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

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

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

Abstract. In this paper, we propose the most competent 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 to the crypto-economy. Our implementation result shows that our protocol outperforms existing solutions, in terms of efficiency and security.
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 ring insider (non-signing users who are included in the ring), while we show that the RingCT1.0 is not secure in this strong model.
Our RingCT3.0 protocol relies on our brand new designed ring signature scheme as an underlying primitive, which is believed to be the most efficient ring signature scheme up-to-date (in terms of signature size) without trusted setup. Our ring signature scheme is derived from our novel design of an efficient set membership proof of n public keys, with the proof size of $O(\log n)$. It is the first set membership proof without trusted setup for public keys in the base group, instead of in the exponent. These two primitives are of independent interest.

1 Introduction

Blockchain is a distributed ledger of transaction records between nodes, without relying on a trusted authority. Transaction records are synchronized to all nodes by a consensus algorithm, in order to provide a globally agreed, immutable history. However, all transaction data, including the sender's public key, the recipient's public key and the transaction amount, are publicly available. It may not be desirable for sensitive transaction in the area of FinTech. As a result, privacy-preserving blockchain received a lot of attention.

Monero, Dash and Zcash are three popular privacy-preserving cryptocurrencies having total market capitalization of USD 4 billion. They are ranked at 10, 13 and 21 of all cryptocurrencies as of October 2018. They used different cryptographic techniques: Monero [32] used linkable ring signatures, Pedersen commitment and Diffie-Hellman key agreement; Dash used coin shuffling; Zcash [5] used 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 distributed ledger is maintained by *miners*. They 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 0.002-0.008 BTC/kB (which is over USD 100/kB at that time); since then, Bitcoin's fee rate is relatively stable at 0.0001 BTC/kB (which is about USD 1/kB). Monero has a relatively 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 April 2018, the average transaction fee of Bitcoin is USD 1 and the daily transaction fee of the whole Bitcoin system is USD 160,000. 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 to the crypto-economy. 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, which is widely used in privacy-preserving blockchain applications. Ring signature [34] 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. Ring signature provides perfect anonymity, such that it is impossible to decide whether two signatures are issued by the same user. 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 [27], 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) [28] is a proposed enhancement to the Bitcoin protocol for hiding payment values in the blockchain. To further achieve sender anonymity, a number of coin mixing protocols are proposed with CT.

In cryptocurrency Monero, linkable ring signature is used with CT to give a *Ring Confidential Transaction* (RingCT) protocol [32]. For a set of M transaction inputs, the M transaction inputs 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 [32] 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 [22,36,30]. By launching an analysis of the Monero blockchain data, the signer anonymity can be revoked with a significant non-negligible probability.

The RingCT1.0 paper [32] does not give any notion and security model of RingCT, which are then later formalized in [35] 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 [9]), the M key images for linking double-spending transactions, and the computation of N committed output transaction amount.

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

RingCT	Communication		Prover	Verifier
	$\mathbb{G}$	$\mathbb{Z}_p$	(# $\mathbb{G}$ exponentiation)	(# $\mathbb{G}$ exponentiation)
RingCT1.0 [32]	$M+1$	$(M+1)(n+1)$	$2(M+1)(n-1)\mathrm{e}_2 + 2(M+1)\mathrm{e}_1$	$2(M+1)n \cdot \mathrm{e}_2$
This paper	$2\log nM + 9$	M+8	$2(\mathrm{e}_{nM+1} + \mathrm{e}_{\frac{nM}{2}+1} + \ldots + e_1)$ $+\mathrm{e}_{2nM+1} + \mathrm{e}_{2nM} + \mathrm{e}_{nM+1} + \mathrm{e}_{M+3}$ $+\mathrm{e}_M + (2nM+1)\mathrm{e}_3 + 4nM\mathrm{e}_2 + \mathrm{e}_1$	$\mathrm{e}_{2nM+2\log nM+4}$ $+\mathrm{e}_4 + \mathrm{e}_5$ $+\mathrm{e}_{M+1}$

Table 1: Summary 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).

Ring Signatures	Signature Size		Prover	Verifier
	$\mathbb{G}$	$\mathbb{Z}_p$	(# $\mathbb{G}$ exponentiation)	(# $\mathbb{G}$ exponentiation)
[21]	$4\log n$	$3\log n + 1$	$\log n \cdot \mathrm{e}_{n+1} + \frac{3}{2}\log n(\mathrm{e}_2 + \mathrm{e}_1)$	$\mathrm{e}_{n+\log n+1} + n(\mathrm{e}_3 + \mathrm{e}_2)$
[7]	$\log n + 12$	$\frac{3}{2}\log n + 6$	$\frac{1}{2}\log n \cdot \mathrm{e}_{n+1} + 2\mathrm{e}_{2\log n+1}$ $+(\frac{1}{2}\log n + 3)\mathrm{e}_2 + 4\mathrm{e}_1$	$2\mathrm{e}_{n+\frac{1}{2}\log n+1}$ $+2\mathrm{e}_{2\log n+2} + 4\mathrm{e}_2$
This paper	$2\log n + 7$	7	$\mathrm{e}_{2n+1} + \mathrm{e}_{n+1} + 3\mathrm{e}_2$ $+(\mathrm{e}_{2n} + \mathrm{e}_n + \ldots + \mathrm{e}_1)$	$\mathrm{e}_{2n+2\log n+4}$ $+\mathrm{e}_4 + \mathrm{e}_3$

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

has ring signature size of $O(M + \log n)$ and the original RingCT1.0 [32] 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 [32] (98kB). In blockchain applications, the transaction fee is proportional to the transaction size. This improvement in the size provides significant savings of the transaction fee for privacy-preserving cryptocurrencies such as Monero. 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 [32] is not secure in this anonymity model for insider attack. Then we will show that our RingCT3.0 is secure in this strong 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 without trusted setup with the proof size of $O(\log n)$. We believe these two primitives are of independent interest and contributions.

2 Background

We give some background for the paper.

2.1 System Model

Public blockchain is a distributed system where users can join the system freely at anytime. User can have different roles:

– Sender/Recipient: The sender acts as the *signer* of a signature scheme to confirm that he wants to spend some money to the recipient in a transaction.

– Miner: The miner acts as the *verifier* of the signature scheme and checks if the signature (and the transaction) is valid. If it is true, all miners run the consensus protocol of the blockchain to agree on the order of all transactions.

In public blockchain, we assume that system parameters are simply agreed by all users and they do not need to be generated by a trusted party. For example, common hash function and elliptic curve can be used.

For the data model in the system, we use the model of Unspent Transaction Outputs (UTXO) as used in many blockchain systems. If a user wants to spend his digital assets in a transaction, he has to refer to the specific assets that he wants to spend. If he spends the same asset twice, the verifier will notice it and will reject the transaction.

2.2 Notations and Intractability Assumptions

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.

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 $C = \boldsymbol{g}^{\boldsymbol{a}} = \prod_{i=1}^{n} g_i^{a_i}$.

Intractability Assumptions. Denote $\mathbb{G}$ as a cyclic group of order p generated by a security parameter 1^λ. This paper uses the following intractability assumptions.

Discrete Logarithm (DL) Assumption. Given (g, g^a) where $g \in \mathbb{G}, a \in \mathbb{Z}_p$, no probabilistic polynomial time (PPT) adversary can output a with non-negligible probability.

q-Decisional Diffie-Hellman Inversion (DDHI) Assumption. Given $(g, g^a, \ldots, g^{a^q}, T)$ where $g \in \mathbb{G}, a \in \mathbb{Z}_p$, no PPT adversary can decide if $T = g^{1/a}$ with non-negligible probability.

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 [9] 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. 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 and h is not known.

Our set membership proof is fundamentally different from the existing approaches. [21,7,8] proved that for a set of commitments, one of them is committed to g^0. They used n polynomials of degree $\log n$ to hide the prover index and ran 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 [9] to reduce the proof size to $\log n$. Details of the set membership proof is given in Appendix B. A comparison with the state-of-the-art $\log n$-size set membership proofs is shown Table 5. Our scheme is the most efficient in terms of communication size for ECC group, where $|\mathbb{G}| \approx |\mathbb{Z}_p|$. The computation of prover and verifier are slightly less efficient than the existing schemes, but the difference is smaller if we consider the acceleration of computing multi-exponentiation.

Linkable Ring Signatures for RingCT3.0. 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 Appendix C. However, turning it into linkable ring signature is a non-trivial task. Simply adding a key image to the ring signature does not give a secure linkable ring signature. It is because in one of the security models of linkable ring signature, the adversary is allowed to know all the secret keys in the ring. This is not compatible with the proof of ring signature, where the adversary does not know the pairwise discrete logarithm between public keys in the ring. As a result, the representation of public keys in the ring has to be changed for linkable ring signature.

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 pairwise 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 pairwise DL . 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.

Therefore, 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 1).

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 [9] to further compress the 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 [35], 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 [35].
- 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 [35]. We consider the anonymity against insider attacks. Note that the original RingCT [32] is not secure in this strong 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 [35] only considered outsider security (i.e., not against the recipient and other members of the ring). In this paper, we define two stronger models for anonymity: anonymity against recipient (who knows all the output account secret keys and their amounts) and anonymity against ring insider (who knows some input account secret keys and their amounts). The collusion of recipient and ring insider will inevitably lower the level of anonymity. Therefore, we do not capture it in our security model.

Anonymity of RingCT1.0. We first review the original RingCT1.0 [32]. 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 $\mathbb{A}_S$. The original RingCT1.0 [32] 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 as follows:

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

The real signers must be located in the same column. It is because RingCT1.0 [32] 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. 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 [32] is not secure against our model of anonymity against ring insider.

Anonymity of RingCT3.0. In order to provide anonymity against ring insider, we have to break the distribution of real signer in $\mathbb{A}_{\mathsf{in}}$ in RingCT1.0 [32]. 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 agrument for the set $\boldsymbol{Y} = \boldsymbol{Y_1}||\ldots||\boldsymbol{Y_M}$, such that one element from each $\boldsymbol{Y_k}$ is committed in 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 [35]. 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 [32]. 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.
  - **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.
- $(cn, \mathsf{ck}) \leftarrow \mathbf{Mint}(\mathsf{pk}, \mathsf{a})$: 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.
- $(act, a\mathsf{sk})/\perp \leftarrow \mathbf{AccountGen}(\mathsf{sk}, \mathsf{pk}, cn, \mathsf{ck}, \mathsf{a})$: it takes as input a user key pair (sk, 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 $a\mathsf{sk} \doteq (\mathsf{sk}, \mathsf{ck}, \mathsf{a})$.
- $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{C}k_{\mathsf{out}})/\perp \leftarrow \mathbf{Spend}(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$: 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 $a\mathsf{sk}_i \in \mathbb{K}_S$.
- $1/0/-1 \leftarrow \mathbf{Verify}(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S})$: 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.

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.

A RingCT protocol is called perfectly correct if for all PPT adversaries $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{l|l} \mathbf{Verify}(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}) = 1 : & \begin{array}{l} pp \leftarrow \mathbf{Setup}(1^\lambda); \\ \forall j \in [1, M], a_j \leftarrow_R \mathbb{Z}_p : \\ \quad (\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathbf{KeyGen}(); (cn_j, \mathsf{ck}_j) \leftarrow \mathbf{Mint}(\mathsf{pk}, \mathsf{a}_j); \\ \quad (act_j, a\mathsf{sk}_j) \leftarrow \mathbf{AccountGen}(\mathsf{sk}_j, \mathsf{pk}_j, cn_j, \mathsf{ck}_j, \mathsf{a}_j); \\ \mathbb{A}_S \doteq \{act_j\}_{j\in[1,M]}, \mathbb{K}_S \doteq \{a\mathsf{sk}_j\}_{j\in[1,M]}; \\ (\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{O}) \leftarrow \mathcal{A}(pp, \mathbb{A}_S, \mathbb{K}_S); \\ (\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{C}k_{\mathsf{out}}) \leftarrow \mathbf{Spend}(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O}). \end{array} \end{array}\right] = 1.$$

We give the definition of some oracles and lists that will be used in the security models in Table 3 and 4.

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 section 3.2, we define two stronger models for anonymity: anonymity against recipient (who knows all the output amounts) and anonymity against ring insider (who knows some input account secret keys and their amounts).

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

Definition 1. *A RingCT protocol is called anonymous against recipient if for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *in the following experiment* Expr_{AR}*:*

$pp \leftarrow \boldsymbol{Setup}(1^\lambda)$;
$(\mathtt{m}, \{\mathsf{pk}_{k,i}\}_{k\in[1,M],i\in[1,n]}, \{\mathsf{pk}_{\mathsf{out},j}\}_{j\in[1,N]}) \leftarrow \mathcal{A}_1^{\mathcal{O}rc}(pp)$, *where* $(\mathsf{sk}_{k,i}, \mathsf{pk}_{k,i}) \in \mathcal{U}$ *and* $((\mathsf{pk}_{k,i}, \cdot), \cdot) \notin \mathcal{L}$;
for $k \in [1, M], i_k^* \leftarrow_R [1, n], \mathsf{a}_{k,i}, \mathsf{a}_{\mathsf{out},j} \leftarrow_R$ *Range, where* $\sum_k \mathsf{a}_{k,i_k^*} = \sum_j \mathsf{a}_{\mathsf{out},j}$.
$(cn_{k,i}, \boldsymbol{ck}_{k,i}) \leftarrow \mathbf{Mint}(\mathsf{pk}_{k,i}, \mathsf{a}_{k,i})$,
$(act_{k,i}, a\mathsf{sk}_{k,i}) \leftarrow \mathbf{AccountGen}(\mathsf{sk}_{k,i}, \mathsf{pk}_{k,i}, cn_{k,i}, \boldsymbol{ck}_{k,i}, \mathsf{a}_{k,i})$;
$\mathbb{A}_{\mathsf{in}} \doteq \{act_{k,i}\}, \mathbb{A}_S \doteq \{act_{k,i_k^*}\}, \mathbb{K}_S \doteq \{a\mathsf{sk}_{k,i_k^*}\}, \mathbb{O} \doteq \{\mathsf{pk}_{\mathsf{out},j}, \mathsf{a}_{\mathsf{out},j}\}$;
$(\mathbb{A}_{\mathsf{out}}^*, \pi^*, \mathbb{S}^*, \mathbb{C}k_{\mathsf{out}}^*) \leftarrow \boldsymbol{Spend}(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$;
$(k^*, \boldsymbol{ind}^*) \leftarrow \mathcal{A}_2^{\mathcal{O}rc}(pp, (\mathbb{A}_{\mathsf{out}}^*, \pi^*, \mathbb{S}^*, \mathbb{O}, \mathbb{C}k_{\mathsf{out}}^*, \mathbb{A}_{\mathsf{in}}))$,

it holds that: $\left|\Pr[\mathcal{A} \text{ wins in } \mathsf{Expr}_{AR}] - \frac{1}{n}\right| \leq \mathsf{negl}(\lambda)$. $\mathcal{A}$ *wins if* $\boldsymbol{ind}^* = i_k^*$ *and* $\mathbb{A}_{\mathsf{in}} \cap \mathcal{C} = \emptyset$.

