinbase output while attaching a self-transfer output to every transaction, the transaction fees would become prohibitive.

To mitigate this problem, we further propose a *multi-self-transfer mechanism*. A P2Pool miner can spend multiple coinbase outputs in

a single multi-input transaction to at least two non-mining accounts, so that the amount held by each non-mining account is hidden. In this way, the miner can hide its identity while only sending one transaction, thereby reducing the transaction fees.

Our stronger security model captures this attack in the privacy property (see Figure 6 in Section 4.2), which requires that at least one honest receiver's view key remains hidden if an honest miner account is chosen in the ring.

3.2 Anonymity Reduction Attack

We discover a theoretical *anonymity reduction attack* for transactions involving miner accounts in the ring. We start from a hypothetical case. If a miner account acc_M holding the amount $\$a_M$ is included in the ring of a transaction as a decoy account with one input only ($\mathfrak{S} = 1$), and the transaction fee exceeds $\$a_M$ ($\$a^{\mathsf{fee}} > \$a_M$), then acc_M must not be the real sender. However, a transaction with such a high transaction fee is not realistic because $\$a_M$ is lower bounded by the block reward (~USD100 as of November 2024) while $\$a^{\mathsf{fee}}$ is usually less than USD0.01 in practice.[1]

Next, we consider a more realistic case where the view keys of some outputs in the transaction tx_B with amount $\$a^{\mathcal{T}}$ are known to the adversary Eve. If Eve observes that $\$a_M < \$a^{\mathsf{fee}} + \$a^{\mathcal{T}}$, then Eve can confirm that acc_M must not be the real sender, as the Attack 2 in Figure 4. Hence, the anonymity space of the transaction is reduced. This attack applies when a view-key adversary learns one or more output amounts that, together with the transaction fee, exceed the known coinbase value of a miner account included in the input ring.

The anonymity reduction attack can be generalized for transactions with multiple inputs ($\mathfrak{S} \geq 1$). Suppose that there are $\mathfrak{S}$ miner accounts $\mathsf{acc}_{M_1}, \ldots, \mathsf{acc}_{M_{\mathfrak{S}}}$ with amount $\$a_{M_1}, \ldots \$a_{M_{\mathfrak{S}}}$ respectively included in the ring (in a single ring as in Omniring and SuccinCT, or in $\mathfrak{S}$ rings as in CLSAG and RingCT 3.0). If Eve observes that $\sum_{i=1}^{\mathfrak{S}} \$a_{M_i} < \$a^{\mathsf{fee}} + \sum_{k=1}^{\mathfrak{T}} \$a_k^{\mathcal{T}}$, then the transaction outputs must not come from the set of miners $\mathsf{acc}_{M_1}, \ldots, \mathsf{acc}_{M_{\mathfrak{S}}}$. Therefore, the effective anonymity space of the transaction is reduced.

Attacks Intuition. The key insight behind this attack is identical to the miner privacy attack, due to the transparency of miner coinbase outputs and transaction fees. In the case that the input rings of a transaction include coinbase outputs and the transaction's total output amount exceeds the sum of the amounts in these coinbase outputs, an adversary can rule out the possibility that all real inputs come from those miners, thereby reducing the anonymity space.

Our Approach. We propose a *non-coinbase input requirement*, as the Solution 2 in Figure 4: for the transaction with $\mathfrak{S}$ inputs, (1) there should be at most $\mathfrak{S} - 1$ miner accounts in the single ring structure like Omniring and SuccinCT, (2) for the $\mathfrak{S}$ rings structure like CLSAG and RingCT 3.0, there should be at most $\mathfrak{S} - 1$ rings including miner accounts. It implies that at least one non-mining account is included in the real spender set, indicating that at least one input amount is hidden from the adversary. Therefore, the adversary cannot infer whether the real spender set includes any miner account, so the anonymity set is not reduced.

[1] This attack does not require the view key.

Our stronger security model also captures this attack (see Figure. 5 in Section 4.2) by requiring that there are at most $\mathfrak{S} - 1$ miner accounts in the input ring. For the case of hidden $\mathfrak{S}$ such as ZGSX23 [31], we also identify an open problem in Appendix B.

4 Security Analysis

In this paper, we improve the security model of the confidential transaction in [11] by:

- Considering the transaction fee included in each transaction, also the block reward (which affects the balance definition), the creation of the miner's account (in which the amount owned by this account is known by the public), and using the miner account for spending (which affects the privacy definition).
- Integrating the view key directly into the security model, and consider the security properties when the view key is given to the adversary.

We showed that the combination of the view key and the miner's account may threaten the security of the confidential transaction protocol in the previous section. It demonstrates the importance of our new security model.

4.1 Security Notion

The confidential transaction protocol consists of a tuple of polynomial time algorithms (**Setup**, **SAKeyGen**, **OTAccGen**, **View**, **Receive**, **Spend**, **Verify**, **Mine**):

- $(\mathsf{pp}, \mathbb{B}) \leftarrow \mathbf{Setup}(1^\lambda, 1^\iota, 1^\beta)$: The setup algorithm takes as input a security parameter 1^λ and integers $1^\iota, 1^\beta$, where 1^ι represents an upper bound 2^ι on the number of outputs in a single transaction[2] and 1^β represents an upper bound 2^β of the amount given to each transaction output. It outputs the public parameters pp and a public blockchain $\mathbb{B}$ (an append-only data structure which is initially empty).
- $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vdk}) \leftarrow \mathbf{SAKGen}(\mathsf{pp}, q)$: The stealth address key generation algorithm takes as inputs the public parameter pp and the maximum number that mpk can be used q, and outputs a master public key mpk, a master secret key msk and a view secret key vdk.
- $(\mathsf{ck}, \mathsf{acc}, \mathsf{info}) \leftarrow \mathbf{OTAccGen}(\mathsf{mpk}, a, \kappa)$: The one-time account generation algorithm takes a master public key mpk, an amount $a \in [0, 2^\beta - 1]$, and some index κ[3] as inputs. It outputs a coin key ck, an account acc, and the view information info.
- $a/\perp \leftarrow \mathbf{View}(\mathsf{vdk}, \mathsf{acc}, \mathsf{info})$: The view algorithm inputs a viewing key vdk, an account acc, and viewing information info. It outputs an amount a stored in the account acc, or outputs $\perp$ if vdk is not a valid viewing key for acc.
- $(\mathsf{ck}, a, \mathsf{sk}, \mathsf{tag})/\perp \leftarrow \mathbf{Receive}(\mathsf{msk}, \mathsf{acc}, \mathsf{info})$: The receive algorithm takes as inputs a master secret key msk, an account acc and some viewing information info. It outputs a coin key ck, an amount a, a secret key sk and a tag, or outputs $\perp$ if msk is not a valid master secret key for acc.

[2] ι is set as 4 since Monero v8 protocol.

[3] One can generate different one-time accounts for the same mpk with the same amount a in a transaction, by using a different (output) index.

- $\sigma \leftarrow \textbf{Spend}(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\mathsf{fee}}, \mu)$: The spend algorithm takes the following inputs:
  - $\mathcal{R} = \{\mathsf{acc}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}$: a set of ring accounts $\mathsf{acc}_i^{\mathcal{R}}$, which can be a target account of a previous transaction or a miner account of a previous block stored in $\mathbb{B}$. The total number of miner accounts in $\mathcal{R}$ is at most $\mathfrak{S} - 1$[4].
  - $\mathcal{S} = \{\mathsf{ind}_j, \mathsf{ck}_j^{\mathcal{S}}, a_j^{\mathcal{S}}, \mathsf{sk}_j, \mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}$: a set of tuples consisting of an index $\mathsf{ind}_j \in [1, \mathfrak{R}]$, a coin key $\mathsf{ck}_j^{\mathcal{S}}$, an amount $a_j^{\mathcal{S}}$, a secret key sk_j, and a tag tag_j of $\mathsf{acc}_{\mathsf{ind}_j}^{\mathcal{R}}$.
  - $\mathcal{T} = \{\mathsf{ck}_k^{\mathcal{T}}, a_k^{\mathcal{T}}, \mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}$: a set of tuples consisting of a coin key $\mathsf{ck}_k^{\mathcal{T}}$, an amount $a_k^{\mathcal{T}}$, a target account $\mathsf{acc}_k^{\mathcal{T}}$ and the viewing information $\mathsf{info}_k^{\mathcal{T}}$. Also $2 \le \mathfrak{T} \le 2^{\iota}$[5]. and $0 \le a_k^{\mathcal{T}} \le 2^{\beta} - 1$.
  - a^{fee}: the transaction fee given to the miner, where $0 \le a^{\mathsf{fee}} \le 2^{\beta} - 1$.
  - μ: an optional message to be signed.

  It outputs a proof σ.
- $0/1/\perp \leftarrow \textbf{Verify}(\mathsf{tx}, \sigma)$: The verify algorithm takes as inputs a transaction tx and a signature σ. A transaction tx defined by a function TX as follows:

$$\begin{aligned}\mathsf{tx} &= \mathsf{TX}(\mathcal{R}, \mathcal{S}, \mathcal{T}, a_{\mathsf{fee}}, \mu)\\ &:= \left(\{\tilde{\mathsf{acc}}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}, \{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}, \{\mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}, a^{\mathsf{fee}}, \mu\right),\end{aligned}$$

  where $\tilde{\mathsf{acc}}_i^{\mathcal{R}}$ is the reference pointer to the transaction output $\mathsf{acc}_i^{\mathcal{R}}$ in $\mathbb{B}$. It outputs 0 if σ is invalid signature. It outputs $\perp$ if any $\tilde{\mathsf{acc}}_i^{\mathcal{R}}$ is an invalid pointer, $\mathfrak{T} = 1$ or $\mathfrak{T} > 2^{\iota}$, or any tag_j appears in $\mathbb{B}$. Else, it outputs 1 for valid.
- $(\mathsf{B}, \mathsf{acc}_M) \leftarrow \textbf{Mine}(\{\mathsf{tx}, \sigma\}, a_M)$: The mining algorithm takes as inputs a set of transaction-signature pairs $\{\mathsf{tx}, \sigma\}$, and the mining reward $a_M \in [0, 2^{\beta} - 1]$. It outputs a block of transactions B and the miner's account acc_M[6] receiving a_M and all a^{fee} in $\{\mathsf{tx}\}$. The block B and the miner's account acc_M will be added to $\mathbb{B}$[7].

We define two extra functions that are useful in our security definitions.

- $0/1 \leftarrow \textbf{CheckAmount}(\mathsf{acc}, \mathsf{ck}, a)$. The amount checking algorithm takes as inputs an account acc, a coin key ck, and an amount a. It outputs a bit b indicating the consistency of the amount.
- $0/1 \leftarrow \textbf{CheckTag}(\mathsf{acc}, \mathsf{sk}, \mathsf{tag})$. The tag checking algorithm takes as inputs an account acc, a secret key sk, and a tag tag. It outputs a bit b indicating the consistency of the tag.

Definition 4.1 (Correctness). A confidential transaction protocol is correct if the following holds for all $\lambda, \iota, \beta, q \in \mathbb{N}$, and for all $(\mathsf{pp}, \mathbb{B}) \in \textbf{Setup}(1^{\lambda}, 1^{\iota}, 1^{\beta})$.

- Honestly generated payments should be received correctly. Concretely, for any $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vdk}) \in \textbf{SAKGen}(\mathsf{pp}, q)$, any amount $a \in [0, 2^{\beta} - 1]$, any index κ, any $(\mathsf{ck}, \mathsf{acc}, \mathsf{info}) \in \textbf{OTAccGen}(\mathsf{mpk}, a, \kappa)$, and any $(\mathsf{ck}', a', \mathsf{sk}, \mathsf{tag}) \in \textbf{Receive}(\mathsf{msk}, \mathsf{acc}, \mathsf{info})$, it holds that $(\mathsf{ck}, a) = (\mathsf{ck}', a')$.
- Honestly generated transactions should be recognized as valid. Concretely, for any tuple $(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\mathsf{fee}}, \mu)$ with syntax as defined above that satisfies
  - $2 \le \mathfrak{T} \le 2^{\iota}$,
  - for all $k \in \mathfrak{T}$, $a_k^{\mathcal{T}} \in [0, 2^{\beta} - 1]$,
  - $a^{\mathsf{fee}} \in [0, 2^{\beta} - 1]$,
  - for all $i \in \mathfrak{R}$, $\mathsf{acc}_i^{\mathcal{R}} \in \mathbb{B}$,
  - the total number of miner accounts in $\mathcal{R}$ is at most $\mathfrak{S} - 1$,
  - for all $j \in \mathfrak{S}$, $\textbf{CheckTag}(\mathsf{acc}_{\mathsf{ind}_j}^{\mathcal{R}}, \mathsf{sk}_j, \mathsf{tag}_j) = 1$,
  - for all $j \in \mathfrak{S}$, $\textbf{CheckAmount}(\mathsf{acc}_{\mathsf{ind}_j}^{\mathcal{R}}, \mathsf{ck}_j^{\mathcal{S}}, a_j^{\mathcal{S}}) = 1$,
  - for all $k \in \mathfrak{T}$, $\textbf{CheckAmount}(\mathsf{acc}_k^{\mathcal{T}}, \mathsf{ck}_k^{\mathcal{T}}, a_k^{\mathcal{T}}) = 1$,
  - $\sum_{j=1}^{\mathfrak{S}} a_j^{\mathcal{S}} - a_{\mathsf{fee}} = \sum_{k=1}^{\mathfrak{T}} a_k^{\mathcal{T}}$,

  and for any proof $\sigma \in \textbf{Spend}(\mathcal{R}, \mathcal{S}, \mathcal{T}, a_{\mathsf{fee}}, \mu)$, it holds that $\textbf{Verify}(\mathsf{tx}, \sigma) = 1$, where $\mathsf{tx} = \mathsf{TX}(\mathcal{R}, \mathcal{S}, \mathcal{T}, a_{\mathsf{fee}}, \mu)$.
- Honest miners receive all transaction fees and the block reward. Concretely, for any $(\mathsf{mpk}_M, \mathsf{msk}_M, \mathsf{vdk}_M) \in \textbf{SAKGen}(\mathsf{pp}, q)$, any amount $a_M \in [0, 2^{\beta} - 1]$, any honestly generated transaction-signature pair $(\mathsf{tx}_\ell, \sigma_\ell)$ defined above with transaction fee a_ℓ^{fee}, any $(\mathsf{B}, \mathsf{acc}_M) \leftarrow \textbf{Mine}(\{\mathsf{tx}_\ell, \sigma_\ell\}, \mathsf{mpk}_M, a_M)$, $(\mathsf{ck}_M, \mathsf{acc}_M, \mathsf{info}) \in \textbf{OTAccGen}(\mathsf{mpk}_M, a_M - \sum_\ell a_\ell^{\mathsf{fee}}, 0)$, and any $(\mathsf{ck}', a', \mathsf{sk}, \mathsf{tag}) \in \textbf{Receive}(\mathsf{msk}_M, \mathsf{acc}_M, \mathsf{info})$, it holds that $(\mathsf{ck}_M, a_M - \sum_\ell a_\ell^{\mathsf{fee}}) = (\mathsf{ck}', a')$.
- Correctly received payments should have well-defined amounts and tags. Concretely, for any $(\mathsf{ck}, a, \mathsf{sk}, \mathsf{tag}) \in \textbf{Receive}(\mathsf{msk}, \mathsf{acc}, \mathsf{info})(\neq \perp)$, it holds that $\textbf{CheckAmount}(\mathsf{acc}, \mathsf{ck}, a) = 1$ and $\textbf{CheckTag}(\mathsf{acc}, \mathsf{sk}, \mathsf{tag}) = 1$.
- Honestly generated payment should be viewed correctly. Concretely, for any $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vdk}) \in \textbf{SAKGen}(\mathsf{pp}, q)$, any amount $a \in [0, 2^{\beta} - 1]$, any index κ, any $(\mathsf{ck}, \mathsf{acc}, \mathsf{info}) \in \textbf{OTAccGen}(\mathsf{mpk}, a, \kappa)$, and any $a' \in \textbf{View}(\mathsf{vdk}, \mathsf{acc}, \mathsf{info})(\neq \perp)$, it holds that $a = a'$.

4.2 Security Models

In this paper, we assume that the blockchain $\mathbb{B}$ is secure: $\mathbb{B}$ is an append-only data structure and only valid transactions can be added.

Privacy. Privacy captures anonymity for both the spenders and the receivers, and the confidentiality of the amounts being transferred. The adversary is allowed to specify a ring with arbitrarily many corrupt accounts, and two honest subsets of the ring which are the potential spenders. The adversary also specifies two sets of receivers and the amounts that they are supposed to receive. A transaction is then created using one of the two specifications of the adversary, who should not be able to tell which specification is used to create the transaction.

