RingCT 3.0 for Blockchain Confidential Transaction: Shorter Size and Stronger Security

Tsz Hon Yuen[1], Shi-Feng Sun[2,3], Joseph K. Liu[2], Man Ho Au[1], Muhammed F. Esgin[2,3], Qingzhao Zhang[4], Dawu Gu[4]

[1] The Univeristy of Hong Kong, Hong Kong
{thyuen, allenau}@cs.hku.hk
[2] Monash University, Australia,
{shifeng.sun, joseph.liu, muhammed.esgin}@monash.edu
[3] Data61, CSIRO, Australia
[4] Shanghai Jiao Tong University, China
{fszqz001, dwgu}@sjtu.edu.cn

Abstract. In this paper, we propose the most efficient blockchain ring confidential transaction protocol (RingCT3.0) for protecting the privacy of the sender's identity, the recipient's identity and the confidentiality of the transaction amount. For a typical 2-input transaction with a ring size of 1024, the ring signature size of our RingCT3.0 protocol is 98% less than the ring signature size of the original RingCT1.0 protocol used in Monero. Taking the advantage of our compact RingCT3.0 transcript size, privacy-preserving cryptocurrencies can enjoy a much lower transaction fee which will have a significant impact on the crypto-economy.
In addition to the significant improvement in terms of efficiency, our scheme is proven secure in a stronger security model. We remove the trusted setup assumption used in RingCT2.0. Our scheme is anonymous against non-signing users who are included in the ring, while we show that the RingCT1.0 is not secure in this improved model. Our implementation result shows that our protocol outperforms existing solutions, in terms of efficiency and security.

1 Introduction

Monero, Dash and Zcash are three popular privacy-preserving cryptocurrencies having total market capitalization of USD 1.5 billion. They are ranked at 16, 26 and 32 of all cryptocurrencies as of December 2019. They use different cryptographic techniques: Monero [13] uses linkable ring signatures, Pedersen commitment and Diffie-Hellman key agreement; Dash uses coin shuffling; Zcash [1] uses general zero-knowledge proof (zk-SNARK). These cryptographic techniques mainly suffer from two drawbacks: inefficient signature generation/verification, or large signature size. The latter is more concerned in public blockchains, since it is directly related to the transaction fee.

Transaction Fee. In public blockchain, the *miners* are motivated for bookkeeping because they earn rewards in terms of new cryptocurrency mined and the

transaction fee from each transaction they recorded. The transaction fee is determined by the size of the transaction data. Different cryptocurrencies have their own *fee rate*, i.e., the price per kB of transaction data. During Nov 2017 to Feb 2018, Bitcoin's fee rate reaches 0.008 BTC/kB (which is over USD 100/kB at that time); since then, Bitcoin's fee rate is relatively stable at 0.0002 BTC/kB (which is about USD 1.6/kB). Monero has a stable fee rate of about 0.0008 XMR/kB (which is about USD 0.2/kB).

Transaction fee depends on the length of the transaction data, which is dominated by the signature length of the senders. In Bitcoin, a *typical* transaction of 2-input-2-output contains 2 ECDSA signatures, the length of which is 1kB. As of December 2019, the average transaction fee of Bitcoin is USD 0.25 and the monthly transaction fee of the whole Bitcoin system is USD 2.5M. In Monero, the total signature size for a typical confidential transaction is 13.2kB. Therefore, any effort to reduce the signature size will have a significant impact. The improvement in signature size is relatively more important than the improvement in computation efficiency for public blockchains.

Ring Signatures in Blockchain. In this paper, we will focus on the ring signature [15], which allows a user to dynamically choose a set of public keys (including his own) and to sign messages on behalf of the set, without revealing his identity. In anonymous e-cash or cryptocurrency system, linkable-anonymity is more suitable than perfect anonymity since a double-spent payment can be detected. In a linkable ring signature [9], given any two signatures, the verifier knows that whether they are generated by the same signer (even though the verifier still does not know who the actual signer is).

RingCT. The first blockchain Confidential Transaction (CT) [10] is a proposed enhancement to the Bitcoin protocol for hiding payment values in the blockchain. In cryptocurrency Monero, linkable ring signature is used with CT to give a *Ring Confidential Transaction* (RingCT) protocol [13]. For M transaction inputs, they correspond to M ring signatures of ring size $O(n)$ each, where n is the number of possible signer. In addition the net transaction amount (which should be equal to a commitment of zero) also corresponds to a ring signature of ring size $O(n)$. Therefore, Monero's RingCT1.0 [13] has $(M + 1)$ signatures of size $O(n)$ each. Since the large signature size limits the the number n of possible signers, the value of n in Monero's official wallet software ranges from 5 to 20 only. As a result, the sender anonymity for RingCT1.0 is at most 1-out-of-20. Due to the small ring size, there are some attacks to the anonymity of Monero users such as [7,17,12].

The RingCT1.0 paper [13] does not give any notion and security model of RingCT, which are then later formalized in [16] and they give a RingCT2.0 protocol with $(M+1)$ signatures of size $O(1)$ by using trusted public parameters. However, the use of trusted public parameters is not desirable in the setting of public blockchain. Note that the above comparison ignores the computation of range proof (e.g., an efficient range proof can be adopted from [4]), the M key images for linking double-spending transactions, and the computation of N committed output transaction amount.

RingCT	Communication		Actual Size for $M = 2$ (Bytes)				
	$\mathbb{G}$	$\mathbb{Z}_p$	$n = 16$	$n = 64$	$n = 256$	$n = 1024$	$n = 4096$
RingCT1.0	$M + 1$	$(M + 1)(n + 1)$	1731	6339	24771	98499	393411
This paper	$2 \log nM + 9$	M+8	947	1079	1211	1343	1475

Table 1: Size of RingCT without trusted setup for a set of M transaction inputs and each input generates a ring signature of ring size n (excluding the range proof, the key images and the input/output accounts), for 128-bit security.

1.1 Our Contributions

Our goal is to construct a cost-efficient blockchain RingCT protocol by using an efficient ring signature scheme without trusted setup, and to prove the security in a stronger security model. Specifically, the contributions of this paper include:

1. We build a novel efficient ring signature scheme to construct RingCT3.0 protocol with the shortest RingCT transcript size, without using trusted setup. As shown in Table 1, our RingCT3.0 has ring signature size of $O(M + \log n)$ and the original RingCT1.0 [13] has ring signature size of $O(Mn)$. Consider a typical transaction (i.e., number of inputs $M = 2$) with a ring size of 1024, our ring signature size (1.3kB) is 98.6% less than the ring signature size of [13] (98kB). It provides significant savings of the transaction fee for privacy-preserving cryptocurrencies. In addition, it becomes practical to include a larger ring size (e.g., to include 10^5 users with less than 1800 bytes for the signature size) and therefore (having a large ring size) makes it extremely difficult to launch an anonymity attack based on blockchain data analysis.
2. We give a strong security model for RingCT. In particular, we give a clearer security model for the balance property, and give a stronger security model for anonymity by considering insider attack. We will show that the original RingCT1.0 protocol in [13] is not secure in this anonymity model for insider attack. Then we will show that our RingCT3.0 is secure in this improved model.
3. Our significant improvements in the RingCT3.0 protocol rely on our proposed brand new ring signature scheme. It is the shortest ring signature without trusted setup in the literature (refer to Table 2). The idea comes from an innovative technique to construct an efficient set membership proof of n public keys in the base group, instead of in the exponent. We believe these two primitives are of independent interest and contributions.