Anonymity against ring insider. The anonymity against ring insider 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 uncorrupred accounts.

Definition 2. *A RingCT protocol is called anonymous against ring insider if for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *in the following experiment* Expr_{AI}*:*

$pp \leftarrow \boldsymbol{Setup}(1^\lambda)$;
$(\mathtt{m}, \{\mathsf{act}_{k,i}\}_{k\in[1,M],i\in[1,n]}, \{\mathsf{pk}_{\mathsf{out},j}\}_{j\in[1,N]}) \leftarrow \mathcal{A}_1^{\mathcal{O}rc}(pp)$, *where* $(\cdot, \mathsf{pk}_{\mathsf{out},j}) \in \mathcal{U}$, *and* $(\mathsf{act}_{k,i}, \mathsf{a}_{k,i}, \boldsymbol{ck}_{k,i}) \in \mathcal{G}$;
for $j \in [1, N], i_k^* \leftarrow_R [1, n], \mathsf{a}_{\mathsf{out},j} \leftarrow_R$ *Range, where* $\sum_k \mathsf{a}_{k,i_k^*} = \sum_j \mathsf{a}_{\mathsf{out},j}$.
$\mathbb{A}_{\mathsf{in}} \doteq \{act_{k,i}\}, \mathbb{A}_S \doteq \{act_{k,i_k^*}\}, \mathbb{K}_S \doteq \{a\mathsf{sk}_{k,i_k^*}\}, \mathbb{O} \doteq \{\mathsf{pk}_{\mathsf{out},j}, \mathsf{a}_{\mathsf{out},j}\}$;
$(\mathbb{A}_{\mathsf{out}}^*, \pi^*, \mathbb{S}^*, \mathbb{C}k_{\mathsf{out}}^*) \leftarrow \boldsymbol{Spend}(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$;
$(k^*, \boldsymbol{ind}^*, n^*) \leftarrow \mathcal{A}_2^{\mathcal{O}rc}(pp, (\mathbb{A}_{\mathsf{out}}^*, \pi^*, \mathbb{S}^*, \mathbb{A}_{\mathsf{in}}))$,

it holds that: $\left|\Pr[\mathcal{A} \text{ wins in } \mathsf{Expr}_{AI}] - \frac{1}{n-n^*}\right| \leq \mathsf{negl}(\lambda)$. $\mathcal{A}$ *wins if* $\boldsymbol{ind}^* = i_k^*$ *and there are* n^* *distinct values of* i *such that* $\mathsf{act}_{k^*,i} \in \mathcal{C}$.

List	Description
$\mathcal{U}$	The list of user key pairs generated by the challenger.
$\mathcal{G}$	The list of accounts, their balance and coin keys generated by Mint or Spend.
$\mathcal{I}$	The list of accounts with public keys generated by the challenger.
$\mathcal{L}$	The list of accounts and account secret keys with public keys generated by the challenger.
$\mathcal{T}$	The list of output generated by the Spend oracle with the corresponding input accounts.
$\mathcal{C}$	The list of corrupted accounts.

Table 3: Definition of Lists

Oracle	Description
$\mathtt{PkGen}(i, b, j)$	on input a query number i, a bit b and an index j, if $b = 0$, it runs algorithm $(\mathsf{ltsk}_i, \mathsf{ltpk}_i) \leftarrow$ **LongTermKeyGen**(), adds $(i, \mathsf{ltsk}_i, \mathsf{ltpk}_i)$ to an initially empty list $\mathcal{U}'$ and returns the long term public key pk_i. If $b = 1$, it retrieves $(j, \mathsf{ltsk}_j, \mathsf{ltpk}_j)$ from $\mathcal{U}'$ and runs $(\mathsf{pk}_i, R_{\mathsf{ot}}) \leftarrow$ **OneTimePKGen**(ltpk_j), $\mathsf{sk}_i \leftarrow$ **OneTimeSKGen**$(\mathsf{pk}_i, R_{\mathsf{ot}}, \mathsf{ltsk}_j)$. It adds $(\mathsf{sk}_i, \mathsf{pk}_i)$ to an initially empty list $\mathcal{U}$ and returns pk_i.
$\mathtt{ActGen}(\mathsf{pk}, \mathsf{a})$	on input a public key pk and an amount a, it runs algorithm $(cn, \mathsf{ck}) \leftarrow$ **Mint**$(\mathsf{pk}, \mathsf{a})$, and outputs (act, ck) for address pk. It adds $(act, \mathsf{a}, \mathsf{ck})$ to an initially empty list $\mathcal{G}$. If $(\mathsf{sk}, \mathsf{pk}) \in \mathcal{U}$, then the associated secret key with account act is $a\mathsf{sk} \doteq (\mathsf{sk}, ck, \mathsf{a})$. It adds act and $(act, a\mathsf{sk})$ to an initially empty list $\mathcal{I}$ and $\mathcal{L}$ respectively.
$\mathtt{Corrupt}(act)$	on input an account act, it retrieves $(act, a\mathsf{sk}) \in \mathcal{L}$. It adds act to an initially empty list $\mathcal{C}$, and finally returns $a\mathsf{sk}$.
$\mathtt{Spend}(\mathtt{m}, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$	takes in a string $\mathtt{m}$, input accounts $\mathbb{A}_{\mathsf{in}}$ containing $\mathbb{A}_S$ and the output set $\mathbb{O}$, it retrieves $(act_i, a\mathsf{sk}_i) \in \mathcal{L}$ for all $act_i \in \mathbb{A}_S$. Denote $\mathbb{K}_S$ as the set of these $a\mathsf{sk}_i$. It runs $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{C}k_{\mathsf{out}}) \leftarrow$ **Spend**$(\mathtt{m}, \mathbb{K}_S, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$ and returns $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{C}k_{\mathsf{out}})$ after adding $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{A}_S)$ to an initially empty list $\mathcal{T}$. For all $act_j = (\mathsf{pk}_j, cn_j) \in A_{\mathsf{out}}$, with a_j as the corresponding amount in $\mathbb{O}$ and ck_j as the corresponding coin key in $\mathbb{C}k_{\mathsf{out}}$, it adds $(act_j, \mathsf{a}_j, \mathsf{ck}_j)$ to the list $\mathcal{G}$. If $(\mathsf{sk}_j, \mathsf{pk}_j) \in \mathcal{U}$, it sets $a\mathsf{sk}_j \doteq (\mathsf{sk}_j, \mathsf{ck}_j, \mathsf{a}_j)$, where ck_j is the corresponding coin key in $\mathbb{C}k_{\mathsf{out}}$. It adds act_j to the list $\mathcal{I}$ and adds $(act_j, a\mathsf{sk}_j)$ to the list $\mathcal{L}$.

Table 4: Definition of Oracles $\mathtt{Orc}$

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.

The following definition of unforgeability captures the property that no PPT adversary can forge a signature where all input accounts are uncorrupted.

Definition 3. *A RingCT protocol is called unforgeable if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\Pr\left[\mathcal{A}\ \textit{Wins}: pp \leftarrow \mathit{Setup}(1^\lambda); (\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}) \leftarrow \mathcal{A}^{\mathcal{O}rc}(pp)\right] \leq \mathsf{negl}(\lambda),$$

where $\mathtt{Orc}$ denotes all oracles $\mathtt{PkGen}$, $\mathtt{ActGen}$, $\mathtt{Spend}$ and $\mathtt{Corrupt}$ defined in Table 4. Finally, $\mathcal{A}$ wins in the experiment if **Verify**$(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}) = 1$, $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \cdot) \notin \mathcal{T}$, *and for all $act_i \in \mathbb{A}_{\mathsf{in}}$, $act_i \in \mathcal{I} \setminus \mathcal{C}$.*

The following definition of equivalence captures the property that no PPT adversary can output a valid signature and coin keys where the sum of input amount is different from the sum of output amount. It holds even if these accounts are corrupted.

Definition 4. *A RingCT protocol is called equivalent w.r.t. insider corruption if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\Pr\left[\mathcal{A}\ \textit{Wins}: pp \leftarrow \mathit{Setup}(1^\lambda);\ (\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{C}k_{\mathsf{out}}, \mathbb{A}mt_{\mathsf{out}}) \leftarrow \mathcal{A}^{\mathcal{O}rc}(pp)\right] \leq \mathsf{negl}(\lambda),$$

where $\mathbb{A}mt_{\mathsf{out}}$ *is a set of output amount.* $\mathcal{A}$ *wins in the experiment if it satisfies the following conditions: (1)* $\textbf{\textit{Verify}}(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}) = 1$*, (2) for all coin key* $\mathsf{ck}_i \in \mathbb{C}k_{\mathsf{out}}$ *and amount* $\mathsf{a}_{\mathsf{out},i} \in \mathbb{A}mt_{\mathsf{out}}$*,* $\textbf{\textit{AccountGen}}(\mathsf{sk}_i, \mathsf{pk}_i, cn_i, \mathsf{ck}_i, \mathsf{a}_{\mathsf{out},i}) \neq \perp$*, (3) for all account* $act_j \in \mathbb{A}_{\mathsf{in}}$*,* $(act_j, \mathsf{a}_{\mathsf{in},j}) \in \mathcal{G}$*,* $\sum_j \mathsf{a}_{\mathsf{in},j} \neq \sum_i \mathsf{a}_{\mathsf{out},i}$*.*

The following definition of linkability captures the property that no PPT adversary can output two valid signatures with distinct serial numbers $\mathbb{S}_1$ and $\mathbb{S}_2$, where there is at most $|\mathbb{S}_1| + |\mathbb{S}_2| - 1$ input account that is corrupted or not generated by the challenger.

Definition 5. *A RingCT protocol is called linkable w.r.t. insider corruption if for all PPT adversaries* $\mathcal{A}$*, it holds that*

$$\Pr\left[\mathcal{A}\ \textit{Wins}: pp \leftarrow \textit{Setup}(1^\lambda);\ \{\mathtt{m}_i, \mathbb{A}_{\mathsf{in},i}, \mathbb{A}_{\mathsf{out},i}, \pi_i, \mathbb{S}_i\}_{i=1,2} \leftarrow \mathcal{A}^{\mathcal{O}rc}(pp)\right] \leq \mathsf{negl}(\lambda).$$

$\mathcal{A}$ *wins in the experiment if* $\textbf{\textit{Verify}}(\mathtt{m}_i, \mathbb{A}_{\mathsf{in}_i}, \mathbb{A}_{\mathsf{out},i}, \pi_i, \mathbb{S}_i) = 1$ *for* $i = 1, 2$*;* $\mathbb{S}_1 \cap \mathbb{S}_2 = \emptyset$*; and* $(\mathbb{A}_{\mathsf{in},1} \cup \mathbb{A}_{\mathsf{in},2}) \cap \mathcal{C} + (\mathbb{A}_{\mathsf{in},1} \cup \mathbb{A}_{\mathsf{in},2}) \setminus \mathcal{I} \leq |\mathbb{S}_1| + |\mathbb{S}_2| - 1$*.*

Non-Slanderability. The non-slanderabiliy property requires that a malicious user cannot slander any honest user after observing an honestly generated spending. It is infeasible for any malicious user to produce a valid spending that shares at least one serial number with a previously generated honest spending.

Definition 6. *A RingCT protocol is called non-slanderable if for all PPT adversaries* $\mathcal{A}$*, it holds that*

$$\Pr\left[\mathcal{A}\ \textit{Wins}: pp \leftarrow \textbf{\textit{Setup}}(1^\lambda); \left(\hat{\mathbb{S}}, \mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}\right) \leftarrow \mathcal{A}^{\mathcal{O}rc}(pp)\right] \leq \mathsf{negl}(\lambda).$$

$\mathcal{A}$ *wins if* $\hat{\mathbb{S}} \cap \mathbb{S}^* \neq \emptyset$*;* $\textbf{\textit{Verify}}(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}) = 1$*;* $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \cdot) \notin \mathcal{T}$*; and* $(\cdot, \cdot, \hat{\mathbb{S}}, \mathbb{A}_S) \in \mathcal{T}$ *and for all* $act \in \mathbb{A}_S$*,* $act \notin \mathbb{C}$*.*

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 [9], 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 [9].

Our basic construction is as follows.

Setup. On input security parameter 1^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 **KeyGen** 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 $\mathsf{pk} = g^{x_1} \cdot g^{H_6((g^{x_2})^r)}$. 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:

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

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. The sender generates a *single* ring signature for each all row k of $\mathbb{A}_{\mathsf{in}}$. 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

$$\begin{aligned}&\boldsymbol{Y_k} = ((\mathsf{pk}_{\mathsf{in},k}^{(1)})^{d_0^{k-1}}(C_{\mathsf{in},k}^{(1)})^{d_1} g_1^{d_2}, \ldots, (\mathsf{pk}_{\mathsf{in},k}^{(n)})^{d_0^{k-1}}(C_{\mathsf{in},k}^{(n)})^{d_1} g_n^{d_2}) \qquad \text{for } k \in [1, M],\\ &\boldsymbol{Y} = \boldsymbol{Y_1}||\ldots||\boldsymbol{Y_M}.\end{aligned}$$

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:

$$\begin{aligned}&B_1 = h^{\alpha_1} \cdot \prod_{k=1}^{M} (\mathsf{pk}_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_0^{k-1}} (C_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_1} g_{\mathsf{ind}_k}^{d_2} \qquad B_2 = h^{\alpha_2} \prod_{k=1}^{M} g_{\mathsf{ind}_k}, \qquad A = h^\beta \boldsymbol{h}^{\boldsymbol{b_R}},\\ &S_1 = h^{r_{\alpha_1} - d_2 r_{\alpha_2}} g^{\sum_{k=1}^{M} r_{\mathsf{sk}_k} d_0^{k-1}} g_c^{d_1 r_\Delta}, \qquad S_2 = h^\rho \boldsymbol{Y}^{\boldsymbol{s_L}} \boldsymbol{h}^{\boldsymbol{s_R}}, \qquad S_3 = \prod_{k=1}^{M} U_k^{r_{\mathsf{sk}_k} d_0^{k-1}}.\end{aligned}$$

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

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:

$$\begin{aligned}&l(X) = \boldsymbol{b_L} - z \cdot \boldsymbol{1}^{nM} + \boldsymbol{s_L} \cdot X,\\ &r(X) = \boldsymbol{y}^{nM} \circ (w \cdot \boldsymbol{b_R} + wz \cdot \boldsymbol{1}^{nM} + \boldsymbol{s_R} \cdot X) + \sum_{k=1}^{M} z^{1+k} \cdot (\boldsymbol{0}^{(k-1)n}||\boldsymbol{1}^n||\boldsymbol{0}^{(M-k)n}).\end{aligned}$$

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

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

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:

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

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 $(U_1, \ldots, U_M)$. Note that $\boldsymbol{l}, \boldsymbol{r}$ can be reduced to logarithm size.

