RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero

Shi-Feng Sun[1,2], Man Ho Au[1] ⋆, Joseph K. Liu[3], Tsz Hon Yuen[4], Dawu Gu[2]

[1]Hong Kong Polytechnic University, Hong Kong
E-mail: `csssun,csallen@comp.polyu.edu.hk`
[2]Shanghai Jiao Tong University, China
E-mail: `dwgu@sjtu.edu.cn`
[3]Monash University, Australia
E-mail: `joseph.liu@monash.edu`
[4] Huawei, Singapore
E-mail: `YUEN.TSZ.HON@huawei.com`

Abstract. In this work, we initially study the necessary properties and security requirements of Ring Confidential Transaction (RingCT) protocol deployed in the popular anonymous cryptocurrency Monero. Firstly, we formalize the syntax of RingCT protocol and present several formal security definitions according to its application in Monero. Based on our observations on the underlying (linkable) ring signature and commitment schemes, we then put forward a new efficient RingCT protocol (RingCT 2.0), which is built upon the well-known Pedersen commitment, accumulator with one-way domain and signature of knowledge (which altogether perform the functions of a linkable ring signature). Besides, we show that it satisfies the security requirements if the underlying building blocks are secure in the random oracle model. In comparison with the original RingCT protocol, our RingCT 2.0 protocol presents a significant space saving, namely, the transaction size is independent of the number of groups of input accounts included in the generalized ring while the original RingCT suffers a linear growth with the number of groups, which would allow each block to process more transactions.

1 Introduction

1.1 Monero: A Blockchain-based Cryptocurrency

A cryptocurrency is a digital asset designed to work as a medium of exchange using cryptography to secure the transactions and to control the creation of additional units of the currency. Bitcoin became the first decentralized cryptocurrency in 2009. Since then, numerous cryptocurrencies have been created. Bitcoin and its derivatives use decentralized control as opposed to centralized electronic money or centralized banking systems. The decentralized control is related to the use of blockchain transaction database in the role of a distributed ledger.

Major advantages of cryptocurrency include decentralized control and anonymous payment, when compared to the traditional credit card or debit card system. However, the anonymity provided by bitcoin has been questioned in the sense it offers pseudonymity instead of offering a true anonymity. For instance, there is a research that identifies ownership relationships between Bitcoin addresses and IP addresses [22]. Bitcoin proxy or even other users may still compute the actual identity of a bitcoin's owner. Although there are various improvements to enhance the anonymity of bitcoin (e.g. [33]), they are far from practical and satisfactory.

One of the first attempt to provide anonymity in cryptocurrency is Dash (released in 2014), which anonymizes the transaction process by mixing coins. Nevertheless, it does not formally provide cryptographic anonymity. Another attempt to provide anonymity in

⋆ Corresponding author

cryptocurrency is ZCash [9] (released in 2016), which uses zero-knowledge succinct non-interactive argument of knowledge (zk-SNARKs) [10]. They provide anonymity with a formal security proof. They used zk-SNARKs to prove the knowledge of pre-image of hash functions in which the proof generation process is rather expensive. Therefore, the efficiency is much worse than the normal bitcoin transaction (for the sender side, it takes a few minutes to perform a spent computation).

Monero is an open-source cryptocurrency created in April 2014 that focuses on privacy, decentralisation and scalability. The current market value of Monero is already over US$1.5B[1], which is one of the largest cryptocurrencies. Unlike many cryptocurrencies that are derivatives of Bitcoin, Monero is based on the CryptoNote protocol and possesses significant algorithmic differences relating to blockchain obfuscation. Monero daemon is mainly based on the original CryptoNote protocol, which deploys "one-time ring signatures" as the core crypto-primitive to provide anonymity. Monero further improves the protocol by using a variant of linkable ring signature [24], which is called **Ring Confidential Transactions (RingCT)** [26].

On 10 January 2017, RingCT has been put into Monero transactions, starting at block #1220516. RingCT transactions are enabled by default at this stage, but it is still possible to send a transaction without RingCT until the next hard fork in September 2017. In the first month after implementation, it has been reported that approximately 50-60% of transactions used the optional RingCT feature.[2]

Upon the enhancement of privacy, a major trade-off is the increase of size for the transaction, due to the size of the linkable ring signature in the RingCT protocol. Although RingCT has already shortened the size of the ring signature by 50% when compared to the original CryptoNote protocol, it is still linear with the number of public keys included in the ring.

1.2 Ring Signature and Linkable Ring Signature

A ring signature scheme (e.g., [28, 35, 1]) allows a member of a group to sign messages on behalf of the group without revealing his identities, i.e. signer anonymity. In addition, it is not possible to decide whether two signatures have been issued by the same group member. Different from a group signature scheme (e.g., [14, 11, 7]), the group formation is spontaneous and there is no group manager to revoke the identity of the signer. That is, under the assumption that each user is already associated with a public key of some standard signature scheme, a user can form a group by simply collecting the public keys of all the group members including his/her own. These diversion group members can be totally unaware of being conscripted into the group.

Ring signature provides perfect (or unconditional) anonymity. However, it may be too strong in some scenario. For example, in the case of anonymous e-voting, it is necessary to detect if someone has submitted his vote more than once so that the second casting should not be counted. Similar concerns should be applied into anonymous e-cash system. A double-spent payment should be discarded. In both scenarios, a linkable-anonymity is necessary, instead of the strongest form, unconditional anonymity. Linkable ring signature [24] provides a perfect characteristic of linkable anonymity: verifier knows nothing about the signer, except that s/he is one of the users in the group (represented by the list of public keys/identities). Yet given any two linkable 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).

1.3 Our Contributions

The contributions of this paper are twofold. First, we give a rigorous security definition and requirement of RingCT protocol. We note that in the original paper of RingCT [26], there

[1] At the time of September 2017. Market info is referenced from https://coinmarketcap.com/

[2] https://web.archive.org/web/20170127204814/http://moneroblocks.info/stats/ringct-transactions

is no rigorous security definition but just a direct instantiation of the protocol. A rigorous security definition would definitely help future researchers to develop better improvement of RingCT. Second, we target to reduce the size of the RingCT protocol. Our new RingCT protocol (we call it RingCT 2.0) is based on the well-known Pedersen commitment, accumulator with one-way domain and signature of knowledge related to the accumulator. The accumulator and the signature of knowledge together perform the functions of a linkable ring signature. In particular, the size of signature in our protocol is independent to the number of groups of input accounts in a transaction. We argue that it can significantly shorten the size of each block, when compared to the original protocol (which is linear with the number of groups of accounts included in the generalized ring for the anonymizing purpose) especially when the number of groups grows larger. More importantly, our construction fits perfectly into the framework of the RingCT definition, which makes it suitable to be deployed in Monero.

2 Related Works

Linkable ring signature was first proposed by Liu et al. [24] in 2004 (they named it as *Linkable Spontaneous Anonymous Group* Signature which is actually linkable ring signature). There are many variants in different types of cryptosystems with different features. We summarize their features in table 1.

Table 1. Comparison of Linkable Ring Signatures

Scheme	Signature Size	Cryptosystem
Liu *et al.* [24]	$\mathcal{O}(n)$	public key
Tsang and Wei [31]	$\mathcal{O}(1)$	public key
Liu and Wong [25]	$\mathcal{O}(n)$	public key
Au *et al.* [2]	$\mathcal{O}(1)$	public key
Au *et al.* [3]	$\mathcal{O}(n)$	certificate-based
Zheng *et al.* [36]	$\mathcal{O}(n)$	public key
Tsang *et al.* [32]	$\mathcal{O}(n)$	public key
Tsang *et al.* [30]	$\mathcal{O}(n)$	identity-based
Chow *et al.* [15]	$\mathcal{O}(1)$	identity-based
Fujisaki and Suzuki [20]	$\mathcal{O}(n)$	public key
Fujisaki [19]	$\mathcal{O}(\sqrt{n})$	public key
Au *et al.* [4]	$\mathcal{O}(1)$	identity-based
Yuen *et al.* [34]	$\mathcal{O}(\sqrt{n})$	public key
Liu *et al.* [23]	$\mathcal{O}(n)$	public key

As we can see from the table, there are only a few constant size linkable ring signature existed in the literature. In our discussion, we focus on public key only because both identity-based and certificate-based cryptosystems are not suitable for blockchain paradigm as they require a Private Key Generator (PKG) to issue user keys, which contradicts to the decentralized concept of blockchain. Among them, [31] requires the Certificate Authority (CA) to generate the user key. [2] is an improvement over [31] but it still requires an interaction between the user and the CA during the user key generation process. Neither of them is suitable for blockchain.

We note that not all linkable ring signature schemes are suitable for Monero. There are some requirements that should be satisfied in order to be compatible with RingCT. We will discuss more on this in Section 4.

3 Preliminaries

In this section, we first give some notations used in the rest of this paper. We use $[n]$ to denote the set of integers $\{1, 2, \ldots, n\}$ for some positive integer $n \in \mathbb{N}$. For a randomized algorithm $\mathrm{A}(\cdot)$, we write $y = \mathrm{A}(x; r)$ to denote the unique output of A on input x and randomness r, and denote by $y \leftarrow \mathrm{A}(x)$ the process of picking randomness r at random and setting $y = A(x; r)$. Also, we write $x \leftarrow S$ for sampling an element uniformly at random from a set S, and use $negl(\lambda)$ to denote some negligible function in a security parameter λ.