2 Background

Vector Notations. For a scalar $c \in \mathbb{Z}_p$ and a vector $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$, we denote by $\boldsymbol{b} = c\boldsymbol{a}$ the vector of $b_i = c \cdot a_i$ for $i \in [1, n]$. Let $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{i=1}^{n} a_i b_i$ denote the inner product between two vectors $\boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{a} \circ \boldsymbol{b} = (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$ denote the Hadamard product.

Ring Signatures	Signature Size		Actual Size (Bytes)				
	$\mathbb{G}$	$\mathbb{Z}_p$	$n = 16$	$n = 64$	$n = 256$	$n = 1024$	$n = 4096$
[6]	$4 \log n$	$3 \log n + 1$	944	1400	1856	2312	2768
[2]	$\log n + 12$	$\frac{3}{2} \log n + 6$	912	1074	1236	1398	1560
This paper	$2 \log n + 7$	7	719	1079	1211	1343	1475

Table 2: Summary of $O(\log n)$-size DL-based ring signatures for n public keys.

We use $\boldsymbol{k}^n$ to denote the vector containing the first n-th powers of $k \in \mathbb{Z}_p$. That is $\boldsymbol{k}^n = (1, k, k^2, \ldots, k^{n-1}) \in \mathbb{Z}_p^n$. We use the vector notation to Pedersen vector commitment. Let $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ be a vector of generators and $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$, then $C = \boldsymbol{g}^{\boldsymbol{a}} = \prod_{i=1}^n g_i^{a_i}$.

3 Overview of RingCT3.0

We give a brief overview on how to improve the efficiency and the security of the RingCT protocol using a step-by-step approach.

3.1 Efficient RingCT3.0 Protocol

We give a new design of ring signature scheme to build an efficient RingCT without using trusted setup. This construction is composed of a number of new primitives and techniques.

Set Membership Proof. Our basic idea is to start with a set membership proof of a set of public keys, without trusted setup. We give the first set membership proof without trusted setup for public keys in the base group. The intuition is that we can have a set of public keys $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$ and a binary vector $\boldsymbol{b}_L = (b_1, \ldots, b_n)$. Denote $\boldsymbol{Y}^{\boldsymbol{b}_L} = \prod_{i=1}^n Y_i^{b_i}$. For a public key $Y_i \in \boldsymbol{Y}$, we set $C = h^\beta Y_i$ for some public group element h and randomness β. We observe that C is a Pedersen commitment of the secret key $x_i = \log_g Y_i$. Also, when $\boldsymbol{b}_L$ only has the bit at position i equal to 1, we have:

$$C = h^\beta Y_i = h^\beta \boldsymbol{Y}^{\boldsymbol{b}_L}.$$

Define $\boldsymbol{b_R}$ as $\boldsymbol{b_R} = \boldsymbol{b_L} - \boldsymbol{1}^n$, where $\boldsymbol{1}^n = (1, \ldots, 1)$ of length n. We give a zero-knowledge proof for the above condition of $\boldsymbol{b}_L$ by showing that:

$$\boldsymbol{b_L} \circ \boldsymbol{b_R} = \boldsymbol{0}^n, \quad \boldsymbol{b_L} - \boldsymbol{b_R} = \boldsymbol{1}^n, \quad \langle \boldsymbol{b_L}, \boldsymbol{1}^n \rangle = 1,$$

where $\circ$ denotes the Hadamard product. Since the zero-knowledge proof hides the knowledge of $\boldsymbol{b_L}$, the position index i of the committed public key is hidden.

In order to give an efficient set membership proof of $\boldsymbol{b_L}$ with length n, we use the inner product argument in [4] to reduce the proof size to $\log n$. One non-trivial tweak of our construction is to ensure the security of the Pedersen commitment C on the public key Y_i. We have to set $h = \text{Hash}(\boldsymbol{Y})$ (Hash denotes

a cryptographic hash function), such that the discrete logarithm (DL) between the public key Y_i and h is not known.

Our set membership proof is fundamentally different from the existing approaches. [6,2,3] prove that for a set of commitments, one of them is committed to 0. They use n polynomials of degree $\log n$ to hide the prover index and run a zero-knowledge proof for the polynomials. Our scheme uses a zero-knowledge proof to prove that $\boldsymbol{b}_L$ is a binary vector with Hamming weight 1 and uses the inner product argument in [4] to reduce the proof size to $\log n$. Details of the set membership proof is given in the full version of the paper [18].

Linkable Ring Signatures. We propose the use of set membership proof for constructing ring signatures directly. The signer can directly give a zero-knowledge proof of knowing: (1) a committed public key ($C = h^{\beta} Y_i$) which is in the set of n public keys, and (2) the secret key which corresponds to the committed public key. Details of the ring signature is given in the full version of the paper [18]. However, turning it into linkable ring signature is a non-trivial task.

Firstly, we convert our ring signature into a linkable ring signature by giving an extra linkability tag (also called key image in Monero) for each signer. The security proof of our ring signature scheme requires that the DL between different users' public key should be unknown to the adversary. However, in the security model of balance and non-slanderability for RingCT, the adversary is allowed to have more than one secret key. If we simply use the users' public keys as the representation of users in the ring, the scheme is not secure since the adversary knows the DL between public keys.

The classical representation of user i in DL-setting is the public key $Y_i = g^{x_i}$ where x_i is the user's secret key. Hence, we give a new proposal of using $Y_i g_i^d$ as the *user representation* in the set $\boldsymbol{Y}$, where Y_i is the public key, g_i is the system parameter and d is the hash of all public keys in the ring. For each user representation $Y_i g_i^d$, the g_i component cannot be canceled out by Y_i due to the exponent d added. Now consider the DL between user representations. Even though the adversary knows the secret keys x_i of other users (which is allowed in the security model), the DL relation between different users' representation is still unknown guaranteed by the DL between g and g_i. (Refer to lemma **??**).

Compressing Multiple Inputs for RingCT3.0. A trivial construction of RingCT with M multiple input is to include M linkable ring signatures and then proves that the sum of input amount is equal to the sum of output amount. As a result, we can obtain a RingCT with signature size $O(M \log n)$. In this paper, we follow the technique of proving multiple range proof in [4] to further compress our RingCT 3.0. In short, we use the first n bit of $\boldsymbol{b_L}$ to represent the first linkable ring signature, the second n bit of $\boldsymbol{b_L}$ to represent the second linkable ring signature and so on. As a result, we have a nM bit of $\boldsymbol{b_L}$ for M inputs. By applying the inner product argument, the correctness of $\boldsymbol{b_L}$ is proven with a proof of size $O(\log nM)$. However, we still need M group elements to show the correctness of M key images. Therefore, our final RingCT 3.0 has a proof size of $O(M + \log n)$.

3.2 Strong Security Model for RingCT3.0

As compared to the formal security model of RingCT proposed in [16], we propose a few improvements:

- We remove the use of trapdoor in system parameters. Therefore, this model is more suitable for public blockchain, as compared with RingCT2.0 [16].
- We give a clearer definition of the balance property. We observe that the balance property requires that any malicious user cannot (1) spend any account of an honest user, (2) spend her own accounts with the sum of input amount being different from that of output amount, and (3) double spend any of her accounts. Therefore, we break down the balance property into unforgeability, equivalency and linkability. As a result, the security of each property can be evaluated easily.
- We give a stronger security model of anonymity than the model in [16]. We consider the anonymity against insider attacks. Note that the original RingCT [13] is not secure in this improved model.