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 0 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 $k \in [1, M]$ as follows.
   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:

$$t = \langle \boldsymbol{l}, \boldsymbol{r} \rangle, \tag{1}$$
$$g^t h^{\tau_x} = g^{\sum_{k=1}^{M} z^{1+k}(1-nz) + w(z-z^2)\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM} \rangle} \cdot T_1^x \cdot T_2^{x^2}, \tag{2}$$
$$h^\mu \boldsymbol{Y^l h'^r} = B_1 \cdot A^w \cdot S_2^x \cdot \boldsymbol{Y}^{-z \cdot \mathbf{1}^{nM}} \cdot \boldsymbol{h'}^{wz \cdot \boldsymbol{y}^{nM} + \sum_{k=1}^{M} z^{1+k} \cdot (\mathbf{0}^{(k-1)n} || \mathbf{1}^n || \mathbf{0}^{(M-k)n})}, \tag{3}$$
$$h^{z_{\alpha_1} - d_2 z_{\alpha_2}} g^{\sum_{k=1}^{M} z_{\mathsf{sk},k} d_0^{k-1}} g_c^{d_1 z_\Delta} = S_1 (B_1 \cdot \prod_{j=1}^{N} C_{\mathsf{out},j}^{d_1} \cdot B_2^{-d_2})^x, \tag{4}$$
$$\prod_{k=1}^{N} U_k^{z_{\mathsf{sk},k} d_0^{k-1}} = S_3 \cdot u^{x \sum_{k=1}^{N} d_0^{k-1}}. \tag{5}$$

5.2 Security

We give the security theorem of our construction. The security proofs are given in the appendix.

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 recipient 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 insider 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.*

5.3 Our Efficient Construction

The last step towards our final construction is to use the improved inner product argument in [9] 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 [9]. 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 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 [9], which is based on the DL assumption in the random oracle model.

6 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 $\geq$ 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%.

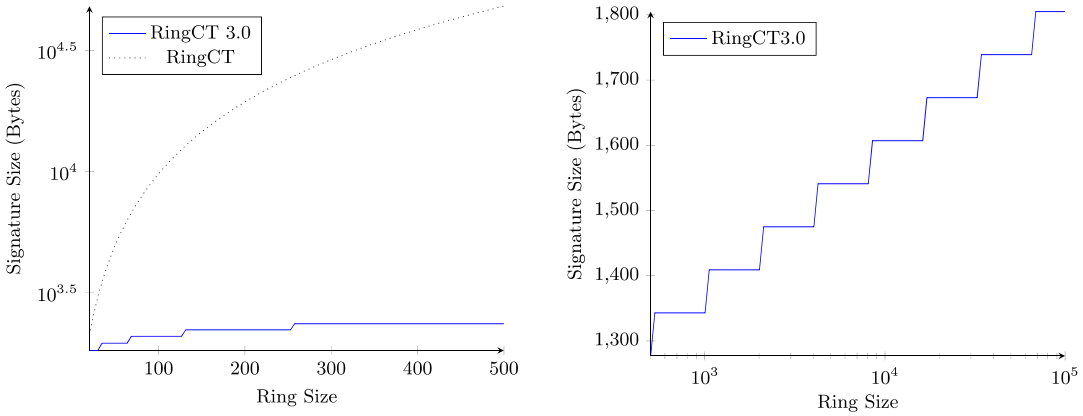
Consider a typical transaction (i.e., number of inputs $M = 2$) with a ring size of 1024, our ring signature size (1.3kB) is 98.6% less than the ring signature size of [32] (98kB). Monero has a relatively stable fee rate of about 0.0008 XMR/kB (which is about USD 0.2/kB). For the ring size of 1024, the 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 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.



(a) small ring size

(b) large ring size

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

7 Conclusion

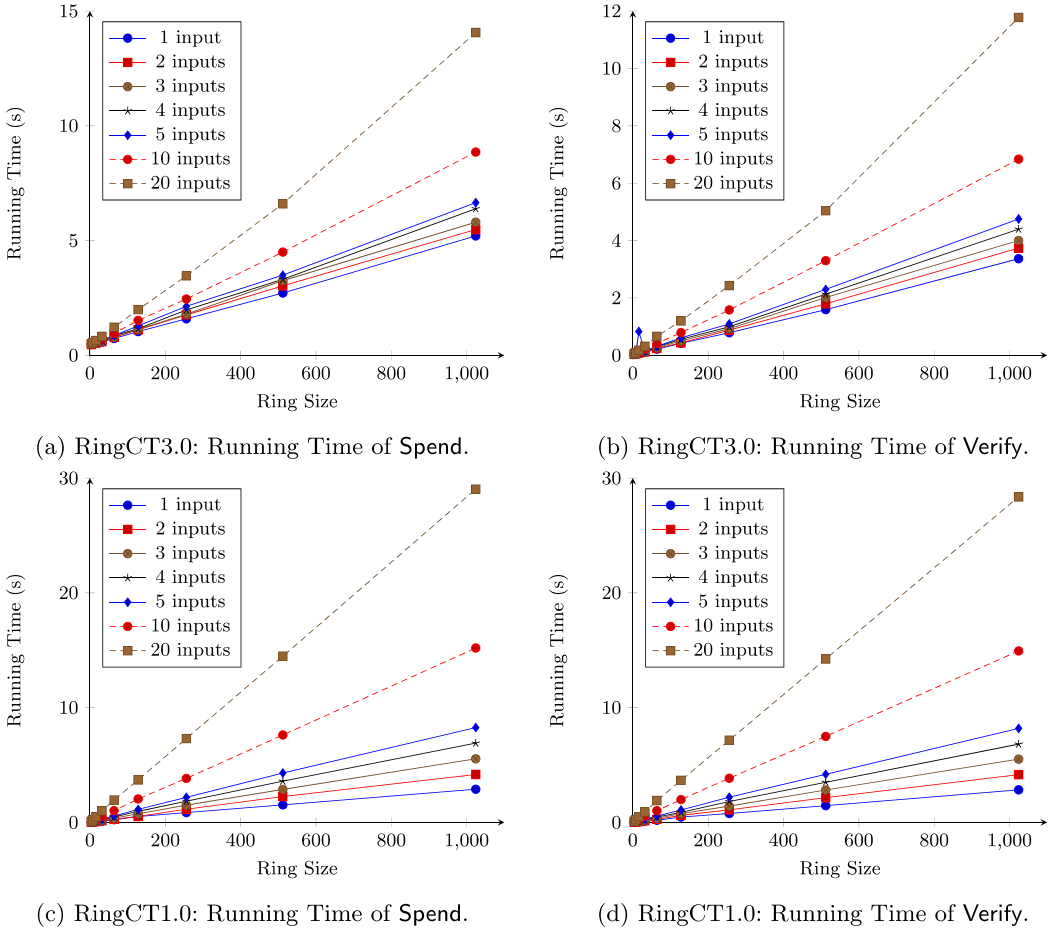
We propose the RingCT3.0 protocol, which is more efficient and more secure than the existing RingCT1.0 used in Monero. For a typical 2-input transaction with a ring size of 1024, the ring signature size of our RingCT3.0 protocol is 98% less than the ring signature size of the RingCT1.0 protocol.

Acknowledgement

We would like to thank Russell W.F. Lai for the discussion of the security proof in the previous version of this paper.

References

1. Au, M.H., Tsang, P.P., Susilo, W., Mu, Y.: Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 295–308. Springer (2009)
2. Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., Yakoubov, S.: Accumulators with applications to anonymity-preserving revocation. In: EuroS&P 2017. pp. 301–315. IEEE (2017)
3. Bayer, S., Groth, J.: Zero-knowledge argument for polynomial evaluation with application to blacklists. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 646–663. Springer (2013)
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), `https://eprint.iacr.org/2018/046`
5. 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)
6. Boneh, D., Corrigan-Gibbs, H.: Bivariate polynomials modulo composites and their applications. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 42–62. Springer (2014)
7. 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)
8. 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)



(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.

9. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE SP 2018. pp. 315–334. IEEE (2018)
10. Camacho, P., Hevia, A., Kiwi, M.A., Opazo, R.: Strong accumulators from collision-resistant hashing. Int. J. Inf. Sec. **11**(5), 349–363 (2012)
11. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer (2008)
12. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 481–500. Springer (2009)
13. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer (2002)
14. Chandran, N., Groth, J., Sahai, A.: Ring signatures of sub-linear size without random oracles. In: Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 423–434. Springer (2007)
15. Damgård, I., Triandopoulos, N.: Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538 (2008), `http://eprint.iacr.org/`
16. Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 127–144. Springer (2015)
17. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 609–626. Springer (2004)

18. Esgin, M.F., Steinfeld, R., Liu, J.K., Liu, D.: Lattice-based zero-knowledge proofs: New techniques for shorter and faster constructions and applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. Lecture Notes in Computer Science, vol. 11692, pp. 115–146. Springer (2019)
19. Esgin, M.F., Steinfeld, R., Sakzad, A., Liu, J.K., Liu, D.: Short lattice-based one-out-of-many proofs and applications to ring signatures. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. Lecture Notes in Computer Science, vol. 11464, pp. 67–88. Springer (2019)
20. Foley, S.N., Gollmann, D., Snekkenes, E. (eds.): ESORICS 2017, LNCS, vol. 10493. Springer (2017)
21. 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)
22. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A traceability analysis of monero's blockchain. In: Foley et al. [20], pp. 153–173
23. Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: Scaling up private payments without trusted setup - formal foundations and constructions of ring confidential transactions with log-size proofs. Cryptology ePrint Archive, Report 2019/580 (2019), `https://eprint.iacr.org/2019/580`
24. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 253–269. Springer (2007)
25. Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 1–31. Springer (2016)
26. Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: Laih, C. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer (2003)
27. 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)
28. Maxwell, G.: Confidential transactions (2015), `https://people.xiph.org/~greg/confidential_values.txt`
29. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: IEEE SP 2013. pp. 397–411. IEEE Computer Society (2013)
30. 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)
31. Nguyen, L.: Accumulators from Bilinear Pairings and Applications. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 275–292. Springer (2005)
32. Noether, S.: Ring Signature Confidential Transactions for Monero. Cryptology ePrint Archive, Report 2015/1098 (2015), `http://eprint.iacr.org/`
33. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE SP 2013. pp. 238–252. IEEE Computer Society (2013)
34. 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)
35. 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. [20], pp. 456–474
36. 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)
37. Zhang, H., Zhang, F., Tian, H., Au, M.H.: Anonymous post-quantum cryptocash. In: Meiklejohn, S., Sako, K. (eds.) FC 2018. Lecture Notes in Computer Science, vol. 10957, pp. 461–479. Springer (2018)

A Security Proofs

Theorem 4. *Our scheme is anonymous against recipient if the q-DDHI assumption holds in* $\mathbb{G}$ *in the ROM, where q is the number of Spend oracle query.*

Proof. Observe that the adversary is given $\mathbb{A}^*_{\mathsf{out}}, \pi^* = (\pi_{\mathsf{range}}, \sigma_{\mathsf{ring}}), \mathbb{S}^*, \mathbb{O}, \mathbb{C}k^*_{\mathsf{out}}$ and $\mathbb{A}_{\mathsf{in}}$. Note that $\mathbb{A}^*_{\mathsf{out}}, \mathbb{O}, \mathbb{C}k^*_{\mathsf{out}}$ only contain information about the output account (including the output amount). Therefore, we only have to check if $\pi^*, \mathbb{S}^*$ reveal the information of the input account or amount.

Suppose the simulator is given the DDHI problem instance $(g, g^a, \ldots, g^{a^q}, T)$ and wants to decide if $T = g^{1/a}$. The simulator picks some distinct random $\zeta, h^*, h_1, \ldots, h_q \in \mathbb{Z}_p$ and $u = g^{\zeta \cdot \prod_{i=1}^{q}(a-h^*+h_i)}$ as part of the system parameters.

For the PkGen query, the simulator picks some random ltsk and computes the corresponding ltpk and one-time pk. Except for one time, the simulator returns $\mathsf{ltpk}^* = (g^{a-h^*}, g^{x_2})$ for some random $x_2 \in \mathbb{Z}_p^*$. The simulator honestly runs the ActGen oracle, except that the $a\mathsf{sk}^*$ corresponding to ltpk^* is unknown. For the Corrupt oracle, the simulator declares failure and exits if the $a\mathsf{sk}^*$ is requested.

If the adversary queries the H_6 oracle, it either returns $\zeta, h^*, h_1, \ldots, h_q$, or a random element in $\mathbb{Z}_p^*$.

If the adversary queries the Spend oracle with input $(\mathtt{m}, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$ with with known ltsks correspond to some public keys in $\mathbb{A}_S$, then the simulator can generate valid signatures for that part. Otherwise $\mathsf{pk} = (g^{a-h^*} g^{\hat{h}})$ corresponds to a public key in $\mathbb{A}_S$, where $\hat{h}$ is the oracle output of H_6. If $\hat{h} \neq h_j$ for some j, the simulator declares failure and exits. If $\hat{h} = h_j$, then the simulator set $U = g^{\zeta \cdot \prod_{i=1, i\neq j}^{q} (a-h^*+h_i)}$. The simulator chooses most of the proof elements $(B_1, B_2, S_2, T_2, \tau_x, \mu, z_{\alpha_1}, z_{\alpha_2}, z_{\mathsf{sk},1}, \ldots, z_{\mathsf{sk},M}, z_\Delta, \boldsymbol{l}, \boldsymbol{r})$ and challenges (x, y, z, w) uniformly at random from their respective domains, and then computes t from equation (1), T_1 from equation (2), A from equation (3), S_1 from equation (4) and S_3 from equation (5). Then the simulator sets the value x, y, z, w in the random oracle H_4, H_5. Then the output for σ_{ring} is complete. The rest of the Spend oracle can be simulated easily.