3.1 Mathematical Assumptions

Bilinear Pairings. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order p, and g be a generator of $\mathbb{G}_1$. A function $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a bilinear map if the following properties hold:

- Bilinearity: $e(A^x, B^y) = e(A, B)^{xy}$ for all $A, B \in \mathbb{G}_1$ and $x, y \in \mathbb{Z}_p$;
- Non-degeneracy: $e(g, g) \neq 1$, where 1 is the identity of $\mathbb{G}_2$;
- Efficient computability: there exists an algorithm that can efficiently compute $e(A, B)$ for all $A, B \in \mathbb{G}_1$.

Decisional Diffie-Hellman (DDH) Assumption. Let $\mathbb{G}$ be a group where $|\mathbb{G}| = q$ and $g \in \mathbb{G}$ such that $\langle g \rangle = \mathbb{G}$. There exists no probabilistic polynomial time (PPT) algorithm that can distinguish the distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) with non-negligible probability over 1/2 in time polynomial in q, where a, b, c are chosen uniformly at random from $\mathbb{Z}_q$.
k-Strong Diffie-Hellman (k-SDH) Assumption. There exists no PPT algorithm which, on input a $k+1$-tuple $(g_0, g_0^{\alpha}, g_0^{\alpha^2}, \ldots, g_0^{\alpha^k}) \in \mathbb{G}^{k+1}$, returns a pair $(w, y) \in \mathbb{G} \times \mathbb{Z}_p^*$, where $\mathbb{G} = \langle g_0 \rangle$ and p is the order of $\mathbb{G}$, such that $w^{\alpha+y} = g_0$, with non-negligible probability and in time polynomial in λ.

3.2 Building Blocks

In this section, we briefly recall the basic primitives used to construct our RingCT protocol, which include accumulator with one-way domain, signature of knowledge and homomorphic commitment scheme.

Accumulators with One-Way Domain. As defined in [16, 6], an accumulator *accumulates* multiple values into one single value such that, for each value accumulated, there is a witness proving that it has indeed been accumulated. Formally, let $\mathcal{F} = \{F_\lambda\}$ be a sequence of families of functions and $\mathcal{X} = \{X_\lambda\}$ a sequence of families of finite sets, such that $F_\lambda = \{f : U_f \times X_f \to U_f\}$ and $X_\lambda \subseteq X_f$ for all $\lambda \in \mathbb{N}$, we call the pair $(\mathcal{F}, \mathcal{X})$ *an accumulator family with one-way domain* if the following conditions hold:

- quasi-commutativity: for all $\lambda \in \mathbb{N}, f \in F_\lambda, u \in U_f$ and $x_1, x_2 \in X_\lambda$, it holds that $f(f(u, x_1), x_2) = f(f(u, x_2), x_1)$. $\{X_\lambda\}$ is always referred to as the domain of this accumulator. For any $X = \{x_1, x_2, \cdots, x_n\} \subset X_\lambda$, we further refer to $f(\cdots f(u, x_1) \cdots x_n)$ as the accumulated value of X over u, which will be denoted by $f(u, X)$ thanks to this quasi-commutative property.
- collision-resistance: for all $\lambda \in \mathbb{N}$ and efficient adversaries $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{l} X \subset X_\lambda \wedge (x \in X_f \backslash X) \\ (w \in U_f) \wedge (f(w, x) = f(u, X)) \end{array} : \begin{array}{l} f \leftarrow F_\lambda; u \leftarrow U_f; \\ (x, w, X) \leftarrow \mathcal{A}(f, U_f, u) \end{array}\right] \leq negl(\lambda).$$

- one-way domain: let $\{Y_\lambda\}, \{R_\lambda\}$ be two sequences of families of sets associated with $\{X_\lambda\}$, such that each R_λ is an *efficiently verifiable, samplable* relation over $Y_\lambda \times X_\lambda$ and it is infeasible to efficiently compute a witness $y' \in Y_\lambda$ for an x sampled from X_λ. That is,

$$\Pr[(y', x) \in R_\lambda : (y, x) \leftarrow \mathsf{Samp}(1^\lambda); y' \leftarrow \mathcal{A}(1^\lambda, x)] \leq negl(\lambda),$$

where Samp denotes the efficient sampling algorithm over R_λ.

- $\mathsf{efficient\ generation}$: there exists an efficient algorithm denoted by $\mathsf{ACC.Gen}$ that on input a security parameter λ outputs a description desc of a random element of F_λ, possibly including some auxiliary information.
- $\mathsf{efficient\ evaluation}$: for $\lambda \in \mathbb{N}, f \in F_\lambda, u \in U_f$ and $X \subset X_\lambda$, $w \in U_f$ is called a witness for the fact that $x \in X$ has been accumulated within $v \doteq f(u, X) \in U_f$ *iff* $f(w, x) = v$. There exists two algorithms denoted by $\mathsf{ACC.Eval}$ and $\mathsf{ACC.Wit}$ that on input (desc, X) and (desc, x, X) can efficiently evaluate the accumulated value $f(u, X)$ and the witness for x in $f(u, X)$, respectively.

For sake of simplicity, we will denote by $\mathsf{ACC} = (\mathsf{ACC.Gen}, \mathsf{ACC.Eval}, \mathsf{ACC.Wit})$ such an *accumulator with one-way domain* in the following.

Signature of Knowledge. Every three-move Proof of Knowledge protocols (PoKs) that is Honest-Verifier Zero-Knowledge (HVZK) can be transformed into a signature scheme by setting the challenge to the hash value of the commitment concatenated with the message to be signed [18]. Signature schemes generated as such are provably secure [27] against existential forgery under adaptively chosen message attack in the random oracle model [8]. They are sometimes referred to as *Signatures of Knowledge*, SoK for short [12]. As an example, we denote by $\mathrm{SoK}\{(x) : y = g^x\}(m)$, where m is the message, the signature scheme derived from the zero-knowledge proof of the discrete logarithm of y using the above technique. Before presenting the formal definition of SoK, we first let $\mathtt{R}$ be a fixed NP-hard relation with the corresponding language $\mathtt{L} = \{y : \exists\ x\ s.t\ (x, y) \in \mathtt{R}\}$. Recall that a relation is called hard if it is infeasible for any efficient algorithm, given some instance y, to compute a valid witness such that $(x, y) \in \mathtt{R}$. In general, signature of knowledge protocol for $\mathtt{R}$ over message space $\mathcal{M}$ comprises of a triple of poly-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verf})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda)$: on input a security parameter λ, the algorithm outputs public parameters par, which will be implicitly taken as part input of the following algorithms. Also, we assume that λ is efficiently recoverable from par.
- $\mathsf{Sign}(m, x, y)$: on input a message $m \in \mathcal{M}$ and a valid pair $(x, y) \in \mathtt{R}$, the algorithm outputs an SoK π.
- $\mathsf{Verf}(m, \pi,\ y)$: on input a message m, an SoK π and a statement y, the algorithm outputs 0/1 indicating the in/validity of the SoK.

Definition 1 (SimExt Security of SoK [13]). *An SoK protocol $\mathsf{SoK} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verf})$ for hard relation $\mathtt{R}$ is called SimExt-secure if it satisfies the correct, simulatable and extractable properties as defined below.*

Correctness *For any message $m \in \mathcal{M}$ and valid pair $(x, y) \in \mathtt{R}$, it holds that*

$$\Pr[\mathsf{Verf}(m, \pi, y) = 1 : \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda); \pi \leftarrow \mathsf{Sign}(m, x, y)] \geq 1 - negl(\lambda),$$

where if the probability is exactly 1 we call SoK perfectly correct.

Simulatability *There exists a poly-time simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimSign})$ such that for any PPT adversary $\mathcal{A}$, it holds that*

$$\begin{aligned}\Big|\Pr\left[b = 1 : (\mathsf{par}, td) \leftarrow \mathsf{SimGen}(1^\lambda); b \leftarrow \mathcal{A}^{\mathsf{Sim}(td,\cdot,\cdot,\cdot)}(\mathsf{par})\right] - \\ \Pr\left[b = 1 : \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda); b \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\cdot,\cdot)}(\mathsf{par})\right]\Big| \leq negl(\lambda),\end{aligned}$$

where Sim receives an input (m, x, y), checks the validity of y and returns $\pi \leftarrow \mathsf{SimSign}$ (m, x, y) if $(x, y) \in \mathtt{R}$. In addition, td is the additional trapdoor information used by Sim to simulate signatures without knowing a witness.

Extraction *In addition to Sim, there exists an efficient extractor Ext such that for any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{c}(x, y) \in \mathtt{R} \vee (m, y) \in Q \\ \vee\ \mathsf{Verf}(m, y, \pi) = 0\end{array} : \begin{array}{l}(\mathsf{par}, td) \leftarrow \mathsf{SimGen}(1^\lambda); \\ (m, y, \pi) \leftarrow \mathcal{A}^{\mathsf{Sim}(td,\cdot,\cdot,\cdot)}(\mathsf{par}) \\ x \leftarrow \mathsf{Ext}(\mathsf{par}, td, m, y, \pi).\end{array}\right] \geq 1 - negl(\lambda).$$

where Q denotes the set of all queries (m, y) that $\mathcal{A}$ has made to Sim.