More concretely, we model privacy in the security experiments $\mathsf{Privacy}^b$ for $b \in \{0, 1\}$ defined in Figure 6, with oracles defined in Figure 5. Let $\mathcal{A}$ be a PPT adversary who, after several queries

[4] This new requirement is based on our analysis in Section 3.2.
[5] This new requirement is based on our analysis in Section 3.1, and Monero also has enforced it since v8 protocol.
[6] The coin key for the miner account is defined as 1 since v4 of Monero in 2017.
[7] For simplicity, this function only considers the miners who successfully mined a block. It implies that all transactions in B are valid. Unsuccessful mining is not represented here.

to the oracles, produces an incomplete input $(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\text{fee}}, \mu)$ to the **Spend** algorithm, along with two instructions I and J. The incomplete input $(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\text{fee}}, \mu)$ corresponds to all malicious information that will be used to generate a transaction and its proof, while the instructions I and J specify how the sets $\mathcal{S}$ and $\mathcal{T}$ should be populated by information held by the honest users. Following the adversary's instructions, the experiment duplicates $(\mathcal{S}, \mathcal{T})$ into $(\mathcal{S}_0, \mathcal{T}_0)$ and $(\mathcal{S}_1, \mathcal{T}_1)$, and populates $(\mathcal{R}, \mathcal{S}_n, \mathcal{T}_n)$ for $n \in \{0, 1\}$ as follows. The instruction I corresponds to the source accounts. The experiment retrieves the account information required to spend by calling TryReceive if possible. It sets $\mathcal{S}_n$ to the appropriate sender information. Similarly, the instruction J corresponds to the target accounts. The experiment creates a one-time account using the master public key of the user and the amount specified in J, and set $\mathcal{T}_n$ to the appropriate receiver information.

Definition 4.2. A confidential transaction protocol is private if for all PPT adversaries $\mathcal{A}$, and all positive integers $\iota, \beta \in \text{poly}(\lambda)$,

$$\left|\Pr[\text{Privacy}^0_{\mathcal{A}}(1^\lambda, 1^\iota, 1^\beta) = 1] - \Pr[\text{Privacy}^1_{\mathcal{A}}(1^\lambda, 1^\iota, 1^\beta) = 1]\right| \leq \text{negl}(\lambda),$$

where $\text{Privacy}^b_{\mathcal{A}}(1^\lambda, 1^\iota, 1^\beta)$ is defined in Figure 6, and also there was no query $\text{ViewO}(e_{\ell,n}, \text{acc}^{\mathcal{T}}_{\ell,n}, \text{info}^{\mathcal{T}}_{\ell,n})$ for all $\ell \in [1, |J|], n = 0, 1$.

Difference with [11]. There are a number of important differences between our new privacy with the privacy model in [11].

(1) The inclusion of the miners account, and detecting the miner accounts in I by running TryReceive.
(2) We explicitly allow either $|I| = 0$ or $|J| = 0$, which means that the adversary can completely control either the source or target accounts. [11] did not mention if this is allowed.
(3) We define that if an honest miner account is chosen in I, then there must be at least one honest receiver. It corresponds to our attack and recommendation in Section 3.1.

The security models of balance and non-slanderability will be given in Appendix H.2.

5 Overview of Our Solution

We give the overview of our scheme SuccinCT. We start to build a confidential transaction using the concept of *sign first, compress later* from DualRing-EC. Specifically, we use a Schnorr-based multi-signature first, then use the Bulletproof to compress the ring (the signer index only) later. It is structurally different from RingCT 3.0 [30] and Omniring [11], which use the Bulletproof to prove the knowledge of all witnesses, including the signer index, the secret key, the transaction amount, etc.

Step 1 - Single Input Case. Suppose that G is the generator of a group, $\{\text{pk}_i\}_{i=[1,\mathfrak{R}]}$ to a set of $\mathfrak{R}$ public keys, μ is the message, and H is a hash function. In order to solve the single binding issue (see Appendix A.2), we have to add an additional constraint to $c_1, \ldots, c_{\mathfrak{R}}$, so that we have a special case of DualRing-EC:

$$G^z = R \cdot \prod_{i=1}^{\mathfrak{R}} \text{pk}_i^{c_i}, \quad \sum_{i=1}^{\mathfrak{R}} c_i = H(\mu, R, \{\text{pk}_i\}), \quad c_i = 0 \ \forall i \neq \text{ind},$$

where $R = G^r$ and $z = r + c_{\text{ind}} \cdot \text{sk}_{\text{ind}}$. However, the additional condition on c_i will leak information about the signer index ind and breaks the anonymity of the ring signature. Hence, we need to use a non-interactive zero-knowledge (NIZK) proof for inner product argument instead of the non-interctive sum argument in DualRing-EC. Specifically, define a binary vector $(b_1, \ldots, b_{\mathfrak{R}})$ so

InitOracles()
MPK := ∅, MSK := ∅, VDK := ∅
Spent := ∅, Σ := ∅

SAKGenO(q)
(mpk, msk, vdk) ← **SAKGen**(pp, q)
MPK := MPK||mpk
MSK := MSK||(msk, q)
VDK := VDK||vdk
return mpk

TryReceive(acc)
$z := -1$
if acc is a transaction output in $\mathbb{B}$, $z = 1$
if acc is a miner account in $\mathbb{B}$, $z = 0$
retrieve info of acc from $\mathbb{B}$
for $\ell \in [1, |\text{MSK}|]$
  (msk, q) := MSK[ℓ]
  if $q = 0$
    continue
  (ck, a, sk, tag) ← Receive(msk, acc, info)
  if (ck, a, sk, tag) ≠ ⊥
    MSK[ℓ] = (msk, $q - 1$)
    return (ck, a, sk, tag, z)
return ⊥

GetViewKey()
return VDK

SpendO($I, \mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\text{fee}}, \mu$)
parse I as $\{s_\ell, \text{ind}_\ell\}_{\ell=1}^{|I|}$
for $\ell \in [1, |I|]$
  $\text{acc}_\ell := \mathcal{R}[\text{ind}_\ell]$
  $(\text{ck}_\ell, a_\ell, \text{sk}_\ell, \text{tag}_\ell, z_\ell) \leftarrow \text{TryReceive}(\text{acc}_\ell)$
  $\mathcal{S}[s_\ell] := (\text{ind}_\ell, \text{ck}_\ell, a_\ell, \text{sk}_\ell, \text{tag}_\ell)$
tx = TX($\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\text{fee}}, \mu$)
$\sigma \leftarrow \text{Spend}(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\text{fee}}, \mu)$
$\Sigma := \Sigma \cup \{\text{tx}, \sigma\}$
if **Verify**(tx, σ) = 0, **return** 0
Spent = Spent $\cup \{\text{tag}_\ell\}_{\ell=1}^{|I|}$
return σ

ViewO(ℓ, acc, info)
return **View**(VDK[ℓ], acc, info)

Figure 5: Oracles.

$\mathsf{Privacy}^{b}_{\mathcal{A}}(1^\lambda, 1^\iota, 1^\beta)$

$(\mathsf{pp}, \mathbb{B}) \leftarrow \mathbf{Setup}(1^\lambda, 1^\iota, 1^\beta)$, InitOracles()
$(I, J, \mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\mathsf{fee}}, \mu) \leftarrow \mathcal{A}^{\mathsf{SAKGenO},\mathsf{SpendO}}(\mathsf{pp}, \mathbb{B})$
$\mathcal{S}_0 := \mathcal{S}, \mathcal{S}_1 := \mathcal{S}, \mathcal{T}_0 := \mathcal{T}, \mathcal{T}_1 := \mathcal{T}$
parse I as $\{(s_\ell, \{\mathsf{ind}_{\ell,n}\}_{n=0,1})\}_{\ell=1}^{|I|}$
parse J as $\{(d_\ell, \{e_{\ell,n}, a^{\mathcal{T}}_{\ell,n}\}_{n=0,1})\}_{\ell=1}^{|J|}$
if $|I| = |J| = 0$ **return** $\perp$
// Preparing honest spenders chosen by the adversary
for $\ell \in [1, |I|]$
  for $n \in [0, 1]$
    $\mathsf{acc}_{\ell,n} := \mathcal{R}[\mathsf{ind}_{\ell,n}]$
    $(\mathsf{ck}_{\ell,n}, a_{\ell,n}, \mathsf{sk}_{\ell,n}, \mathsf{tag}_{\ell,n}, z_{\ell,n}) \leftarrow \mathsf{TryReceive}(\mathsf{acc}_{\ell,n})$
    $\mathcal{S}_n[s_\ell] := (\mathsf{ind}_{\ell,n}, \mathsf{ck}_{\ell,n}, a_{\ell,n}, \mathsf{sk}_{\ell,n}, \mathsf{tag}_{\ell,n})$
  if $\mathsf{tag}_{\ell,0} \neq \mathsf{tag}_{\ell,1} \wedge \{\mathsf{tag}_{\ell,0}, \mathsf{tag}_{\ell,1}\} \cap \mathsf{Spent} \neq \emptyset$
    return $\perp$
// If an honest miner account is chosen,
// there must be at least one honest receiver
  if $|J| = 0 \wedge (z_{\ell,0} = 0 \vee z_{\ell,1} = 0)$
    return $\perp$
// Preparing honest receivers chosen by the adversary
for $\ell \in [1, |J|]$
  for $n \in [0, 1]$
    $(\mathsf{ck}^{\mathcal{T}}_{\ell,n}, \mathsf{acc}^{\mathcal{T}}_{\ell,n}, \mathsf{info}^{\mathcal{T}}_{\ell,n}) := \mathbf{OTAccGen}(\mathsf{MPK}[e_{\ell,n}], a^{\mathcal{T}}_{\ell,n}, d_\ell)$
    $\mathcal{T}_n[d_\ell] := (\mathsf{ck}^{\mathcal{T}}_{\ell,n}, a^{\mathcal{T}}_{\ell,n}, \mathsf{acc}^{\mathcal{T}}_{\ell,n}, \mathsf{info}^{\mathcal{T}}_{\ell,n})$
for $n \in [0, 1]$
  $\mathsf{tx}_n := \mathsf{TX}(\mathcal{R}, \mathcal{S}_n, \mathcal{T}_n, a^{\mathsf{fee}}, \mu)$
  $\sigma_n \leftarrow \mathbf{Spend}(\mathcal{R}, \mathcal{S}_n, \mathcal{T}_n, a^{\mathsf{fee}}, \mu)$
  if $\mathbf{Verify}(\mathsf{tx}_n, \sigma_n) = 0$, **return** $\perp$
$b' \leftarrow \mathcal{A}^{\mathsf{SAKGenO},\mathsf{SpendO},\mathsf{ViewO}}(\mathsf{tx}_b, \sigma_b)$
return b'

Figure 6: Privacy.

that $b_{\mathsf{ind}} = 1$ and $b_i = 0$ for $i \neq \mathsf{ind}$. We need a NIZK proof for:

$$\{(z, b_1, \ldots, b_{\mathfrak{R}}) : \quad G^z = R \cdot (\prod_{i=1}^{\mathfrak{R}} \mathsf{pk}_i^{b_i})^h, \quad h = H(\mu, R, \{\mathsf{pk}_i\}), \\ (b_1, \ldots, b_{\mathfrak{R}}) \text{ is a binary vector with a single 1.}\}$$

Using the Bulletproof approach [4], the last condition on vector $\vec{b} = (b_1, \ldots, b_{\mathfrak{R}})$ is equivalent to:

$$\vec{b} \circ (\vec{1}^{\mathfrak{R}} - \vec{b}) = \vec{0}^{\mathfrak{R}}, \quad \langle \vec{b}, \vec{1}^{\mathfrak{R}} \rangle = 1,$$

where $\vec{1}^{\mathfrak{R}}$ and $\vec{0}^{\mathfrak{R}}$ are unit vector and zero vector of length $\mathfrak{R}$. Now observe that the signature part $z = r + h \cdot \mathsf{sk}_{\mathsf{ind}}$ is almost identical to the Schnorr signature, except that all public keys of the ring are put in the hash function. Since G^z will leak the information about $G^{\mathsf{sk}_{\mathsf{ind}}}$, we also need to put z as the witness of the NIZK proof.

Next, we set the key image as $\mathsf{tag} = U^{\mathsf{sk}_{\mathsf{ind}}}$ for some generator U[8] and define $R_U = U^r$. We also need a NIZK proof for the relation:

$$\{(z) : \ U^z = R_U \cdot \mathsf{tag}^h\}.$$

[8]This tag structure will be discussed later in this section.

Consider the signer's input coin $\mathsf{co}^{\mathcal{R}}_{\mathsf{ind}} = V^a W^{\mathsf{ck}}$ for some amount a, coin key ck and group generators V, W. Denote the output coin as $\mathsf{co}^{\mathcal{T}} = V^a W^{\mathsf{ck}^{\mathcal{T}}}$ and $\mathsf{ck}' = \mathsf{ck} - \mathsf{ck}^{\mathcal{T}}$. Hence we need a NIZK proof π for:

$$\{(\mathsf{ck}', b_1, \ldots, b_{\mathfrak{R}}) : \prod_{i=1}^{\mathfrak{R}} (\mathsf{co}_i^{\mathcal{R}})^{b_i} = \mathsf{co}^{\mathcal{T}} \cdot W^{\mathsf{ck}'}\}.$$

Recall that the Bulletproof proof for the above relations has a size of $O(\log \mathfrak{R})$.

Step 2 - Combining Multiple Input using MuSig. We preserve the structure of Schnorr signature (R, z) with the use of DualRing-EC. When applied to the case of multiple input, we aggregate multiple Schnorr-based ring signatures using the multi-signature scheme MuSig [15].

Suppose that we have the same set of $\mathfrak{R}$ accounts $\mathcal{R} = (\mathsf{pk}_i, \mathsf{co}_i)$ for $i \in [1, \mathfrak{R}]$ for all $\mathfrak{S}$ inputs. Denote the index of jth input account is ind_j for $j \in [1, \mathfrak{S}]$. We define $\mathfrak{S}$ binary vectors

$$\{\vec{b^{(j)}} = (b_1^{(j)}, b_2^{(j)}, \ldots, b_{\mathfrak{R}}^{(j)})\}_{j=1}^{\mathfrak{S}},$$

where $b_i^{(j)} = 0$ for $i \neq \mathsf{ind}_j$ and $b_{\mathsf{ind}_j}^{(j)} = 1$. Hence, the "public key" for the jth input is $(\prod_{i=1}^{\mathfrak{R}} \mathsf{pk}_i^{b_i^{(j)}})$, and the corresponding key image is $\mathsf{tag}_j = U^{\mathsf{sk}_{\mathsf{ind}_j}}$. Denote $\alpha_j = H_a(\mathcal{R}, j)$ for $j \in [1, \mathfrak{S}]$. By using MuSig, the aggregated public key becomes $\prod_{j=1}^{\mathfrak{S}} (\prod_{i=1}^{\mathfrak{R}} \mathsf{pk}_i^{b_i^{(j)}})^{\alpha_j}$. After picking randomness $\{r_j\}_{j \in [1,\mathfrak{S}]}$, the linkable ring multi-signature becomes:

$$R = \prod_{j=1}^{\mathfrak{S}} R^{(j)} := \prod_{j=1}^{\mathfrak{S}} G^{r_j}, \ R_U = \prod_{j=1}^{\mathfrak{S}} R_U^{(j)} := \prod_{j=1}^{\mathfrak{S}} U^{r_j},$$
$$h = H(\mu, R, R_U, \mathcal{R}, \alpha_1, \ldots, \alpha_{\mathfrak{S}}, \mathsf{tag}_1, \ldots, \mathsf{tag}_{\mathfrak{S}}),$$
$$z = \sum_{j=1}^{\mathfrak{S}} r_j + h \cdot \sum_{j=1}^{\mathfrak{S}} \alpha_j \cdot \mathsf{sk}_{\mathsf{ind}_j}.$$

If the multi-signatures are generated by the same party, then the spender can simply use a single randomness $r := \sum_{j=1}^{\mathfrak{S}} r_j$ to compute $R = G^r, R_U = U^r$, and $z = r + h \cdot \sum_{j=1}^{\mathfrak{S}} \alpha_j \cdot \mathsf{sk}_{\mathsf{ind}_j}$. The spender generates a NIZK proof π_1 for the relation:

$$\{(z, \{\vec{b^{(j)}}\}_{j=1}^{\mathfrak{S}}) : \quad G^z = R \cdot \prod_{j=1}^{\mathfrak{S}} \prod_{i=1}^{\mathfrak{R}} (\mathsf{pk}_i^{b_i^{(j)}})^{\alpha_j h}, \tag{1}$$

$$\vec{b^{(j)}} \circ (\vec{1}^{\mathfrak{R}} - \vec{b^{(j)}}) = \vec{0}^{\mathfrak{R}}, \ \langle \vec{b^{(j)}}, \vec{1}^{\mathfrak{R}} \rangle = 1, \ U^z = R_U \cdot \prod_{j=1}^{\mathfrak{S}} (\mathsf{tag}_j)^{\alpha_j h}\}, \tag{2}$$

Suppose that we have $\mathfrak{T}$ output coins $\{\mathsf{co}_k^{\mathcal{T}} = V^{a_k^{\mathcal{T}}} W^{\mathsf{ck}_k^{\mathcal{T}}}\}_{k \in [1,\mathfrak{T}]}$ for amount $a_k^{\mathcal{T}}$ and coin key $\mathsf{ck}_k^{\mathcal{T}}$. Consider that the signer's input coins are $\{\mathsf{co}^{\mathcal{R}}_{\mathsf{ind}_j} = V^{a^{\mathcal{R}}_{\mathsf{ind}_j}} W^{\mathsf{ck}^{\mathcal{R}}_{\mathsf{ind}_j}}\}_{j \in [1,\mathfrak{S}]}$ for some amount $a^{\mathcal{R}}_{\mathsf{ind}_j}$ and randomness $\mathsf{ck}^{\mathcal{R}}_{\mathsf{ind}_j}$, and the transaction fee a^{fee}. By the balance property, we have $\sum_{j=1}^{\mathfrak{S}} a^{\mathcal{R}}_{\mathsf{ind}_j} - a^{\mathsf{fee}} = \sum_{k=1}^{\mathfrak{T}} a_k^{\mathcal{T}}$. Define

$\mathsf{ck}' = \sum_{j=1}^{\mathfrak{S}} \mathsf{ck}_{\mathsf{ind}_j}^{\mathcal{R}} - \sum_{k=1}^{\mathfrak{T}} \mathsf{ck}_k^{\mathcal{T}}$. We need a NIZK proof π_2 for:

$$\{(\mathsf{ck}', \{\vec{b^{(j)}}\}_{j=1}^{\mathfrak{S}}, \{a_k^{\mathcal{T}}, \mathsf{ck}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}) : V^{-a^{\mathsf{fee}}} \cdot \prod_{j=1}^{\mathfrak{S}} \prod_{i=1}^{\mathfrak{R}} (\mathsf{co}_i^{\mathcal{R}})^{b_i^{(j)}} \tag{3}$$

$$= \prod_{k=1}^{\mathfrak{T}} \mathsf{co}_k^{\mathcal{T}} \cdot W^{\mathsf{ck}'},\ \mathsf{co}_k^{\mathcal{T}} = V^{a_k^{\mathcal{T}}} W^{\mathsf{ck}_k^{\mathcal{T}}},\ 0 \le a_k^{\mathcal{T}} \le 2^\beta\ \forall k \in [1, \mathfrak{T}]\}.$$

The output is $(R, R_U, \mathsf{tag}_1, \ldots, \mathsf{tag}_{\mathfrak{S}}, \pi_1, \pi_2)$, where π_1, π_2 are the two NIZK proofs of size $O(\log \mathfrak{S}\mathfrak{R})$ and $O(\log(\mathfrak{S}\mathfrak{R} + \beta\mathfrak{T}))$, resp.