In the challenge phase, the adversary gives $(\mathtt{m}, \{\mathsf{pk}_{k,i}\}_{k\in[1,M], i\in[1,n]}, \{\mathsf{pk}_{\mathsf{out},j}\}_{j\in[1,N]})$ to the simulator. If $\mathsf{pk}^* = g^{a-h^*} g^{h^*} = g^a$ does not corresponds to some $\mathsf{pk}_{k,i}$, the simulator declares failure and exits. Without loss of generality, assume $\mathsf{pk}_{k',i_k^*} = \mathsf{pk}^*$ for some k', i_k^*. According to the security model, the simulator retrieves $(\mathsf{sk}_{k,i}, \mathsf{pk}_{k,i}) \in \mathcal{U}$, picks $\mathsf{a}_{k,i}, \mathsf{a}_{\mathsf{out},j} \leftarrow_R \mathit{Range}$ where $\sum_k \mathsf{a}_{k,i_k^*} = \sum_j \mathsf{a}_{\mathsf{out},j}$. With the knowledge of all input amount, the simulator can honestly generate π_{range}. For the generation of the ring signature part, the generation of σ_{ring} is described as follows. The simulator sets $U_{k'} = g^{\zeta \cdot \prod_{i=1}^{q} (a-h^*+h_i)/a} := g^{\zeta \cdot \sum_{i=0}^{q-1} A_i a^i} T^{\zeta A_{-1}}$, for some constant $A_{q-1}, \ldots, A_0, A_{-1} \in \mathbb{Z}_p$. Note that $A_{-1} \neq 0$ since $h^* \neq h_i$ for all i. The values U_k can be honestly computed for $k \neq k'$. The rest of σ_{ring} is simulated as in the Spend oracle. According to the security model, the adversary is given $\mathbb{A}_{\mathsf{out}}^*, \pi^* = (\pi_{\mathsf{range}}, \sigma_{\mathsf{ring}}), \mathbb{S}^*, \mathbb{O}$, $\mathbb{C}k_{\mathsf{out}}^*$ and $\mathbb{A}_{\mathsf{in}}$, where $\mathbb{S}^*$ is the set of serial numbers, $\mathbb{O}$ is the set of output public keys and output amounts and $\mathbb{A}_{\mathsf{in}}$ is the set of input accounts.

Finally the adversary outputs (k^*, ind_k^*). With probability $1/M$, $k' = k^*$. If the adversary guesses $\mathsf{ind}_k^* = i_k^*$ correctly, then the simulator outputs $T = g^{1/a}$ as the solution to the DDHI problem. Therefore, our scheme is anonymous against recipient if the DDHI assumption holds. □

Theorem 5. *Our scheme is anonymous against ring insider if the q-DDHI assumption holds in* $\mathbb{G}$ *in the ROM and* RP *is a secure zero-knowledge range proof.*

Proof. Observe that the adversary is given $\mathbb{A}_{\mathsf{out}}^*, \pi^* = (\pi_{\mathsf{range}}, \sigma_{\mathsf{ring}})$, $\mathbb{S}^*$ and $\mathbb{A}_{\mathsf{in}}$. Note that $\mathbb{A}_{\mathsf{out}}^*$ only contain the public information about the output account. Therefore, we only have to check if $\pi^*, \mathbb{S}^*$ reveal the information of the input account or amount.

Suppose the simulator is given the DDHI problem instance $(g, g^a, \ldots, g^{a^q}, T)$ and wants to decide if $T = g^{1/a}$. The simulator picks some distinct random $\zeta, h^*, h_1, \ldots, h_q \in \mathbb{Z}_p$ and $u = g^{\zeta \cdot \prod_{i=1}^{q} (a-h^*+h_i)}$ as part of the system parameters.

For the PkGen query, the simulator picks some random ltsk and computes the corresponding ltpk and one-time pk. Except for one time, the simulator returns $\mathsf{ltpk}^* = (g^{a-h^*}, g^{x_2})$ for some random $x_2 \in \mathbb{Z}_p^*$. The simulator honestly runs the ActGen oracle, except that the $a\mathsf{sk}^*$ corresponding to ltpk^* is unknown. For the Corrupt oracle, the simulator declares failure and exits if the $a\mathsf{sk}^*$ is requested.

If the adversary queries the H_6 oracle, it either returns $\zeta, h^*, h_1, \ldots, h_q$, or a random element in $\mathbb{Z}_p^*$.

If the adversary queries the Spend oracle with input $(\mathtt{m}, \mathbb{A}_S, \mathbb{A}_{\mathsf{in}}, \mathbb{O})$ with with known ltsks correspond to some public keys in $\mathbb{A}_S$, then the simulator can generate valid signatures for that part. Otherwise $\mathsf{pk} = (g^{a-h^*} g^{\hat{h}})$ corresponds to a public key in $\mathbb{A}_S$, where $\hat{h}$ is the oracle output of H_6. If $\hat{h} \neq h_j$ for some j, the simulator declares failure and exits. If $\hat{h} = h_j$, then

the simulator set $U = g^{\zeta \cdot \prod_{i=1, i\neq j}^{q}(a-h^*+h_i)}$. The simulator chooses most of the proof elements $(B_1, B_2, S_2, T_2, \tau_x, \mu, z_{\alpha_1}, z_{\alpha_2}, z_{\mathsf{sk},1}, \ldots, z_{\mathsf{sk},M}, z_\Delta, \boldsymbol{l}, \boldsymbol{r})$ and challenges (x, y, z, w) uniformly at random from their respective domains, and then computes t from equation (1), T_1 from equation (2), A from equation (3), S_1 from equation (4) and S_3 from equation (5). Then the simulator sets the value x, y, z, w in the random oracle H_4, H_5. Then the output for σ_{ring} is complete. The rest of the Spend oracle can be simulated easily.

In the challenge phase, the adversary gives $(\mathtt{m}, \{\mathsf{act}_{k,i}\}_{k\in[1,M], i\in[1,n]}, \{\mathsf{pk}_{\mathsf{out},j}\}_{j\in[1,N]})$ to the simulator. If $\mathsf{pk}^* = g^{a-h^*}g^{h^*} = g^a$ does not corresponds to some $\mathsf{act}_{k,i}$, the simulator declares failure and exits. Without loss of generality, assume the corresponding $\mathsf{pk}_{k',i_k^*} = \mathsf{pk}^*$ for some k', i_k^*. According to the security model, the simulator retrieves $(\mathsf{act}_{k,i}, \mathsf{a}_{k,i}, \mathsf{ck}_{k,i}) \in \mathcal{G}$, picks $\mathsf{a}_{\mathsf{out},j} \leftarrow_R$ *Range* where $\sum_k \mathsf{a}_{k,i_k^*} = \sum_j \mathsf{a}_{\mathsf{out},j}$. With the knowledge of all input amount, the simulator can honestly generate π_{range}. For the generation of the ring signature part, the generation of σ_{ring} is described as follows. The simulator sets $U_{k'} = g^{\zeta \cdot \prod_{i=1}^{q}(a-h^*+h_i)/a} := g^{\zeta \cdot \sum_{i=0}^{q-1} A_i a^i} T^{\zeta A_{-1}}$, for some constant $A_{q-1}, \ldots, A_0, A_{-1} \in \mathbb{Z}_p$. Note that $A_{-1} \neq 0$ since $h^* \neq h_i$ for all i. The values U_k can be honestly computed for $k \neq k'$. The rest of σ_{ring} is simulated as in the Spend oracle. According to the security model, the adversary is given $\mathbb{A}^*_{\mathsf{out}}, \pi^* = (\pi_{\mathsf{range}}, \sigma_{\mathsf{ring}}), \mathbb{S}^*$ and $\mathbb{A}_{\mathsf{in}}$, where $\mathbb{S}^*$ is the set of serial numbers and $\mathbb{A}_{\mathsf{in}}$ is the set of input accounts. Recall that if RP is a secure zero-knowledge range proof, then π_{range} does not reveal the transaction input amount.

Finally the adversary outputs $(k^*, \mathsf{ind}_k^*, n^*)$. With probability $1/(n-n^*)$, $k' = k^*$. If the adversary guesses $\mathsf{ind}_k^* = i_k^*$ correctly, then the simulator outputs $T = g^{1/a}$ as the solution to the DDHI problem. Therefore, our scheme is anonymous against recipient if the DDHI assumption holds and RP is a secure zero-knowledge range proof. □

Before giving the formal proof of balance, we first give a lemma.

Lemma 1. *There is no non-trival discrete logarithm relation between each element in $\boldsymbol{Y}$ (formed by the Ring Formation step), $\boldsymbol{h}$ and h if the discrete logarithm assumption holds in $\mathbb{G}$.*

Proof. The part related to $\boldsymbol{h}$ and h is trivial since each element in $\boldsymbol{h}$ is a random $\mathbb{G}$ element and h is the output of a hash function $H_3(\boldsymbol{Y})$. Next we show that it is difficult to find distinct $\boldsymbol{l}, \boldsymbol{l'}$ such that $\boldsymbol{Y}^{\boldsymbol{l}} = \boldsymbol{Y}^{\boldsymbol{l'}}$.

Suppose the simulator is given a DL problem instance (g_0, X) and wants to compute $\log_{g_0} X$. The simulator picks some random $\zeta_i, \eta_1, \eta_2 \in \mathbb{Z}_p$ and sets $g = g_0$, $g_c = g^{\eta_1}, h_c = g^{\eta_2}$ and $g_i = X^{\zeta_i}$. Consider elements $Y_i \doteq g^{\mathsf{sk}_i} h_c^{\mathsf{a}_i d_1} g_c^{d_1 \kappa_i} g_i^{d_2}$ for all i.

Suppose $\boldsymbol{Y}^{\boldsymbol{l}} = \boldsymbol{Y}^{\boldsymbol{l'}}$. Denote $\boldsymbol{l} = (l_1, \ldots, l_n)$ and $\boldsymbol{l'} = (l'_1, \ldots, l'_n)$. Then we have

$$\begin{aligned}
\prod_{i=1}^{n} (g^{\mathsf{sk}_i} h_c^{\mathsf{a}_i d_1} g_c^{d_1 \kappa_i} g_i^{d_2})^{l_i} &= \prod_{i=1}^{n} (g^{\mathsf{sk}_i} h_c^{\mathsf{a}_i d_1} g_c^{d_1 \kappa_i} g_i^{d_2})^{l'_i} \\
g^{\sum_{i=1}^{n} \mathsf{sk}_i (l_i - l'_i)} h_c^{\sum_{i=1}^{n} d_1 \mathsf{a}_i (l_i - l'_i)} g_c^{\sum_{i=1}^{n} d_1 \kappa_i (l_i - l'_i)} &= \prod_{i=1}^{n} g_i^{d_2 (l'_i - l_i)} \\
g_0^{\sum_{i=1}^{n} (\mathsf{sk}_i + \eta_1 d_1 \kappa_i + \eta_2 d_1 \mathsf{a}_i)(l_i - l'_i)} &= X^{\sum_{i=1}^{n} \zeta_i d_2 (l'_i - l_i)}
\end{aligned}$$

Since $\boldsymbol{l}, \boldsymbol{l'}$ are distinct, $l'_i \neq l_i$ for some i. Then the simulator can answer the DL problem as $\frac{\sum_{i=1}^{n} (\mathsf{sk}_i + \eta_1 d_1 \kappa_i + \eta_2 d_1 \mathsf{a}_i)(l_i - l'_i)}{\sum_{i=1}^{n} \zeta_i d_2 (l'_i - l_i)}$. □

Theorem 6. *Our scheme is unforgeable if the DL assumption holds in $\mathbb{G}$ in the random oracle model.*

Proof. Suppose the simulator is given a discrete logarithm problem instance g, g^a. It picks random group elements for $\boldsymbol{h}$, u, h_c and g_c such that the discrete logarithm with respect to g is unknown. It picks a random $\beta \in \mathbb{Z}_p$ ad sets $u = g^{a\beta}$.

For the PkGen query, the simulator picks some random $\mathsf{sk} = \iota \in \mathbb{Z}_p$ and returns $\mathsf{pk} = g^\iota$. Except for one-time, the simulator returns $\mathsf{pk}^* = g^a$. The simulator honestly runs the ActGen oracle, except

that the $a\mathsf{sk}^*$ for input pk^* is unknown. For the $\mathsf{Corrupt}$ oracle, the simulator declares failure and exits if the $a\mathsf{sk}^*$ is requested. For the H_6 oracle, it returns a random $h \in \mathbb{G}$ (therefore the discrete logarithm with respect to g is unknown).

If the adversary queries the Spend oracle, the simulator runs as the **Spend** algorithm if pk^* is not the signing account's public key. If pk^* is the signing account's public key, the simulator can first honestly generate π_{range} using the knowledge of all input amount. It generates the set of output accounts $\mathbb{A}^*_{\mathsf{out}}$ and coin keys $\mathbb{C}k^*_{\mathsf{out}}$ using **Mint**. The generation of the ring signature part σ_{ring} is described as follows. The simulator sets $U = g^{\beta}$ for the key image corresponding to pk^*, and computes other U_k honestly. The simulator chooses most proof elements $(B_1, B_2, S_2, T_2, \tau_x, \mu, z_{\alpha_1}, z_{\alpha_2}, z_{\mathsf{sk},1}, \ldots, z_{\mathsf{sk},M}, z_{\Delta}, \boldsymbol{l}, \boldsymbol{r})$ and challenges (x, y, z, w) uniformly at random from their respective domains, and then computes t from equation (1), T_1 from equation (2), A from equation (3), S_1 from equation (4) and S_3 from equation (5). Then the simulator sets the value x, y, z, w in the random oracle H_4, H_5. Then the output for σ_{ring} is complete. Hence the simulator can answer the Spend oracle query.

Finally, the adversary $\mathcal{A}$ outputs $(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S})$ such that for all $act_i \in \mathbb{A}_{\mathsf{in}}$, $act_i \in \mathcal{I} \setminus \mathcal{C}$, and $(\mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \cdot) \notin \mathcal{T}$. If $\mathcal{A}$ wins, $\mathbf{Verify}(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}) = 1$. Denote the output $\pi = (\pi_{\mathsf{range}}, \sigma_{\mathsf{ring}})$. The simulator rewinds H_5 (in step challenge 2) for three times. For each transcript, denote the challenge as x_i and the responses as $(\tau_{x,i}, \mu_i, z_{\alpha,1,i}, z_{\alpha,2,i}, z_{\mathsf{sk},1,i}, \ldots, z_{\mathsf{sk},M,i}, z_{\Delta,i}, \boldsymbol{l}_i, \boldsymbol{r}_i, t_i)$ for $i \in [1, 3]$. Denote $\boldsymbol{l}_i = (l_{i,1}, \ldots, l_{i,n})$ and $\boldsymbol{r}_i = (r_{i,1}, \ldots, r_{i,n})$.