Homomorphic Commitment Schemes. Informally, a (non-interactive) commitment scheme includes two phases: in commit phase, a sender chooses a value and constructs a commitment to it; later in the reveal phase the sender may open the commitment and reveal the value. After that, the receiver can verify that it is exactly the value that was committed at first. More formally, a commitment scheme consists of a pair of poly-time algorithms ($\mathsf{CKGen}, \mathsf{Com}$): on input a security parameter λ, $\mathsf{CKGen}(1^\lambda)$ outputs a public commitment key ctk, which specifies a message space $\mathcal{M}_{ctk}$ and a commitment space $\mathcal{C}_{ctk}$; on input a message $m \in \mathcal{M}_{ctk}$, $\mathsf{Com}(ctk, m)$ generates a commitment $c \leftarrow \mathsf{Com}(ctk, m)$ to m, where ctk is often omitted when it is clear from the context. Normally, a commitment scheme should satisfy the hiding and binding properties, as defined below.

Definition 2 (Security of HCom [21]). *A non-interactive scheme $\mathsf{HCom} = (\mathsf{CKGen}, \mathsf{Com})$ is called a secure homomorphic commitment scheme if it satisfies the following properties.*

Hiding *This property means that the commitment does not reveal the committed value. More precisely, HCom is called hiding if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\left| \Pr\left[\mathcal{A}(c) = b : \begin{array}{l} ctk \leftarrow \mathsf{CKGen}(1^\lambda); (m_0, m_1) \leftarrow \mathcal{A}(ctk); \\ b \leftarrow \{0,1\}; c \leftarrow \mathsf{Com}(ctk, m_b) \end{array} \right] - \frac{1}{2} \right| \leq negl(\lambda),$$

where $m_0, m_1 \in \mathcal{M}_{ctk}$ and HCom is called perfectly hiding if the probability of $\mathcal{A}$ guessing b is exactly 1/2.

Binding *This property means that a commitment cannot be opened to two different values. More precisely, HCom is called binding if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{c} m_0 \neq m_1 \wedge \\ \mathsf{Com}(m_0; r_0) = \mathsf{Com}(m_1; r_1) \end{array} : \begin{array}{l} ctk \leftarrow \mathsf{CKGen}(1^\lambda); \\ (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(ctk) \end{array} \right] \leq negl(\lambda),$$

where $m_0, m_1 \in \mathcal{M}_{ctk}$ and r_0, r_1 are random coins of Com. HCom is called perfectly binding if the probability is exactly 0. Moreover, we call HCom strongly binding if the probability holds even for the condition $(m_0, r_0) \neq (m_1, r_1)$ rather than $m_0 \neq m_1$.

Homomorphic *For this property, we assume that for each well-formed ctk, the commitment space $\mathcal{C}_{ck}$ is a multiplicative group of order q and both the messages and random coins are from $\mathbb{Z}_q$. This property says that for all $\lambda \in \mathbb{N}$, $ctk \leftarrow \mathsf{CKGen}(1^\lambda)$, $m_0, m_1 \in \mathbb{Z}_q$ and $r_0, r_1 \in \mathbb{Z}_q$, it holds that*

$$\mathsf{Com}(m_0; r_0) \cdot \mathsf{Com}(m_1; r_1) = \mathsf{Com}(m_0 + m_1; r_0 + r_1).$$

4 RingCT Protocol for Monero

In this section, we formalize ring confidential transaction (RingCT) protocol for Monero. Recall that Monero is based on CryptoNote, where each user may have a number of distinct accounts. Each account consists of a one-time address and a coin, and it is always associated with an account key used to authorize its spending. In each transaction, a user can spend many of her/his accounts with the corresponding keys. The goal of ring confidential transaction (RingCT) protocol is to protect the anonymity of spenders as well as the privacy of transactions.

Informally, a RingCT protocol mainly comprises of two phases: the generation and the verification of ring confidential transactions, which are operated by the spender and recipients respectively. When a user would like to spend m of her/his accounts, w.l.o.g., denoted by $A_s = \{(pk_s^{(k)}, cn_s^{(k)})\}_{k\in[m]}$ where $pk_s^{(k)}$ is the user's k-th account address and $cn_s^{(k)}$ is the coin w.r.t. this account, s/he first chooses t output accounts $\{(pk_{out,j}, cn_{out,j})\}_{j\in[t]}$ for all output addresses $R = \{pk_{out,j}\}_{j\in[t]}$ accordingly, such that the sum of balances of her/his input accounts equals to that of output accounts, and then additionally selects $n-1$ groups of input accounts with each containing m different accounts to anonymously spend A_s for some payments (i.e., creating a ring confidential transaction). Whenever receiving this transaction from the P2P blockchain network, the miners check the validity of the transaction with public information along with it and add it to a (new) block if valid.

By a thorough analysis of the protocol in [26], we find that the RingCT protocol essentially involves ring signatures and commitments (that are used to hide account balance). To be compatible within the protocol, these two cryptographic primitives should satisfy the following properties simultaneously:

– Public keys generated by the key generation algorithm of ring signature should be homomorphic.
– Commitments should be homomorphic with respect to (w.r.t.) the same operation as public keys.
– Commitments to zero are well-formed public keys, each corresponding secret key of which can be derived from the randomness of commitments.

To further capture the essential properties and securities required by the ring confidential transaction protocol for Monero, we initiate the formalization of RingCT protocol and its security models, the details of which are shown in the following subsections.

4.1 Technical Description

In general, a RingCT protocol consists of a tuple of poly-time algorithms (Setup, KeyGen, Mint, Spend, Verify), the syntax of which are described as follows:

– $pp \leftarrow \mathsf{Setup}(1^\lambda)$: the *Setup* algorithm takes a security parameter $\lambda \in \mathbb{N}$, and outputs the public system parameters pp. All algorithms below have implicitly pp as part of their inputs.
– $(sk, pk) \leftarrow \mathsf{KeyGen}(pp)$: the *key generation* algorithm takes as input pp and outputs a public and secret key pair (pk, sk). In the context of Monero, pk is always set as a one-time address, which together with a coin constitutes an account.
– $(cn, ck) \leftarrow \mathsf{Mint}(pk, a)$: the *Mint* algorithm takes as input an amount a and a valid address pk s.t. $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, and outputs a coin cn for pk as well as the associated coin key ck[3]. The coin cn together with address pk forms an account $act \doteq (pk, cn)$, the corresponding secret key of which is $ask \doteq (sk, ck)$ that is required for authorizing its spending.
– $(tx, \pi, S) \leftarrow \mathsf{Spend}(\mathtt{m}, K_s, A_s, A, R)$: on input a group A_s of accounts together with the corresponding account secret keys K_s, an arbitrary set A of groups of input accounts containing A_s, a set R of out addresses and some transaction string $\mathtt{m} \in \{0,1\}^*$, the algorithm outputs a transaction tx (containing $\mathtt{m}$, A and A_R which denotes the set of output accounts w.r.t. R), a proof π and a set S of serial numbers.
– $1/0 \leftarrow \mathsf{Verify}(tx, \pi, S)$: on input the transaction tx containing $\mathtt{m}, A$ and A_R, proof π and serial numbers S, the algorithm verifies whether a set of accounts with serial numbers S is spent properly for the transaction tx towards addresses R, and outputs 1 or 0, meaning a `valid` or `invalid` spending respectively.

4.2 Security Definitions

A RingCT protocol should at least satisfy the properties formalized below.

Definition 3 (Perfect Correctness). *This 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. Specifically, a RingCT protocol $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ is called perfectly correct if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\Pr\left[\mathsf{Verify}(tx,\pi,S)=1 : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda);\ (\mathtt{m}, A, R) \leftarrow \mathcal{A}(pp, A_s, K_s) \\ \text{where } (A_s, K_s) = \big\{\big((pk,cn),(sk,ck)\big)\big\} \text{ s.t.} \\ (pk,sk) \leftarrow \mathsf{KeyGen}(pp), (cn,ck) \leftarrow \mathsf{Mint}(pk,a); \\ (tx,\pi,S) \leftarrow \mathsf{Spend}(\mathtt{m}, K_s, A_s, A, R). \end{array}\right] = 1.$$

[3] We note that ck will be privately sent to the user possessing account address pk, e.g., by private public key encryption.

Definition 4 (Balance). *This property requires that any malicious user cannot (1) spend any account without her control and (2) spend her own/controllable accounts with a larger output amount. Specifically, a RingCT protocol $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ is called balanced w.r.t. insider corruption if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\Pr\left[\mathcal{A}\ \mathit{Wins}: \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda);\ (\{act'_i\}_{i=1}^{\mu}, \{\mathcal{S}_i\}_{i=1}^{\nu}) \\ \leftarrow \mathcal{A}^{\mathtt{AddGen},\mathtt{ActGen},\mathtt{Spend},\mathtt{Corrupt}}(pp) \end{array}\right] \leq negl(\lambda),$$

where all oracles `AddGen`*,* `ActGen`*,* `Spend` *and* `Corrupt` *are defined as below:*

– `AddGen`*(i): on input a query number i, picks randomness τ_i, runs algorithm $(sk_i, pk_i) \leftarrow \mathsf{KeyGen}(pp; \tau_i)$ and returns address pk_i.*
– `ActGen`*(i, a_i): on input address index i and an amount a_i, runs algorithm $(cn_i, ck_i) \leftarrow \mathsf{Mint}(pk_i, a_i)$, then adds i and account $act_i = (pk_i, cn_i)$ to initially empty lists $\mathcal{I}$ and $\mathcal{G}$ respectively, and outputs (act_i, ck_i) for address pk_i, where pk_i is assumed to have been generated by* `AddGen`*. The associated secret key with account act_i is $ask_i \doteq (sk_i, ck_i)$. The oracle also uses ask_i to determine the serial number s_i of act_i and adds it to initially empty list $\mathcal{S}$.*
– `Spend`*$(\mathtt{m}, A_s, A, R)$: takes in transaction string $\mathtt{m}$, input accounts A containing A_s and output addresses R, runs $(tx, \pi, S) \leftarrow \mathsf{Spend}(\mathtt{m}, K_s, A_s, A, R)$ and returns (tx, π, S) after adding it to list $\mathcal{T}$, where $A_s \subset \mathcal{G}$ and we assume that at least one account/address in A_s has not been corrupted so far.*
– `Corrupt`*(i): on input query number $i \in \mathcal{I}$, uses account key ask_i to determine the serial number s_i of account act_i with address pk_i, then adds s_i and (s_i, a_i) to lists $\mathcal{C}$ and $\mathcal{B}$ respectively, where a_i is the balance of the account with address pk_i, and finally returns τ_i.*