Anonymity against Insider Attacks. We observe that anonymity for RingCT protocol is more complicated than the anonymity of linkable ring signatures. Given the knowledge of the input and output amount, the level of anonymity of RingCT protocol may be lowered. For a transaction with multiple input accounts, multiple linkable ring signatures are generated. Yet, they are correlated when validating the balance of input and output amount. This extra relationship may lower the level of anonymity.

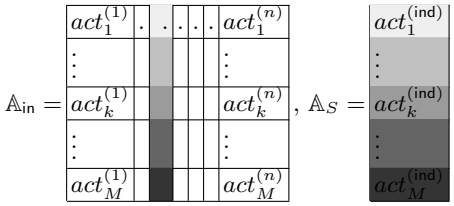
The previous model of anonymity [16] only considered outsider security (i.e., not against the recipient and other members of the ring). In this paper, we define two improved models for anonymity: anonymity against recipient (who knows all the output account secret keys and their amounts) and anonymity against ring insiders (who knows some input account secret keys and their amounts). The collusion of recipient and ring insiders will inevitably lower the level of anonymity. Therefore, we do not capture it in our security model.[5]

Anonymity of RingCT1.0. We first review the original RingCT1.0 [13]. Denote $\mathbb{A}_{\mathsf{in}}$ as the set of all input accounts and $\mathbb{A}_S$ as the set of real signers. Arrange $\mathbb{A}_{\mathsf{in}}$ as an $M \times n$ matrix with each row containing only one account in $\mathbb{A}_S$. [13] requires that all signing accounts are in the same column in $\mathbb{A}_{\mathsf{in}}$. One ring signature is generated for each row. The graphical representation is shown in the following figure.

The real signers must be located in the same column. It is because RingCT1.0 [13] includes an extra ring signature, where each "ring public key" is computed by the product of all coins in each column, divided by all output coins. It is used to guarantee the balance of the input and output amount.

We observe that if the adversary knows one of the secret key in the first column, he can check if any of the key images is generated from this secret key.

[5] This property is different from the simple ring signature setting [14] or the tumbler setting [11], since we also consider different transaction amount in different UTXO.



If not, the adversary can rule out the possibility that the real signer is from the first column. The level of anonymity is already lowered. By knowing $n-1$ secret keys in different columns, the adversary can find out which M input accounts are the real signer. Therefore, the original RingCT1.0 [13] is not secure against our model of anonymity against ring insiders.

Anonymity of RingCT3.0. In order to provide anonymity against ring insiders, we have to break the distribution of real signer in $\mathbb{A}_{\mathsf{in}}$ in RingCT1.0 [13]. We achieve this by three steps: (1) for the k-th row, we change the user representation of our RingCT3.0 as:

$$\boldsymbol{Y_k} = \{(\mathsf{pk}_{\mathsf{in},k}^{(1)})^{d_0^{k-1}}(C_{\mathsf{in},k}^{(1)})^{d_1}g_1^{d_2}, \ldots, (\mathsf{pk}_{\mathsf{in},k}^{(n)})^{d_0^{k-1}}(C_{\mathsf{in},k}^{(n)})^{d_1}g_n^{d_2}\},$$

for some hash values d_0, d_1, d_2, and system parameters $g_1, \ldots, g_n$; (2) compute a batch inner product argument for the set $\boldsymbol{Y} = \boldsymbol{Y_1}||\ldots||\boldsymbol{Y_M}$, such that one element from each $\boldsymbol{Y_k}$ is committed in a commitment B_1; (3) prove the sum of amounts committed in B_1 is equal to the sum of amounts for all the output coins $C_{\mathsf{out},j}$. The second step is done by computing B_1 as the commitment of the multiplication of one element in each $\boldsymbol{Y_k}$ for all k. The third step is done via showing that the coins committed in B_1 divides by $\prod_j C_{\mathsf{out},j}^{d_1}$ is a commitment to zero. It can be done without the need of using ring signatures.

4 Security Model for RingCT

We give the security definitions and models for RingCT which is modified from [16]. A RingCT protocol consists of a tuple of polynomial time algorithms (**Setup**, **KeyGen**, **Mint**, **AccountGen**, **Spend**, **Verify**), the syntax of which are described as follows:

- $pp \leftarrow \mathbf{Setup}(1^\lambda)$: it takes a security parameter $\lambda \in \mathbb{N}$, and outputs the system parameters pp. All algorithms below have implicitly pp as part of their inputs.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathbf{KeyGen}()$: In order to provide anonymity to the recipient, the concept of *stealth address* was used in RingCT1.0 [13]. It can be viewed as dividing the algorithm into two parts: generating a long-term key pairs for each user, and generating a one-time key pairs for each transaction.
  - **LongTermKeyGen**. It outputs a long term secret key ltsk and a long term public key ltpk.

  - **OneTimePKGen**. On input a long term public key ltpk, it outputs pk and the auxiliary information R_{ot}.
  - **OneTimeSKGen**. On input a one-time public key pk, an auxiliary information R_{ot} and a long term secret key ltsk, it outputs the one-time secret key sk.
- $(cn, \mathsf{ck}) \leftarrow$ **Mint**$(\mathsf{pk}, \mathsf{a})$: it takes as input a public key pk and an amount a, outputs a coin cn for pk as well as the associated coin key ck.
- $(act, a\mathsf{sk})/\perp \leftarrow$ **AccountGen**$(\mathsf{sk}, \mathsf{pk}, cn, \mathsf{ck}, \mathsf{a})$: it takes as input a user key pair $(\mathsf{sk}, \mathsf{pk})$, a coin cn, a coin key ck and an amount a. It returns $\perp$ if ck is not the coin key of cn with amount a. Otherwise, it outputs the account $act \doteq (\mathsf{pk}, cn)$ and the corresponding account secret key is $a\mathsf{sk} \doteq (\mathsf{sk}, \mathsf{ck}, \mathsf{a})$.
- $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{C}k_{\mathsf{out}})/\perp \leftarrow$ **Spend**$(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$: it takes as input a group $\mathbb{A}_S$ of accounts together with the corresponding account secret keys $\mathbb{K}_S$, an arbitrary set $\mathbb{A}_{\mathsf{in}}$ of groups of input accounts containing $\mathbb{A}_S$, a set $\mathbb{O}$ of output public keys with the corresponding output amounts, and some transaction string $\mathtt{m} \in \{0,1\}^*$, it outputs $\perp$ if the sum of output amount in $\mathbb{O}$ is different from the sum of input amount in $\mathbb{K}_S$. Otherwise, it outputs a set of output accounts $\mathbb{A}_{\mathsf{out}}$, a proof π, a set $\mathbb{S}$ of serial numbers and a set of output coin keys $\mathbb{C}k_{\mathsf{out}}$. Each serial number $S_i \in \mathbb{S}$ corresponds to one account secret key $a\mathsf{sk}_i \in \mathbb{K}_S$.
- 1/0/-1 $\leftarrow$ **Verify**$(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S})$: it takes as input a message $\mathtt{m}$, a set of input accounts $\mathbb{A}_{\mathsf{in}}$, a set of output accounts $\mathbb{A}_{\mathsf{out}}$, a proof π and a set of serial numbers $\mathbb{S}$, the algorithm outputs -1 if the serial numbers in $\mathbb{S}$ is spent previously. Otherwise, it checks if the proof π is valid for the transaction tx, and outputs 1 or 0, meaning a `valid` or `invalid` spending respectively.