- To extract $B_1 A^w$, it picks some $\eta_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^2 \eta_i = 1$, $\sum_{i=1}^2 \eta_i x_i = 0$. By equation (3), we have:

$$\begin{aligned} B_1 A^w &= h^{\sum_{i=1}^2 \eta_i \mu_i} \boldsymbol{Y}^{\sum_{i=1}^2 \eta_i \boldsymbol{l}_i + z \cdot \boldsymbol{1}^{nM}} \boldsymbol{h'}^{\sum_{i=1}^2 \eta_i \boldsymbol{r}_i - wz \cdot \boldsymbol{y}^{nM} - \sum_{k=1}^M z^{1+k} \cdot (\boldsymbol{0}^{(k-1)n} || \boldsymbol{1}^n || \boldsymbol{0}^{(M-k)n})} \\ &:= h^{\gamma'} \boldsymbol{Y}^{\boldsymbol{b'_L}} \boldsymbol{h}^{w \cdot \boldsymbol{b'_R}}, \end{aligned} \tag{6}$$

  for some $\gamma', \boldsymbol{b'_L}, \boldsymbol{b'_R}$.
- To extract S_2, it picks some $\eta'_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^2 \eta'_i = 0, \sum_{i=1}^2 \eta'_i x_i = 1$. By equation (3), we have:

$$S_2 = h^{\sum_{i=1}^2 \eta'_i \mu_i} \boldsymbol{Y}^{\sum_{i=1}^2 \eta'_i \boldsymbol{l}_i} \cdot \boldsymbol{h'}^{\sum_{i=1}^2 \eta'_i \boldsymbol{r}_i} := h^{\rho'} \boldsymbol{Y}^{\boldsymbol{s'_L}} \boldsymbol{h}^{\cdot \boldsymbol{s'_R}},$$

  for some $\rho', \boldsymbol{s'_L}, \boldsymbol{s'_R}$.

Putting back the extracted values $B_1 A^w$ and S_2 into equation (3), we have:

$$h^{\mu} \boldsymbol{Y}^{\cdot \boldsymbol{l}} \boldsymbol{h'}^{\boldsymbol{r}} = (h^{\gamma'} \boldsymbol{Y}^{\boldsymbol{b'_L}} \boldsymbol{h}^{w \cdot \boldsymbol{b'_R}}) \cdot (h^{\rho'} \boldsymbol{Y}^{\boldsymbol{s'_L}} \boldsymbol{h}^{\boldsymbol{s'_R}})^x \cdot \boldsymbol{Y}^{-z \cdot \boldsymbol{1}^{nM}} \cdot \boldsymbol{h'}^{wz \cdot \boldsymbol{y}^{nM} + \sum_{k=1}^M z^{1+k} \cdot (\boldsymbol{0}^{(k-1)n} || \boldsymbol{1}^n || \boldsymbol{0}^{(M-k)n})}.$$

By lemma 1, there is no non-trival discrete logarithm relation between each element in $\boldsymbol{Y}$, $\boldsymbol{h'}$ and h if the discrete logarithm assumption holds in $\mathbb{G}$. Then we have:

$$\begin{aligned} \boldsymbol{l} &= \boldsymbol{b'_L} - z \cdot \boldsymbol{1}^{nM} + \boldsymbol{s'_L} \cdot x, \\ \boldsymbol{r} &= \boldsymbol{y}^{nM} \circ (w \cdot \boldsymbol{b'_R} + wz \cdot \boldsymbol{1}^{nM} + \boldsymbol{s'_R} \cdot x) + \sum_{k=1}^{M} z^{1+k} \cdot (\boldsymbol{0}^{(k-1)n} || \boldsymbol{1}^n || \boldsymbol{0}^{(M-k)n}). \end{aligned}$$

By the same set of 3 rewinding transcripts, we can also extract the commitments T_1, T_2 as follows.

- To extract T_1, it picks some $\delta_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^3 \delta_i = 0$, $\sum_{i=1}^3 \delta_i x_i = 1, \sum_{i=1}^3 \delta_i x_i^2 = 0$. By equation (2), we have:

$$T_1 = g^{\sum_{i=1}^3 \delta_i t_i} h^{\sum_{i=1}^3 \delta_i \tau_{x,i}} := g^{t'_1} h^{r'_1}$$

  for some t'_1, r'_1.
- To extract T_2, it picks some $\delta'_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^3 \delta'_i = 0$, $\sum_{i=1}^3 \delta'_i x_i = 0, \sum_{i=1}^3 \delta'_i x_i^2 = 1$. By equation (2), we have:

$$T_2 = g^{\sum_{i=1}^3 \delta'_i t_i} h^{\sum_{i=1}^3 \delta'_i \tau_{x,i}} := g^{t'_2} h^{r'_2}$$

  for some t'_2, r'_2.

Putting back the extracted values T_1 and T_2 into equation (2), we have:

$$g^t h^{\tau_x} = g^{\sum_{k=1}^{M} z^{1+k}(1-nz)+w(z-z^2)\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle} \cdot (g^{t'_1} h^{r'_1})^x \cdot (g^{t'_2} h^{r'_2})^{x^2}.$$

Since h is a random group element by the simulation of H_6, we have:

$$t = \sum_{k=1}^{M} z^{1+k}(1-nz) + w(z-z^2)\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle + t'_1 x + t'_2 x^2.$$

Denote $t'_0 = \sum_{k=1}^{M} z^{1+k}(1-nz) + w(z-z^2)\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle$.

By equation (1), we have $t = \langle \boldsymbol{l}, \boldsymbol{r}\rangle$. Observe that we already extracted $\boldsymbol{l}$, $\boldsymbol{r}$ as:

$$\begin{aligned}
\langle \boldsymbol{l}, \boldsymbol{r}\rangle &= (\boldsymbol{b'_L} - z\cdot \mathbf{1}^{nM} + \boldsymbol{s'_L}\cdot x)\cdot(\boldsymbol{y}^{nM} \circ (w\cdot \boldsymbol{b'_R} + wz\cdot \mathbf{1}^{nM} + \boldsymbol{s'_R}\cdot x) + \sum_{k=1}^{M} z^{1+k}\cdot(\mathbf{0}^{(k-1)n}||\mathbf{1}^n||\mathbf{0}^{(M-k)n}))\\
&= w\langle \boldsymbol{b'_L}, \boldsymbol{b'_R}\circ \boldsymbol{y}^{nM}\rangle + wz\langle \boldsymbol{b'_L} - \boldsymbol{b'_R}, \boldsymbol{y}^{nM}\rangle + \sum_{k=1}^{M} z^{1+k}\langle \boldsymbol{b'_L}, \mathbf{0}^{(k-1)n}||\mathbf{1}^n||\mathbf{0}^{(M-k)n}\rangle\\
&\qquad - wz^2\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle - \sum_{k=1}^{M} z^{2+k}\langle \mathbf{1}^{nM}, \mathbf{0}^{(k-1)n}||\mathbf{1}^n||\mathbf{0}^{(M-k)n}\rangle + t''_1 x + t''_2 x^2\\
&= w\langle \boldsymbol{b'_L}, \boldsymbol{b'_R}\circ \boldsymbol{y}^{nM}\rangle + wz\langle \boldsymbol{b'_L} - \boldsymbol{b'_R}, \boldsymbol{y}^{nM}\rangle - wz^2\langle \mathbf{1}^{nM}, \boldsymbol{y}^{nM}\rangle\\
&\qquad + \sum_{k=1}^{M} z^{1+k}\langle \boldsymbol{b'_L}, \mathbf{0}^{(k-1)n}||\mathbf{1}^n||\mathbf{0}^{(M-k)n}\rangle - \sum_{k=1}^{M} nz^{2+k} + t''_1 x + t''_2 x^2,
\end{aligned}$$

for some $t''_1, t''_2 \in \mathbb{Z}_p$. Since the above holds for all w, x, y, z, we have:

$$\boldsymbol{b'_L}\circ \boldsymbol{b'_R} = \mathbf{0}^{nM}, \quad \boldsymbol{b'_L} - \boldsymbol{b'_R} = \mathbf{1}^{nM}, \quad \boldsymbol{b'_L} := \boldsymbol{b'_{L,1}}||\ldots||\boldsymbol{b'_{L,M}}, \quad \langle \boldsymbol{b'_{L,k}}, \mathbf{1}^n\rangle = 1 \text{ for } k\in[1,M].$$

Therefore, it implies that $\boldsymbol{b'_{L,k}}$ is a binary vector with one bit equal to 1. Denote that bit as ind_k. Putting back $\boldsymbol{b'_{L,k}}$ in equation (6), we have:

$$B_1 A^w = h^{\gamma'} \cdot \prod_{k=1}^{M} (\mathsf{pk}_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_0^{k-1}} (C_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_1} g_{\mathsf{ind}_k}^{d_2} \cdot \boldsymbol{h}^{w\cdot \boldsymbol{b'_R}},$$

for some index ind_k. Since the above is true for all w, then we have $B_1 = h^{\alpha'} \prod_{k=1}^{M} (\mathsf{pk}_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_0^{k-1}} (C_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_1} g_{\mathsf{ind}_k}^{d_2}$ for some $\alpha' \in \mathbb{Z}_p$.

By the same set of rewinding transcripts, we can also extract from equation (4):

$$\begin{aligned}
B_1 \cdot \prod_{j=1}^{N} C_{\mathsf{out},j}^{d_1} \cdot B_2^{-d_2} &= h^{\frac{z_{\alpha_1,1}-z_{\alpha_1,2}}{x_1-x_2} - d_2\cdot\frac{z_{\alpha_2,1}-z_{\alpha_2,2}}{x_1-x_2}} g^{\frac{\sum_{k=1}^{M}(z_{\mathsf{sk},k,1}-z_{\mathsf{sk},k,2})d_0^{k-1}}{x_1-x_2}} g_c^{\frac{d_1(z_{\Delta,1}-z_{\Delta,2})}{x_1-x_2}} \cdot \prod_{j=1}^{N} C_{\mathsf{out},j}^{d_1} \cdot B_2^{-d_2}\\
&:= h^{\alpha''_1 + d_2\alpha''_2} \cdot \prod_{k=1}^{M} g^{\mathsf{sk}'_k d_0^{k-1}} \cdot g_c^{d_1\Delta'} \cdot \prod_{j=1}^{N} C_{\mathsf{out},j}^{d_1} \cdot B_2^{-d_2} \qquad (7)
\end{aligned}$$

Since the above is true for all h, d_0, d_1 and d_2, then we have $\mathsf{pk}_{\mathsf{in},k}^{(\mathsf{ind}_k)} = g^{\mathsf{sk}'_k}$. Hence if $g^{\mathsf{sk}'_k} = \mathsf{pk}_*$, then the simulator returns sk'_k as the solution to the DL problem. It happens with probability $1/n$.

Theorem 7. *Our scheme is equivalent w.r.t. insider corruption if the DL assumption holds in* $\mathbb{G}$ *in the random oracle model and* RP *is a secure zero-knowledge range proof.*

Proof. Suppose the simulator is given a discrete logarithm problem instance g, g^a. It picks random group elements for $\boldsymbol{h}$ u, and h_c such that the discrete logarithm with respect to g is unknown. It sets $g_c = g^a$.

For the AddGen query, the simulator picks some random $\mathsf{sk} = \iota \in \mathbb{Z}_p$ and returns $\mathsf{pk} = g^\iota$. The simulator honestly runs the ActGen oracle, the $\mathsf{Corrupt}$ oracle and the Spend oracle.

In the challenge phase, the adversary outputs $(\mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S}, \mathbb{C}k_{\mathsf{out}}, \mathbb{A}mt_{\mathsf{out}})$. If $\mathcal{A}$ wins, then for all coin key $\kappa_i \in \mathbb{C}k_{\mathsf{out}}$, amount $\mathsf{a}_{\mathsf{out},i} \in \mathbb{A}mt_{\mathsf{out}}$ and output coins $C_{\mathsf{out},i}$ in $\mathbb{A}_{\mathsf{out}}$,

$$C_{\mathsf{out},i} = g_c^{\kappa_i} h_c^{\mathsf{a}_{\mathsf{out},i}}.$$

Denote the output $\pi = (\pi^*_{\mathsf{range}}, \sigma_{\mathsf{ring}})$. Recall from the proof of unfogeability that we have $B_1 = h^{\alpha'} \prod_{k=1}^{M} (\mathsf{pk}_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_0^{k-1}} (C_{\mathsf{in},k}^{(\mathsf{ind}_k)})^{d_1} g_{\mathsf{ind}_k}^{d_2}$. Together with equation (7), we have

$$\frac{\prod_{k=1}^{M} C_{\mathsf{in},k}^{(\mathsf{ind}_k)}}{\prod_{j=1}^{N} C_{\mathsf{out},j}} = g_c^{\Delta'}, \tag{8}$$

by the random choice of d_1.

Recall that if $\mathcal{A}$ wins, then for all account $act_j \in \mathbb{A}_{\mathsf{in}}$, $(act_j, \mathsf{a}_{\mathsf{in},j}, \kappa_{\mathsf{in},k}) \in \mathcal{G}$, It implies that $C_{\mathsf{in},k}^{(\mathsf{ind}_k)} = h_c^{\mathsf{a}_{\mathsf{in},k}} g_c^{\kappa_{\mathsf{in},k}}$ and $\mathsf{a}_{\mathsf{in},k}$ is within a valid range. Similarly, the range proof π^*_{range} of the challenge signature shows that $C_{\mathsf{out},j} = h_c^{\mathsf{a}_{\mathsf{out},j}} g_c^{\kappa_{\mathsf{out},j}}$ and $\mathsf{a}_{\mathsf{out},j}$ is within a valid range. Therefore, we have:

$$\prod_{k=1}^{M} h_c^{\mathsf{a}_{\mathsf{in},k}} g_c^{\kappa_{\mathsf{in},k}} = \prod_{j=1}^{N} h_c^{\mathsf{a}_{\mathsf{out},j}} g_c^{\kappa_{\mathsf{out},j}} \cdot g_c^{\Delta'}.$$

If $Z \doteq \sum_{j=1}^{N} \mathsf{a}_{\mathsf{out},j} - \sum_{k=1}^{M} \mathsf{a}_{\mathsf{in},k} \neq 0$, then:

$$h_c^Z = g_c^{\sum_{k=1}^{M} \kappa_{\mathsf{in},k} + \Delta' - \sum_{j=1}^{N} \kappa_{\mathsf{out},j}}.$$

Then the simulator returns $\frac{\sum_{k=1}^{M} \kappa_{\mathsf{in},k} + \Delta' - \sum_{j=1}^{N} \kappa_{\mathsf{out},j}}{Z}$ as the solution to the DL problem.

Theorem 8. *Our scheme is linkable w.r.t. insider corruption if the DL assumption holds in* $\mathbb{G}$ *in the random oracle model.*

Proof. Suppose the simulator is given a DL problem instance g, g^a. It picks random elements in $\mathbb{G}$ for $\boldsymbol{h}$, u, h_c and g_c.