At last, $\mathcal{A}$ outputs all her spends with some new accounts $(act'_1, act'_2, \cdots, act'_\mu, \mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_\nu)$ such that $\mathcal{S}_i = (tx_i, \pi_i, S_i)$, where all spends are payed to, w.l.o.g., the challenger with account address pk_c[4]*, i.e., $tx_i = (\mathtt{m}_i, A_i, A_{\{pk_c\}})$, and $A_i \subset \mathcal{G} \cup \{act'_i\}_{i=1}^{\mu}$ for all $i \in [\nu]$. We call $\mathcal{A}$ wins in the experiment if her outputs satisfy the following conditions:*

1. *$\mathsf{Verify}(tx_i, \pi_i, S_i) = 1$ for all $i \in [\nu]$.*
2. *$\mathcal{S}_i \notin \mathcal{T} \wedge S_i \subset \mathcal{S}$ for all $i \in [\nu]$, and $S_j \cap S_k = \emptyset$ for any different $j, k \in [\nu]$.*
3. *Let $S_i = \{s_{i,j}\}$ and $E = \bigcup_{i=1}^{\nu}\{a_{i,j} : (s_{i,j}, a_{i,j}) \in \mathcal{B} \wedge s_{i,j} \in S_i \cap \mathcal{C}\}$, it holds that $\sum_{a_{i,j} \in E} a_{i,j} < \sum_{i=1}^{\nu} a_{out,i}$, where $a_{out,i}$ denotes the balance of output account in $\mathcal{S}_i$.*

Definition 5 (Anonymity). *This property requires that two proofs of spending with the same transaction string $\mathtt{m}$, input accounts A, output addresses R and distinct spent accounts $A_{s_0}, A_{s_1} \in A$ are (computationally) indistinguishable, meaning that the spender's accounts are successfully hidden among all the honestly generated accounts. Specifically, a RingCT protocol $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ is called anonymous if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, it holds that*

$$\left|\Pr\left[b' = b : \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda);\ (\mathtt{m}, A_{s_0}, A_{s_1}, A, R) \leftarrow \\ \mathcal{A}_1^{\mathtt{AddGen},\mathtt{ActGen},\mathtt{Spend},\mathtt{Corrupt}}(pp);\ b \leftarrow \{0,1\}, \\ (tx^*, \pi^*, S^*) \leftarrow \mathsf{Spend}(\mathtt{m}, K_{s_b}, A_{s_b}, A, R); \\ b' \leftarrow \mathcal{A}_2^{\mathtt{Spend},\mathtt{Corrupt}}(pp, (tx^*, \pi^*, S^*)) \end{array}\right] - \frac{1}{2}\right| \leq negl(\lambda),$$

where all oracles are defined as before, $A_{s_i} \in A$ and $A_{s_i} \subset \mathcal{G}$ for $i \in \{0, 1\}$. In addition, the following restrictions should be satisfied:

– *For all $i \in \{0, 1\}$, any account in A_{s_i} has not been corrupted.*
– *Any query in the form of $(\cdot, A_s, \cdot, \cdot)$ s.t. $A_s \cap A_{s_i} \neq \emptyset$ has not been issued to* `Spend` *oracle.*

[4] Note that in this case, assuming pk_c has been generated by `AddGen`, the challenger knows all balances of the spent accounts and output accounts involved in the adversarial spends $\{\mathcal{S}\}_{i=1}^{\nu}$.

Definition 6 (Non-Slanderability). *This property requires that a malicious user cannot slander any honest user after observing an honestly generated spending. That is, 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. Specifically, a RingCT protocol* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ *is called non-slanderable if for all PPT adversaries* $\mathcal{A}$*, it holds that*

$$\Pr\left[\mathcal{A}\ \ Wins: \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda);\ \left((\hat{tx}, \hat{\pi}, \hat{S}), (tx^*, \pi^*, S^*)\right) \\ \leftarrow \mathcal{A}^{\mathtt{AddGen},\mathtt{ActGen},\mathtt{Spend},\mathtt{Corrupt}}(pp) \end{array}\right] \leq negl(\lambda),$$

where all oracles are defined as before, and $(\hat{tx}, \hat{\pi}, \hat{S})$ *is one output of the oracle* $\mathtt{Spend}$ *for some* $(\mathtt{m}, A_s, A, R)$*. We call* $\mathcal{A}$ *succeeds if the output satisfies the following conditions: (1)* $\mathsf{Verify}(tx^*, \pi^*, S^*) = 1$*; (2)* $(tx^*, \pi^*, S^*) \notin \mathcal{T}$*; (3)* $\hat{S} \cap \mathcal{C} = \emptyset$ *but* $\hat{S} \cap S^* \neq \emptyset$*.*

We note that our non-slanderability definition already covers *linkability* property of a linkable ring signature. Thus we do not need to explicitly define linkability.

5 Our RingCT 2.0 Protocol

In this section, we present a new RingCT protocol under our formalized syntax. Specifically, our protocol is constructed based on a generic accumulator with one-way domain ACC, a signature of knowledge SoK and the well-known Pedersen commitment. Proceeding to present the details, we first give an intuition of our protocol. Without loss of generality, we denote all, say, n groups of input accounts by $A = \{(pk_{in,i}^{(k)}, cn_{in,i}^{(k)})\}_{i\in[n],k\in[m]}$ (including the group of m accounts the user intends to spend) and set the spender's group as the s-th group, i.e., $A_s = \{(pk_{in,s}^{(k)}, cn_{in,s}^{(k)})\}_{k\in[m]}$. Conceptually, our idea is to arrange the account groups key into a matrix in which each group corresponds to a column. To shorten the size of transaction, the public keys in the same row is accumulated into one value. Then, the spender proves that he is using one account of each row in the spending. To ensure that the spender is using the account of the same column, the accumulated elements in protocol are formed as $pk_{in,i}^{(k)} \cdot u^s$ instead of $pk_{in,i}^{(k)}$, as shown in the matrix below.

To further guarantee the total balance in each transaction is conserved, the spender computes extra public keys $\widetilde{pk}_i$ based on the input accounts and the output accounts. Looking ahead, knowledge of the secret key that corresponds to $\widetilde{pk}_i$ implies that the balance in accounts A_i is equal to the balance of the output accounts.

$$\begin{pmatrix} pk_{in,1}^{(1)} \cdot u^1 & \cdots & pk_{in,s}^{(1)} \cdot u^s & \cdots & pk_{in,n}^{(1)} \cdot u^n \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ pk_{in,1}^{(k)} \cdot u^1 & \cdots & pk_{in,s}^{(k)} \cdot u^s & \cdots & pk_{in,n}^{(k)} \cdot u^n \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ pk_{in,1}^{(m)} \cdot u^1 & \cdots & pk_{in,s}^{(m)} \cdot u^s & \cdots & pk_{in,n}^{(m)} \cdot u^n \\ \boxed{\widetilde{pk}_1 \cdot u^1} & \cdots & \boxed{\widetilde{pk}_s \cdot u^s} & \cdots & \boxed{\widetilde{pk}_n \cdot u^n} \end{pmatrix} \begin{array}{c} \Rightarrow \\ \vdots \\ \Rightarrow \\ \vdots \\ \Rightarrow \\ \Rightarrow \end{array} \begin{pmatrix} v_1 \\ \vdots \\ v_k \\ \vdots \\ v_m \\ v_{m+1} \end{pmatrix}$$

5.1 Protocol Description

Let $\mathsf{ACC} = (\mathsf{ACC.Gen}, \mathsf{ACC.Eval}, \mathsf{ACC.Wit})$ be an accumulator with one-way domain and $\mathsf{SoK} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verf})$ be a signature of knowledge as defined in Section 3.2. Based on these primitives and the Pedersen commitment, our RingCT protocol $\mathsf{RCT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ is designed as follows:

$\mathsf{Setup}(1^\lambda)$: on input a security parameter λ, the algorithm prepares a collision-resistant accumulator f with one-way domain $\mathbb{G}_q$, together with its description desc, by calling $\mathsf{ACC.Gen}(1^\lambda)$, and generates par by running $\mathsf{Gen}(1^\lambda)$. Then it randomly picks generators h_0, $h_1, u, \tilde{h} \in \mathbb{G}_q$, and outputs the system parameters $pp = \left(1^\lambda, \mathsf{desc}, \mathsf{par}, h_0, h_1, u, \tilde{h}\right)$.

$\mathsf{KeyGen}(pp)$: on input pp, the algorithm generates a key pair $(sk, pk) := (x, y = h_0^x) \in \mathbb{Z}_q \times \mathbb{G}_q$ by executing the sampling algorithm of the one-way relation associated with the domain of f. In the context of Monero, the public key pk is always set as a one-time address, which combining with a coin constitutes a user's account.