The formal security model is given in the full version of the paper [18]. We give a high level description as follows.

Perfect Correctness. The perfect correctness property requires that a user can spend any group of her accounts w.r.t. an arbitrary set of groups of input accounts, each group containing the same number of accounts as the group she intends to spend.

Anonymity. The anonymity of RingCT is more complicated than the anonymity of linkable ring signatures, due to the extra knowledge of transaction amount. The previous model of anonymity in RingCT1.0 only considered outsider security only (i.e., not against the recipient and other members of the ring). As introduced in §3.2, we define two stronger models for anonymity: anonymity against recipients (who know all the output amounts) and anonymity against ring insiders (who know some input account secret keys and their amounts).

Anonymity against recipients. The anonymity against recipients property requires that without the knowledge of any input account secret key and input amount (which are within a valid *Range*: from 0 to a maximum value), the spender's accounts are successfully hidden among all the honestly generated accounts, even when the output accounts and the output amounts are known.

Anonymity against ring insiders. The anonymity against ring insiders property requires that without the knowledge of output account secret key and output amount (which are within a valid *Range*), the spender's accounts are successfully hidden among all uncorrupted accounts.

Balance. The balance property requires that any malicious user cannot (1) spend any account of an honest user, (2) spend her own accounts with the sum of input amount being different from that of output amount, and (3) double spend any of her accounts. Therefore, the balance property can be modeled by three security models: unforgeability, equivalence and linkability.

Non-Slanderability. The non-slanderability property requires that a malicious user cannot prevent any honest user from spending. It is infeasible for any malicious user to produce a valid spending that shares at least one serial number with a honestly generated spending.

5 RingCT 3.0

For the ease of presentation, we first present a basic construction of RingCT3.0 with linkable ring signature. Then we optimize our construction to $\log(n)$-size by using the inner-product argument in [4], where n is the size of the ring.

5.1 Our Basic Construction

We give our basic construction in this section. Our scheme uses a zero-knowledge range proof of a value committed in a Pederson commitment. Denote $\mathsf{RP} = (\mathsf{RSetup}, \mathsf{RProof}, \mathsf{RVerify})$ as a zero-knowledge range proof for the statement:

$$PoK : \{(\mathsf{a}, \kappa) : C = h_c^{\mathsf{a}} g_c^{\kappa} \wedge \mathsf{a} \in [R_{\mathsf{min}}, R_{\mathsf{max}}]\}.$$

The range proof can be instantiated by the Bulletproof [4].

Our basic construction is as follows.

Setup. On input security parameter λ and the maximum size of ring n_{max}, it picks a group $\mathbb{G}$ of prime order p and some generators $g_c, h_c, g, u \in \mathbb{G}$, $\boldsymbol{g} = (g_1, \ldots, g_{n_{\mathsf{max}}})$, $\boldsymbol{h} = (h_1, \ldots, h_{n_{\mathsf{max}}}) \in \mathbb{G}^{n_{\mathsf{max}}}$. Suppose that $H_j : \{0,1\}^* \to \mathbb{Z}_p$ for $j = 1, 2, 4, 5$, $H_3 : \{0,1\}^* \to \mathbb{G}$ and $H_6 : \mathbb{G} \to \mathbb{Z}_p$ are collision resistant hash functions. It also runs RSetup of the range proof. Assume these parameters are known in the system.

KeyGen. The algorithm is divided as follows.

- **LongTermKeyGen**. The user picks his long term secret key $\mathsf{ltsk} \doteq (x_1, x_2) \in \mathbb{Z}_p^2$ and computes his long term public key $\mathsf{ltpk} \doteq (g^{x_1}, g^{x_2})$.
- **OneTimePKGen**. On input a long term public key $\mathsf{ltpk} = (g^{x_1}, g^{x_2})$, it picks a random $r_{\mathsf{ot}} \in \mathbb{Z}_p$ and computes a one-time public key $\mathsf{pk} = g^{x_1} \cdot g^{H_6((g^{x_2})^{r_{\mathsf{ot}}})}$. It outputs pk and the auxiliary information $R_{\mathsf{ot}} \doteq g^{r_{\mathsf{ot}}}$.

– **OneTimeSKGen**. On input a one-time public key pk, an auxiliary information R_{ot} and a long term secret key $\mathsf{ltsk} = (x_1, x_2)$, it checks if $\mathsf{pk} = g^{x_1} \cdot g^{H_6(R_{\mathsf{ot}}^{x_2})}$. If it is correct, then it outputs the one-time secret key $\mathsf{sk} = x_1 + H_6(R_{\mathsf{ot}}^{x_2})$.

Mint. On input a public key pk, an amount $\mathsf{a} \in \mathbb{Z}_p$, the algorithm chooses $\kappa \in \mathbb{Z}_p$ uniformly at random and computes the coin $C = g_c^{\kappa} h_c^{\mathsf{a}}$. It returns the coin C and the coin key $\mathsf{ck} = \kappa$.

AccountGen. On input a user key pair $(\mathsf{sk}, \mathsf{pk})$, a coin C and a coin key $\mathsf{ck} = \kappa$ (where the pair (pk, C) is listed as the output of a transaction) and an amount a, it checks if $C = g_c^{\kappa} h_c^{\mathsf{a}}$. If it is true, then it outputs the account $act \doteq (\mathsf{pk}, C)$ and the corresponding account secret key is $a\mathsf{sk} \doteq (\mathsf{sk}, \mathsf{ck}, \mathsf{a})$.

Spend. On input a set of M signer's input accounts $\mathbb{A}_S$ with a set of account secret keys $\{a\mathsf{sk}_k = (\mathsf{sk}_k, \kappa_{\mathsf{in},k}, \mathsf{a}_{\mathsf{in},k})\}_{k\in[1,M]}$, a set of nM input accounts $\mathbb{A}_{\mathsf{in}}$ which contains $\mathbb{A}_S$ (where $n < n_{\mathsf{max}}$ is the size of the ring), a set of N output amount $\{\mathsf{a}_{\mathsf{out},j}\}_{j\in[1,N]}$ corresponding to N recipient's public keys $\{\mathsf{pk}_{\mathsf{out},j}\}_{j\in[1,N]}$, and a transaction message $\mathtt{m}$, it first checks the amount balance. If $\sum_{k=1}^{M} \mathsf{a}_{\mathsf{in},k} \neq \sum_{j=1}^{N} \mathsf{a}_{\mathsf{out},j}$, the transaction amount is not correct and it returns $\perp$.

Arrange $\mathbb{A}_{\mathsf{in}}$ as an $M \times n$ matrix with each row containing only one account in $\mathbb{A}_S$. Denote the column index ind_k as the position of the k-th element in $\mathbb{A}_S$ appearing in row k, column ind_k of $\mathbb{A}_{\mathsf{in}}$. The graphical representation is as follows:

$$\mathbb{A}_{\mathsf{in}} = \begin{array}{|c|c|c|c|c|c|c|} \hline act_1^{(1)} & & & & & & act_1^{(n)} \\ \hline \vdots & & & & & & \vdots \\ \hline act_k^{(1)} & & & & & & act_k^{(n)} \\ \hline \vdots & & & & & & \vdots \\ \hline act_M^{(1)} & & & & & & act_M^{(n)} \\ \hline \end{array}, \quad \mathbb{A}_S = \begin{array}{|c|} \hline act_1^{(\mathsf{ind}_1)} \\ \hline \vdots \\ \hline act_k^{(\mathsf{ind}_k)} \\ \hline \vdots \\ \hline act_M^{(\mathsf{ind}_M)} \\ \hline \end{array}$$

The spend protocol can be roughly separated into two parts. The first part is mainly about the balance of the input and output amount. The second part is mainly about the ring signature providing anonymity of the sender.