Step 3 - Combining Multiple NIZK Proofs. We use Bulletproof++ [6] to give the shortest NIZK proof for inner products. In particular, they used a different bitwise proof relation: $(b_i - \frac{1}{2})^2 = \frac{1}{4}$.

Denote $(\gamma_1, \gamma_2) = H_b(h, \mathsf{co}_1^{\mathcal{T}}, \ldots, \mathsf{co}_{\mathfrak{T}}^{\mathcal{T}})$. We define the combined NIZK proof π by using $(Eq.1 + \gamma_1 \times Eq.2 + \gamma_2 \times Eq.3)$.

$$\{(z, \mathsf{ck}', \{\vec{b^{(j)}}\}_{j=1}^{\mathfrak{S}}, \{a_k^{\mathcal{T}}, \mathsf{ck}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}) : (G \cdot U^{\gamma_1})^z \cdot (\prod_{k=1}^{\mathfrak{T}} \mathsf{co}_k^{\mathcal{T}} \cdot W^{\mathsf{ck}'})^{\gamma_2}$$

$$= (R \cdot R_U^{\gamma_1} \cdot V^{-a^{\mathsf{fee}}\gamma_2}) \cdot \prod_{j=1}^{\mathfrak{S}} \prod_{i=1}^{\mathfrak{R}} (\mathsf{pk}_i^{\alpha_j h} \mathsf{co}_i^{\mathcal{R}\gamma_2})^{b_i^{(j)}} \cdot (\prod_{j=1}^{\mathfrak{S}} (\mathsf{tag}_j)^{\alpha_j h})^{\gamma_1}$$

$$(\vec{b^{(j)}} - \frac{\vec{1}^{\mathfrak{R}}}{2})^2 = \frac{\vec{1}^{\mathfrak{R}}}{4}, \quad \langle \vec{b^{(j)}}, \vec{1}^{\mathfrak{R}} \rangle = 1, \tag{4}$$

$$\mathsf{co}_k^{\mathcal{T}} = V^{a_k^{\mathcal{T}}} W^{\mathsf{ck}_k^{\mathcal{T}}}, \quad 0 \le a_k^{\mathcal{T}} \le 2^\beta\ \forall k \in [1, \mathfrak{T}]\}.$$

To sum up, it outputs a linkable ring signature (R, R_U, π) with key images $(\mathsf{tag}_1, \ldots, \mathsf{tag}_{\mathfrak{S}})$, and the output accounts (including the output one-time address and coins), where π is the Bulletproof++ of size $O(\log(\mathfrak{S}\mathfrak{R} + \beta\mathfrak{T}))$.

High-level idea - New structure for one-time key pairs and tags. The tag $= U^{\mathsf{sk}_{\mathsf{ind}}}$ is not pseudorandom under the related key attack. Therefore, if the adversary obtains another $\mathsf{tag}' = U^{\mathsf{sk}_{\mathsf{ind}}+\delta}$ such that δ is known to the adversary, then tag and tag$'$ can be linked. This is the case where Alice sends two transactions to Bob and Alice learns δ when creating one-time public keys for Bob. When Bob spends those coins later, Alice can link those two tags using δ.

In the literature, there are two ways to generate pseudorandom tags secure under related key attack. CLSAG uses $\mathsf{tag} = H(\mathsf{pk}_{\mathsf{ind}})^{\mathsf{sk}_{\mathsf{ind}}}$. This tag requires another costly way to hide $\mathsf{pk}_{\mathsf{ind}}$ in the hash function. RingCT3.0 or Omniring uses $\mathsf{tag} = U^{1/\mathsf{sk}_{\mathsf{ind}}}$ to make the tag non-linear to $\mathsf{sk}_{\mathsf{ind}}$. However, this non-linearity is not compatible with the MuSig method to aggregate all output proofs.

We propose an alternative structure for one-time key pairs. The user picks a random degree q polynomial $f(x)$ in $\mathbb{Z}_p$. The user's long-term public key is $g^{f(x)}$, represented by group elements $F_1, \ldots, F_q$ for each coefficient in the polynomial. When the sender wants to send coins to a receiver, he generates a random R and sets the one-time public key as $g^{f(R)}$ and implies that $\mathsf{sk}_{\mathsf{ind}} = f(R)$. When the receiver wants to use the coin, he can set $\mathsf{tag} = U^{f(R)}$. If the long-term key pair is used less than q times, tag remains pseudorandom due to the randomness of the polynomial interpolation. Our approach incurs an additional limitation to the use of long-term key pairs. However, it gives us a more efficient construction. The value q can be flexible for each individual user.

6 SuccinCT

6.1 Building Blocks

In this paper, we use five building blocks: (vector) Pedersen commitment, tagging scheme, MuSig [15], Diffie-Hellman key agreement, and NIZK. The definition of other building blocks are given in Appendix C.

6.1.1 Tagging scheme. The tagging scheme is formally defined in [11]. We give the construction of a tagging scheme below.

- $\mathsf{pp}_{\mathsf{Tag}} \leftarrow \mathsf{TagSetup}(1^\lambda)$: it takes the security parameter 1^λ as input and outputs the public parameters $\mathsf{pp}_{\mathsf{Tag}}$, which include the generators $G, U \in \mathbb{G}$.
- $X \leftarrow \mathsf{TagKGen}(x)$: it takes a key $x \in \mathbb{Z}_p$ as the input, and outputs the public key $X := G^x \in \mathbb{G}$.
- $\mathsf{tag} \leftarrow \mathsf{TagEval}(x)$: it takes a key $x \in \mathbb{Z}_p$ as the input, and outputs the public key $\mathsf{tag} := U^x \in \mathbb{G}$.

If the discrete logarithm and the decisional Diffie-Hellman assumption holds, the tagging scheme satisfies one-wayness, and pseudorandomness defined in [11]. The full proof is in Appendix H.1. Furthermore, the scheme is additive homomorphic over TagKGen:

$$\mathsf{TagKGen}(x_1) \cdot \mathsf{TagKGen}(x_2) = \mathsf{TagKGen}(x_1 + x_2),$$

for all $x_1, x_2 \in \mathbb{Z}_p$. Consider a degree q polynomial $f(x) := \sum_{i=0}^q \theta_i x^i \in \mathbb{Z}_p[x]$, we have:

$$\mathsf{TagKGen}(f(x)) = \prod_{i=0}^{q} (\mathsf{TagKGen}(x^i))^{\theta_i}.$$

We do not require the related-input property for the tagging scheme itself. If the adversary only knows $x_1, \ldots, x_q \in \mathbb{Z}_p$ to generate $\mathsf{TagKGen}(f(x_1)), \ldots, \mathsf{TagKGen}(f(x_q))$, there is no related-input if f is a random polynomial in $\mathbb{Z}_p[x]$.

6.1.2 Diffie-Hellman Key Agreement. Our scheme works on the hashed Diffie-Hellman key agreement (DHKA), where one party uses a static key and the other party uses an ephemeral key. We give a definition of DHKA here using the one-round key agreement syntax [20].

- $\mathsf{pp}_{\mathsf{KA}} \leftarrow \mathsf{KASetup}(1^\lambda)$: it takes the security parameter 1^λ as input and outputs the public parameters $\mathsf{pp}_{\mathsf{KA}}$, which include the generators $G' \in \mathbb{G}$ and a hash function $H_n^\kappa : \mathbb{G} \to \mathbb{Z}_p$.
- $M_1 \leftarrow \mathsf{KAmsg}_1(a)$: it outputs a message $M_1 = G'^a$.
- $M_2 \leftarrow \mathsf{KAmsg}_2(b)$: it outputs a message $M_2 = G'^b$.
- $K \leftarrow \mathsf{KAkey}_1(a, M_2)$: it outputs a session key $K = H_n^\kappa(M_2^a)$.
- $K \leftarrow \mathsf{KAkey}_2(b, M_1)$: it outputs a session key $K = H_n^\kappa(M_1^b)$.

Correctness means that for all a, b in the randomness space,

$$\mathsf{KAkey}_2(b, \mathsf{KAmsg}_1(a)) = \mathsf{KAkey}_1(a, \mathsf{KAmsg}_2(b))$$

A KA scheme is non-malleable if $K \leftarrow \mathsf{KAkey}_1(a, M_2)$ is indistinguishable to a random key, even when given an oracle $\mathsf{KAkey}_1(a, \cdot)$. A KA scheme has pseudorandom second messages if M_2 is indistinguishable from random, even to someone who chooses M_1 adversarially.

The non-malleable property of hashed DHKA is equivalent to the oracle DH assumption. The hashed DHKA has pseudorandom second messages if the CDH assumption holds [20].

6.2 SuccinCT Construction

Setup$(1^\lambda, 1^\iota, 1^\beta)$. The setup algorithm generates $\mathsf{pp}_{\mathsf{HC}} \leftarrow \mathsf{HCSetup}(1^\lambda)$, $\mathsf{pp}_{\mathsf{Tag}} \leftarrow \mathsf{TagSetup}(1^\lambda)$, and $\mathsf{pp}_{\mathsf{KA}} \leftarrow \mathsf{KASetup}(1^\lambda)$. In $\mathsf{pp}_{\mathsf{KA}}$, denote $H_n : \{0,1\}^* \to \mathbb{Z}_p$, for any string $\kappa \in \{0,1\}^*$ and any $X \in \mathbb{G}$, define $H_n^\kappa(X) := H_n(X||\kappa)$. We further require that all schemes use the same cyclic group $\mathbb{G}$ of prime order p with a generator G. It picks a random generator $U \in \mathbb{G}$. Denote $H_a : \{0,1\}^* \to \mathbb{Z}_p$, $H_b : \{0,1\}^* \to \mathbb{Z}_p$, $H_c : \{0,1\}^* \to \mathbb{Z}_p$, $H : \{0,1\}^* \to \mathbb{Z}_p$ as secure hash functions. It runs $\mathsf{pp}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZKSetup}(1^\lambda, \mathsf{pp}_{\mathsf{HC}}, \mathsf{pp}_{\mathsf{KA}}, \mathsf{pp}_{\mathsf{Tag}}, G, U, H_n, H_a, H_b, H_c, H)$. Denote $\mathbb{B}$ as the blockchain which is initially empty. It outputs the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{HC}}, \mathsf{pp}_{\mathsf{Tag}}, \mathsf{pp}_{\mathsf{KA}}, \mathsf{pp}_{\mathsf{NIZK}}, H_n, H_a, H_b, H_c, H)$ and the blockchain $\mathbb{B}$.

SAKGen(pp, q). The Stealth Address Key Generation picks some random $\theta_0, \ldots, \theta_q \in \mathbb{Z}_p$. The long-term secret key is the polynomial $f(x) = \sum_{i=0}^{q} \theta_i x^i$ and the long-term public key is $\mathsf{lpk} = \{F_i := \mathsf{TagKGen}(\theta_i)\}_{i=0}^{q}$. It picks a random view secret key $\mathsf{vdk} \in \mathbb{Z}_p$ and computes $\mathsf{vek} \leftarrow \mathsf{KAmsg}_1(\mathsf{vdk})$. It returns master public key $\mathsf{mpk} := (\mathsf{lpk}, \mathsf{vek})$, master secret key $\mathsf{msk} := (\mathsf{lsk}, \mathsf{vdk}, q)$, and view secret key vdk.

For the miner's address, we set $q = 0$ and it has no view key. It picks a random $\mathsf{lsk} \in \mathbb{Z}_p$ and set $\mathsf{lpk} = \mathsf{TagKGen}(\mathsf{lsk})$. It returns master public key $\mathsf{mpk} := (\mathsf{lpk}, \emptyset)$, master secret key $\mathsf{msk} := (\mathsf{lsk}, \emptyset, 0)$, and view secret key $\mathsf{vdk} = \emptyset$.

Although the length of the master public key mpk is linear to the maximum number of use q, only the 32 bytes one-time public key pk will be used to generate signatures in transactions, while mpk in $O(q)$ is stored locally at the wallet. Consider the case that short addresses at Monero wallets are represented as QR codes supporting up to 3 KBs of data, setting $q \approx 90$ is appropriate since mpk contains $q + 2$ group elements $\mathbb{G}$ with 32 bytes each.

OTAccGen$(\mathsf{mpk}, a, \kappa)$. The One-Time Account Generation algorithm takes as input $\mathsf{mpk} = (\mathsf{lpk}, \mathsf{vek})$, the amount a, and an index κ. If $\kappa \geq 1$, it generates a random ephemeral key $\mathsf{tdk} \in \mathbb{Z}_p$ and computes $\mathsf{tek} \leftarrow \mathsf{KAmsg}_2(\mathsf{tdk})$, $s \leftarrow \mathsf{KAkey}_2(\mathsf{tdk}, \mathsf{vek})$. A one-time public key pk is derived as $\mathsf{pk} := \mathsf{TagKGen}(f(s))$ for some polynomial f using $(F_0, \ldots, F_q) \in \mathsf{lpk}$. The output amount is encoded as $\mathsf{ctxt}_a = a \oplus_8 H_n(\text{“}amount\text{”}, s)$, where $\oplus_8$ is an XOR operation between the first $\beta/8$ bytes of each operand. It computes $\mathsf{ck} = H_n(\text{“}commitment_mask\text{”}, s)$ and a coin $\mathsf{co} := \mathsf{Com}(a; \mathsf{ck})$. It outputs the account $\mathsf{acc} = (\mathsf{pk}, \mathsf{co}, \mathsf{ctxt}_a, \kappa)$, the coin key ck, and the view information $\mathsf{info} := \mathsf{tek}$.

If $\kappa = 0$ (i.e., the miner's account), it defines $\mathsf{pk} = \mathsf{lpk}$. It outputs the account $\mathsf{acc} = (\mathsf{pk}, \mathsf{co} := \mathsf{Com}(a; 1), a, 0)$, the coin key $\mathsf{ck} := 1$, and the view information $\mathsf{info} := \emptyset$.

View$(\mathsf{vdk}, \mathsf{acc}, \mathsf{info})$. The view algorithm checks if vdk corresponds to the account $\mathsf{acc} = (\mathsf{pk}, \mathsf{co}, \mathsf{ctxt}_a, \kappa)$ by using $\mathsf{info} = \mathsf{tek}$. It computes $s \leftarrow \mathsf{KAkey}_1(\mathsf{vdk}, \mathsf{tek})$. If $\mathsf{pk} \neq \mathsf{TagKGen}(f(s))$, it returns $\perp$. Else, it computes $a = \mathsf{ctxt}_a \oplus_8 H_n(\text{“}amount\text{”}, s)$ and $\mathsf{ck} = H_n(\text{“}commitment_mask\text{”}, s)$. If $\mathsf{co} \neq \mathsf{Com}(a; \mathsf{ck})$, it returns $\perp$. Otherwise, it returns a.