$\mathsf{Mint}(pk, a)$: on input address pk and an amount $a \in \mathbb{Z}_q$, the algorithm mints a coin for pk: chooses $r \in \mathbb{Z}_q$ uniformly at random, computes commitment $c = h_0^r h_1^a$, where r is called a secret hiding factor and a is the balance of account pk, and then returns $(cn, ck) = (c, (r, a))$. The coin cn together with pk forms the account $act \doteq (pk, cn)$, to which the corresponding secret key is $ask \doteq (sk, ck)$.

$\mathsf{Spend}(\mathtt{m}, K_s, A_s, A, R)$: on input a set of secret keys K_s associated with the group of input accounts A_s, some transaction string $\mathtt{m} \in \{0,1\}^*$, an arbitrary set A of groups of input accounts containing A_s, and a set R of output addresses, the algorithm produces an SoK π and the corresponding serial numbers S w.r.t. A_s as follows. Without loss of generality, we denote all, say, n groups (including the group the user intends to spend) of input accounts by $A = \{(pk_{in,i}^{(k)}, cn_{in,i}^{(k)})\}_{i\in[n],k\in[m]}$ and set the spender's group as the s-th group, i.e., $A_s = \{(pk_{in,s}^{(k)}, cn_{in,s}^{(k)})\}_{k\in[m]}$, the corresponding secret keys of which are $K_s = \{ask_s^{(k)} = (sk_{in,s}^{(k)}, (r_{in,s}^{(k)}, a_{in,s}^{(k)}))\}_{k\in[m]}$, and denote the intended output addresses by $R = \{pk_{out,j}\}_{j\in[t]}$.

1. Set $a_{out,j} \in \mathbb{Z}_q$ for all output address $pk_{out,j} \in R$, such that the input and output balances satisfy $\sum_{k=1}^{m} a_{in,s}^{(k)} = \sum_{j=1}^{t} a_{out,j}$, then pick uniformly at random $r_{out,j} \in \mathbb{Z}_q$ and mint coin $cn_{out,j} = c_{out,j} = h_0^{r_{out,j}} h_1^{a_{out,j}}$. After that, add output account $act_{out,j} = (pk_{out,j}, cn_{out,j})$ to A_R, and privately send the coin key $ck_{out,j} = (r_{our,j}, a_{out,j})$ to the user holding address $pk_{out,j}$.
2. Compute $\widetilde{sk}_s = \sum_{k=1}^{m} sk_{in,s}^{(k)} + \sum_{k=1}^{m} r_{in,s}^{(k)} - \sum_{j=1}^{t} r_{out,j}$ and $\widetilde{pk}_i = \prod_{k=1}^{m} pk_{in,i}^{(k)} \cdot \prod_{k=1}^{m} cn_{in,i}^{(k)} / \prod_{j=1}^{t} cn_{out,j}$ for each $i \in [n]$. Clearly, it holds that $\widetilde{pk}_s = h_0^{\widetilde{sk}_s}$, which follows from the fact that $\sum_{k=1}^{m} a_{in,s}^{(k)} = \sum_{j=1}^{t} a_{out,j}$. For convenience, we denote $\widetilde{pk}_i$ and $\widetilde{sk}_s$ by $y_i^{(m+1)}$ and $x_s^{(m+1)}$ respectively hereafter, i.e., $\widetilde{pk}_i \doteq y_i^{(m+1)}$ and $\widetilde{sk}_s \doteq x_s^{(m+1)}$.
3. Generate a proof π that the group of coins A_s was spent properly for a transaction tx, which consists of $\mathtt{m}$, input accounts A and output accounts $A_R = \{act_{out,j}\}$, as follows. For clarity, we denote $sk_{in,s}^{(k)} = x_s^{(k)}$ for all $k \in [m]$ and $pk_{in,i}^{(k)} = y_i^{(k)}$ for all $i \in [n]$ and $k \in [m]$. Recall that $\widetilde{pk}_i \doteq y_i^{(m+1)}$ for all $i \in [n]$ and $\widetilde{sk}_s \doteq x_s^{(m+1)}$.
   (a) For each $k \in [m+1]$, compute the accumulated value $v_k = \mathsf{ACC.Eval}(\mathsf{desc}, \{y_i^{(k)} \cdot u^i\})$ and the witness $w_s^{(k)} = \mathsf{ACC.Wit}(\mathsf{desc}, \{y_i^{(k)} \cdot u^i | i \neq s\})$ for the fact that $y_s^{(k)} \cdot u^s$ has been accumulated within v_k (i.e., computing the witness $w_s^{(k)}$ s.t. $f(w_s^{(k)}, y_s^{(k)} \cdot u^s) = v_k$). Then compute $s_k = \tilde{h}^{x_s^{(k)}}$ for all $k \in [m]$. For simplicity, we denote $z_s^{(k)} = y_s^{(k)} \cdot u^s$ hereafter.
   (b) Use Sign to produce a signature of knowledge π on tx as:
   $$\mathrm{SoK}\left\{ (\{w_k, z_k, x_k\}_{k=1}^{m+1}, \gamma) : \begin{array}{c} f(w_{m+1}, z_{m+1}) = v_{m+1} \wedge z_{m+1} = h_0^{x_{m+1}} u^\gamma \wedge \\ f(w_1, z_1) = v_1 \wedge z_1 = h_0^{x_1} u^\gamma \wedge s_1 = \tilde{h}^{x_1} \wedge \\ \vdots \\ f(w_m, z_m) = v_m \wedge z_m = h_0^{x_m} u^\gamma \wedge s_m = \tilde{h}^{x_m} \end{array} \right\}(tx)$$
   (c) Eventually, return (tx, π, S), where $S = \{s_1, s_2, \ldots, s_m\}$. We note that the serial number s_k is uniquely determined by the address key $sk_{in,s}^{(k)}$ for every $k \in [m]$, and thus they can be used to prevent double-spending.

$\mathsf{Verify}(tx, \pi, S)$: receiving a transaction tx containing $\mathtt{m}, A$ and A_R, the associated SoK π for tx and the serial numbers $S = \{s_i\}$, the recipient verifies that a set of accounts with serial numbers $\{s_i\}$ from input accounts A was spent for a transaction tx (towards output addresses R) with string $\mathtt{m}$, as follows:

1. Use $A = \{(pk_{in,i}^{(k)}, cn_{in,i}^{(k)})\}_{i\in[n],k\in[m]}$ and $A_R = \{(pk_{out,j}, cn_{out,j})\}_{j\in[t]}$ (contained in tx) to compute $\widetilde{pk}_i = \prod_{k=1}^{m} pk_{in,i}^{(k)} \cdot \prod_{k=1}^{m} cn_{in,i}^{(k)} / \prod_{j=1}^{t} cn_{out,j}$ for all $i \in [n]$, and then compute the accumulated values $v_k = \mathsf{ACC.Eval}(\mathsf{desc}, \{pk_{in,i}^{(k)} \cdot u^i\})$ for all $k \in [m]$ and $v_{m+1} = \mathsf{ACC.Eval}(\mathsf{desc}, \{\widetilde{pk}_i \cdot u^i\})$.
2. Take as input accumulated values $(v_1, \cdots, v_{m+1})$, serial numbers $S = (s_1, \cdots, s_m)$, transaction tx and π to verify whether it is a *valid* spending by checking $\mathsf{Verf}(tx, (v_1, \cdots, v_{m+1}, s_1, \cdots, s_m), \pi) \overset{?}{=} 1$. If true, accept this transaction, otherwise reject it.

The correctness of this protocol follows directly from that of the underlying signature of knowledge protocol SoK. We do not give more details here.

Remark 1. Compared with the conference version, there is a modification of how s_i is computed in our RingCT 2.0 protocol. The conference version, which requires zero-knowledge proof-of-knowledge of (sk_i, pk_i) for the relation $s_i = H(pk_i)^{sk_i}$ is extremely expensive in practice. The current version, which requires $s_i = \tilde{h}^{sk_i}$ can be instantiated using standard techniques and can be done in a much more efficient manner.

5.2 Security Analysis

In this part, the securities of our RingCT protocol are collectively analyzed under the formalized security models, which are indicated as the following theorems.

Theorem 1. *Assuming the discrete logarithm (DL) problem in $\mathbb{G}_q$ is hard, ACC is an accumulator with one-way domain and SoK is a SimExt-secure signature of knowledge, then the proposed RingCT protocol RCT is balanced w.r.t. insider corruption.*

Theorem 2. *Let HCom_P be the Pedersen commitment, ACC be an accumulator with one-way domain and SoK a SimExt-secure signature of knowledge, then the proposed RingCT protocol RCT is anonymous under the DDH assumption.*

Theorem 3. *Assuming the DL problem in $\mathbb{G}_q$ is hard, ACC is an accumulator with one-way domain and SoK is a SimExt-secure signature of knowledge, then the proposed RingCT protocol RCT is non-slanderable w.r.t. insider corruption.*

Proofs for Theorem 1, 2 and 3 can be found in the Appendix.

5.3 Instantiations

Our RingCT protocol is constructed based on a well-known homomorphic commitment, i.e., the Pedersen commitment, a generic accumulator with one-way domain ACC and a signature of knowledge SoK for a specific language related to ACC. Next we give an instantiation of ACC and SoK, and briefly recall the Pedersen commitment for completeness.