We first give some sub-protocols for the *balance* property as follows.

1. Generate One-Time Public Key: The sender converts all recipient's long term public keys to one-time public keys by **OneTimePKGen**. The auxiliary information is appended to the transaction message $\mathtt{m}$.
2. Generate Output Coins. It first runs $(C_{\mathsf{out},j}, \kappa_{\mathsf{out},j}) \leftarrow \mathbf{Mint}(\mathsf{a}_{\mathsf{out},j})$, for all $j \in [1, N]$. It sets $\mathbb{A}_{\mathsf{out}} = \{(\mathsf{pk}_{\mathsf{out},j}, C_{\mathsf{out},j})\}_{j\in[1,N]}$ as the set of N output accounts.
   The sender can later privately send the amount $\mathsf{a}_{\mathsf{out},j}$ and coin key $\kappa_{\mathsf{out},j}$ to each secret key owner of $\mathsf{pk}_{\mathsf{out},j}$. Denote $\mathbb{C}k_{\mathsf{out}}$ as the set of all coin keys.

3. Generate Range Proof. It runs the RProof of the range proof for all $\mathsf{a}_{\mathsf{out},j}$ where $j \in [1, N]$. Denote π_{range} as the set of output of RProof for all j.
4. Prepare Balance Proof. Denote the coin for $act_k^{(\mathsf{ind}_k)}$ as $C_{\mathsf{in},k}^{(\mathsf{ind}_k)}$. Recall that the coin key of $C_{\mathsf{in},k}^{(\mathsf{ind}_k)}$ is $(\mathsf{a}_{\mathsf{in},k}, \kappa_{\mathsf{in},k})$. If sum of input amount is equal to the sum of output amount, we have $\prod_{k=1}^{M} C_{\mathsf{in},k}^{(\mathsf{ind}_k)} / \prod_{j=1}^{N} C_{\mathsf{out},j} = g_c^{\sum_{k=1}^{M} \kappa_{\mathsf{in},k} - \sum_{j=1}^{N} \kappa_{\mathsf{out},j}}$. Denote $\Delta \doteq \sum_{k=1}^{M} \kappa_{\mathsf{in},k} - \sum_{j=1}^{N} \kappa_{\mathsf{out},j}$.

Next, we give some sub-protocols for the *ring signature* part. Denote $act_k^{(i)} = (\mathsf{pk}_{\mathsf{in},k}^{(i)}, C_{\mathsf{in},k}^{(i)})$ for $i \in [1, n]$ and the signer index is ind_k. The sender runs as follows.

1. Generate One-Time Secret Key: The sender converts his long term secret key to one-time secret keys by **OneTimeSKGen**.
2. Generate Key Images. Denote $(\mathsf{sk}_k, \cdot, \cdot)$ as the account secret key for $act_k^{(\mathsf{ind}_k)}$. It computes the key image $U_k = u^{\frac{1}{\mathsf{sk}_k}}$.
3. Ring Formation. Denote the concatenated string str as the concatenation of $\{act_k^{(1)} || \ldots || act_k^{(n)}\}_{k \in [1,M]}$. The prover computes $d_0 = H_2(0, \mathsf{str})$, $d_1 = H_2(1, \mathsf{str})$ and $d_2 = H_2(2, \mathsf{str})$. The prover sets $\boldsymbol{Y} = \boldsymbol{Y_1} || \ldots || \boldsymbol{Y_M}$, where:

$$\boldsymbol{Y_k} = ((\mathsf{pk}_{\mathsf{in},k}^{(1)})^{d_0^{k-1}} (C_{\mathsf{in},k}^{(1)})^{d_1} g_1^{d_2}, \ldots, (\mathsf{pk}_{\mathsf{in},k}^{(n)})^{d_0^{k-1}} (C_{\mathsf{in},k}^{(n)})^{d_1} g_n^{d_2}) \quad \text{for } k \in [1, M].$$

4. Prepare Signer Index. For $k \in [1, M]$, the sender generates a binary vector $\boldsymbol{b_{L,k}} = (b_{k,1}, \ldots, b_{k,n})$, where $b_{k,i} = 1$ when $i = \mathsf{ind}_k$ and $b_{k,i} = 0$ otherwise. Define $\boldsymbol{b_L} = \boldsymbol{b_{L,1}} || \ldots || \boldsymbol{b_{L,M}}$ and $\boldsymbol{b_R} = \boldsymbol{b_L} - \boldsymbol{1^n}$. We will prove in zero knowledge that $\boldsymbol{b_{L,k}}$ is a binary vector with only one bit equal to 1. It is equivalent to showing: $\boldsymbol{b_L} \circ \boldsymbol{b_R} = \boldsymbol{0}^n, \boldsymbol{b_L} - \boldsymbol{b_R} = \boldsymbol{1}^n, \langle \boldsymbol{b_{L,k}}, \boldsymbol{1}^n \rangle = 1$ for $k \in [1, M]$.
5. Signature Generation. It consists of the following steps.

   - *Commit 1.* It sets $h = H_3(\boldsymbol{Y})$, picks random $\alpha_1, \alpha_2, \beta, \rho, r_{\alpha_1}, r_{\alpha_2}, r_{\mathsf{sk}_1}, \ldots, r_{\mathsf{sk}_M}, r_\delta \in \mathbb{Z}_p$, $\boldsymbol{s_L}, \boldsymbol{s_R} \in \mathbb{Z}_p^{nM}$ and computes:

$$B_1 = h^{\alpha_1} \prod_{k=1}^{M} (\mathsf{pk}_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_0^{k-1}} (C_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_1} g_{\mathsf{ind}_k}^{d_2}, \quad B_2 = h^{\alpha_2} \prod_{k=1}^{M} g_{\mathsf{ind}_k}, \quad A = h^\beta \boldsymbol{h^{b_R}},$$

$$S_1 = h^{r_{\alpha_1} - d_2 r_{\alpha_2}} g^{\sum_{k=1}^{M} r_{\mathsf{sk}_k} d_0^{k-1}} g_c^{d_1 r_\Delta}, \quad S_2 = h^\rho \boldsymbol{Y^{s_L}} \boldsymbol{h^{s_R}}, \quad S_3 = \prod_{k=1}^{M} U_k^{r_{\mathsf{sk}_k} d_0^{k-1}}.$$

   Observe that $B_1 = h^{\alpha_1} \boldsymbol{Y^{b_L}}$.

   - *Challenge 1.* Denote the concatenated string $\mathsf{str}' = \boldsymbol{Y} || B_1 || B_2 || A || S_1 || S_2 || S_3 || U_1 || \ldots || U_M$. It computes $y = H_4(1, \mathsf{str})$, $z = H_4(2, \mathsf{str})$ and $w = H_4(3, \mathsf{str})$.