For the PkGen query, the simulator picks some random $\mathsf{sk} = \iota \in \mathbb{Z}_p$ and returns $\mathsf{pk} = g^\iota$. Except for one-time, the simulator returns $\mathsf{pk}^* = g^a$. The simulator honestly runs the ActGen oracle, except that the $a\mathsf{sk}^*$ for input pk^* is unknown. For the $\mathsf{Corrupt}$ oracle, the simulator declares failure and exits if the $a\mathsf{sk}^*$ is requested. If the adversary queries the Spend oracle, the simulator runs as in the proof of unforgeability.

Finally, the adversary $\mathcal{A}$ outputs $(\mathtt{m}_i, \mathbb{A}_{\mathsf{in},i}, \mathbb{A}_{\mathsf{out},i}, \pi_i, \mathbb{S}_i)$ for $i = 1, 2$, such that all U in $\mathbb{S}_1$ and $\mathbb{S}_2$ are distinct. From the proof of unforgeability, the simulator can rewinds x and from equation (7):

$$\mathsf{sk}'_k = \frac{z_{\mathsf{sk},k,1} - z_{\mathsf{sk},k,2}}{x_1 - x_2}$$

for the random choice of d_0. From equation (5), we also have:

$$\prod_{k=1}^{M} U_k^{(z_{\mathsf{sk},k,1} - z_{\mathsf{sk},k,2}) d_0^{k-1}} = u^{(x_1 - x_2) \sum_{k=1}^{N} d_0^{k-1}}.$$

Since the above holds for all d_0, we have $U_k = u^{\frac{1}{\mathsf{sk}'_k}}$ and $g^{\mathsf{sk}'_k}$ refers to one public key in $\mathbb{A}_{\mathsf{in},1} \cup \mathbb{A}_{\mathsf{in},2}$. There are $|\mathbb{S}_1| + |\mathbb{S}_2|$ distinct values of U.

If $\mathcal{A}$ wins, then $(\mathbb{A}_{\mathsf{in},1} \cup \mathbb{A}_{\mathsf{in},2}) \cap \mathcal{C} + (\mathbb{A}_{\mathsf{in},1} \cup \mathbb{A}_{\mathsf{in},2}) \setminus \mathcal{I} \leq |\mathbb{S}_1| + |\mathbb{S}_2| - 1$. It means that there exists at least one U corresponding to one public key $g^{\mathsf{sk}'}$ with account $act^* \doteq (g^{\mathsf{sk}'}, \cdot) \in \mathcal{I} \setminus \mathcal{C}$. With probability at least $\frac{1}{q_a+q_s-q_c}$, $g^{\mathsf{sk}'} = g^a$, where q_a, q_s, q_c is the number of oracle queries to the ActGen, Spend, $\mathsf{Corrupt}$ oracles respectively. Then the simulator returns sk' as the solution to the DL problem.

Theorem 9. *Our scheme is non-slanderable w.r.t. insider corruption if the DL assumption holds in $\mathbb{G}$ in the random oracle model.*

Proof. Suppose the simulator is given a DL problem instance g, g^a. It picks random elements in $\mathbb{G}$ for $\boldsymbol{h}$, u, h_c and g_c.

For the PkGen query, the simulator picks some random $\mathsf{sk} = \iota \in \mathbb{Z}_p$ and returns $\mathsf{pk} = g^\iota$. Except for one-time, the simulator returns $\mathsf{pk}^* = g^a$. The simulator honestly runs the ActGen oracle, except that the $a\mathsf{sk}^*$ for input pk^* is unknown. For the $\mathsf{Corrupt}$ oracle, the simulator declares failure and exits if the $a\mathsf{sk}^*$ is requested. If the adversary queries the Spend oracle, the simulator runs as in the proof of unforgeability.

Finally, the adversary $\mathcal{A}$ outputs $(\hat{\mathbb{S}}, \mathtt{m}, \mathbb{A}_{\mathsf{in}}, \mathbb{A}_{\mathsf{out}}, \pi, \mathbb{S})$, where $(\cdot, \cdot, \hat{\mathbb{S}}, \mathbb{A}_S) \in \mathcal{T}$.

If $\mathsf{pk}^* \notin \mathbb{A}_S$, the simulator declares failure and exits (it happens with probability $(1 - \frac{1}{q_p})^{|\mathbb{A}_S|}$, where q_p is the number of PkGen oracle queries.). By the winning condition, there exists some $U \in \hat{\mathbb{S}} \cap \mathbb{S}^*$. Following the proof of linkability, the simulator can extract sk' such that $U = u^{\frac{1}{\mathsf{sk}'}}$ and $g^{\mathsf{sk}'} \in \mathbb{A}_S$. With probability $\frac{1}{|\mathbb{A}_S|}$, $g^{\mathsf{sk}'} = \mathsf{pk}^*$. Then the simulator return sk' as the solution to the DL problem.

B Set Membership Proof without Trusted Setup

We first review the definition of set membership proof in [11] and then we give our new construction without using trusted setup.

Definition 7. *[11] Let $C = (Gen, Com, Open)$ be the generation, the commit and the open algorithm of a commitment scheme. For an instance c, a proof of set membership with respect to commitment scheme C and set Φ is a proof of knowledge for the following statement:*

$$PK\{(\mu, \rho) : c \leftarrow Com(\mu; \rho) \wedge \mu \in \Phi\}.$$

The security model for set membership proof follows the standard definitions of zero-knowledge proof: perfect completeness, computational soundness and perfect zero-knowledge.

In this section, we consider the following modified set membership proof for a set Φ of base group elements :

$$PK\{(\mu, \rho) : c = g^\mu h^\rho \wedge g^\mu \in \Phi\}.$$

Related Works. Generally speaking, accumulator with the property of zero-knowledge can be viewed as a special case of set membership proof. As summarized in [16,2], it includes the RSA-based [13,24,2], pairing-based [31,15,1,12,16], Merkle-tree-based [6] and lattice-based [25] constructions. However, all existing RSA-based and pairing-based accumulators, and [6] require a trusted setup. The scheme in [6] also requires a trusted setup for generating composite order group. Furthermore, the pairing-based approach can only accumulate values in the exponent. It means that if we want to accumulate public keys, eventually we have to prove the knowledge of a secret key as a proof of double discrete logarithm, which is inefficient. For the Merkle-tree-based accumulator [10], it can be combined with zk-SNARK [33] to achieve zero-knowledge. Nevertheless, zk-SNARK also needs trusted setup. Recently, zk-STARK [4] is proposed to remove the need of trusted setup. However, zk-STARK proofs is 1000 times longer than zk-SNARK proofs and hence it is not practical. For the lattice-based accumulator without trusted setup [25], the asymptotic proof size is $O(\log(n))$, but the actual size is impractical (> 30MB).

B.1 Our Basic Construction

Our construction is essentially a set membership proof for group elements which is the domain of public keys. It is the first set membership proof for public keys in the base group, instead of in the exponent. The intuition of our scheme is introduced in the previous section. Our construction is as follows.

- **Setup.** On input security parameter 1^λ and the maximum size of the set of membership public key N, it picks a group $\mathbb{G}$ of prime order p and some generators $g \in \mathbb{G}, \boldsymbol{h} = (h_1, \ldots, h_N) \in \mathbb{G}^N$. Suppose that $H_j : \{0,1\}^* \to \mathbb{Z}_p$ for $j = 1, 2, 3, 4$, $H_6 : \{0,1\}^* \to \mathbb{G}$ are collision resistant hash functions. Let $C = (Gen, Com, Open)$ be the Pedersen commitment scheme. Assume these parameters are known in the system.
- **PKGen.** It randomly picks $x \in \mathbb{Z}_p$ and outputs a public key $Y = g^x$.
- **Prove.** On input the set of $n \leq N$ public keys as $\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_n)$ and denote the set member $\sigma = Y_{i^*} \in \boldsymbol{Y}$, with corresponding secret key x_{sk,i^*}. The prover runs as follows.
  1. *Prepare Index.* The prover generates a binary vector $\boldsymbol{b_L} = (b_1, \ldots, b_n)$, where $b_i = 1$ when $i = i^*$ and $b_i = 0$ otherwise. Define $\boldsymbol{b_R} = \boldsymbol{b_L} - \boldsymbol{1^n}$. It proves in zero knowledge that $\boldsymbol{b_L}$ is a binary vector with only one bit equal to 1. It is equivalent to showing:
  $$\boldsymbol{b_L} \circ \boldsymbol{b_R} = \boldsymbol{0}^n, \quad \boldsymbol{b_L} - \boldsymbol{b_R} = \boldsymbol{1}^n, \quad \langle \boldsymbol{b_L}, \boldsymbol{1}^n \rangle = 1.$$
  2. *Commit 1.* It computes $h = H_6(\boldsymbol{Y})$. It picks random $\alpha, \beta, \rho, r_\alpha, r_{sk} \in \mathbb{Z}_p$, $\boldsymbol{s_L}, \boldsymbol{s_R} \in \mathbb{Z}_p^n$ and computes:
  $$A_1 = h^\alpha \boldsymbol{Y}^{\boldsymbol{b_L}} = h^\alpha Y_{i^*}, \qquad A_2 = h^\beta \boldsymbol{h}^{\boldsymbol{b_R}}, \qquad S_1 = h^{r_\alpha} g^{r_{sk}}, \qquad S_2 = h^\rho \boldsymbol{Y}^{\boldsymbol{s_L}} \boldsymbol{h}^{\boldsymbol{s_R}}.$$
  Note that A_1 is the Pedersen commitment of the secret key of Y_{i^*} for randomness α.
  3. *Challenge 1.* Denote the concatenated string $\mathsf{str} = \boldsymbol{Y}||A_1||A_2||\ S_1||S_2$. It computes $y = H_2(\mathsf{str})$, $z = H_3(\mathsf{str})$ and $w = H_4(\mathsf{str})$.
  4. *Commit 2.* It can construct two degree 1 polynomials of variable X:
  $$\begin{aligned} l(X) &= \boldsymbol{b_L} - z \cdot \boldsymbol{1}^n + \boldsymbol{s_L} \cdot X, \\ r(X) &= \boldsymbol{y}^n \circ (w \cdot \boldsymbol{b_R} + wz \cdot \boldsymbol{1}^n + \boldsymbol{s_R} \cdot X) + z^2 \cdot \boldsymbol{1}^n. \end{aligned}$$
  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
  $$\begin{aligned} t_0 &= w\langle \boldsymbol{b_L}, \boldsymbol{b_R} \circ \boldsymbol{y}^n \rangle + zw\langle \boldsymbol{b_L} - \boldsymbol{b_R}, \boldsymbol{y}^n \rangle + z^2 \langle \boldsymbol{b_L}, \boldsymbol{1}^n \rangle - wz^2 \langle \boldsymbol{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \boldsymbol{1}^n, \boldsymbol{1}^n \rangle, \\ &= z^2 + w(z - z^2)\langle \boldsymbol{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \boldsymbol{1}^n, \boldsymbol{1}^n \rangle. \end{aligned}$$
  It picks random $\tau_1, \tau_2 \in \mathbb{Z}_p$, and computes:
  $$T_1 = g^{t_1} h^{\tau_1}, \quad T_2 = g^{t_2} h^{\tau_2}.$$
  5. *Challenge 2.* It computes $x = H_1(w, y, z, T_1, T_2)$.
  6. *Response.* It computes:
  $$\begin{aligned} \tau_x &= \tau_1 \cdot x + \tau_2 \cdot x^2, \\ \mu &= \alpha + \beta \cdot w + \rho \cdot x, \\ z_\alpha &= r_\alpha + \alpha \cdot x, \\ z_{sk} &= r_{sk} + x_{sk,i^*} \cdot x, \\ \boldsymbol{l} &= l(x) = \boldsymbol{b_L} - z \cdot \boldsymbol{1}^n + \boldsymbol{s_L} \cdot x, \\ \boldsymbol{r} &= r(x) = \boldsymbol{y}^n \circ (w \cdot \boldsymbol{b_R} + wz \cdot \boldsymbol{1}^n + \boldsymbol{s_R} \cdot x) + z^2 \cdot \boldsymbol{1}^n, \\ t &= \langle \boldsymbol{l}, \boldsymbol{r} \rangle. \end{aligned}$$

It outputs A_1 and $\sigma = (A_2, S_1, S_2, T_1, T_2, \tau_x, \mu, z_\alpha, z_{sk}, \boldsymbol{l}, \boldsymbol{r}, t)$.

– **Verify.** On input a set of public keys $\boldsymbol{Y}$, A_1 and the proof $\sigma = (A_2, S_1, S_2, T_1, T_2, \tau_x, \mu, z_\alpha, z_{sk}, \boldsymbol{l}, \boldsymbol{r}, t)$, denote the concatenated string $\mathsf{str} = \boldsymbol{Y}||A_1||A_2||S_1||S_2$. It computes $h = H_6(\boldsymbol{Y})$, $y = H_2(\mathsf{str})$, $z = H_3(\mathsf{str})$, $w = H_4(\mathsf{str})$ and $x = H_1(w, y, z, T_1, T_2)$. Define $\boldsymbol{h'} = (h'_1, \ldots, h'_n) \in \mathbb{G}^n$ such that $h'_i = h_i^{y^{-i+1}}$ for $i \in [1, n]$. It checks if all of the following hold:

$$t = \langle \boldsymbol{l}, \boldsymbol{r} \rangle, \tag{9}$$

$$g^t h^{\tau_x} = g^{z^2 + w(z-z^2)\langle \mathbf{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \mathbf{1}^n, \mathbf{1}^n \rangle} \cdot T_1^x \cdot T_2^{x^2}, \tag{10}$$

$$h^\mu \boldsymbol{Y}^{\boldsymbol{l}} \boldsymbol{h'}^{\boldsymbol{r}} = A_1 \cdot A_2^w \cdot S_2^x \cdot \boldsymbol{Y}^{-z \cdot \mathbf{1}^n} \cdot \boldsymbol{h'}^{wz \cdot \boldsymbol{y}^n + z^2 \cdot \mathbf{1}^n}, \tag{11}$$

$$h^{z_\alpha} g^{z_{sk}} = S_1 A_1^x. \tag{12}$$

Theorem 10. *The set membership proof is secure if the discrete logarithm assumption holds in $\mathbb{G}$ in the random oracle model.*

Proof. **Completeness.** The completeness of the protocol is straightforward.

Zero-knowledge. The simulator chooses most of the proof elements $(A_1, S_2, T_2, T_3, \tau_x, \mu, z_\alpha, z_{sk}, \boldsymbol{l}, \boldsymbol{r})$ and challenges (x, y, z, w) uniformly at random from their respective domains, and then computes t from equation (9), T_1 from equation (10), A_2 from equation (11) and S_1 from equation (12). Then the simulator sets the value x, y, z, w in the random oracle H_1, H_2, H_3, H_4. Hence it achieves the zero-knowledge property .