Pedersen commitment. As shown in [21], the Pedersen commitment is naturally a homomorphic commitment scheme. Its key generation algorithm CKGen and commit algorithm Com are described respectively as:

– $\mathsf{CKGen}(1^\lambda)$: on input a security parameter 1^λ, outputs the commitment key $ctk = (\mathbb{G}_q, q, g, h)$, where $\mathbb{G}_q$ is a cyclic group of prime order q and g, h are random generators of $\mathbb{G}_q$.
– $\mathsf{Com}(ctk, m)$: to commit to a message $m \in \mathbb{Z}_q$, the algorithm randomly picks $r \in \mathbb{Z}_q$ and computes $c = g^m h^r$.

As well known, this commitment scheme is perfectly hiding and computationally strongly binding under the discrete logarithm assumption in $\mathbb{G}_q$.

Accumulator with one-way domain. A specific (universal) accumulator for DDH groups presented in [6] is well suited to our protocol, the algorithms of which are described as follows:

– $\mathsf{ACC.Gen}(1^\lambda)$: generate cyclic groups $\mathbb{G}_1 = \langle g_0 \rangle$ and $\mathbb{G}_2$ of prime order p, equipped with a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, and an accumulating function $\mathsf{g} \circ \mathsf{f} : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \mathbb{G}_1$, where f is defined as $\mathsf{f} : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ such that $\mathsf{f} : (u, x) \mapsto u(x + \alpha)$ for some auxiliary information α randomly chosen from $\mathbb{Z}_p^*$ (for simplicity, u is always set as the identity element of $\mathbb{Z}_p^*$) and g is defined as $\mathsf{g} : \mathbb{Z}_p^* \to \mathbb{G}_1$ such that $\mathsf{g} : x \mapsto g_0^x$. The domain of accumulatable elements is $\mathbb{G}_q = \langle h \rangle$, which is a cyclic group of prime order q such that $\mathbb{G}_q \subset \mathbb{Z}_p^*$. At last, output the description $\mathsf{desc} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_q, e, g_0, g_0^\alpha, g_0^{\alpha^2}, \cdots, g_0^{\alpha^n}, \mathsf{g} \circ \mathsf{f})$, where n is the maximum number of elements to be accumulated.
– $\mathsf{ACC.Eval}(\mathsf{desc},\ X)$: compute the accumulated value $\mathsf{g} \circ \mathsf{f}(1, X)$ for X by evaluating $\prod_{i=0}^n (g_0^{\alpha^i})^{u_i}$ with public information $\{g_0^{\alpha^i}\}_{i \in [n]}$, where u_i is the coefficient of the polynomial $\prod_{x \in X}(x + \alpha) = \prod_{i=0}^n (u_i \alpha^i)$.
– $\mathsf{ACC.Wit}(\mathsf{desc},\ x_s, X)$: the relation Ω w.r.t. this accumulator is defined as $\Omega(w, x, v) = 1$ *iff* $e(w, g_0^x g_0^\alpha) = e(v, g_0)$, a witness w_s for the element $x_s \in X := \{x_1, x_2, \cdots, x_n\}$ s.t. $s \in [n]$ is computed as $w_s = \mathsf{g} \circ \mathsf{f}(1, X \backslash \{x_s\}) = \prod_{i=0}^{n-1} (g_0^{\alpha^i})^{u_i}$ with public information $\{g_0^{\alpha^i}\}_{i \in [n-1]}$, where u_i is the coefficient of the polynomial $\prod_{i=1, i \neq s}^n (x_i + \alpha) = \prod_{i=0}^{n-1} (u_i \alpha^i)$.

Regarding this accumulator, the domain of accumulatable elements is $\mathbb{G}_q = \langle h \rangle$, the one-way relation for which is defined as $\mathtt{R}_q \doteq \{(y, x) \in \mathbb{Z}_q \times \mathbb{G}_q : x = h^y\}$. Moreover, the relation $\mathtt{R}_q$ is *efficiently verifiable*, *efficiently samplable* and *one-way*, as defined before.

Theorem 4 ([6]). *Under the n-SDH assumption in group $\mathbb{G}_1$, the above accumulator ACC is a secure universal accumulator. Moreover, under the DL assumption in group $\mathbb{G}_q$, it is an accumulator with one-way domain.*

Remark 2. It is easy to see that one drawback of our RingCT in contrast to the original is that the accumulator needs a trusted setup. Similar to Zcash [9], this can be partially solved by using multiparty computation technique.

As to the SoK associated with our RingCT protocol (instantiated with this ACC), it can be obtained by applying the Fiat-Shamir paradigm [17] to a generalized interactive zero-knowledge protocol from [6]. In the following, we present more details on the zero-knowledge protocol equipped with our RingCT.

Let g_0, g_1, g_2 and $h_0, h_1, h_2, \tilde{h}, u$ be independent generators of $\mathbb{G}_1$ and $\mathbb{G}_q \subset \mathbb{Z}_p^*$ respectively. The other notations are the same as before. To achieve our goal, the protocol can be decomposed as several efficient zero-knowledge protocols as below.

The first protocol, denoted by PoK_1, is to prove the knowledge of (x_k, z_k, γ) s.t. $z_k = h_0^{x_k} \cdot u^\gamma$ and $s_k = \tilde{h}^{x_k}$ without revealing any information of x_k, γ and z_k. More precisely, we have:

$$\mathrm{PoK}_1 \left\{ (z_k, r_k, x_k, \gamma, t) : C_k = g_0^{z_k} g_1^{r_k} \wedge z_k = h_0^{x_k} \cdot u^\gamma \wedge s_k = \tilde{h}^{x_k} \wedge D = h_1^\gamma h_2^t \right\}.$$

To instantiate this protocol, we further treat it as the composition of the following sub-protocols:

$$\mathrm{PoK}_1 \left\langle \begin{array}{l} \mathrm{PoK}_{1,1}\{(z_k, r_k) : C_k = g_0^{z_k} g_1^{r_k}\} \\ \mathrm{PoK}_{1,2}\{(x_k, r_k, \gamma, t) : C_k = g_0^{h_0^{x_k} \cdot u^\gamma} g_1^{r_k} \wedge s_k = \tilde{h}^{x_k} \wedge D = h_1^\gamma h_2^t\} \end{array} \right.$$

The instantiation of $\mathrm{PoK}_{1,1}$ is a standard generalization of Schnorr's protocol. Regarding the instantiation of $\mathrm{PoK}_{1,2}$, it makes use of the zero-knowledge proof-of-knowledge of double discrete logarithms [11], which can be obtained by extending the protocol presented in [6].

The second protocol denoted by PoK_2, is to demonstrate that the committed element z_k in C_k is indeed accumulated in value v_k, without revealing any information about w_k s.t. $\Omega(w_k, z_k, v_k) = 1$. More concretely, we have:

$$\mathrm{PoK}_2 \left\{ (w_k, z_k, r_k) : e(w_k, g_0^{z_k} g_0^\alpha) = e(v_k, g_0) \wedge C_k = g_0^{z_k} g_1^{r_k} \right\}.$$

By using the similar technique in [5], a prover knowing (w_k, z_k, r_k) can compute this protocol by first computing quantities $w_{k,1} = g_0^{\tau_1} g_1^{\tau_2}, w_{k,2} = w_k g_1^{\tau_1}$ for some $\tau_1, \tau_2 \leftarrow \mathbb{Z}_p$, and then computing the following proof-of-knowledge protocol:

$$\mathrm{PoK}_2' \Big\{ (\tau_1, \tau_2, z_k, \delta_1, \delta_2, r_k) : w_{k,1} = g_0^{\tau_1} g_1^{\tau_2} \wedge w_{k,1}^{z_k} = g_0^{\delta_1} g_1^{\delta_2} \wedge \\ \tfrac{e(w_{k,2}, g_0^{\alpha})}{e(v_k, g_0)} = e(w_{k,2}, g_0)^{-z_k} e(g_1, g_0)^{\delta_1} e(g_1, g_0^{\alpha})^{\tau_1} \wedge C_k = g_0^{z_k} g_1^{r_k} \Big\}.$$

where $\delta_1 = \tau_1 z_k$ and $\delta_2 = \tau_2 z_k$.

Combining protocols PoK_1 and PoK_2, we get an interactive zero-knowledge protocol for our RingCT:

$$\mathrm{PoK} \Big\{ (w_k, z_k, x_k, \gamma) : e(w_k, g_0^{z_k} g_0^{\alpha}) = e(v_k, g_0) \wedge z_k = h_0^{x_k} u^{\gamma} \wedge s_k = \tilde{h}^{x_k} \Big\}.$$

In the above presentation, we give more priority to clarity than efficiency; the protocols may be optimized for even better performance.

Remark 3. In our RingCT protocol, a spender needs to compute such a zero-knowledge protocol for m times (cf. the third step of algorithm Spend), each time corresponding to an accumulation of n input accounts. In addition, the spender needs to calculate a simplified version of PoK, where $s_k = \tilde{h}^{x_k}$ is omitted.

We also note that, to avoid the use of "negative amount" in transactions, the user in our protocol has to show that the transaction amount $a \in [0, 2^{64}]$ as well if we use 64 bits to represent a transaction amount, which can be realized in a similar way as the original RingCT [26].

6 Efficiency Analysis

In this section, we give a brief comparison of the efficiency of our RingCT protocol with that of [26]. In particular, the anonymity of our protocol relies heavily on the underlying accumulator with one-way domain, which compacts a group of input accounts to a shorter value, while the RingCT protocol given in [26] is directly constructed on the basis of a linkable ring signature. As shown in [26], the size of signature in their protocol increases linearly with the number n of group accounts. More concretely, it is almost $\mathcal{O}(n(m+1))$ where m is the number of accounts contained in each group. In contrast, the communication complexity of our protocol is $\mathcal{O}(m)$, which is independent of the number of groups. Clearly, the proposed protocol presents a significant space/bandwidth saving when n is large enough. More details on the comparison are given in Table 2 below.