Receive$(\mathsf{msk}, \mathsf{acc}, \mathsf{info})$. The receiving algorithm checks if $\mathsf{msk} = (\mathsf{lsk}, \mathsf{vdk}, q)$ is the owner of the account $\mathsf{acc} = (\mathsf{pk}, \mathsf{co}, \mathsf{ctxt}_a, \kappa)$ by using a viewing information info. It first runs $a \leftarrow$**View**$(\mathsf{vdk}, \mathsf{acc}, \mathsf{info})$ and gets ck, s during the computation. If $a = \perp$ or it has received from q different accounts before, return $\perp$. Otherwise, the algorithm derives the one-time secret key of the account as $\mathsf{sk} = f(s)$. It also generates the tag of the account as $\mathsf{tag} := \mathsf{TagEval}(\mathsf{sk})$. It outputs $(\mathsf{ck}, a, \mathsf{sk}, \mathsf{tag})$.

If $\mathsf{vdk} = \emptyset$ (i.e., the miner's account), $\mathsf{acc} = (\mathsf{mpk} := (\mathsf{lpk}, \emptyset), \mathsf{co} := \mathsf{Com}(a; 1), a, 0)$, and $\mathsf{lpk} = \mathsf{TagKGen}(\mathsf{lsk})$, it outputs $(\mathsf{ck} := 1, a, \mathsf{sk} := \mathsf{lsk}, \mathsf{tag} := \mathsf{TagEval}(\mathsf{sk}))$.

Spend$(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\mathsf{fee}}, \mu)$. The spending algorithm takes as input a set of input accounts $\mathcal{R} = \{\mathsf{acc}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}$, a set of spender information $\mathcal{S} = \{\mathsf{ind}_j, \mathsf{ck}_j^{\mathcal{S}}, a_j^{\mathcal{S}}, \mathsf{sk}_j, \mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}$, a set of output information $\mathcal{T} = \{\mathsf{ck}_k^{\mathcal{T}}, a_k^{\mathcal{T}}, \mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}$ (created from **OTAccGen**$(\cdot, a_k^{\mathcal{T}}, k)$), a transaction fee a^{fee} and a message μ.

(1) Ring Multi-signature. It computes $\alpha_j = H_a(\mathcal{R}, \{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}, j)$ for $j \in [1, \mathfrak{S}]$ and $\beta = H_b(\{\alpha_j\}_{j=1}^{\mathfrak{S}})$. It picks a random $r \in \mathbb{Z}_p$ and computes:

$$R = (G \cdot U^\beta)^r,\ h = H(\mu, R, \beta),\ z = r + h(\sum_{j=1}^{\mathfrak{S}} \alpha_j \mathsf{sk}_j).$$

(2) Coin Balance. Note that by the balance requirement, we have $\sum_{j=1}^{\mathfrak{S}} a_j^{\mathcal{S}} - a^{\mathsf{fee}} = \sum_{k=1}^{\mathfrak{T}} a_k^{\mathcal{T}}$. It computes $\mathsf{ck}' = \sum_{j=1}^{\mathfrak{S}} \mathsf{ck}_j^{\mathcal{S}} - \sum_{k=1}^{\mathfrak{T}} \mathsf{ck}_k^{\mathcal{T}}$.

(3) NIZK Proof. Pick a randomness $t \in \mathbb{Z}_p$ and compute the vector commitment $T = \mathsf{Com}(z, \mathsf{ck}', \{\vec{b^{(j)}}\}_{j=1}^{\mathfrak{S}}; t)$. Compute $\gamma = H_c(h, T, \mathsf{acc}_1^{\mathcal{T}}, \ldots, \mathsf{acc}_{\mathfrak{T}}^{\mathcal{T}}, a^{\mathsf{fee}})$. Construct the binary vectors $\vec{b}^{(j)} = (b_1^{(j)}, b_2^{(j)}, \ldots, b_{\mathfrak{R}}^{(j)})$, where $b_{\mathsf{ind}_j}^{(j)} = 1$ for $j \in [1, \mathfrak{S}]$ and all other bits are equal to 0. Denote $\mathsf{co}_k^{\mathcal{T}}$ as the coin for the output account $\mathsf{acc}_k^{\mathcal{T}}$. Denote $\mathsf{co}_i^{\mathcal{R}}$ and $\mathsf{pk}_i^{\mathcal{R}}$ as the coin and public key for the input account $\mathsf{acc}_i^{\mathcal{R}}$. It computes the NIZK proof $\pi \leftarrow \mathsf{Prove}(\mathsf{stmt}, \mathsf{wit})$ for the following witness wit and statement stmt:

$$\{(z, t, \mathsf{ck}', \{\vec{b^{(j)}}\}_{j=1}^{\mathfrak{S}}, \{\mathsf{ck}_k^{\mathcal{T}}, a_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}) :\ (G \cdot U^\beta)^z \cdot (\prod_{k=1}^{\mathfrak{T}} \mathsf{co}_k^{\mathcal{T}} \cdot \mathsf{Com}(0; \mathsf{ck}'))^\gamma$$
$$= (R \cdot V^{-a^{\mathsf{fee}}\gamma}) \cdot \prod_{j=1}^{\mathfrak{S}} \prod_{i=1}^{\mathfrak{R}} [(\mathsf{pk}_i^{\mathcal{R}})^{\alpha_j h} (\mathsf{co}_i^{\mathcal{R}})^\gamma]^{b_i^{(j)}} \cdot (\prod_{j=1}^{\mathfrak{S}} (\mathsf{tag}_j)^{\alpha_j h})^\beta, \tag{5}$$
$$T = \mathsf{Com}(z, \mathsf{ck}', \{\vec{b^{(j)}}\}_{j=1}^{\mathfrak{S}}; t),\ (\vec{b^{(j)}} - \frac{\vec{1}}{2}^{\mathcal{R}})^2 = \frac{\vec{1}}{4}^{\mathcal{R}},\ \langle \vec{b^{(j)}}, \vec{1}^{\mathcal{R}} \rangle = 1,$$
$$\forall j \in [1, \mathfrak{S}],\quad \mathsf{co}_k^{\mathcal{T}} = \mathsf{Com}(a_k^{\mathcal{T}}; \mathsf{ck}_k^{\mathcal{T}}),\ 0 \leq a_k^{\mathcal{T}} \leq 2^\beta,\ \forall k \in [1, \mathfrak{T}].\}$$

It outputs the proof $\sigma = (R, T, \pi)$ for the transaction tx with input accounts $\{\mathsf{acc}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}$, key images $\{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}$, output account $\{\mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}$, transaction fee a^{fee}, and a message μ. By using the same tdk in **OTAccGen** for all output accounts, all $\mathsf{info}_k^{\mathcal{T}}$ are the same and we can simply store one copy of it in tx.

Verify(tx, σ). The verification algorithm takes as input a transaction tx, a proof σ and the blockchain $\mathbb{B}$. Denote $\mathsf{tx} = (\{\check{\mathsf{acc}}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}},$

$\{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}, \{\mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}, a^{\mathsf{fee}}, \mu)$ and $\sigma = (R, T, \pi)$. It retrieves the accounts $\{\mathsf{acc}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}$ from the blockchain $\mathbb{B}$ using the reference pointers $\{\tilde{\mathsf{acc}}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}$.

It computes $\alpha_j = H_a(\{\mathsf{acc}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}, \{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}, j)$ for $j \in [1, \mathfrak{S}]$, $\beta = H_b(\{\alpha_j\}_{j=1}^{\mathfrak{S}}), h = H(\mu, R, \beta), \gamma = H_c(h, T, \mathsf{acc}_1^{\mathcal{T}}, \ldots, \mathsf{acc}_{\mathfrak{T}}^{\mathcal{T}}, a^{\mathsf{fee}})$ to form the statement stmt. It returns $\perp$ if (1) not all tag_j are distinct; or (2) any tag_j appears in some transaction in $\mathbb{B}$. It returns 0 if $\mathbf{Verify}(\mathsf{stmt}, \pi) = 0$. Otherwise, it returns 1.

$\mathbf{Mine}(\{\mathsf{tx}, \sigma\}, \mathsf{mpk}_M, a_M)$. The mining algorithm takes as input a set of transaction-signature pairs $\{\mathsf{tx}, \sigma\}$ such that $\mathbf{Verify}(\mathsf{tx}, \sigma) = 1$, the miner's public key mpk_M and the block reward a_M. It creates a block of transactions B. Calculate the total amount a^* as the sum of a_M and all transaction fees in B. Define the miner's account as $\mathsf{acc}_M := (\mathsf{mpk}_M, \mathsf{co} := \mathsf{Com}(a^*; 1), a^*, 0)$. It returns $(\mathsf{B}, \mathsf{acc}_M)$.

Remark. We use Bulletproof++ as the NIZK proof, which contains the commitment to all witnesses inside the protocol. Therefore, the vector commitment T can be replaced directly by the commitment in the NIZK proof.

6.3 Security Proofs

Theorem 6.1. *SuccinCT has balance, privacy, and non-slanderability if the discrete logarithm assumption, the DDH assumption, and the oracle DH assumption hold in the random oracle model.*

The security proofs of balance, privacy, and non-slanderability are given in Appendix H.3 due to the space limit.

7 Evaluation

Since existing related schemes do not provide implementations and are based on Bulletproof rather than the more advanced Bulletproof++ used in our work, we implemented SuccinCT independently in Go. While the implementation of other related work is not available, we evaluate SuccinCT independently and argue that it is expected to be practically efficient, as its Bulletproof(++) component involves fewer witnesses than those of RingCT 3.0, Omniring, and ZGSX23.

Our implementation utilizes the cloudflare/bn256 library elliptic curve operations and a suite of supporting utility functions. In addition, we integrate the Fiat-Shamir transformation [5] with the Keccak hash function to convert interactive zero-knowledge proofs into non-interactive ones. These utilities enable homomorphic commitments, tagging algorithms, key agreement schemes, and non-interactive zero-knowledge proofs. The modular design of our implementation supports extensibility and optimization, paving the way for future enhancements or adaptations.

We run our experiments on a Mac mini (Macmini9,1) equipped with an Apple M1 chip (8 cores and 8 GB of memory). SuccinCT was evaluated across various ring sizes—128, 256, 512, and 1024—with each configuration tested over 100 runs to measure spending and verification times. All computations were restricted to a single thread to ensure consistency and controlled conditions, isolating the performance of SuccinCT's cryptographic algorithms from hardware-specific optimizations or multithreading effects.

Execution Time Analysis. The execution times for generating (spending) and verifying transactions are summarized in Table 3. Spending time increases modestly with ring size, starting at 2.005 ms (±0.324 ms) for a ring size of 128 and rising to 3.562 ms (±0.159 ms) for 1024. Verification time shows a similar trend, growing from 1.246 ms (±0.221 ms) for 128 to 2.607 ms (±0.170 ms) for 1024. These results indicate that the execution times for both spending and verification scale with the ring size, while remaining within low milliseconds for all configurations.

Table 3: Running time for various ring sizes. Mean and standard deviation are shown as mean ± std for 100 runs.

Ring Size	Spending Time (ms)	Verification Time (ms)
128	2.00509 ± 0.323716	1.24612 ± 0.221242
256	2.18062 ± 0.097861	1.39342 ± 0.123428
512	2.64914 ± 0.128827	1.80817 ± 0.206267
1024	3.56213 ± 0.158974	2.60732 ± 0.169522

Factors affecting the transaction size. Factors affecting the transaction size of SuccinCT and other CT protocols are given in Appendix I. Interestingly, SuccinCT is relatively balanced between the key image size, proof size and output size. Comparatively, CLSAG and RingCT3.0 have a larger portion on the proof size, Omniring has a larger portion on the output size, and ZGSX23 has a larger portion on the key image size.

Transaction size of different CT protocols. We compare the overall transaction size of CT protocols (CLSAG, RingCT3.0, Omniring, ZGSX23, BulletCT, and SuccinCT) in Figure 7 for a classical two-input two-output transaction. The transaction size of CLSAG is linear to $\mathfrak{R}$. Although the proof size of the other five schemes is logarithmic to $\mathfrak{R}$, the overall transaction size will be dominated by the input size $\mathfrak{R}|\mathcal{I}|$ for larger ring. Therefore, it shows a linear trend when $\mathfrak{R}$ is more than 4096. ZGSX23 is in the middle because of the $\mathfrak{R}$ link tags (compared to the $\mathfrak{S} = 2$ link tags in other schemes). SuccinCT has the smallest overall transaction size, and it is more significant for $\mathfrak{R} < 256$ because the impact of proof size prevails the impact of input size when $\mathfrak{R}$ is relatively small. The details of computation complexity are given in Appendix I.4.

Anonymity Space vs. Privacy. Although Figure 2 showed that ZGSX23 [31] performed better than other previous schemes based on the same anonymity space, this comparison may not be accurate. We argue that a larger anonymity space does not imply a higher level of privacy. Besides, BulletCT [26] also mentioned several limitations of the any-out-of-many proofs for RingCT schemes. The details are given in Appendix D.

8 Conclusion

In this paper, we identify miner privacy as a previously overlooked security gap in CTs and propose general mitigation mechanisms for existing schemes. We further propose SuccinCT, a DualRing-based CT protocol that provides native support for miner privacy. Our analysis shows that SuccinCT achieves the smallest overall transaction size among the compared schemes, and our implementation demonstrates low-millisecond spending and verification times for practical ring sizes.

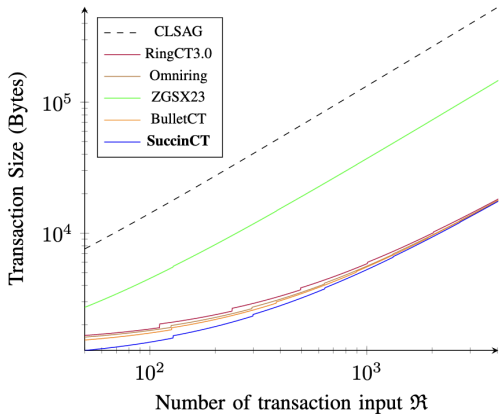

Figure 7: The transaction size of CT protocols for a two-input, two-output transaction with different ring size $\mathfrak{R}$.

Acknowledgments

We reported our findings to the Monero Project Vulnerability Response Process (ID: 2864144) in November 2024. Additionally, we engaged with the Monero team to discuss potential solutions for addressing these issues in their next release. The Monero team has already taken action in response to this attack.

We thank the Monero team for helpful discussions on attacks against different types of miners and on FCMP.

References

[1] Abdalla, M., Bellare, M., Rogaway, P.: The oracle diffie-hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. Lecture Notes in Computer Science, vol. 2020, pp. 143–158. Springer (2001)
[2] Alupotha, J., Boyen, X., McKague, M.: Aggregable confidential transactions for efficient quantum-safe cryptocurrencies. IEEE Access **10**, 17722–17747 (2022)
[3] Alupotha, J., Boyen, X., McKague, M.: LACT+: practical post-quantum scalable confidential transactions. Cryptogr. 7(2), 24 (2023)
[4] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: SP 2018. pp. 315–334. IEEE Computer Society (2018)
[5] Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D., Wichs, D.: Fiat-shamir: from practice to theory. In: STOC 2019. pp. 1082–1090. ACM (2019)
[6] Eagen, L., Kanjalkar, S., Ruffing, T., Nick, J.: Bulletproofs++: Next generation confidential transactions via reciprocal set membership arguments. In: EUROCRYPT 2024. vol. 14655, pp. 249–279. Springer (2024)
[7] Esgin, M.F., Zhao, R.K., Steinfeld, R., Liu, J.K., Liu, D.: Matrict: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In: CCS 2019. pp. 567–584. ACM (2019)
[8] Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Efficient set membership proofs using mpc-in-the-head. Proc. Priv. Enhancing Technol. (2), 304–324 (2022)
[9] Goodell, B., Noether, S., Blue, A.: Concise linkable ring signatures and forgery against adversarial keys. IACR Cryptol. ePrint Arch. p. 654 (2019)
[10] koe, Alonso, K.M., Noether, S.: Zero to monero: Second edition: A technical guide to a private digital currency; for beginners, amateurs, and experts (2020)
[11] Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: Scaling private payments without trusted setup. In: CCS 2019. pp. 31–48. ACM (2019)
[12] Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. Lecture Notes in Computer Science, vol. 3108, pp. 325–335. Springer (2004)
[13] Lyubashevsky, V., Nguyen, N.K., Seiler, G.: SMILE: set membership from ideal lattices with applications to ring signatures and confidential transactions. In: CRYPTO 2021. LNCS, vol. 12826, pp. 611–640. Springer (2021)
[14] Maxwell, G.: Confidential transactions (2015)
[15] Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Des. Codes Cryptogr. **87**(9), 2139–2164 (2019)
[16] Noether, S., Mackenzie, A.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
[17] Park, S., Sealfon, A.: It wasn't me! - repudiability and claimability of ring signatures. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. Lecture Notes in Computer Science, vol. 11694, pp. 159–190. Springer (2019)
[18] Parker, L.K.: Fcmp++ (2025)
[19] Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer (2001)
[20] Rosulek, M., Trieu, N.: Compact and malicious private set intersection for small sets. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS 2021. pp. 1166–1181. ACM (2021)
[21] Rucknium: Fully specified estimation plan for optimal static parametric estimation of arbitrary distributions (ospead) (2022), https://github.com/Rucknium/OSPEAD/blob/main/OSPEAD-Fully-Specified-Estimation-Plan-PUBLIC.pdf
[22] van Saberhagen, N.: Cryptonote v 2.0 (2013)
[23] Sun, S., Au, M.H., Liu, J.K., Yuen, T.H.: Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In: ESORICS 2017. LNCS, vol. 10493, pp. 456–474. Springer (2017)
[24] Torres, W.A.A., Kuchta, V., Steinfeld, R., Sakzad, A., Liu, J.K., Cheng, J.: Lattice ringct V2.0 with multiple input and multiple output wallets. In: ACISP 2019. LNCS, vol. 11547, pp. 156–175. Springer (2019)
[25] Torres, W.A.A., Steinfeld, R., Sakzad, A., Liu, J.K., Kuchta, V., Bhattacharjee, N., Au, M.H., Cheng, J.: Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice ringct v1.0). In: ACISP 2018. LNCS, vol. 10946, pp. 558–576. Springer (2018)
[26] Wang, N., Wang, Q., Liu, D., Esgin, M.F., Abuadbba, A.: BulletCT: Towards more scalable ring confidential transactions with transparent setup. USENIX Security 2025 (2025)
[27] Yu, J., Au, M.H.A., Veríssimo, P.J.E.: Re-thinking untraceability in the cryptonote-style blockchain. In: 32nd IEEE Computer Security Foundations Symposium (CSF). pp. 94–107 (2019)
[28] Yu, Z., Au, M.H., Yu, J., Yang, R., Xu, Q., Lau, W.F.: New empirical traceability analysis of cryptonote-style blockchains. In: Financial Cryptography and Data Security (FC). pp. 133–149 (2019)
[29] Yuen, T.H., Esgin, M.F., Liu, J.K., Au, M.H., Ding, Z.: Dualring: Generic construction of ring signatures with efficient instantiations. In: CRYPTO 2021. vol. 12825, pp. 251–281. Springer (2021)
[30] Yuen, T.H., Sun, S., Liu, J.K., Au, M.H., Esgin, M.F., Zhang, Q., Gu, D.: Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. In: FC 2020,. LNCS, vol. 12059, pp. 464–483. Springer (2020)
[31] Zheng, T., Gao, S., Song, Y., Xiao, B.: Leaking arbitrarily many secrets: Any-out-of-many proofs and applications to ringct protocols. In: SP 2023. pp. 2533–2550. IEEE (2023)