   - *Commit 2.* It can construct two degree 1 polynomials of variable X:

$$l(X) = \boldsymbol{b_L} - z\boldsymbol{1}^{nM} + \boldsymbol{s_L} \cdot X,$$

$$r(X) = \boldsymbol{y}^{nM} \circ (w\boldsymbol{b_R} + wz\boldsymbol{1}^{nM} + \boldsymbol{s_R} \cdot X) + \sum_{k=1}^{M} z^{1+k} \cdot (\boldsymbol{0}^{(k-1)n} || \boldsymbol{1}^n || \boldsymbol{0}^{(M-k)n}).$$

Denote $t(X) = \langle l(X), r(X)\rangle$, which is a degree 2 polynomial. We can write $t(X) = t_0 + t_1 X + t_2 X^2$, and t_0, t_1, t_2 can be computed by using $(\boldsymbol{b_L}, \boldsymbol{b_R}, \boldsymbol{s_L}, \boldsymbol{s_R}, w, y, z)$. In particular, observe that

$$\begin{aligned} t_0 &= w\langle \boldsymbol{b_L}, \boldsymbol{b_R} \circ \boldsymbol{y}^{nM}\rangle + zw\langle \boldsymbol{b_L} - \boldsymbol{b_R}, \boldsymbol{y}^{nM}\rangle + \sum_{k=1}^{M} z^{1+k}\langle \boldsymbol{b_L}, \mathbf{0}^{(k-1)n}||\mathbf{1}^n||\\ &\quad \mathbf{0}^{(M-k)n}\rangle - wz^2\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle - \sum_{k=1}^{M} z^{2+k}\langle \mathbf{1}^{nM}, \mathbf{0}^{(k-1)n}||\mathbf{1}^n||\mathbf{0}^{(M-k)n}\rangle,\\ &= \sum_{k=1}^{M} z^{1+k} + w(z-z^2)\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle - \sum_{k=1}^{M} nz^{2+k}. \end{aligned}$$

It picks random $\tau_1, \tau_2 \in \mathbb{Z}_p$, and computes: $T_1 = g^{t_1}h^{\tau_1}, T_2 = g^{t_2}h^{\tau_2}$.

- *Challenge 2.* It computes $x = H_5(w, y, z, T_1, T_2, \mathtt{m})$.

- *Response.* It computes:

$$\begin{aligned} &\tau_x = \tau_1 \cdot x + \tau_2 \cdot x^2, \quad \mu = \alpha_1 + \beta \cdot w + \rho \cdot x, \quad z_{\alpha_1} = r_{\alpha_1} + \alpha_1 \cdot x,\\ &z_{\alpha_2} = r_{\alpha_2} + \alpha_2 \cdot x, \quad z_{\Delta} = r_{\Delta} + \Delta \cdot x, \quad z_{\mathsf{sk},k} = r_{\mathsf{sk},k} + \mathsf{sk}_k \cdot x \ \text{ for } k \in [1, M],\\ &\boldsymbol{r} = \boldsymbol{y}^{nM} \circ (w\boldsymbol{b_R} + wz\mathbf{1}^{nM} + \boldsymbol{s_R} \cdot x) + \sum_{k=1}^{M} z^{1+k} \cdot (\mathbf{0}^{(k-1)n}||\mathbf{1}^n||\mathbf{0}^{(M-k)n}),\\ &\boldsymbol{l} = \boldsymbol{b_L} - z \cdot \mathbf{1}^{nM} + \boldsymbol{s_L} \cdot x, \quad t = \langle \boldsymbol{l}, \boldsymbol{r}\rangle. \end{aligned}$$

It outputs $\sigma_{\mathsf{ring}} = (B_1, B_2, A, S_1, S_2, S_3, T_1, T_2, \tau_x, \mu, z_{\alpha_1}, z_{\alpha_2}, z_{\mathsf{sk},1}, \ldots, z_{\mathsf{sk},M}, z_{\Delta}, \boldsymbol{l}, \boldsymbol{r}, t)$ and the key image $(U_1, \ldots, U_M)$.

Output. Denote $\mathbb{S}$ as a set of serial number $\{U_1, \ldots, U_M\}$. Then the output of the spend algorithm is $(\mathbb{A}_{\mathsf{out}}, \pi = (\pi_{\mathsf{range}}, \sigma_{\mathsf{ring}}), \mathbb{S}, \mathbb{C}k_{\mathsf{out}})$.

Verify. On input a message $\mathtt{m}$, a set of input accounts $\mathbb{A}_{\mathsf{in}}$, a set of output accounts $\mathbb{A}_{\mathsf{out}}$, a proof π and a set $\mathbb{S}$ of serial numbers and a set $\mathbb{U}$ of serial numbers in the past, then it checks:

1. If there exists any U in both $\mathbb{S}$ and $\mathbb{U}$, returns -1 and exits since it is a double spending of the previous transaction. We can use Bloom filter on $\mathbb{U}$ to speed up the detection of double spending.
2. It runs the RVerify algorithm of the range proof with input from π_{range} and the output coins in $\mathbb{A}_{\mathsf{out}}$.
3. It checks the ring signature σ_{ring} and key images $U_k \in \mathbb{S}$ for $k \in [1, M]$. It computes d_0, d_1, d_2 and $\boldsymbol{Y}$ as in the Ring Formation of the **Spend** protocol, using $\mathbb{A}_{\mathsf{in}}$. Denote the concatenated string $\mathsf{str} = \boldsymbol{Y}||B_1||B_2||A||S_1||S_2||S_3||U_1||\ldots||U_M$. It computes $h = H_3(\boldsymbol{Y})$, $y = H_4(1, \mathsf{str})$, $z = H_4(2, \mathsf{str})$, $w = H_4(3, \mathsf{str})$ and $x = H_5(w, y, z, T_1, T_2, \mathtt{m})$. Define $\boldsymbol{h'} = (h'_1, \ldots, h'_{nM}) \in \mathbb{G}^{nM}$ such that $h'_i = h_i^{y^{-i+1}}$ for $i \in [1, nM]$. It returns 1 if all of the following hold

and returns 0 otherwise:

$$t = \langle \boldsymbol{l}, \boldsymbol{r} \rangle, \tag{1}$$

$$g^t h^{\tau_x} = g^{\sum_{k=1}^{M} z^{1+k}(1-nz)+w(z-z^2)\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle} \cdot T_1^x \cdot T_2^{x^2}, \tag{2}$$

$$h^{\mu} \boldsymbol{Y}^{\boldsymbol{l}} \boldsymbol{h'}^{\boldsymbol{r}} = B_1 A^w S_2^x \boldsymbol{Y}^{-z \cdot \mathbf{1}^{nM}} \boldsymbol{h'}^{wz \cdot \boldsymbol{y}^{nM} + \sum_{k=1}^{M} z^{1+k} \cdot (\mathbf{0}^{(k-1)n} || \mathbf{1}^n || \mathbf{0}^{(M-k)n})}, \tag{3}$$

$$h^{z_{\alpha_1} - d_2 z_{\alpha_2}} g^{\sum_{k=1}^{M} z_{\mathsf{sk},k} d_0^{k-1}} g_c^{d_1 z_\Delta} = S_1 (B_1 \cdot \prod_{j=1}^{N} C_{\mathsf{out},j}^{d_1} \cdot B_2^{-d_2})^x, \tag{4}$$

$$\prod_{k=1}^{N} U_k^{z_{\mathsf{sk},k} d_0^{k-1}} = S_3 \cdot u^{x \sum_{k=1}^{N} d_0^{k-1}}. \tag{5}$$

Security Analysis. The security proofs of the following theorems are given in the full version of the paper [18].