Soundness. Suppose the adversary returns a proof σ for a set of public keys $\boldsymbol{Y^*} = \{Y_1, \ldots, Y_n\}$, the extractor rewinds challenge 2 for 3 times. For each transcript, denote the challenge as x_i and the responses as $(\tau_{x,i}, \mu_i, z_{\alpha,i}, z_{sk,i}, \boldsymbol{l}_i, \boldsymbol{r}_i, t_i)$ for $i \in [1, 3]$. Denote $\boldsymbol{l}_i = (l_{i,1}, \ldots, l_{i,n})$ and $\boldsymbol{r}_i = (r_{i,1}, \ldots, r_{i,n})$.

– To extract $A_1 A_2^w$, it picks some $\eta_i \in \mathbb{Z}_p$ such that: $\sum_{i=1}^2 \eta_i = 1, \sum_{i=1}^2 \eta_i x_i = 0$. By equation (11), we have:

$$\begin{aligned} A_1 A_2^w &= h^{\sum_{i=1}^2 \eta_i \mu_i} \boldsymbol{Y}^{\sum_{i=1}^2 \eta_i \cdot \boldsymbol{l}_i + z \cdot \mathbf{1}^n} \boldsymbol{h'}^{\sum_{i=1}^2 \eta_i \boldsymbol{r}_i - wz \cdot \boldsymbol{y}^n - z^2 \cdot \mathbf{1}^n} \\ &:= h^{\alpha'} \boldsymbol{Y}^{\boldsymbol{b'_L}} \boldsymbol{h}^{w \cdot \boldsymbol{b'_R}}, \end{aligned} \tag{13}$$

for some $\alpha', \boldsymbol{b'_L}, \boldsymbol{b'_R}$.

– To extract S_2, it picks some $\eta'_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^2 \eta'_i = 0, \sum_{i=1}^2 \eta'_i x_i = 1$. By equation (11), we have:

$$S_2 = h^{\sum_{i=1}^2 \eta'_i \mu_i} \boldsymbol{Y}^{\sum_{i=1}^2 \eta'_i \boldsymbol{l}_i} \cdot \boldsymbol{h'}^{\sum_{i=1}^2 \eta'_i \boldsymbol{r}_i} := h^{\rho'} \boldsymbol{Y}^{\boldsymbol{s'_L}} \boldsymbol{h}^{\cdot \boldsymbol{s'_R}},$$

for some $\rho', \boldsymbol{s'_L}, \boldsymbol{s'_R}$.

Putting back the extracted values into equation (11), we have:

$$h^\mu \boldsymbol{Y}^{\boldsymbol{l}} \boldsymbol{h'}^{\boldsymbol{r}} = (h^{\alpha'} \boldsymbol{Y}^{\boldsymbol{b'_L}} \boldsymbol{h}^{w \cdot \boldsymbol{b'_R}}) \cdot (h^{\rho'} \boldsymbol{Y}^{\boldsymbol{s'_L}} \boldsymbol{h}^{\boldsymbol{s'_R}})^x \cdot \boldsymbol{Y}^{-z \cdot \mathbf{1}^n} \cdot \boldsymbol{h'}^{wz \cdot \boldsymbol{y}^n + z^2 \cdot \mathbf{1}^n}.$$

Since the above holds for all x, and assuming the DL assumption holds between h, g and $\boldsymbol{h}$, we have

$$\begin{aligned} \boldsymbol{l} &= \boldsymbol{b'_L} - z \cdot \mathbf{1}^n + \boldsymbol{s'_L} \cdot x, \\ \boldsymbol{r} &= \boldsymbol{y}^n \circ (w \cdot \boldsymbol{b'_R} + wz \cdot \mathbf{1}^n + \boldsymbol{s'_R} \cdot x) + z^2 \cdot \mathbf{1}^n. \end{aligned}$$

By the same set of 3 rewinding transcripts, we can also extract some other commitments as follows.

– To extract T_1, it picks some $\delta_i \in \mathbb{Z}_p$ such that $\sum_{i=1}^3 \delta_i = 0, \sum_{i=1}^3 \delta_i x_i = 1, \sum_{i=1}^3 \delta_i x_i^2 = 0$. By equation (10), we have:

$$T_1 = g^{\sum_{i=1}^3 \delta_i t_i} h^{\sum_{i=1}^3 \delta_i \tau_{x,i}} := g^{t'_1} h^{r'_1},$$

for some t'_1, r'_1.

– To extract T_2, it picks some $\delta'_i \in \mathbb{Z}_p$ such that: $\sum_{i=1}^3 \delta'_i = 0, \sum_{i=1}^3 \delta'_i x_i = 0, \sum_{i=1}^3 \delta'_i x_i^2 = 1$. By equation (10), we have:

$$T_2 = g^{\sum_{i=1}^3 \delta'_i t_i} h^{\sum_{i=1}^3 \delta'_i \tau_{x,i}} := g^{t'_2} h^{r'_2},$$

for some t'_2, r'_2.

Putting back the extracted values into equation (10), we have:

$$g^t h^{\tau_x} = g^{z^2 + w(z-z^2)\langle \mathbf{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \mathbf{1}^n, \mathbf{1}^n \rangle} \cdot (g^{t'_1} h^{r'_1})^x \cdot (g^{t'_2} h^{r'_2})^{x^2}.$$

By the random oracle model, the DL between $h = H_6(\boldsymbol{Y})$ and g is not known by the adversary. So we have:

$$t = z^2 + w(z-z^2)\langle \mathbf{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \mathbf{1}^n, \mathbf{1}^n \rangle + t'_1 x + t'_2 x^2.$$

By equation (9), we have $t = \langle \boldsymbol{l}, \boldsymbol{r} \rangle$. Observe that:

$$\begin{aligned} \langle \boldsymbol{l}, \boldsymbol{r} \rangle &= (\boldsymbol{b'_L} - z \cdot \mathbf{1}^n + \boldsymbol{s'_L} \cdot x)(\boldsymbol{y}^n \circ (w \cdot \boldsymbol{b'_R} + wz \cdot \mathbf{1}^n + \boldsymbol{s'_R} \cdot x) + z^2 \cdot \mathbf{1}^n) \\ &= w\langle \boldsymbol{b'_L}, \boldsymbol{b'_R} \circ \boldsymbol{y}^n \rangle + wz\langle \boldsymbol{b'_L} - \boldsymbol{b'_R}, \boldsymbol{y}^n \rangle + z^2 \langle \boldsymbol{b'_L}, \mathbf{1}^n \rangle - wz^2 \langle \mathbf{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \mathbf{1}^n, \mathbf{1}^n \rangle + t''_1 x + t''_2 x^2, \end{aligned}$$

for some $t''_1, t''_2 \in \mathbb{Z}_p$. Since the above holds for all w, x, y, z, we have:

$$\boldsymbol{b'_L} \circ \boldsymbol{b'_R} = \mathbf{0}^n, \quad \boldsymbol{b'_L} - \boldsymbol{b'_R} = \mathbf{1}^n, \quad \langle \boldsymbol{b'_L}, \mathbf{1}^n \rangle = 1.$$

Therefore, it implies that $\boldsymbol{b'_L}$ is a binary vector with one bit equal to 1. By equation (11), we have $A_1 A_2^w = h^{\alpha'} Y_{i^*} \boldsymbol{h}^{w \cdot \boldsymbol{b'_R}} = h^{\alpha'} g^{x_{sk,i^*}} \boldsymbol{h}^{w \cdot \boldsymbol{b'_R}}$ for some index i^*. Since the above is true for all w, then we have $A_1 = h^{\alpha'} g^{x_{sk,i^*}}$.

By the same set of rewinding transcripts, from equation 12, we also have:

$$A_1 = h^{\frac{z_{\alpha,1} - z_{\alpha,2}}{x_1 - x_2}} g^{\frac{z_{sk,1} - z_{sk,2}}{x_1 - x_2}} := h^{\alpha'} g^{sk'}.$$

By the random oracle model, the DL between h and g is not known by the adversary. So we can extract $x_{sk,i^*} = sk'$, where A_1 is a commitment to $Y_{i^*} = g^{x_{sk,i^*}}$. It completes the soundness proof. □

B.2 Set Membership Proof with Logarithm Size

Our scheme in the last section is linear size of n for the part of $\boldsymbol{l}$ and $\boldsymbol{r}$. Observe that the verifier can compute $A_1 \cdot A_2^w \cdot S_2^x \cdot \boldsymbol{Y}^{-z \cdot \mathbf{1}^n} \cdot \boldsymbol{h'}^{wz \cdot \boldsymbol{y}^n + z^2 \cdot \mathbf{1}^n}$. We note that verifying both equations (9) and (11) is equivalent to verifying the witness $\boldsymbol{l}$ and $\boldsymbol{r}$ satisfying the inner-product relation. Therefore, it can be fitted into the improved inner-product argument framework from [9] to give a zero knowledge proof π of $\boldsymbol{l}, \boldsymbol{r}$ such that:

$$P = \boldsymbol{Y'^l h'^r} \quad \wedge \quad t = \langle \boldsymbol{l}, \boldsymbol{r} \rangle.$$

The size of π is $2 \cdot \lceil \log_2(n) \rceil$ elements in $\mathbb{G}$ and 2 elements in $\mathbb{Z}_p$. The signer's work is dominated by $\log n + 1$ multi-exponentiations in $\mathbb{G}$ of size $2n, n, n/2, \ldots, 1$ respectively. The verifier's work is dominated by a single multi-exponentiations in $\mathbb{G}$ of size $2n + 2\log_2 n + 1$.

To sum up, the set membership proof output is $\sigma = (A_1, A_2, S_1, S_2, T_1, T_2, \tau_x, \mu, z_\alpha, z_{sk}, t, \pi)$, which has size $2 \cdot \lceil \log_2(n) \rceil + 6$ elements in $\mathbb{G}$ and 7 elements in $\mathbb{Z}_p$. The signer's work is dominated by three multi-exponentiations in $\mathbb{G}$ of size $2n + 1$, $2n$ and $n + 1$ respectively. The verifier's work is dominated by two multi-exponentiations in $\mathbb{G}$ of size $2n + 2\log_2 n + 1$ and $n + 4$ respectively.

We implemented our scheme in Intel Core i5 3.1GHz, 8Gb RAM, MacOS 10.13.4. We used the BouncyCastle's 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. The results are shown in Figure 3. It can be seen that the proof and verification time is almost linear to the size of the set, which is consistent to our analysis.

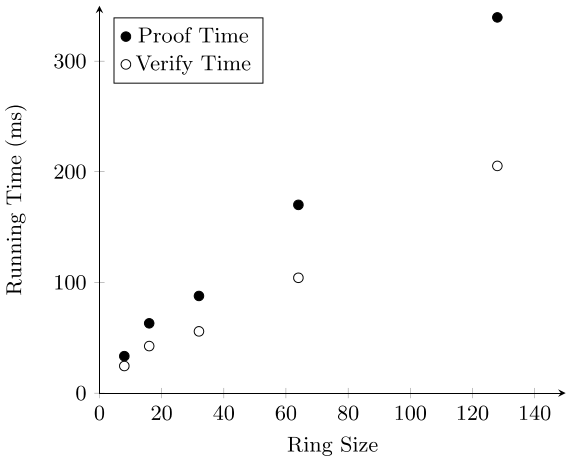


Fig. 3: Efficiency of Our Set Membership Proof.

Membership Proof	Communication		Prover (# $\mathbb{G}$ exponentiation)	Verifier (# $\mathbb{G}$ exponentiation)
	$\mathbb{G}$	$\mathbb{Z}_p$		
[3]	$4\log n + 5$	$3\log n + 4$	$(4\log n + 4)\mathrm{e}_2$	$(2\log n + 1)\mathrm{e}_3 + \mathrm{e}_{\log n+3} + \mathrm{e}_2$
[21]	$4\ \log n$	$3\ \log n\ + 1$	$\frac{5}{2}\log n \cdot \mathrm{e}_2 + \frac{3}{2}\log n \cdot \mathrm{e}_1$	$\mathrm{e}_{\log n+3} + n(\mathrm{e}_3 + \mathrm{e}_2)$
[7]	$\log n\ + 12$	$\frac{3}{2}\log n + 6$	$\frac{1}{2}\log n \cdot \mathrm{e}_{n+1} + 2\mathrm{e}_{2\log n+1}$ $+(\frac{1}{2}\log n + 3)\mathrm{e}_2 + 4\mathrm{e}_1$	$2\mathrm{e}_{n+\frac{1}{2}\log n+1}$ $+2\mathrm{e}_{2\log n+2} + 4\mathrm{e}_2$
[8] Scheme 1	7	$4\ \log n\ + 4$	$5\mathrm{e}_{\log n+1}$ $+(\log n + 2)\mathrm{e}_2$	$\mathrm{e}_{3\log n+1} + \mathrm{e}_{\log n+2}$ $+\mathrm{e}_{\log n+1} + \mathrm{e}_2$
[8] Scheme 2	$2.7\ \sqrt{\log n}\ + 5$	$1.9\ \log n\ +$ $2.7\ \sqrt{\log n}\ + 4$	$5\mathrm{e}_{3.2\log n+1}$ $+2.7\log n \cdot \mathrm{e}_2$	$\mathrm{e}_{9.5\log n+1} + \mathrm{e}_{3.2\log n+1}$ $+\mathrm{e}_{1.6\log n+2} + \mathrm{e}_2$
This paper	$2\ \log n + 6$	7	$\mathrm{e}_{2n+1} + \mathrm{e}_{n+1} + 3\mathrm{e}_2$ $+(\mathrm{e}_{2n} + \mathrm{e}_n + \ldots + \mathrm{e}_1)$	$\mathrm{e}_{2n+2\log n+4}$ $+\mathrm{e}_4 + \mathrm{e}_3$

Table 5: Summary of set membership proofs for n members with $O(\log n)$ size. Denote e_i as the multi-exponentiation of length i.

B.3 Comparison

As introduced in the previous section (§3.1), our set membership proof is fundamentally different from the existing approaches [21,7,8]. A comparison with the state-of-the-art $\log n$-size set membership proofs is shown Table 5. Our scheme is the most efficient in terms of communication size for ECC group, where $|\mathbb{G}| \approx |\mathbb{Z}_p|$.

B.4 Extensions for Range Proof

We can construct set membership proof for arbitrary set of integers, if we modify our scheme by using $\mathbb{G}$ as a group of composite order $N = pq$. By considering the set as $(x_1, x_2, \ldots, x_n)$ and we use $Y_i = g^{x_i}$ in the above construction.

As compared to the range proof in [9], it only allows the proof for a continuous range from zero to a maximum value. It is because the range proof is essentially performed in a bitwise manner. In our scheme, we can perform proof of arbitrary disjoint range, e.g. proving $x \in [2, 5] \cup [8, 10] \cup [12, 14]$. It may be useful in some privacy-preserving smart contract scenario. For example, it can be used to show that a transaction is performed between Monday to Friday in January 2018. It is equivalent to prove that the date variable $x \in [1, 5] \cup [8, 12] \cup [19, 23] \cup [26, 30]$.