A Related works

A.1 (Linkable) Ring signature.

Ring signature schemes [19] allow the signer to anonymously sign a message on behalf of a spontaneously chosen group (i.e., *ring*). The set of non-signers is sometimes called decoy or mixin. They are widely used in privacy-preserving protocols such as e-voting, whistleblowing and privacy-preserving cryptocurrencies.

Linkable ring signature [12] is a ring signature with a *key image* (or linkability tag), so that if the same signer signed two different linkable ring signatures, their key images are the same. Hence, these two signatures are *linked*. Linkable ring signatures was mentioned in CryptoNote v2.0 [22] to ensure anonymity in cryptocurrencies while still preventing double-spending.

A.2 DualRing

DualRing [29] is a novel generic ring signature construction containing two rings: a ring of commitments and a ring of challenges. DualRing consists of $\mathfrak{R}$ challenges and a single response in the signature, where $\mathfrak{R}$ is the number of parties in the ring. In the elliptic curve-based instantiation DualRing-EC, the challenge ring is formed by a modular addition operation, so the proof of knowledge of $\mathfrak{R}$ challenges can be efficiently compressed into logarithmic

size of $\mathfrak{R}$ by using a Bulletproof-styled proof. DualRing-EC is the state-of-the-art ring signature scheme in terms of signature size.

We observe that in DualRing-EC [29], the signature $(c_1, \ldots, c_n, z)$ for a message μ has the relations:

$$G^z = R \cdot \prod_{i=1}^{\mathfrak{R}} \mathsf{pk}_i^{c_i}, \quad c_1 + \ldots + c_{\mathfrak{R}} = H(\mu, R, \{\mathsf{pk}_i\}),$$

where the real signer index is $\mathsf{ind} \in [1, \mathfrak{R}]$, $R = G^r \prod_{i \neq \mathsf{ind}} \mathsf{pk}_i^{-c_i}$ and $z = r + c_{\mathsf{ind}} \cdot \mathsf{sk}_{\mathsf{ind}}$. In the security proof, the hash function H is rewinded and the simulator can obtain δ_z, δ_{c_i} for $i \in [1, \mathfrak{R}]$ such that there is at least one $\delta_{c_i} \neq 0$ and:

$$G^{\delta_z} = \prod_{i=1}^{\mathfrak{R}} \mathsf{pk}_i^{\delta_{c_i}}.$$

It means that the simulator can only extract δ_z which is a linear combination of secret keys $\sum_{i=1}^{\mathfrak{R}} \mathsf{sk}_i \cdot \delta_{c_i}$. Since the extracted δ_z cannot be binded to one single $\mathsf{sk}_{\mathsf{ind}}$, it cannot be used to form a linkable ring signature directly, since the key image cannot be binded to one single $\mathsf{sk}_{\mathsf{ind}}$.

In order to solve the single binding issue, we have to add an additional constraint to $c_1, \ldots, c_{\mathfrak{R}}$:

$$c_{i*} = H(\mu, R, \{\mathsf{pk}_i\}), \quad c_i = 0 \ \forall i \neq \mathsf{ind}.$$

Hence, we have a special case of DualRing-EC, so that

$$G^z = R \cdot \prod_{i=1}^{\mathfrak{R}} \mathsf{pk}_i^{c_i}, \quad \sum_{i=1}^{\mathfrak{R}} c_i = H(\mu, R, \{\mathsf{pk}_i\}), \quad c_i = 0 \ \forall i \neq \mathsf{ind},$$

where $R = G^r$ and $z = r + c_{\mathsf{ind}} \cdot \mathsf{sk}_{\mathsf{ind}}$.

B Anonymity Reduction Attack for Hidden $\mathfrak{S}$

In ZGSX23 [31], the number of transaction input $\mathfrak{S}$ is hidden and only the ring size $\mathfrak{R}$ is known. The anonymity space is $2^{\mathfrak{R}}$. Suppose that there are $\mathfrak{M}$ miners' accounts in the ring and the sum of some output amounts $a^{\mathcal{T}}$ is known to the adversary Eve. If there is any combination of the sum of the miner accounts' amount less than $a^{\mathcal{T}} + a^{\mathsf{fee}}$, then Eve can remove this combination from the anonymity space. If $a^{\mathcal{T}} + a^{\mathsf{fee}}$ is larger than the sum of the amounts from all $\mathfrak{M}$ miners' accounts, the anonymity space is reduced to $2^{\mathfrak{R}-\mathfrak{M}}$.

Our approach defends against the anonymity reduction attack by setting $\mathfrak{S}$ with respect to the number of miner accounts in the input ring. However, it does not work in ZGSX23 since $\mathfrak{S}$ is hidden to protect the anonymity space. To the best of our knowledge, there is no simple or general solution for ZGSX23. One trick is to limit the total transaction output amount to be less than any block reward, but it is not a practical solution. We leave this problem as future work.

C Building Blocks

C.1 (Vector) Pedersen commitment.

We first give the construction of the Pedersen commitment below.

- $\mathsf{pp}_{\mathsf{HC}} \leftarrow \mathsf{HCSetup}(1^\lambda)$: it takes the security parameter 1^λ as input and outputs the public parameters $\mathsf{pp}_{\mathsf{HC}}$, which include the generators $V, W \in \mathbb{G}$.
- $C \leftarrow \mathsf{Com}(a; r)$: it takes a message $a \in \mathbb{Z}_p$ and a randomness $r \in \mathbb{Z}_p$ as inputs, and outputs the commitment $C := V^a W^r \in \mathbb{G}$.

The Pedersen commitment is perfectly hiding, and is binding if the discrete logarithm assumption holds. The Pedersen commitment is additive homomorphic:

$$\mathsf{Com}(a_1; r_1) \cdot \mathsf{Com}(a_2; r_2) = \mathsf{Com}(a_1 + a_2; r_1 + r_2),$$

for all $a_1, a_2, r_1, r_2 \in \mathbb{Z}_p$. In this paper, we also use the vector Pedersen commitment which commits a vector of elements in $\mathbb{Z}_p$ using a vector of generators. The scheme is very similar and we omit the details.

C.2 MuSig.

We give the (simplified) Schnorr-based multi-signature scheme MuSig [15] below. No interaction is needed for the signing protocol with the modification that all secret keys are owned by a single party.

- $\mathsf{pp}_{\mathsf{MS}} \leftarrow \mathsf{MSSetup}(1^\lambda)$: it takes the security parameter 1^λ as input and outputs the public parameters $\mathsf{pp}_{\mathsf{MS}}$, which include the generators $G \in \mathbb{G}$ and two hash functions $H_a : \mathbb{G}^* \to \mathbb{Z}_p$ and $H : \mathbb{G}^2 \times \{0,1\}^* \to \mathbb{Z}_p$.
- $X \leftarrow \mathsf{MSKGen}(x)$: it takes a key $x \in \mathbb{Z}_p$ as the input, and outputs the public key $X := G^x \in \mathbb{G}$.
- $\sigma \leftarrow \mathsf{MSSign}((x_1, X_1), \ldots, (x_n, X_n), m)$: it takes n key pairs (x_i, X_i) for $i \in [1, n]$ and a message $m \in \{0,1\}^*$ as input. Let $\mathbb{L} := \{X_1, \ldots, X_n\}$ and $a_i := H_a(\mathbb{L}, X_i)$. It picks a random $r \in \mathbb{Z}_p$ and outputs the signature $\sigma := (R, z)$, where:

$$R = G^r, \quad c = H(\prod_{i=1}^{n} X_i^{a_i}, R, m), \quad z = r + c \sum_{i=1}^{n} a_i x_i.$$

- $0/1 \leftarrow \mathsf{MSVerify}(\mathbb{L}, m, \sigma)$: it takes n public keys $\mathbb{L} := (X_1, \ldots, X_n)$, a message m, and signature $\sigma = (R, z)$ as inputs. It computes $a_i = H_a(\mathbb{L}, X_i)$ and $c = H(\prod_{i=1}^{n} X_i^{a_i}, R, m)$. It outputs 1 if $G^z = R \cdot (\prod_{i=1}^{n} X_i^{a_i})^c$, or outputs 0 otherwise.

The Schnorr-based multi-signature scheme satisfies unforgeability if the discrete logarithm assumption holds in the random oracle model [15].

C.3 NIZK

Our scheme works on any NIZK proof for inner product relation. We give a definition of NIZK here.

- $\mathsf{pp}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZKSetup}(1^\lambda, \mathsf{pp}_{\mathsf{HC}}, \mathsf{pp}_{\mathsf{Tag}}, \mathsf{pp}_{\mathsf{MS}})$: it takes the security parameter 1^λ and the public parameters $\mathsf{pp}_{\mathsf{HC}}, \mathsf{pp}_{\mathsf{Tag}}, \mathsf{pp}_{\mathsf{MS}}$ as input and outputs the public parameters $\mathsf{pp}_{\mathsf{NIZK}}$.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{stmt}, \mathsf{wit})$: it takes a statement stmt and a witness wit as input, and outputs a proof π.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{stmt}, \pi)$: it takes statement stmt and a proof π as input, and outputs 1 if π is valid, or 0 otherwise.

The NIZK proof Bulletproof++ [6] satisfies completeness, special honest verifier zero-knowledge and witness-extended emulation if the discrete logarithm assumption holds in the random oracles.

D Anonymity Space vs. Privacy

Number of possible signers. ZGSX23 [31] suggested that their anonymity level is better than Omniring [11]. However, it is only true under the assumption that all possible combinations are equally likely. However in reality, it is much more likely to have a very small $\mathfrak{S}$. In Bitcoin, the number of one-input transaction is around 70-90% in 2023-2024. Hence, it is highly unlikely that Monero will have a lot of transactions with 5+ inputs. In addition, a subset of possible combinations within the anonymity space may be eliminated based on the analysis of decoy input selection of existing transactions [27, 28]. It is difficult to compare confidential transaction protocols based on the same anonymity space.

Deniability of linkable ring signature. All confidential transaction protocols detect double spending by the use of key images, which is a one-way function on the sender's one-time secret key. If two signed transactions have the same key image, then double spending is detected. On the other hand, a non-sender in the ring can generate a zero-knowledge proof of *deniability* [17], so that *none* of the key image is a one-way function on his secret key. It can be viewed as an *anonymity reduction* attack by the ring insider. For Omniring, the anonymity space is reduced to $\binom{\mathfrak{R}-1}{\mathfrak{S}}$ (i.e., a drop of $\frac{\mathfrak{S}}{\mathfrak{R}}$). For ZGSX23, the anonymity space is reduced to $2^{\mathfrak{R}-1}$ (i.e., a drop of $\frac{1}{2}$). Hence ZGSX23 is more affected by this attack if $\mathfrak{R} > 2\mathfrak{S}$.

Tag proliferation and accumulator incompatibility. The tags of ZGSX23 signature are linear to the ring size $\mathfrak{R}$, which significantly inflate the overall transaction size and deteriorate the scalability concerns. In addition, ZGSX23 proposed the use of accumulators to compress the key images, relying on cryptographic assumptions different from DL. For example, efficient RSA-based accumulators require trusted setups, which may expose trapdoor secrets, heighten cryptographic complexities, and compromise the security of permissionless blockchain systems.

E Data Formats in Monero

We review some data formats and data structure in Monero [10]. In Monero, it represents both elements in $\mathbb{G}$ and $\mathbb{Z}_p$ as 32 bytes. In the following, we consider a transaction with a set size $\mathfrak{R}$, source input size $\mathfrak{S}$ and target output size $\mathfrak{T}$.

Input offsets. It includes the *indices* of $\mathfrak{R}$ previous transaction outputs. Since bigger indices require more storage space, Monero just uses the 'absolute' position of the first index from each ring, and the 'relative' positions of the other ring members' indices. For example, with real indices $\{7, 11, 15, 20\}$ we just need to report $\{7, 4, 4, 5\}$. Verifiers can compute the last index with $7+4+4+5 = 20$. In Monero, the input offsets are stored as variable length integers. In practice, there are around 25 million transactions in Monero as of 2024. Hence, each offset can be represented by 4 bytes (representing $>$ 4 billion transactions). Hence, the total size for input offsets is $\leq 4\mathfrak{R}$ bytes.

Monero performs some sanity checks in regards to how the key offsets are chosen. For example, the median of all key offsets must be within the last 40% of all outputs. Over time, Monero has increased the default ring size $\mathfrak{R}$. Monero enforces a ring size of 11 (10 decoys plus the real transaction) in 2018, and it is further increased to 16 in 2022.

Input key images. Each input key image is represented as a group element. In Monero, the input key images size is $32\mathfrak{R}$ bytes.

One-time output addresses. For each target recipient with a public spend key X_s and a public view key X_v, the one-time output address is computed as $X' = G^{H_n(X_v^{8r}||n)} \cdot X_s$[9], where H_n is the hash function Keccak256; r is the randomness; n is the output index; and G is the generator of Monero's elliptic curve. Therefore, the total size of all one-time output addresses is $32\mathfrak{T}$ bytes.

In order to spend it, the recipient computes the corresponding one-time spend key $x' = H_n(R^{8x_v}||n) + x_s$, where $R = G^r$; x_v is the secret view key; and x_s is the secret spend key. Since r is generated by the sender of the transaction, the value of R is stored in the *extra* field of the transaction.

Since fork version 12, Monero transactions must have more than one output. Transactions that do not comply will be rejected by the network. Most transactions naturally have two outputs, the second one being the change output (the output amount can be zero). Enforcing at least two outputs for all transactions makes it more likely for transactions to be indistinguishable.

Encoded output amounts. The amount commitment is of the form $a \ \mathrm{XOR}_8 \ H_n(\text{“}amount\text{”}, H_n(R^{8x_v}||n))$, where a is the output amount; XOR_8 is an XOR operation between the first 8 bytes of each operand; and $H_n(R^{8x_v}||n)$ is the same secret key used to derive the above one-time address (and the following Pedersen blinding factor) for the output. Hence, the total size of encoded output amounts is $8\mathfrak{T}$ bytes.

Anyone with knowledge of the secret view key x_v can therefore compute $H_n(\text{“}amount\text{”}, H_n(R^{8x_v}||n))$ by extracting the transaction public key R from the extra field and using the output index n, and then compute a by using XOR_8.

Output commitments. Blinding factors in Pedersen commitments are not generated randomly. Instead, they are generated in a similar way to how the one-time addresses are derived:

$$r = H_n(\text{“}commitment_mask\text{”}||H_n(R^{8x_v}||n)).$$

For each output amount a, the corresponding output commitment is $\mathrm{co} = V^a W^r$. Hence, the total size of output commitments is $32\mathfrak{T}$ bytes.