Theorem 1 (Balance). *Our scheme is unforgeable if the DL assumption holds in* $\mathbb{G}$ *in the random oracle model (ROM). Our scheme is equivalent w.r.t. insider corruption if the DL assumption holds in* $\mathbb{G}$ *in the ROM and* RP *is a secure zero-knowledge range proof. Our scheme is linkable w.r.t. insider corruption if the DL assumption holds in* $\mathbb{G}$ *in the ROM.*

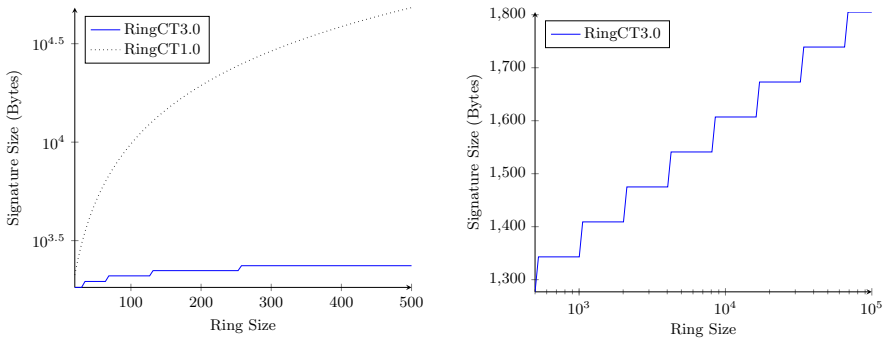
Theorem 2 (Anonymity). *Our scheme is anonymous against recipients if the q-DDHI assumption holds in* $\mathbb{G}$ *in the ROM, where q is the number of Spend oracle query. Our scheme is anonymous against ring insiders if the q-DDHI assumption holds in* $\mathbb{G}$ *in the ROM and* RP *is a secure zero-knowledge range proof.*

Theorem 3 (Non-slander). *Our scheme is non-slanderable w.r.t. insider corruption if the DL assumption holds in* $\mathbb{G}$ *in the random oracle model.*

5.2 Our Efficient Construction

The last step towards our final construction is to use the improved inner product argument in [4] to compress the $O(n)$-size vector $\boldsymbol{l}, \boldsymbol{r}$ in the ring signature part to a $O(\log n)$-size proof. Denote **IPProve**, **IPVerify** as the inner product argument. Details of the algorithm can be found in [4]. We give the modified **Spend'** and **Verify'** algorithms as follows.

- **Spend'**. On input $(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$, it runs $(\mathbb{A}_{\mathsf{out}}, \pi = (\pi_{\mathsf{range}}, \sigma_{\mathsf{ring}}), \mathbb{S}, \mathbb{C}k_{\mathsf{out}}) \leftarrow$ **Spend**$(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$. For each $\sigma_{\mathsf{ring}} = (B_1, B_2, A, S_1, S_2, S_3, T_1, T_2, \tau_x, \mu, z_{\alpha_1}, z_{\alpha_2}, z_{\mathsf{sk},1}, \ldots, z_{\mathsf{sk},M}, z_\Delta, \boldsymbol{l}, \boldsymbol{r}, t)$, it computes $P = \boldsymbol{Y}^{\boldsymbol{l}} \boldsymbol{h'}^{\boldsymbol{r}}$, where $\boldsymbol{Y}$ and $\boldsymbol{h'}$ are defined in **Spend**. it runs $(\boldsymbol{L}, \boldsymbol{R}, a, b) \leftarrow$ **IPProve**$(\boldsymbol{Y}, \boldsymbol{h'}, t, P, \boldsymbol{l}, \boldsymbol{r})$. Note that $\boldsymbol{L}, \boldsymbol{R}$ are vectors of $\mathbb{G}$ with size $\log n$. It sets $\sigma'_{\mathsf{ring}} = (B_1, B_2, A, S_1, S_2, S_3, T_1, T_2, \tau_x, \mu, z_{\alpha_1}, z_{\alpha_2}, z_{\mathsf{sk},1}, \ldots, z_{\mathsf{sk},M}, z_\Delta, t, P, \boldsymbol{L}, \boldsymbol{R}, a, b)$. The algorithm outputs $(\mathbb{A}_{\mathsf{out}}, \pi = (\pi_{\mathsf{range}}, \sigma'_{\mathsf{ring}}), \mathbb{S}, \mathbb{C}k_{\mathsf{out}})$.
- **Verify'**. On input $(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi = (\pi_{\mathsf{range}}, \sigma'_{\mathsf{ring}}), \mathbb{S})$, denote $\sigma'_{\mathsf{ring}} = (B_1, B_2, A, S_1, S_2, S_3, T_1, T_2, \tau_x, \mu, z_{\alpha_1}, z_{\alpha_2}, z_{\mathsf{sk},1}, \ldots, z_{\mathsf{sk},M}, z_\Delta, t, P, \boldsymbol{L}, \boldsymbol{R}, a, b)$. It runs $0/1 \leftarrow$ **IPVerify**$(\boldsymbol{Y}, \boldsymbol{h'}, t, P, (\boldsymbol{L}, \boldsymbol{R}, a, b))$, where $\boldsymbol{Y}$ and $\boldsymbol{h'}$ are defined in **Verify**. It



(a) small ring size

(b) large ring size

Fig. 1: Comparison of RingCT and RingCT3.0 for a transaction with 2 inputs.

outputs 0 if **IPVerify** outputs 0. Otherwise, it runs as the **Verify** algorithm, except that equation (3) is modified to:

$$h^{\mu}P = B_1 \cdot A^{w} \cdot S_2^{x} \cdot \boldsymbol{Y}^{-z\cdot\mathbf{1}^{nM}} \cdot \boldsymbol{h'}^{wz\cdot\boldsymbol{y}^{nM}+\sum_{k=1}^{M} z^{1+k}\cdot(\mathbf{0}^{(k-1)n}||\mathbf{1}^{n}||\mathbf{0}^{(M-k)n})}.$$

The security of our final construction follows from the security of the improved inner product argument in [4], which is based on the DL assumption in the random oracle model.

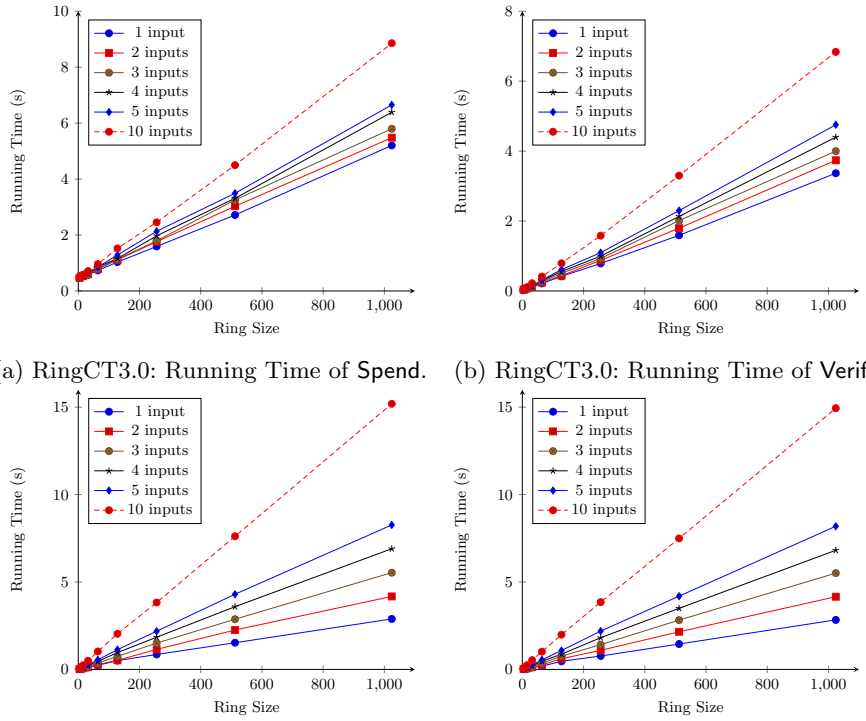
6 Efficiency Analysis

Proof Size of RingCT. RingCT3.0 has size of $O(M + \log n)$ excluding the key images and committed outputs, where n is the size of the ring and M is the number of transaction input. As shown in Figure 1a, RingCT3.0 is significantly shorter than Monero's RingCT1.0 even for small ring size (ring size ≥ 16) and hence RingCT3.0 can reduce the transaction fee by more than 90%. Since the proof size increases logarithmically, the sender can increase the anonymity level by increasing the ring size without increasing the cost drastically. Increasing the ring size of 1000 only increases the transaction fee by 45%.