Table 2. Comparison of RingCT Protocols

Ref.	Spender	Verifier	Communication
[26]	$2.1mn\cdot \exp_1$ $+mn\cdot \mathrm{H}_p$	$2.2mn\cdot\exp_1$ $+mn\cdot\mathrm{H}_p$	$m\vert\mathbb{G}_1\vert+1\vert\mathrm{H}\vert$ $+mn\vert\mathbb{Z}_p\vert$
Ours	$0.4(m+1)(n+1)\cdot \exp_1$ $+(m+1)\cdot p$ $+(m+1)\cdot\exp_T$	$3.2\lambda(m+1)\cdot\exp_q +$ $(1.1\lambda+0.4n+4)(m+1)\cdot\exp_1$ $+3(m+1)\cdot\mathrm{p}+2.2(m+1)\cdot\exp_T$	$(\lambda+5)(m+1)\vert\mathbb{G}_1\vert+$ $(3\lambda+6)(m+1)\vert\mathbb{Z}_p\vert+$ $3\lambda(m+1)\vert\mathbb{Z}_q\vert+(m+1)\vert\mathbb{G}_T\vert$

"n": the number of group of input accounts; "m": the number of input accounts in each group; "λ": the length of element in $\mathbb{Z}_p$; "$\exp_1$": an exponentiation operation in group $\mathbb{G}_1$; "$\exp_T$": an exponentiation operation in group $\mathbb{G}_T$; "$\exp_q$": an exponentiation operation in group $\mathbb{G}_q \subset \mathbb{Z}_p^*$; "$p$": a bilinear pairing operation; H_p: a map-to-point hash function; $|\mathrm{H}|$: the output size of a hash function H; $|\mathbb{G}_1|$: the length of element in group $\mathbb{G}_1$, similarly for $|\mathbb{Z}_p|, |\mathbb{Z}_q|$ and $|\mathbb{G}_T|$.

In the comparison, we only take into account of the expensive operations, mainly exponentiation and bilinear pairing. We also take some pre-computations (that are not counted in the table) and optimizations in both [26] and our protocol. For example, when computing protocol PoK'_2, the initial message, say $w_{k,1}, g_1^{\tau_1}$, and pairings $e(g_1, g_0), e(g_1, g_0^{\alpha})$ can be pre-computed, and multi-exponentiation, e.g., $g_0^{\tau_1} g_1^{\tau_2}$ can be optimized with almost 1.1 exponentiation operation. Similar strategy has been taken to the computation of other sub-protocols, we omit the details here.

7 Conclusion

In this work, we first formalize the syntax of RingCT protocol and present some rigorous security definitions by capturing the necessary security requirements for its application in Monero, which is the core part for Monero. Next, we propose a new RingCT protocol (RingCT 2.0) based on a specific homomorphic commitment, accumulator with one-way domain and the related signature of knowledge. The size of the RingCT 2.0 protocol is independent to the number of groups of input accounts included in the generalized ring while the original RingCT protocol suffers from the linear growing size with the number of groups. We believe the significant space complexity improvement in RingCT 2.0 will improve the overall efficiency of Monero especially for a pretty high level of anonymity.

Acknowledgement

This work is supported by National Natural Science Foundation of China (61602396, 61472083).

References

1. M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n Signatures from a Variety of Keys. In *ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2002.
2. M. H. Au, S. S. M. Chow, W. Susilo, and P. P. Tsang. Short linkable ring signatures revisited. In *EuroPKI*, volume 4043 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2006.
3. M. H. Au, J. K. Liu, W. Susilo, and T. H. Yuen. Certificate based (linkable) ring signature. In *Information Security Practice and Experience, Third International Conference, ISPEC 2007, Hong Kong, China, May 7-9, 2007, Proceedings*, volume 4464 of *Lecture Notes in Computer Science*, pages 79–92. Springer, 2007.
4. M. H. Au, J. K. Liu, W. Susilo, and T. H. Yuen. Secure id-based linkable and revocable-iff-linked ring signature with constant-size construction. *Theor. Comput. Sci.*, 469:1–14, 2013.
5. M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-taa. In *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, pages 111–125, 2006.
6. M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, pages 295–308, 2009.
7. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *Proc. EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer-Verlag, 2003.
8. M. Bellare and P. Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
9. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.
10. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 781–796, 2014.

11. J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *Proc. CRYPTO 97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.
12. J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *CRYPTO 1997*, volume 1294 of *LNCS*, pages 410–424. Springer-Verlag, 1997.
13. M. Chase and A. Lysyanskaya. On signatures of knowledge. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 78–96, 2006.
14. D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.
15. S. S. M. Chow, W. Susilo, and T. H. Yuen. Escrowed linkability of ring signatures and its applications. In *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 175–192. Springer, 2006.
16. Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous Identification in Ad Hoc Groups. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 609–626. Springer, 2004.
17. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.
18. U. Fiege, A. Fiat, and A. Shamir. Zero Knowledge Proofs of Identity. In *STOC '87: 19th Annual ACM conference on Theory of Computing*, pages 210–217, New York, NY, USA, 1987. ACM Press.
19. E. Fujisaki. Sub-linear size traceable ring signatures without random oracles. In *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 393–415. Springer, 2011.
20. E. Fujisaki and K. Suzuki. Traceable ring signature. In *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2007.
21. J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 253–280, 2015.
22. P. Koshy, D. Koshy, and P. D. McDaniel. An analysis of anonymity in bitcoin using P2P network traffic. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 469–485, 2014.
23. J. K. Liu, M. H. Au, W. Susilo, and J. Zhou. Linkable ring signature with unconditional anonymity. *IEEE Trans. Knowl. Data Eng.*, 26(1):157–165, 2014.
24. J. K. Liu, V. K. Wei, and D. S. Wong. Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups (Extended Abstract). In *ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*. Springer, 2004.
25. J. K. Liu and D. S. Wong. Linkable ring signatures: Security models and new schemes. In *ICCSA (2)*, volume 3481 of *Lecture Notes in Computer Science*, pages 614–623. Springer, 2005.
26. S. Noether. Ring Signature Confidential Transactions for Monero. Cryptology ePrint Archive, Report 2015/1098, 2015. http://eprint.iacr.org/.
27. D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In *EUROCRYPT 1996*, volume 1070 of *LNCS*, pages 387–398, 1996.
28. R. L. Rivest, A. Shamir, and Y. Tauman. How to Leak a Secret. In *Proc. ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer-Verlag, 2001.
29. S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen. RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero (Full Version). Cryptology ePrint Archive, Report 2017, 2017. http://eprint.iacr.org/.
30. P. P. Tsang, M. H. Au, J. K. Liu, W. Susilo, and D. S. Wong. A Suite of Non-Pairing ID-Based Threshold Ring Signature Schemes with Different Levels of Anonymity. In *ProvSec 2010*, volume 6402 of *Lecture Notes in Computer Science*, pages 166–183. Springer, 2010.
31. P. P. Tsang and V. K. Wei. Short Linkable Ring Signatures for E-Voting, E-Cash and Attestation. In *ISPEC 2005*, volume 3439 of *Lecture Notes in Computer Science*. Springer, 2005.
32. P. P. Tsang, V. K. Wei, T. K. Chan, M. H. Au, J. K. Liu, and D. S. Wong. Separable Linkable Threshold Ring Signatures. In *Proc. INDOCRYPT 2004*, Lecture Notes in Computer Science, pages 384–398. Springer-Verlag, 2004.

33. D. A. Wijaya, J. K. Liu, R. Steinfeld, S. Sun, and X. Huang. Anonymizing bitcoin transaction. In *ISPEC 2016*, volume 10060 of *Lecture Notes in Computer Science*, pages 271–283. Springer, 2016.
34. T. H. Yuen, J. K. Liu, M. H. Au, W. Susilo, and J. Zhou. Efficient linkable and/or threshold ring signature without random oracles. *Comput. J.*, 56(4):407–421, 2013.
35. F. Zhang and K. Kim. ID-Based Blind Signature and Ring Signature from Pairings. In *Proc. ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 533–547. Springer-Verlag, 2002.
36. D. Zheng, X. Li, K. Chen, and J. Li. Linkable ring signatures from linear feedback shift register. In *EUC Workshops*, volume 4809 of *Lecture Notes in Computer Science*, pages 716–727. Springer, 2007.

A Proof for Theorems

A.1 Proof for Theorem 1

Proof. Suppose that there is an efficient adversary $\mathcal{A}$ that can break the balance property of our RingCT protocol RCT with non-negligible probability ϵ, then we can construct an efficient algorithm $\mathcal{A}'$ to solve the DL problem in $\mathbb{G}_q$.