Anyone with knowledge of the secret view key x_v can therefore compute $r = H_n(\text{“}commitment_mask\text{”}||H_n(R^{8x_v}||n))$ by extracting the transaction public key R from the extra field and using the output index n, and then validate co using a and r.

Extra. Like the name suggests, this field holds extra information for the transaction. It is usually occupied by the output R values discussed above. More frequently, there is a single common R value for all transaction outputs, referred to as the transaction public key, which is 32 bytes.

Payment ID. 8 bytes for an integrated address. There should be no more than one per tx.

Transaction fee. Stored as a variable length integer, so ≤ 9 bytes.

Unlock time. Stored as a variable length integer, so ≤ 9 bytes.

[9] It is mentioned in [10] that 8 is the cofactor of the twisted Edward curve used, to ensure the resulting point is in G's subgroup.

Signature. For RCTTypeBulletproof2 we need $(2\mathfrak{R}+1)\cdot\mathfrak{S}\cdot 32$ bytes of storage, and the aggregate Bulletproof range proof needs $(2\cdot\log_2(64\cdot\mathfrak{T})+9)\cdot 32$ bytes of storage.

F Mixin Selection in Monero

Monero's decoy selection algorithm has evolved to prevent analysis techniques that might reveal the true sender in a transaction. Earlier selection methods (like uniform random selection) had weaknesses that could expose the real input through statistical analysis. The algorithm currently uses a Gamma distribution to select mixin addresses, which better reflects the natural spending patterns of users on the network. It was enforced in the v13 upgrade in October 2018.

F.1 Mixin Distribution

Monero uses a Gamma distribution to model the time that typically elapses between when an output is created and when it's spent [10]. This means that decoy addresses are chosen with a tendency toward more recent outputs, reflecting realistic user spending behavior. The Gamma distribution parameters are calibrated to fit empirical spending data, where recent outputs are more likely to be reused while older outputs are less likely.

When selecting decoys, Monero's algorithm starts by selecting a real output to spend from the user's wallet. Then, based on the Gamma distribution, it samples from the entire blockchain to pick decoy outputs. It ensures that decoy outputs include both recent and older outputs, making the selection pattern more natural.

A new distribution called Optimal Static Parametric Estimation of Arbitrary Distributions (OSPEAD) for the Monero decoy selection algorithm [21] will be implemented in future update.

F.2 Miner's Account

In Monero, newly mined outputs have a maturity period (~60 blocks) before they can be spent. During this period, they are not eligible to be selected as inputs or decoys in any transaction. This rule helps prevent certain types of analysis and adds security to the network. After this maturity period, miner outputs are treated like any other output on the blockchain and can be selected as decoys in other transactions. This is possible because, once matured, these outputs are indistinguishable from other transaction outputs.

Miner outputs (especially recent ones) can appear as decoys. However, because mining rewards are relatively infrequent compared to regular transactions, they may appear less often than other transaction outputs in the decoy pool.

G Definitions of Cryptographic Primitives

Discrete Logarithm (DL) Assumption. Given (G, Y) where $G, Y \in \mathbb{G}$, no PPT adversary can output $a = \log_G Y$ with non-negligible probability. *For all PPT adversaries $\mathcal{A}$ such that*

$$\Pr[Y = G^a \mid a \leftarrow \mathcal{A}(\mathbb{G}, G, Y)] \leq \mathsf{negl}(\lambda).$$

Computational Diffie-Hellman (CDH) Assumption. Given (G, G^a, G^b) where $G \in \mathbb{G}$; $a, b \in \mathbb{Z}_p$, no PPT adversary can output G^{ab} with non-negligible probability. *For all PPT adversaries $\mathcal{A}$ such that*

$$\Pr[Y = G^{ab} \mid Y \leftarrow \mathcal{A}(\mathbb{G}, G, G^a, G^b)] \leq \mathsf{negl}(\lambda).$$

Decisional Diffie-Hellman (DDH) Assumption. Given (G, G^a, G^b, G^c) where $G \in \mathbb{G}$; $a, b, c \in \mathbb{Z}_p$, no PPT adversary can decide if $c = ab$ with non-negligible probability. *For all PPT adversaries $\mathcal{A}$ such that*

$$\left|\Pr[1 \leftarrow \mathcal{A}(\mathbb{G}, G, G^a, G^b, G^c)] - \Pr[1 \leftarrow \mathcal{A}(\mathbb{G}, G, G^a, G^b, G^{ab})]\right| \leq \mathsf{negl}(\lambda).$$

Oracle Diffie-Hellman (ODH) Assumption [1]. Given (G, G^a, G^b, G^c) where $G \in \mathbb{G}$; $a, b, c \in \mathbb{Z}_p$ and a hash function $H : \mathbb{G} \to \{0,1\}^\ell$, no PPT adversary can decide if $c = ab$ with non-negligible probability. *For all PPT adversaries $\mathcal{A}$ such that*

$$\Big|\Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_a}(\mathbb{G}, G, G^a, G^b, H(G^c))] - \Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_a}(\mathbb{G}, G, G^a, G^b, H(G^{ab}))]\Big| \leq \mathsf{negl}(\lambda),$$

where the oracle $\mathcal{O}_a(X)$ outputs $H(X^a)$ for $X \neq G^b$.

Zero-Knowledge Proof. A prover can convince a verifier that a statement holds without revealing any information about a witness.

An argument consists of three PPT algorithms $(\mathcal{K}, \mathcal{P}, \mathcal{V})$, which are CRS (Common Reference String) generator $\mathcal{K}$, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. A CRS τ is produced by $\mathcal{K}$ on input 1^λ and a transcript tr is produced by $\mathcal{P}$ and $\mathcal{V}$ on inputs s and t, which is denoted by $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t)\rangle$. We write $\langle \mathcal{P}(s), \mathcal{V}(t)\rangle = b$ to denote that the verifier $\mathcal{V}$ accepts $b = 1$ or rejects $b = 0$. We define the language:

$$\mathcal{L} = \{u \mid \exists w : (\tau, u, w) \in \mathcal{R}\},$$

where w is a witness and u is a set of statements u in the relation $\mathcal{R}$. An argument of knowledge $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ should satisfy perfect completeness and statistical witness-extended emulation.

Perfect completeness. A prover with a witness w for $u \in \mathcal{L}$ can convince the verifier of this fact. *For any non-uniform polynomial time adversary $\mathcal{A}$, $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ has perfect completeness if*

$$\Pr[(\tau, u, w) \notin \mathcal{R} \text{ or } \langle \mathcal{P}(\tau, u, w), \mathcal{V}(\tau, u)\rangle = 1 \mid \tau \leftarrow \mathcal{K}(1^\lambda), (u, w) \leftarrow \mathcal{A}(\tau)] = 1.$$

Statistical witness-extended emulation. Given an adversary that produces an acceptable argument with probability ϵ, there exists an emulator that produces a similar argument with probability ϵ together with a witness w. *For any deterministic polynomial time prover $\mathcal{P}^*$, $(S, \mathcal{P}, \mathcal{V})$ has witness-extended emulation if there exists a polynomial time emulator $\mathcal{E}$ such that for any pair of interactive adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ such that*

$$\begin{aligned}&\Pr[\mathcal{A}_1(tr) = 1 \mid \tau \leftarrow \mathcal{K}(1^\lambda), (u, s) \leftarrow \mathcal{A}_2(\tau), \\ &\qquad tr \leftarrow \langle \mathcal{P}^*(\tau, u, s), \mathcal{V}(\tau, u)\rangle] \\ &\approx \Pr[\mathcal{A}_1(tr) = 1 \wedge (tr \text{ is accepting} \Rightarrow (\tau, u, w) \in \mathcal{R}) \mid \\ &\qquad \tau \leftarrow \mathcal{K}(1^\lambda), (u, s) \leftarrow \mathcal{A}_2(\tau), (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(\tau, u)],\end{aligned}$$

where the oracle $\mathcal{O} = \langle \mathcal{P}^*(\tau, u, s), \mathcal{V}(\tau, u)\rangle$ can rewind to some point and resume with new randomness for the verifier $\mathcal{V}$ from this point onward.

$\mathsf{OneWay}_{\mathcal{A}}(1^\lambda)$

$\mathsf{pp} \leftarrow \mathbf{TagSetup}(1^\lambda)$
$x \leftarrow \chi, X \leftarrow \mathsf{TagKGen}(x)$
$\mathsf{tag} \leftarrow \mathsf{TagEval}(x)$
$x' \leftarrow \mathcal{A}(\mathsf{pp}, X, \mathsf{tag})$
return $(\mathsf{tag} = \mathsf{TagEval}(x'))$.

$\mathsf{PR}^b_{\mathcal{A}}(1^\lambda)$

$\mathsf{pp} \leftarrow \mathbf{TagSetup}(1^\lambda)$
$x \leftarrow \chi, X \leftarrow \mathsf{TagKGen}(x)$
if $b = 0$, $\mathsf{tag} \leftarrow \mathsf{TagEval}(x)$
else $\mathsf{tag} \leftarrow \phi$
$b' \leftarrow \mathcal{A}(\mathsf{pp}, X, \mathsf{tag})$
return b'.

Figure 8: One-wayness and pseudorandomness experiments for tagging schemes.

Public Coin. *An argument of knowledge $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is called public coin if all messages sent from the verifier to the prover are chosen uniformly at random and independently of the prover's messages, i.e., the challenges correspond to the verifier's randomness ρ.*

Perfect Special Honest-Verifier Zero-Knowledge (SHVZK). *A public coin argument of knowledge $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is a perfect SHVZK argument of knowledge for $\mathcal{R}$ if there exists a PPT simulator $\mathcal{S}$ such that for all pairs of interactive adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$*

$$\Pr[(\tau, u, w) \in \mathcal{R} \wedge \mathcal{A}_1(tr) = 1 \mid \tau \leftarrow \mathcal{K}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\tau), tr \leftarrow \langle \mathcal{P}(\tau, u, w), \mathcal{V}(\tau, u; \rho)\rangle]$$
$$= \Pr[(\tau, u, w) \in \mathcal{R} \wedge \mathcal{A}_1(tr) = 1 \mid \tau \leftarrow \mathcal{K}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\tau), tr \leftarrow \mathcal{S}(u, \rho)].$$

There are efficient standard techniques to convert an SHVZK argument into a public-coin full zero-knowledge argument (with a cheating verifier) when a common reference string is available.

H Security Proofs

H.1 Models and Proofs of Tagging Scheme

In this paper, we use the one-wayness and pseudorandomness property of the tagging scheme, without restricting it to related-input security. Denote the key space as χ and the tag space as ϕ. We give the definition below.

Definition H.1. A tagging scheme is said to be one-way if for any PPT adversary $\mathcal{A}$, $\Pr[\mathsf{OneWay}_{\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$. It is pseudorandom if for any PPT adversary $\mathcal{A}$, $|\Pr[\mathsf{PR}^0_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{PR}^1_{\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$, where $\mathsf{OneWay}_{\mathcal{A}}$ and $\mathsf{PR}^b_{\mathcal{A}}$ are defined in Figure 8.

Theorem H.2. *Our tagging scheme in Section 6.1.1 has one-waynesss and pseudorandomness if the DL assumption and the DDH assumption holds.*

Proof. One-wayness. Suppose that the simulator $\mathcal{S}$ is given the DL instance (G, X). $\mathcal{S}$ picks some random $v \in \mathbb{Z}_p$ and sets $U = G^v$, $\mathsf{pp} = (G, U)$. $\mathcal{S}$ sends $(\mathsf{pp}, X, \mathsf{tag} = X^v)$ to the adversary $\mathcal{A}$. If $\mathcal{A}$ breaks the one-waynesss, he returns x' such that $\mathsf{tag} = U^{x'}$. It implies that $X^v = U^{x'} = G^{vx'}$. $\mathcal{S}$ returns x' as the solution to the DL problem.

Pseudorandomness. Suppose that the simulator $\mathcal{S}$ is given the DDH instance (G, G^a, G^b, G^c). $\mathcal{S}$ sets $U = G^a$, $\mathsf{pp} = (G, U)$. $\mathcal{S}$ sends $(\mathsf{pp}, X = G^b, \mathsf{tag} = G^c)$ to the adversary $\mathcal{A}$. If $c = ab$, then $\mathcal{S}$ simulates the $\mathsf{PR}^0_{\mathcal{A}}$ game. If $c \neq ab$, then $\mathcal{S}$ simulates the $\mathsf{PR}^1_{\mathcal{A}}$ game. If $\mathcal{A}$ breaks the pseudorandomness, then $\mathcal{S}$ solves the DDH problem. □

H.2 Security Model of CT

Balance. Balance roughly means that a spender cannot double-spend, or spend more than what it possesses.

Definition H.3. A confidential transaction is balanced if:

(1) **CheckTag** and **CheckAmount** are binding. For any PPT adversary $\mathcal{A}$, for all positive integers $\iota, \beta \in \mathsf{poly}(\lambda)$, we have

$$\Pr\left[\begin{array}{c|c} \mathbf{CheckAmount}(\mathsf{acc}, \mathsf{ck}, a) = 1, & \\ \mathbf{CheckAmount}(\mathsf{acc}, \mathsf{ck}', a') = 1, & \\ (\mathsf{ck}, a) \neq (\mathsf{ck}', a') & \\ (\mathsf{pp}, \mathbb{B}) \in \mathbf{Setup}(1^\lambda, 1^\iota, 1^\beta) & \\ (\mathsf{acc}, \mathsf{ck}, a, \mathsf{ck}', a') \leftarrow \mathcal{A}(\mathsf{pp}, \mathbb{B}) & \end{array}\right] \leq \mathsf{negl}(\lambda),$$

$$\Pr\left[\begin{array}{c|c} \mathbf{CheckTag}(\mathsf{acc}, \mathsf{sk}, \mathsf{tag}) = 1, & \\ \mathbf{CheckTag}(\mathsf{acc}, \mathsf{sk}', \mathsf{tag}') = 1, & \\ (\mathsf{sk}, \mathsf{tag}) \neq (\mathsf{sk}', \mathsf{tag}') & \\ (\mathsf{pp}, \mathbb{B}) \in \mathbf{Setup}(1^\lambda, 1^\iota, 1^\beta) & \\ (\mathsf{acc}, \mathsf{sk}, \mathsf{tag}, \mathsf{sk}', \mathsf{tag}') \leftarrow \mathcal{A}(\mathsf{pp}, \mathbb{B}) & \end{array}\right] \leq \mathsf{negl}(\lambda),$$

(2) For all PPT adversaries $\mathcal{A}$, and all positive integers $\iota, \beta \in \mathsf{poly}(\lambda)$, there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that $\Pr[\mathsf{Balance}_{\mathcal{A}, \mathcal{E}_{\mathcal{A}}}(1^\lambda, 1^\iota, 1^\beta) = 1] \leq \mathsf{negl}(\lambda)$, where $\mathsf{Balance}_{\mathcal{A}, \mathcal{E}_{\mathcal{A}}}(1^\lambda, 1^\iota, 1^\beta)$ is defined in Figure 9.

Non-Slanderability. Slandering is an act of producing a valid proof on behalf of another user. Non-slanderability is a property that prevents the above attack, which is known as denial-of-spending attack in the literature. Formally, we model non-slanderability by the adversary producing a transaction-proof tuple, after several queries to the oracles (includes getting all view keys). The adversary is successful if the tuple is valid and not produced by the spend oracle, and some of the tags specified in the slandering transaction collide with those that are signed by the spend oracle.

H.3 SuccinCT's Proof

Before describing the security proofs of SuccinCT, we define the two extra functions from our security definitions.

- $0/1 \leftarrow \mathbf{CheckAmount}(\mathsf{acc}, \mathsf{ck}, a)$. The amount checking algorithm takes as inputs an account $\mathsf{acc} = (\mathsf{pk}, \mathsf{co}, \mathsf{ctxt}_a, \kappa)$, a coin key ck, and an amount a. It outputs 1 if $\mathsf{co} = \mathsf{Com}(a; \mathsf{ck})$, or 0 otherwise.