Consider a typical transaction (i.e., number of inputs $M = 2$) with a ring size of 1024, our ring signature size (1.3kB) is 98.6% less than the ring signature size of [13] (98kB). For the ring size of 1024, the cost of the ring signature part for RingCT1.0 is already about USD 20, which is not practical. On the other hand, the cost of the ring signature part for RingCT3.0 is only about USD 0.27.

Running Time of RingCT. We implemented the RingCT3.0 in Ubuntu 16.04, Intel Core i5-6200U 2.3GHz, 8GB RAM. We used the BouncyCastle's Java library for Curve 25519 in our implementation. Each element in $\mathbb{G}$ is represented by 33 bytes and each element in $\mathbb{Z}_p$ is represented by 32 bytes.

We compare the running time of the Spend protocol of RingCT3.0 in Figure 2a and RingCT1.0 in Figure 2c. Our RingCT3.0 outperforms RingCT1.0 when



(a) RingCT3.0: Running Time of Spend.

(b) RingCT3.0: Running Time of Verify.

(c) RingCT1.0: Running Time of Spend.

(d) RingCT1.0: Running Time of Verify.

Fig. 2: Performance of RingCT3.0 and RingCT1.0.

the ring size exceeds 64. Our RingCT3.0 is better for larger ring size and more user input. When the ring size is 1024 and the input size is 20, RingCT3.0 is about 2 times faster than RingCT1.0.

We compare the running time of the Verify protocol of RingCT3.0 in Figure 2b and RingCT1.0 in Figure 2d. Our RingCT3.0 outperforms RingCT1.0 when the ring size exceeds 32. When the ring size reaches 1024 and the input size is 20, RingCT3.0 is more than 2 times faster than RingCT1.0.

In general, the running time of RingCT3.0 is comparatively shorter than the time of generating a block of transactions, which is 2 minutes in Monero and 10 minutes in Bitcoin. Therefore, RingCT3.0 will not be the bottleneck of the blockchain system.

7 Comparison with Omniring

Recently, a parallel and independent work on RingCT, called *Omniring*, is proposed [8]. They also used the inner product argument as a building block. However, they used a different ring formation, especially for the case of multiple input. Even for M inputs, the total ring size is still n (rather than the total ring size

of nM in our RingCT3.0). The signature size of Omniring is $2\log(3+2n+4M)+9$ $\mathbb{G}$ or $\mathbb{Z}_p$ elements. Our scheme is $2\log(nM)+M+17$ $\mathbb{G}$ or $\mathbb{Z}_p$ elements. In practice n is much larger than M (e.g., $n \geq 1024$ and $M < 5$).

Another major difference between our paper and [8] is the modeling on privacy. They use a single model on privacy to capture the indistinguishability of all possible combinations of transaction input. We use two different models to capture the anonymity against ring insiders and recipients. We illustrate the differences with a simple example. Consider the Omniring [8] with a transaction of two inputs, one output and ring size 8:

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	→	Output
$2	$8	$3	$7	$4	$6	$12	$13		$10

Since their anonymity model allows the adversary to know the account balance for all parties, their security proof guarantees that no PPT adversary can distinguish between the case of "Input 1 and 2 are real signers", "Input 3 and 4 are real signers" and "Input 5 and 6 are real signers". The level of anonymity is only limited to the number of possible combinations of transaction input, which is 1/3 in this case. In practice, a signer does not know the transaction amount of other UTXOs in RingCT. The chances of having many possible combinations of transaction input is relatively low, if the signer forms the ring by randomly picking UTXOs in the blockchain. It is possible that there is only 1 legitimate combination. In this case, the honest signer has no security guarantees according to the anonymity model in [8].

On the other hand, our RingCT 3.0 have a structure as follows:

	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	→	Output
Ring 1	$2	$8	$3	$7	$4	$6	$12	$13		$10
Ring 2	$7	$11	$6	$12	$4	$7	$5	$2		

Our anonymity against ring insiders shows that the real signer in ring 1 (resp. ring 2) is hidden between input 1 to 8 of ring 1 (resp. ring 2), since the output amount is hidden from the adversary. Our anonymity against recipients shows that the real signer in ring 1 (resp. ring 2) is also hidden between input 1 to 8 of ring 1 (resp. ring 2), since the input amount are hidden from the adversary. The level of anonymity is 1/8 in both cases.

8 Conclusion

We propose the RingCT3.0 protocol, which is more efficient and more secure than the existing RingCT1.0 used in Monero. For a typical 2-input transaction with a ring size of 1024, the ring signature size of our RingCT3.0 protocol is 98% less than the ring signature size of the RingCT1.0 protocol.

References

1. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE SP 2014. pp. 459–474. IEEE Computer Society (2014)
2. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C.: Short accountable ring signatures based on DDH. In: Pernul, G., Ryan, P.Y.A., Weippl, E.R. (eds.) ESORICS 2015. LNCS, vol. 9326, pp. 243–265. Springer (2015)
3. Bootle, J., Groth, J.: Efficient batch zero-knowledge arguments for low degree polynomials. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10770, pp. 561–588. Springer (2018)
4. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE S&P 2018. pp. 315–334. IEEE (2018)
5. Foley, S.N., Gollmann, D., Snekkenes, E. (eds.): ESORICS 2017, LNCS, vol. 10493. Springer (2017)
6. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 253–280. Springer (2015)
7. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A traceability analysis of monero's blockchain. In: Foley et al. [5], pp. 153–173
8. Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: Scaling private payments without trusted setup. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) CCS 2019. pp. 31–48. ACM (2019)
9. 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. LNCS, vol. 3108, pp. 325–335. Springer (2004)
10. Maxwell, G.: Confidential transactions (2015), `https://people.xiph.org/~greg/confidential_values.txt`
11. Meiklejohn, S., Mercer, R.: Möbius: Trustless tumbling for transaction privacy. PoPETs **2018**(2), 105–121 (2018)
12. Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., Christin, N.: An empirical analysis of traceability in the monero blockchain. PoPETs **2018**(3), 143–163 (2018)
13. Noether, S.: Ring Signature Confidential Transactions for Monero. Cryptology ePrint Archive, Report 2015/1098 (2015), `http://eprint.iacr.org/`
14. 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)
15. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer (2001)
16. 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: Foley et al. [5], pp. 456–474
17. Wijaya, D.A., Liu, J.K., Steinfeld, R., Liu, D.: Monero ring attack: Recreating zero mixin transaction effect. In: IEEE TrustCom. pp. 1196–1201. IEEE (2018)
18. 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. Cryptology ePrint Archive, Report 2019/508 (2019), `https://eprint.iacr.org/2019/508`