There exists range proof in composite order group, with proof size independent of the size of the range [26]. To prove a number $x \in [A, B]$, Lipmaa [26] used the Lagrange theorem for showing $x - A \geq 0$ and $B - x \geq 0$. If we want to use [26] to prove set membership for M disjoint ranges,

then the prover needs to have M disjunction of the Lipmaa's proof. On the other hand, the proof size of our scheme is $O(\log N)$, where N is the total size of the disjoint ranges. Therefore, our scheme is more efficient for multiple disjoint ranges.

B.5 Application to Zerocoin

Zerocoin [29] was the first anonymous cryptocurrency proposal which supports large anonymity sets. The original Zerocoin protocol is used by many cryptocurrencies (Zcoin, PIXV, SmartCash, Zoin, and HexxCoin) with a combined market capitalization of about USD 660 million at the time of writing. Each zerocoin is a commitment C to a so-called serial number S. When a zerocoin is spent, the spender reveals the serial number S and proves in zero-knowledge of C that she knows C is a commitment to the serial number S and C belongs to a large set of minted zerocoins $\mathcal{C}$.

[29] used RSA-based accumulator for zero-knowledge proof of C. Therefore, it suffers from the drawback of having trusted setup. We propose the use of our set membership proof of C for the set of minted zerocoins $\mathcal{C}$. Our set membership proof can be easily modified to fit the Zerocoin protocol, by replacing the RSA-based accumulator. The only modification needed is changing the proof of knowledge of secret key, to the proof of knowledge of randomness used in the zerocoin commitment C.

C Efficient Ring Signatures without Trusted Setup

In this section, we give an efficient ring signature without trusted setup, whose signature size is logarithmic in the size of the ring.

Backgrounds. The classical ring signatures [34] for a set of n public keys are constructed by computing n "pseudo-signatures" and only one real secret key (out of all n secret keys) is used to sign. Many ring signatures use disjunctive proofs to demonstrate possession of one-out-of-n secret keys. However, they have signature size of $O(n)$, with the exception of [14] having $O(\sqrt{n})$ size.

Ring signatures can also be constructed by cryptographic accumulator [17]. Accumulator allows the signer to "compress" n public keys into a constant size *value* and there is a *witness* showing that the signer public key is in the set of public keys. Then, a ring signature can be constructed by a zero-knowledge proof of knowing (1) the *witness* and the signer public key that correspond to the accumulated *value*, and (2) the secret key that corresponds to the signer public key. The advantage of this approach is constant signature size. However, the existing RSA-based and pairing-based accumulators both require a trusted setup for generating system parameters. It is not desirable for system without mutually trusted party.[5] The accumulator without trusted setup by lattice [25] is impractical.

Some new ring signature schemes are proposed using zero-knowledge argument for low degree polynomials. In [21], the signer's index b can be expressed as a binary string $(b_1, \dots b_m)$, where $m = \log n$. For $i \in [1, m]$, each secret b_i is hidden in a polynomial p_i of degree m. The zero-knowledge proof is the evaluation of these polynomials using the challenge value as input. [7] used u-ary representation of b instead of binary. These ring signatures have size of $O(\log(n))$.

The logarithmic ring signature from lattice-based accumulator [25], one-out-of-many proof [19] and one-shot zero-knowledge proof [18] are still at least 100 times longer than our discrete logarithm-based construction. For a ring size of 1024, the signature size of [25] is 59.1MB for $\lambda = 100$ and the signature size of [19] is 1.021MB for $\lambda = 133$. For a ring size of 4096, the signature size of [18] is 103KB for $\lambda = 128$. They are still far less efficient than the DL-based construction.

[5] Some may suggest the use of multi-party computation for generating such system parameters. However, we have to ensure that all participated parties did not collude and they performed secure erasure. In practice, it is not a simple task. Zcash performed some complicated procedures (`https://z.cash/technology/paramgen.html`) to convince the general public that the system parameters are properly generated.

Our Ring Signature by Set Membership Proof. 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 which is in the set of n public keys, and (2) the secret key which corresponds to the committed public key. Compared to the accumulator-based two-step approach, we avoid the need to generate *witness* and accumulated *value* first and then to compute zero-knowledge proof on them. Theoretically, our proposal is a more direct approach of constructing ring signatures. In practice, we can also avoid the use of trusted setup for existing RSA-based and pairing-based accumulators. The major obstacle of our approach is how to construct an efficient set membership proof of "a committed public key is in the set of n public keys".

We give a concrete instantiation of using set membership proof for constructing ring signatures. We use the set membership proof together with the zero-knowledge proof of the secret key corresponding to the committed public key. As a result, we obtain an efficient ring signature without trusted setup, whose signature size is logarithmic to the size of the ring.

Note that this approach is not efficient with the existing set membership proofs [11,21,8]. The non-interactive version of [11] does not allow dynamic set formation. The set membership proof of [21,8] is for the set of integers in the exponents. If the public keys are considered as the set, then a further zero-knowledge proof of knowledge of the corresponding secret key requires an inefficient zero-knowledge proof of double discrete logarithms. Our new set membership proof enables this new paradigm of ring signatures. [6]

Compared with existing accumulator-based ring signatures, the RSA-based or pairing-based constructions require trusted setup. It is not desirable for distributed systems without trusted authority. Our construction does not require such trusted setup. The lattice-based construction is not practical [37]. A number of ring signatures are proposed recently using zero-knowledge argument for polynomial evaluation [21,7]. They both achieve signature size of $O(\log n)$. Table 2 shows the comparison with these efficient ring signatures, except [37] due to its large impractical size. Our construction is the most efficient in terms of signature size (we consider ECC where $|\mathbb{G}| \approx |\mathbb{Z}_p|$) and verifier's computation. In terms of prover efficiency, our scheme is the fastest for larger ring size ($n > 128$).

C.1 Definitions

A ring signature scheme consists of some PPT algorithms (**Setup**, **KeyGen**, **Sign**, **Verify**) for generating a system parameter available to all users, generating keys for users, signing messages and verifying ring signatures. We follow the standard definition of ring signatures algorithms and its security model (correctness, unforgeability and perfect anonymity) in [21].

C.2 Ring Signature with Logarithm Size

- **Setup.** On input security parameter 1^λ and the maximum size of ring N, it picks a group $\mathbb{G}$ of prime order p and some generators $g \in \mathbb{G}, \boldsymbol{h} = (h_1, \ldots, h_N) \in \mathbb{G}^N$. Suppose that $H_j : \{0,1\}^* \to \mathbb{Z}_p$ for $j = 4, 5$ and $H_3 : \{0,1\}^* \to \mathbb{G}$ are collision resistant hash functions. Assume these parameters are known in the system.
- **KeyGen.** For each user, he picks a random $sk \in \mathbb{Z}_p$ as his secret key and computes his public key as $Y = g^{sk}$.
- **Sign.** On input the set of $n \leq N$ public keys as $\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_n)$, the signer index $i^* \in [1, n]$, the signer secret key sk^* and a message $\mathtt{m}$, the signer runs as follows:
  1. *Prepare Signer Index.* The sender generates a binary vector $\boldsymbol{b_L} = (b_1, \ldots, b_n)$, where $b_i = 1$ when $i = i^*$ and $b_i = 0$ otherwise. Define $\boldsymbol{b_R} = \boldsymbol{b_L} - \mathbf{1^n}$. We will prove in zero knowledge that $\boldsymbol{b_L}$ is a binary vector with only one bit equal to 1. It is equivalent to showing:

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

[6] [7,21] use the property that "DL-based public key is equal to Pedersen commitment of zero" and then the ring signature includes the one-out-of-many proofs that one Pedersen commitment is committed to zero. Our scheme does not involve the commitment to zero.

2. *Signature Generation.* It consists of the following steps.

   Commit 1. It sets $h = H_3(\boldsymbol{Y})$, picks random $\alpha, \beta, \rho, r_\alpha, r_{\mathsf{sk}}, \in \mathbb{Z}_p$, $\boldsymbol{s_L}, \boldsymbol{s_R} \in \mathbb{Z}_p^n$ and computes:

$$B = h^\alpha \boldsymbol{Y}^{\boldsymbol{b_L}} = h^\alpha Y_{i^*}, \qquad A = h^\beta \boldsymbol{h}^{\boldsymbol{b_R}}, \qquad S_1 = h^{r_\alpha} g^{r_{\mathsf{sk}}}, \qquad S_2 = h^\rho \boldsymbol{Y}^{\boldsymbol{s_L}} \boldsymbol{h}^{\boldsymbol{s_R}}.$$

   Challenge 1. Denote the concatenated string $\mathsf{str}' = \boldsymbol{Y}||B||A||\ S_1||S_2||U$. 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:

$$\begin{aligned} l(X) &= \boldsymbol{b_L} - z \cdot \boldsymbol{1}^n + \boldsymbol{s_L} \cdot X, \\ r(X) &= \boldsymbol{y}^n \circ (w \cdot \boldsymbol{b_R} + wz \cdot \boldsymbol{1}^n + \boldsymbol{s_R} \cdot X) + z^2 \cdot \boldsymbol{1}^n. \end{aligned}$$

   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

$$\begin{aligned} t_0 &= w\langle \boldsymbol{b_L}, \boldsymbol{b_R} \circ \boldsymbol{y}^n \rangle + zw\langle \boldsymbol{b_L} - \boldsymbol{b_R}, \boldsymbol{y}^n \rangle + z^2 \langle \boldsymbol{b_L}, \boldsymbol{1}^n \rangle - wz^2 \langle \boldsymbol{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \boldsymbol{1}^n, \boldsymbol{1}^n \rangle, \\ &= z^2 + w(z - z^2)\langle \boldsymbol{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \boldsymbol{1}^n, \boldsymbol{1}^n \rangle. \end{aligned}$$

   It picks random $\tau_1, \tau_2 \in \mathbb{Z}_p$, and computes:

$$T_1 = g^{t_1} h^{\tau_1}, \quad T_2 = g^{t_2} h^{\tau_2}.$$

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

   Response. It computes:

$$\begin{aligned} \tau_x &= \tau_1 \cdot x + \tau_2 \cdot x^2, \\ \mu &= \alpha + \beta \cdot w + \rho \cdot x, \\ z_\alpha &= r_\alpha + \alpha \cdot x, \\ z_{\mathsf{sk}} &= r_{\mathsf{sk}} + \mathsf{sk} \cdot x, \\ \boldsymbol{l} &= l(x) = \boldsymbol{b_L} - z \cdot \boldsymbol{1}^n + \boldsymbol{s_L} \cdot x, \\ \boldsymbol{r} &= r(x) = \boldsymbol{y}^n \circ (w \cdot \boldsymbol{b_R} + wz \cdot \boldsymbol{1}^n + \boldsymbol{s_R} \cdot x) + z^2 \cdot \boldsymbol{1}^n, \\ t &= \langle \boldsymbol{l}, \boldsymbol{r} \rangle. \end{aligned}$$

   It outputs $\sigma_{\mathsf{ring}} = (B, A, S_1, S_2, T_1, T_2, \tau_x, \mu, z_\alpha, z_{\mathsf{sk}}, z_\delta, \boldsymbol{l}, \boldsymbol{r}, t)$.
   Note that the two vectors $\boldsymbol{l}, \boldsymbol{r}$ with length n can be reduced to $\pi \doteq (P, \boldsymbol{L}, \boldsymbol{R}, a, b)$ (which consists of $2\log(n) + 1$ elements in $\mathbb{G}$ and 2 elements in $\mathbb{Z}_p$) by the inner product argument.

– **Verify.** On input a set of public keys $\boldsymbol{Y}$, the signature σ, the message $\mathtt{m}$, denote the concatenated string $\mathsf{str} = \boldsymbol{Y}||B||A||S_1||S_2||U$. 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 $x = H_5(v, w, y, z, T_1, T_2, \mathtt{m})$. Define $\boldsymbol{h'} = (h'_1, \ldots, h'_n) \in \mathbb{G}^n$ such that $h'_i = h_i^{y^{-i+1}}$ for $i \in [1, n]$. It checks if all of the following hold:

$$\begin{aligned} t &= \langle \boldsymbol{l}, \boldsymbol{r} \rangle, \\ g^t h^{\tau_x} &= g^{z^2 + w(z-z^2)\langle \boldsymbol{1}^n, \boldsymbol{y}^n \rangle - z^3 \langle \boldsymbol{1}^n, \boldsymbol{1}^n \rangle} \cdot T_1^x \cdot T_2^{x^2}, \\ h^\mu \boldsymbol{Y}^{\boldsymbol{l}} \boldsymbol{h'}^{\boldsymbol{r}} &= B \cdot A^w \cdot S_2^x \cdot \boldsymbol{Y}^{-z \cdot \boldsymbol{1}^n} \cdot \boldsymbol{h'}^{wz \cdot \boldsymbol{y}^n + z^2 \cdot \boldsymbol{1}^n}, \\ h^{z_\alpha} g^{z_{\mathsf{sk}}} &= S_1 B^x. \end{aligned}$$

   It returns 1 if all of the above hold and returns 0 otherwise.
   If the signature is compressed by the inner product argument, it also runs the verification for it.

Security. The security for unforgeability and anonymity can be reduced to the discrete logarithm assumption in the random oracle model. We do not repeat the security proof here as it is very similar to the proof of the security of RingCT3.0 and the set membership proof.

Efficiency. To sum up, the ring signature output is $\sigma = (B, A, S_1, S_2, T_1, T_2, \tau_x, \mu, z_\alpha, z_{\mathsf{sk}}, t, \pi)$, which has size $2 \cdot \lceil \log_2(n) \rceil + 7$ elements in $\mathbb{G}$ and 7 elements in $\mathbb{Z}_p$. The signer's work is dominated by three multi-exponentiations in $\mathbb{G}$ of size $2n+1$, $2n$ and $n+1$, respectively. The verifier's work is dominated by two multi-exponentiations in $\mathbb{G}$ of size $2n + 2\log_2 n + 1$ and $n+4$, respectively.

D Comparison with Omniring

Recently, a parallel and independent work on RingCT, called *Omniring*, is proposed [23]. They also used the inner product argument from the Bulletproof 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 [23] 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 insider and recipient. We illustrate the differences with a simple example. Consider the Omniring [23] 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 1
$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.

Recall that a honest signer does not know the transaction amount of other input. If the signer is unfortunate that he picks the Omniring as:

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

Then the honest signer has no security guarantees according to the anonymity model in [23]: there is only one possible case, namely "Input 3 and 4".

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 1
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 insider 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 recipient 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.