$\mathsf{Balance}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda, 1^\iota, 1^\beta)$

$(\mathsf{pp}, \mathbb{B}) \leftarrow \mathbf{Setup}(1^\lambda, 1^\iota, 1^\beta)$
$(\mathsf{tx}, \sigma) \leftarrow \mathcal{A}(\mathsf{pp}, \mathbb{B})$
$(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\mathsf{fee}}, \mu) \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{pp}, \mathbb{B}, \mathsf{tx}, \sigma)$,
where $\mathcal{R} = \{\mathsf{acc}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}} \in \mathbb{B}$,
$\mathcal{S} = \{\mathsf{ind}_j, \mathsf{ck}_j^{\mathcal{S}}, a_j^{\mathcal{S}}, \mathsf{sk}_j, \mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}$,
$\mathcal{T} = \{\mathsf{ck}_k^{\mathcal{T}}, a_k^{\mathcal{T}}, \mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}$,
$b_0 := \mathbf{Verify}(\mathsf{tx}, \sigma)$,
$b_1 := (\mathsf{tx} = \mathsf{TX}(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\mathsf{fee}}, \mu))$,
$b_2 := \left(\forall j \in [1, \mathfrak{S}], \mathbf{CheckTag}(\mathsf{acc}_{\mathsf{ind}_j}^{\mathcal{R}}, \mathsf{sk}_j, \mathsf{tag}_j) = 1\right)$,
$b_3 := \left(\forall j \in [1, \mathfrak{S}], \mathbf{CheckAmount}(\mathsf{acc}_{\mathsf{ind}_j}^{\mathcal{R}}, \mathsf{ck}_j^{\mathcal{S}}, a_j^{\mathcal{S}}) = 1\right)$,
$b_4 := \left(\forall k \in [1, \mathfrak{T}], \mathbf{CheckAmount}(\mathsf{acc}_k^{\mathcal{T}}, \mathsf{ck}_k^{\mathcal{T}}, a_k^{\mathcal{T}}) = 1\right)$,
$b_5 := \left(\sum_{j=1}^{\mathfrak{S}} a_j^{\mathcal{S}} - a_{\mathsf{fee}} = \sum_{k=1}^{\mathfrak{T}} a_k^{\mathcal{T}}\right)$,
return $b_0 \wedge \neg(b_1 \wedge b_2 \wedge b_3 \wedge b_4 \wedge b_5)$.

Figure 9: Balance.

$\mathsf{NonSlander}_{\mathcal{A}}(1^\lambda, 1^\iota, 1^\beta)$

$(\mathsf{pp}, \mathbb{B}) \leftarrow \mathbf{Setup}(1^\lambda, 1^\iota, 1^\beta)$, InitOracles()
$(\mathsf{tx}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{SAKGenO},\mathsf{SpendO},\mathsf{GetViewKey}}(\mathsf{pp}, \mathbb{B})$
parse tx^* as $(\{\tilde{\mathsf{acc}}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}, \{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}, \mathcal{T} = \{\mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}$, $a^{\mathsf{fee}}, \mu)$,
$b_0 := \mathbf{Verify}(\mathsf{tx}^*, \sigma^*)$,
$b_1 := (\mathsf{tx}^*, \sigma^*) \notin \Sigma$,
$b_2 := \{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}} \cap \mathsf{Spent} \neq \emptyset$,
return $b_0 \wedge b_1 \wedge b_2$.

Figure 10: Non-slanderability.

- $0/1 \leftarrow \mathbf{CheckTag}(\mathsf{acc}, \mathsf{sk}, \mathsf{tag})$. The tag checking algorithm takes as inputs an account $\mathsf{acc} = (\mathsf{pk}, \mathsf{co}, \mathsf{ctxt}_a, \kappa)$, a secret key sk, and a tag tag. It outputs 1 if $\mathsf{pk} = \mathsf{TagKGen}(\mathsf{sk})$ and $\mathsf{tag} = \mathsf{TagEval}(\mathsf{sk})$, or 0 otherwise.

Theorem H.4. *SuccinCT is balance if the discrete logarithm assumption holds in the random oracle model, the NIZK proof has witness-extended emulation and the vector Pedersen commitment is binding.*

Proof. Suppose that the adversary $\mathcal{A}$ outputs a transaction $\mathsf{tx} = (\{\tilde{\mathsf{acc}}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}, \{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}, \{\mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}, a^{\mathsf{fee}}, \mu)$ and a signature $\sigma = (R, R_U, \pi)$. If $\mathcal{A}$ wins the game, it means that $\mathbf{Verify}(\mathsf{tx}, \sigma) = 1$.

We construct an extractor $\mathcal{E}_{\mathcal{A}}$ below. It retrieves the accounts $\mathcal{R} := \{\mathsf{acc}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}$ from the blockchain $\mathbb{B}$ using the reference pointers $\{\tilde{\mathsf{acc}}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}$.

By the witness-extended emulation of the NIZK proof π, $\vec{b^{(j)}}$ is a binary vector of Hamming weight 1 for all $j \in [1, \mathfrak{S}]$. Denote the position of the 1 bit as ind_j. In addition, $\mathsf{co}_k^{\mathcal{T}}$ is the commitment of amount $a_k^{\mathcal{T}}$ with a valid range proof using the coin key $\mathsf{ck}_k^{\mathcal{T}}$ for all $k \in [1, \mathfrak{T}]$. It implies that $b_4 = 1$. $\mathcal{E}_{\mathcal{A}}$ can reconstruct $\mathcal{T} = \{\mathsf{ck}_k^{\mathcal{T}}, a_k^{\mathcal{T}}, \mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}$.

$\mathcal{E}_{\mathcal{A}}$ retrieves previous transactions $\mathsf{acc}_{\mathsf{ind}_j}^{\mathcal{R}}$ from the blockchain $\mathbb{B}$. By the witness-extended emulation of the NIZK proofs in the previous transactions, it can also extract the amounts $a_j^{\mathcal{S}}$ (with a valid range proof) and the coin key $\mathsf{ck}_j^{\mathcal{S}}$ for all $j \in [1, \mathfrak{S}]$. It implies that $b_3 = 1$.

$\mathcal{E}_{\mathcal{A}}$ rewinds H_c to set two sets of random challenges γ_1, γ_2 and obtains two sets of responses satisfying Eq. 5. By the binding properties of the commitment $T = \mathsf{Com}(z, \mathsf{ck}', \{\vec{b^{(j)}}\}_{j\in[1,\mathfrak{S}]}; t)$, these two sets of responses share the same $z, \mathsf{ck}', \{\vec{b^{(j)}}\}_{j\in[1,\mathfrak{S}]}, t$. Denote the verification of these two sets of responses as Eq. 5.1 and 5.2. It computes ι_1, ι_2 such that: $\iota_1 + \iota_2 = 0$, and $\iota_1\gamma_1 + \iota_2\gamma_2 = 1$. Then, the simulator can compute Eq.5.1 $\times\iota_1$+ Eq.5.2 $\times\iota_2$ and obtain:

$$\prod_{k=1}^{\mathfrak{T}} \mathsf{co}_k^{\mathcal{T}} \cdot \mathsf{Com}(0; \mathsf{ck}') = V^{-a^{\mathsf{fee}}} \cdot \prod_{j=1}^{\mathfrak{S}} \mathsf{co}_{\mathsf{ind}_j}^{\mathcal{R}}.$$

Since $\mathsf{co}_i^{\mathcal{R}}$ is the output coin of a previous transaction in $\mathbb{B}$, $\mathsf{co}_i^{\mathcal{R}}$ is also a commitment of amount with a valid range proof. Hence this equation becomes:

$$\prod_{k=1}^{\mathcal{T}} V^{a_k^{\mathcal{T}}} W^{\mathsf{ck}_k^{\mathcal{T}}} \cdot W^{\mathsf{ck}'} = V^{-a^{\mathsf{fee}}} \cdot \prod_{j=1}^{\mathfrak{S}} V^{a_j^{\mathcal{S}}} W^{\mathsf{ck}_j^{\mathcal{S}}}.$$

If the discrete logarithm assumption holds for (V, W), it implies that $b_5 = 1$.

$\mathcal{E}_{\mathcal{A}}$ computes ω_1, ω_2 such that: $\omega_1 + \omega_2 = 1$, and $\omega_1\gamma_1 + \omega_2\gamma_2 = 0$. Then, $\mathcal{E}_{\mathcal{A}}$ can compute Eq.5.1 $\times\omega_1$+ Eq.5.2 $\times\omega_2$ and obtain:

$$(G \cdot U^\beta)^z = R \cdot \prod_{j=1}^{\mathfrak{S}} (\mathsf{pk}_{\mathsf{ind}_j}^{\mathcal{R}} \cdot \mathsf{tag}_j^\beta)^{\alpha_j h}.$$

Next, $\mathcal{E}_{\mathcal{A}}$ rewinds H to set two sets of random challenges h_1, h_2 and obtains two sets of responses z_1, z_2 satisfying the above equation. $\mathcal{E}_{\mathcal{A}}$ computes φ_1, φ_2 such that: $\varphi_1 + \varphi_2 = 0$, and $\varphi_1 h_1 + \varphi_2 h_2 = 1$. $\mathcal{E}_{\mathcal{A}}$ obtains $z' = \varphi_1 z_1 + \varphi_2 z_2$ such that:

$$(G \cdot U^\beta)^{z'} = \prod_{j=1}^{\mathfrak{S}} (\mathsf{pk}_{\mathsf{ind}_j}^{\mathcal{R}} \cdot \mathsf{tag}_j^\beta)^{\alpha_j}. \tag{6}$$

Next rewind H_b to obtain two sets of random challenges β_1, β_2 and obtains two different extracted z_1', z_2'.

$$G^{z_1'-z_2'} \cdot U^{\beta_1 z_1' - \beta_2 z_2'} = \prod_{j=1}^{\mathfrak{S}} (\mathsf{tag}_j^{\beta_1-\beta_2})^{\alpha_j}.$$

Denote $z^* = \frac{z_1'-z_2'}{\beta_1-\beta_2}$ and $x^* = \frac{\beta_1 z_1' - \beta_2 z_2'}{\beta_1-\beta_2}$. By putting $\prod_{j=1}^{\mathfrak{S}} \mathsf{tag}_j^{\alpha_j} = G^{z^*} U^{x^*}$ back to Eq.6, we have:

$$(G \cdot U^\beta)^{z'} = \prod_{j=1}^{\mathfrak{S}} (\mathsf{pk}_{\mathsf{ind}_j}^{\mathcal{R}})^{\alpha_j} \cdot (G^{z^*} U^{x^*})^\beta. \tag{7}$$

Denote that $\prod_{j=1}^{\mathfrak{S}} (\mathsf{pk}_{\mathsf{ind}_j}^{\mathcal{R}})^{\alpha_j} = G^{\varphi_1} U^{\varphi_2}$ for some $\varphi_1, \varphi_2 \in \mathbb{Z}_p$. By the discrete logarithm assumption between G and U, we can see from Eq. 7 that:

$$z' = \varphi_1 + z^*\beta, \quad z'\beta = \varphi_2 + x^*\beta.$$

It implies that $z^*\beta^2 + (\varphi_1 - x^*)\beta - \varphi_2 = 0$. Since this holds for all β, it implies that $z^* = 0$, $\varphi_2 = 0$ and $\varphi_1 = x^*$. Hence:

$$\prod_{j=1}^{\mathfrak{S}} \mathsf{tag}_j^{\alpha_j} = U^{x^*}, \quad \prod_{j=1}^{\mathfrak{S}} (\mathsf{pk}_{\mathsf{ind}_j}^{\mathcal{R}})^{\alpha_j} = G^{x^*}.$$

Denote $\mathsf{sk}_j = \log_G \mathsf{pk}_{\mathsf{ind}_j}^{\mathcal{R}}$. Hence, we have $x^* = \sum_{j=1}^{\mathfrak{S}} \mathsf{sk}_j \alpha_j$. Next, $\mathcal{E}_{\mathcal{A}}$ rewinds H_a to have $\mathfrak{S}$ sets of random challenges $(\alpha_{1,l}, \ldots \alpha_{\mathfrak{S},l})$ for $l \in [1, \mathfrak{S}]$ and obtains $\mathfrak{S}$ different $x_1^*, \ldots, x_{\mathfrak{S}}^*$ satisfying:

$$x_l^* = \sum_{j=1}^{\mathfrak{S}} \mathsf{sk}_j \alpha_{j,l},$$

for $l \in [1, \mathfrak{S}]$. By these $\mathfrak{S}$ equations with $\mathfrak{S}$ unknowns sk_l, all sk_l can be computed by $\mathcal{E}_{\mathcal{A}}$ with probability $1 - \frac{1}{p}$ for randomly chosen $\alpha_{j,l}$. Then we have

$$\prod_{j=1}^{\mathfrak{S}} \mathsf{tag}_j^{\alpha_j} = U^{x^*} = U^{\sum_{j=1}^{\mathfrak{S}} \mathsf{sk}_j \alpha_j}.$$

Hence we obtain $\mathsf{tag}_j = U^{\mathsf{sk}_j}$. It implies that $b_2 = 1$.

$\mathcal{E}_{\mathcal{A}}$ can reconstruct $\mathcal{S} = \{\mathsf{ind}_j, \mathsf{ck}_j^{\mathcal{S}}, a_j^{\mathcal{S}}, \mathsf{sk}_j, \mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}$. Hence $\mathsf{tx} = \mathsf{TX}(\mathcal{R}, \mathcal{S}, \mathcal{T}, a^{\mathsf{fee}}, \mu)$ and $b_1 = 1$

We have $b_1 = b_2 = b_3 = b_4 = b_5 = 1$, which contradicts that $\mathcal{A}$ wins with non-negligible probability. The balance property is achieved. □

Theorem H.5. *SuccinCT has privacy if the DDH assumption holds in the random oracle model, the NIZK proof has SHVZK, the vector Pedersen commitment is hiding, the Diffie-Hellman key agreement is non-malleable and has pseudorandom second message, and the tagging scheme has pseudorandomness.*

Proof. We construct an algorithm $\mathcal{B}$ to run the privacy game b with the adversary $\mathcal{A}$. Suppose that $\mathcal{B}$ honestly computes $(\mathsf{tx}_b, \sigma_b)$. Denote $\mathsf{tx}_b = (\{\tilde{\mathsf{acc}}_i^{\mathcal{R}}\}_{i=1}^{\mathfrak{R}}, \{\mathsf{tag}_j\}_{j=1}^{\mathfrak{S}}, \{\mathsf{acc}_k^{\mathcal{T}}, \mathsf{info}_k^{\mathcal{T}}\}_{k=1}^{\mathfrak{T}}, a^{\mathsf{fee}}, \mu)$ and $\sigma = (R, T, \pi)$.

We first consider the signature part $\sigma = (R, T, \pi)$. If the NIZK proof π has SHVZK, π can be replaced by a simulated proof first. By the hiding property of the vector Pedersen commitment, then T can be replaced by a random element in $\mathbb{G}$. R does not contain any information about the bit b.

Next, we consider the transaction part tx_b. By the pseudorandomness of the tagging scheme, we can replace $\{\mathsf{tag}_j\}_{j=s_1,\ldots,s_{|I|}}$ with random elements in $\mathbb{G}$. By the non-malleability of the Diffie-Hellman key agreement, the session key s_k for output account $k = d_1, \ldots, d_{|J|}$ can be replaced by a random element in $\mathbb{Z}_p$. All queries to ViewO can be handled by the KAkey_1 oracle of the non-malleability game. Hence, the output account public key $\mathsf{pk}_k^{\mathcal{T}} = \mathsf{TagKGen}(f_k(s))$ in $\mathsf{acc}_k^{\mathcal{T}}$ is also pseudorandom if $\mathsf{TagKGen}(f_k(\cdot)))$ is called by TryReceive for less than or equal to q times. By the hiding property of the Pedersen commitment, the output coin $\mathsf{co}_k^{\mathcal{T}}$ in $\mathsf{acc}_k^{\mathcal{T}}$ is also pseudorandom. By the pseudorandom second message property of the Diffie-Hellman key agreement, $infoView_k^{\mathcal{T}}$ can be replaced by a random element in $\mathbb{G}$.

Hence, the adversary has a negligible probability of winning the privacy game. □

Theorem H.6. *SuccinCT has non-slanderability if the discrete logarithm assumption holds in the random oracle model, the NIZK proof has witness-extended emulation and the vector Pedersen commitment is binding.*

Proof. Suppose that the simulator is given the discrete logarithm instance (G, Y) and try to output $y = \log_G Y$. It picks a random $\zeta \in \mathbb{Z}_p$ and sets $U = G^{\zeta}$.

For one of the $\mathsf{SAKGenO}(q)$ queries, it picks $\theta_i \in \mathbb{Z}_p$ and sets $F_i = Y^{\theta_i}$ for all $i \in [1, q]$. The rest of the keys are honestly generated. Denote the long-term secret key as $f^*(x) = \sum_{i=0}^{q} \theta_i y x^i$, which is unknown to the simulator.

If the SpendO oracle requires the use of the long-term secret key $f^*(s)$ for some session key s, then the simulator sets $\mathsf{tag}^* = U^{f^*(s)} = U^{\sum_{i=0}^{q} \theta_i y s^i} = Y^{\sum_{i=0}^{q} \theta_i \zeta s^i}$. The NIZK proof in π can be simulated without the knowledge of $f^*(s)$.

If the adversary $\mathcal{A}$ wins by outputting $(\mathsf{tx}^*, \sigma^*)$, the simulator can extract sk_j and ind_j such that $\mathbf{CheckTag}(\mathsf{acc}_{\mathsf{ind}_j}^{\mathcal{R}}, \mathsf{sk}_j, \mathsf{tag}_j) = 1$ (similar to the balance proof). It implies that $\mathsf{tag}_j = U^{\mathsf{sk}_j}$ for $j \in [1, \mathfrak{S}]$. If $b_2 = 1$, it means that at least one of the tag_j was created from the SpendO oracle. It aborts if tag^* is not equal to such tag_j. If $b_1 = 1$, it means that $(\mathsf{tx}^*, \sigma^*)$ is not the output of from the SpendO oracle. By the witness-extended emulation of the NIZK proof π, the simulator can extract $\mathsf{sk}^* = \log_U \mathsf{tag}^*$. It means that:

$$U^{\mathsf{sk}^*} = U^{\sum_{i=0}^{q} \theta_i y s^i}.$$

Hence, the simulator can output $y = \frac{\mathsf{sk}^*}{\sum_{i=0}^{q} \theta_i s^i}$ as the solution to the discrete logarithm problem. If the discrete logarithm assumption holds, then no PPT adversary win the non-slanderability with non-negligible probability. □

I Empirical Analysis of CT Protocols

We analyse CT protocols by comparing the transaction size with different ring size $\mathfrak{R}$ under two different settings:

(1) The number of input $\mathfrak{S}$ and output $\mathfrak{T}$ are fixed to 2.
(2) $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$ as in [31].

Although we do not agree with the analysis of anonymity space as stated in Appendix D, we still keep the setting of $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$ for comparison. It is because it can give us insight for transactions with larger input and output size.

I.1 SuccinCT

Figure 12 shows the transaction size of SuccinCT for a classical two-input two-output transaction. It shows that the proof size is the dominating factor for $\mathfrak{R} < 256$. For $\mathfrak{R} > 512$, the input size is the dominating factor. Therefore, it is not practical to use $\mathfrak{R} > 512$ unless there is a more compact way to represent the input of the transaction (discussed in Appendix J.2).

Figure 11 shows the transaction size of SuccinCT when $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$ (the setting in [31]). It shows that the key image size and the output size also grows linearly with $\mathfrak{S}$ and $\mathfrak{T}$ respectively. In this case, the proof size is the dominating factor only when $\mathfrak{R} < 128$.

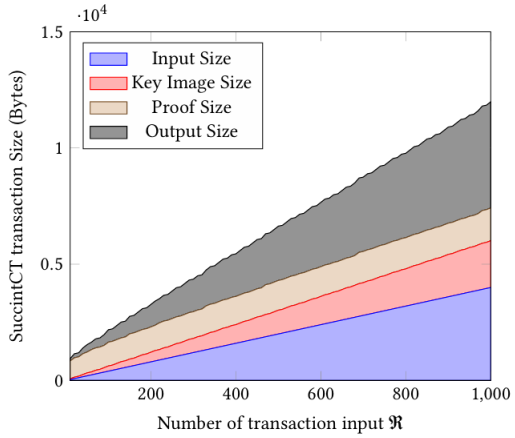

Figure 11: Breakdown of SuccinCT transaction size for different $\mathfrak{R}$, where $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$.

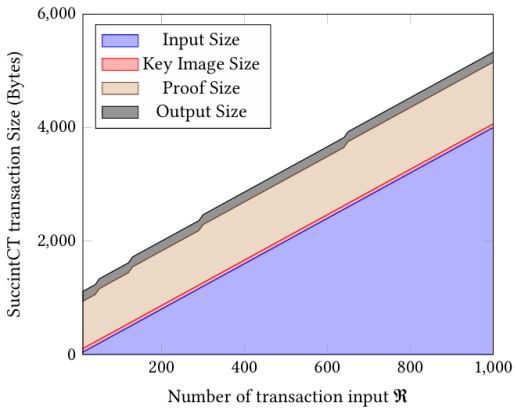

Figure 12: Breakdown of SuccinCT transaction size for different $\mathfrak{R}$, where $\mathfrak{S}$ and $\mathfrak{T}$ are fixed to 2.

I.2 Other CTs

CLSAG. As shown in Figure 13 and 14, the proof size is the dominating factor for CLSAG.

RingCT3.0. As shown in Figure 15, the proof size is the dominating factor for RingCT 3.0. If the input and output sizes are fixed to 2, Figure 16 shows that the proof size is the dominating factor for $\mathfrak{R} < 400$, and the input size is the dominating factor for a larger $\mathfrak{R}$.

Omniring. As shown in Figure 17, the output size is the dominating factor for Omniring. If the input and output sizes are fixed to 2, Figure 18 shows that the proof size and the output size are the dominating factor for $\mathfrak{R} < 256$, and the input size is the dominating factor for a larger $\mathfrak{R}$.

ZGSX23. As shown in Figure 19 and 20, the key image size is the dominating factor for ZGSX23.

BulletCT. As shown in Figure 21 and 22, the transaction size of BulletCT is almost identical with our SuccinCT, except that the proof size of BulletCT is about 20% larger than SuccinCT.

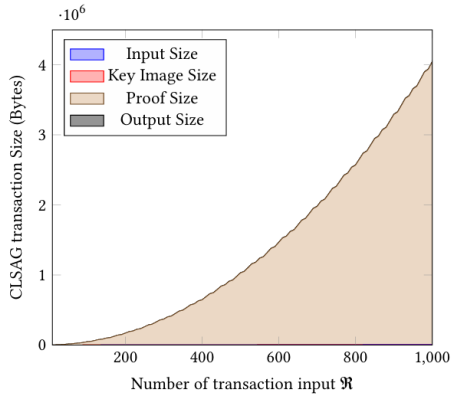

Figure 13: Breakdown of CLSAG transaction size for different $\mathfrak{R}$, where $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$.

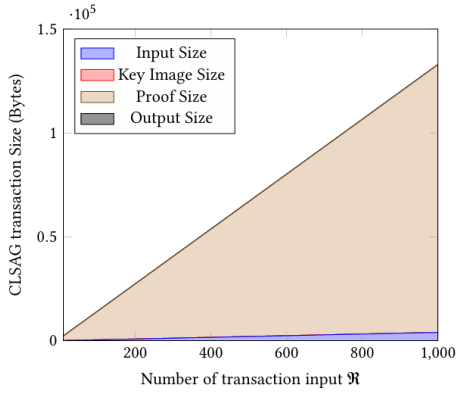

Figure 14: Breakdown of CLSAG transaction size for different $\mathfrak{R}$, where $\mathfrak{S}$ and $\mathfrak{T}$ are fixed to 2.

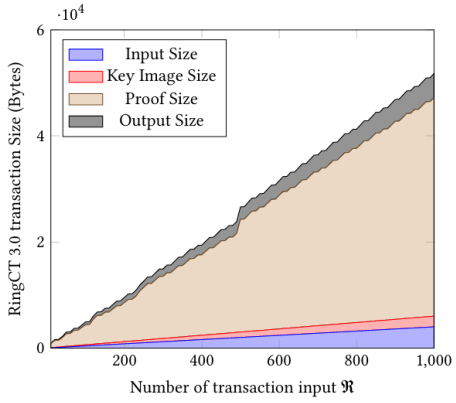

Figure 15: Breakdown of RingCT 3.0 transaction size for different $\mathfrak{R}$, where $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$.

I.3 Comparison

The figures in this section give the following insights for transaction size when $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$:

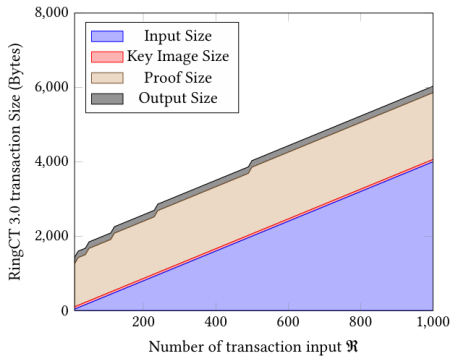

Figure 16: Breakdown of RingCT 3.0 transaction size for different $\mathfrak{R}$, where $\mathfrak{S}$ and $\mathfrak{T}$ are fixed to 2.

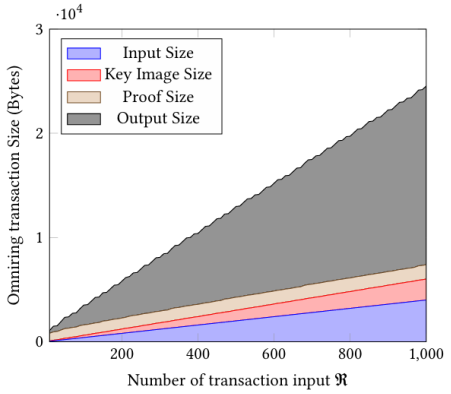

Figure 17: Breakdown of Omniring transaction size for different $\mathfrak{R}$, where $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$.

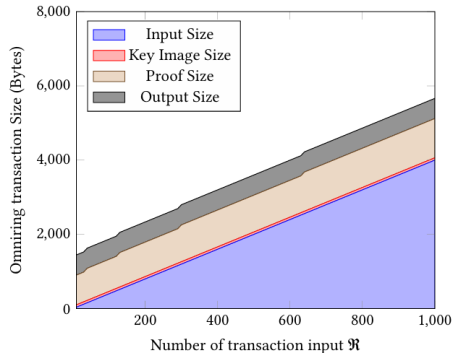

Figure 18: Breakdown of Omniring transaction size for different $\mathfrak{R}$, where $\mathfrak{S}$ and $\mathfrak{T}$ are fixed to 2.

- CLSAG and RingCT 3.0 are dominated by the proof size.
- Omniring is dominated by the output size.
- ZGSX23 is dominated by the key image size.
- BulletCT and SuccinCT have no dominating factor.

The figures in this section give the following insights for transaction size when $\mathfrak{S} = \mathfrak{T} = 2$:

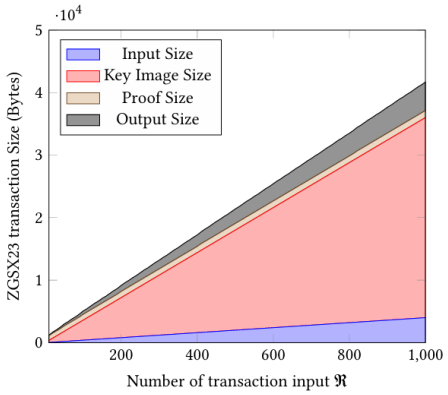

Figure 19: Breakdown of ZGSX23 transaction size for different $\mathfrak{R}$, where $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$.

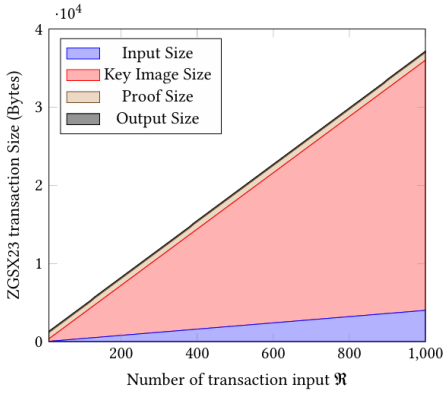

Figure 20: Breakdown of ZGSX23 transaction size for different $\mathfrak{R}$, where $\mathfrak{S}$ and $\mathfrak{T}$ are fixed to 2.

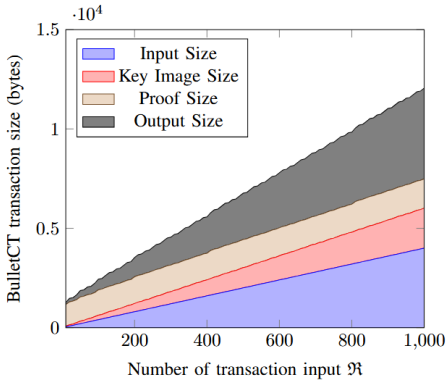

Figure 21: Breakdown of BulletCT transaction size for different $\mathfrak{R}$, where $\mathfrak{S} = \mathfrak{T} = \mathfrak{R}/16$.

- CLSAG is dominated by the proof size.
- RingCT 3.0 is dominated by the proof size for a small ring, and is dominated by the input size for a large ring.

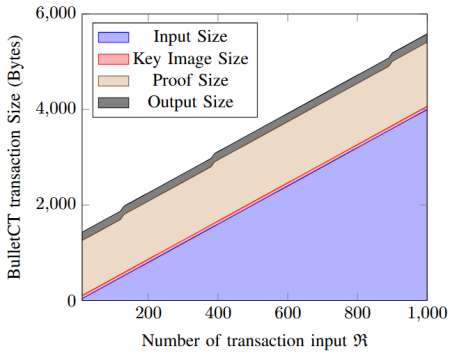

Figure 22: Breakdown of BulletCT transaction size for different $\mathfrak{R}$, where $\mathfrak{S}$ and $\mathfrak{T}$ are fixed to 2.

- Omniring is dominated by the proof size and output size for a small ring, and is dominated by the input size for a large ring.
- ZGSX23 is dominated by the key image size.
- BulletCT and SuccinCT are dominated by the proof size for a small ring, and are dominated by the input size for a large ring.

From the figures, we can observe that CLSAG and RingCT 3.0 are dominated by the proof size. Omniring is mainly dominated by the output size. ZGSX23 is dominated by the key image size. BulletCT and SuccinCT have relatively balanced sizes.

I.4 Computation Complexity

We assume that the spender will run the **Receive** algorithm first to generate his secret key sk and tag before running the **Spend** algorithm. We only count the exponentiations on the elliptic curve since they are the most expensive computation.

We first analyse the **Spend** protocol of SuccinCT. In step (1), it has 2 exponentiations. In step (3), it has $\mathfrak{S}+3$ exponentiations for computing T. It generates the one-time account and coins for each output by **OTAccGen**, which consists of 1 exponentiation for tek, $\mathfrak{T}$ exponentiations for all s, $\mathfrak{T}(q+1)$ exponentiations for all pk (assuming that all parties use the same degree q polynomial) and $2\mathfrak{T}$ exponentiations for all co. They add up to $\mathfrak{T}(q+4)+1$ exponentiations. Finally, it contains the Bulletproof++ with witness size $w = \mathfrak{T}(\beta+2)+\mathfrak{S}\mathfrak{R}+3$, which has $O(w)$ exponentiations. For **Verify**, the running time is mainly the verification of Bulletproof++, which requires a single multi-exponentiation of $O(w/\log(w))$ points, together with the $O(\mathfrak{R}+\mathfrak{S})$ exponentiations in the statement.

For Omniring, the **Spend** protocol generates the one-time account and coins for each output by **OTAccGen**, which consists of $\mathfrak{T}$ exponentiation for pk, $2\mathfrak{T}$ exponentiations for all co, $4\mathfrak{T}$ exponentiations for the encryption of all coin keys. They add up to $7\mathfrak{T}$ exponentiations. They also has a Bulletproof with witness size $w_{\mathsf{omni}} = \mathfrak{T}(\beta+2)+\mathfrak{S}(\mathfrak{R}+3)$, which has $O(w_{\mathsf{omni}})$ exponentiations. For **Verify**, the running time is mainly the verification of Bulletproof, which requires a single multi-exponentiation of $O(w_{\mathsf{omni}}/\log(w_{\mathsf{omni}}))$ points. We do not give the exact number of computation since this data is not available in Bulletproof or Bulletproof++.

J Discussion on the Input Size

J.1 Evaluating the Input Size

The representation of the input ring in Monero is known as the delta encoding followed by variable length encoding. For sorted indices, the expected delta encoding length scales as $\mathfrak{R} \cdot H(P)$, where $H(P)$ is the entropy of the delta distribution. If deltas follow a known distribution (e.g., the OSPEAD distribution suggested by Monero), the length is proportional to the entropy of the distribution.

For simplicity, we assume that each input takes 4 bytes on average in the paper.

J.2 Shortening the Input Size

The input size is one dominating factor for the transaction size of SuccinCT. The only known method to compress it is to use the input offset in Appendix E. However, it is still linear to the size of the ring $\mathfrak{R}$ (around $4\mathfrak{R}$ bytes). Another complication is that the mixin should be chosen according to some secure distribution in Appendix F. Representing the entire ring by using a compact representation has not been studied before.

We show that it is theoretically possible to give a compact representation of the ring, by assuming that there exists a public mapping ϕ and its inverse ϕ^{-1} such that:

- Suppose that the total number of transaction is less than a prime $\hat{p}$. Given some random $z_1, \ldots, z_{\mathfrak{R}} \in \mathbb{Z}_{\hat{p}}$, $\phi(z_j)$ maps to a secure distribution of the ring.

The choice of the mapping is based on the secure distribution used. The study of such distribution is out of scope of the paper. Readers can refer to [21, 27, 28] for details.

The mixin can be selected by the following steps:

(1) Suppose that the indices of the spenders in $\mathcal{S}$ are ind_j. It computes $z_{\mathsf{ind}_j} = \phi^{-1}(\mathsf{ind}_j)$ for $j \in [1, \mathfrak{S}]$.
(2) Compute a polynomial f of degree $\mathfrak{S}-1$ by using polynomial interpolation, such that $f(\mathsf{ind}_j) = z_{\mathsf{ind}_j} \in \mathbb{Z}_{\hat{p}}$ for $j \in [1, \mathfrak{S}]$.
(3) Compute $z_i = f(i)$ for $i \in [1, \mathfrak{R}]$. The ring $\mathcal{R}$ is represented by indices $\phi(f(i)) = \phi(z_i)$.

As a result, the input ring can be represented by a degree $\mathfrak{S}-1$ polynomial f, which consists of $\mathfrak{S}$ elements in $\mathbb{Z}_{\hat{p}}$. If $\hat{p}$ is less than 4 bytes (for >4 billion transactions), then the input ring can be represented by $4\mathfrak{S}$ bytes. This new representation is $O(\mathfrak{S})$ and outperforms the existing one.