Given a random DL instance $(h_0, h_1 = h_0^\alpha)$, the algorithm $\mathcal{A}'$ arming to compute α generates desc and par by running $\mathsf{ACC.Gen}(1^\lambda)$ and $\mathsf{Gen}(1^\lambda)$, respectively. It then chooses uniformly at random $\beta \in \mathbb{Z}_q$, computes $\tilde{h} = h_0^\beta$ and sets $pp = (\mathsf{desc}, \mathsf{par}, h_0, h_1, u, \tilde{h})$. Assume without loss of generality that $\mathcal{A}$ makes at most q_{ad}, q_{ac} and q_{co} queries to $\mathtt{AddGen}$, $\mathtt{ActGen}$ and $\mathtt{Corrupt}$, respectively. $\mathcal{A}'$ first randomly picks $j^* \in [q_{ad}]$ and chooses uniformly at random $x_i \in \mathbb{Z}_q$ for all $i \in [q_{ad}]$, and then $\mathcal{A}'$ sets the queried addresses as

$$pk_i = \begin{cases} h_0^{x_i}, & i \neq j^* \\ h_1^{x_i}, & i = j^* \end{cases}$$

where the corresponding secret key for pk_i is implicitly set as $sk_i = x_i$ for all $i \in [q_{ad}] \backslash \{j^*\}$ and $sk_{j^*} = \alpha \cdot x_{j^*}$. After that, $\mathcal{A}'$ simulates all the oracles as below:

- $\mathtt{AddGen}(i)$: for the i-th query, simply returns address pk_i.
- $\mathtt{ActGen}(i, a_i)$: on input index i and an amount a_i, chooses random $r_i \in \mathbb{Z}_q$ and sets $c_i = h_0^{r_i} h_1^{a_i}$. Then, returns $(cn_i, ck_i) = (c_i, (r_i, a_i))$ and adds i, $act_i = (pk_i, cn_i)$ and $s_i = \tilde{h}^{sk_i}$ to lists $\mathcal{I}$, $\mathcal{G}$ and $\mathcal{S}$, respectively.
- $\mathtt{Spend}(\mathtt{m}, A_s, A, R)$: on input a transaction string $\mathtt{m}$, input accounts A containing $A_s \subset \mathcal{G}$ and output addresses R, the algorithm $\mathcal{A}'$ generates (tx, π, S) as in real RingCT protocol with the corresponding account keys K_s if $act_{j^*} \notin A_s$. Otherwise, $\mathcal{A}'$ generates π by invoking the simulator Sim of SoK with the appropriately accumulated values and the corresponding serial numbers (to A_s) as input. By the SimExt security of SoK, the simulated spending without witness is computationally indistinguishable from the real.
- $\mathtt{Corrupt}(i)$: on input query $i \in \mathcal{I}$, $\mathcal{A}'$ aborts if $i = j^*$. Otherwise, returns x_i, and adds s_i and (s_i, a_i) to lists $\mathcal{C}$ and $\mathcal{B}$ by searching on the previous queries made to $\mathtt{ActGen}$ oracle.

Finally, $\mathcal{A}$ outputs $(act'_1, act'_2, \cdots, act'_\mu, \mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_\nu)$, such that $\mathcal{S}_i = (tx_i, \pi_i, S_i)$ and all spends are payed to $\mathcal{A}'$. Suppose that $\mathcal{A}$ wins in this simulated game, meaning that the outputs of $\mathcal{A}$ satisfy that $\sum_{a_{i,j} \in E} a_{i,j} < \sum_{i=1}^{\nu} a_{out,i}$, where $S_i = \{s_{i,j}\}$, $E = \bigcup_{i=1}^{\nu} \{a_{i,j} : (s_{i,j}, a_{i,j}) \in \mathcal{B} \wedge s_{i,j} \in S_i \cap \mathcal{C}\}$, and $a_{out,i}$ denotes the balance of output account in $\mathcal{S}_i$. For simplicity, we further denote $E_i = \{a_{i,j} : (s_{i,j}, a_{i,j}) \in \mathcal{B} \wedge s_{i,j} \in S_i \cap \mathcal{C}\}$. Then, we show how to use $\mathcal{A}$ to solve the DL problem in $\mathbb{G}_q$. Precisely, the discrete logarithm α of h_1 to base h_0 can be computed in the following cases:

- $\forall i \in [\nu]$, $S_i \backslash \mathcal{C} = \emptyset$: this case denoted as case_1 means that all spent accounts are under $\mathcal{A}$'s control. Thus, we know from $\sum_{a_{i,j} \in E} a_{i,j} < \sum_{i=1}^{\nu} a_{out,i}$ that there exists some $i^* \in [\nu]$ s.t. $\sum_{a_{i^*,j} \in E_{i^*}} a_{i^*,j} < a_{out,i^*}$. Note that in this case we have $E_{i^*} = \{a_{i^*,j} : (s_{i^*,j}, a_{i^*,j}) \in \mathcal{B} \wedge s_{i^*,j} \in S_{i^*}\}$. Through the serial numbers in $S_{i^*} = \{s_{i^*,j}\}_{j=1}^m$

we can find the group $\{act_{i^*,j}\}$ of correspondingly spent accounts in A_{i^*}, where each $act_{i^*,j} = (pk_{i^*,j}, cn_{i^*,j})$ and the associated secret key is $ask_{i^*,j} = (sk_{i^*,j}, (r_{i^*,j}, a_{i^*,j}))$. Then we are able to compute the extra public key $\widetilde{pk}_{i^*}$ corresponding to the spent accounts:

$$\begin{aligned}\widetilde{pk}_{i^*} &= \prod_{j=1}^{m} pk_{i^*,j} \cdot \prod_{j=1}^{m} cn_{i^*,j}/cn_{out,i^*}\\ &= \prod_{j=1}^{m} h_0^{sk_{i^*,j}} \cdot \prod_{j=1}^{m} (h_0^{r_{i^*,j}} h_1^{a_{i^*,j}})/(h_0^{r_{out,i^*}} h_1^{a_{out,i^*}})\\ &= h_0^{\sum_{j=1}^{m} sk_{i^*,j} + \sum_{j=1}^{m} r_{i^*,j} - r_{out,i^*}} \cdot h_0^{\alpha(\sum_{j=1}^{m} a_{i^*,j} - a_{out,i^*})},\end{aligned}$$

where $cn_{out,i^*} = h_0^{r_{out,i^*}} h_1^{a_{out,i^*}}$ denotes the output coin in tx_{i^*}. Moreover, according to the definition of balance, we know that $\mathcal{S}_i \notin \mathcal{T}$ for all $i \in [\nu]$, thus by using extractor Ext we can efficiently extract a valid witness $\Big(\{(w_{i^*,j}, z_{i^*,j}, sk_{i^*,j})\}_{j=1}^{m}, (\widetilde{w}_{i^*}, \widetilde{z}_{i^*}, \widetilde{sk}_{i^*}), \gamma_{i^*}\Big)$ s.t. $\widetilde{pk}_{i^*} = h_0^{\widetilde{sk}_{i^*}}$ for the statement w.r.t. $\mathcal{S}_{i^*}$. With $\widetilde{sk}_{i^*}$ the DL instance (h_0, h_1) can be solved by computing

$$\alpha = \frac{\widetilde{sk}_{i^*} - (\sum_{j=1}^{m} sk_{i^*,j} + \sum_{j=1}^{m} r_{i^*,j} - r_{out,i^*})}{\sum_{j=1}^{m} a_{i^*,j} - a_{out,i^*}}.$$

Recall that $\sum_{j=1}^{m} a_{i^*,j} < a_{out,i^*}$, hence we have $\sum_{j=1}^{m} a_{i^*,j} - a_{out,i^*} \neq 0$.

– $\exists\ i \in [\nu]$, $S_i \backslash \mathcal{C} \neq \emptyset$: this case denoted as case_2 means that $\mathcal{A}$ successfully spends some account without her control. Let $\mathcal{J} = \{i \in [\nu] : S_i \backslash \mathcal{C} \neq \emptyset\}$, $\mathcal{S}_{\mathcal{J}} = \bigcup_{i \in \mathcal{J}} (\mathcal{S}_i \backslash \mathcal{C})$ and the size of $\mathcal{S}_i \backslash \mathcal{C}$ be ℓ_i. Then we check if $s_{j^*} \in \mathcal{S}_{\mathcal{J}}$. If not, simply outputs a random guess. Otherwise, there exists $i^* \in \mathcal{J}$ s.t. $s_{j^*} \in S_{i^*}$. To this point, we can use td to extract a valid witness as the above case, where the witness includes sk_{j^*} s.t. $pk_{j^*} = h_0^{sk_{j^*}} = h_0^{\alpha \cdot x_{j^*}}$, and then solve the discrete logarithm of h_1 to h_0 by computing $\alpha = sk_{j^*}/x_{j^*}$.

At last, we give a brief analysis of the probability of $\mathcal{A}'$ successfully solving the DL problem. For clarity, we denote by abort_c the event that $\mathcal{A}'$ aborts in the $\mathtt{Corrupt}$ phase, and $\overline{\mathsf{abort}_c}$ the complementary event. Clearly, if abort_c does not happen, the simulated game is almost identical to the real from the point of $\mathcal{A}$'s view, except that the spending queries $(\mathtt{m}, A_s, A, R)$ s.t. $act_{j^*} \in A_s$ are computed by calling the simulator of SoK. Therefore, $\mathcal{A}$ wins this game with probability at least $\epsilon - \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$. Conditioned on $\overline{\mathsf{abort}_c}$, $\mathcal{A}'$ always succeeds to compute α if $\mathcal{A}$ wins in the first case; otherwise, the probability of $\mathcal{A}'$ to compute α is at least $\Pr[s_{j^*} \in \mathcal{S}_{\mathcal{J}}] = (\sum_{i \in \mathcal{J}} \ell_i)/q_{ad}$. Moreover, from the simulation we know that $\Pr[\overline{\mathsf{abort}_c}] = 1 - q_{co}/q_{ac}$. Combining them together, we get that the probability that $\mathcal{A}'$ succeeds to solve the discrete logarithm of h_1 to h_0 is

$$\begin{aligned}&\Pr[\mathcal{A}'(h_0, h_1 = h_0^{\alpha}) = \alpha]\\ &= \Pr[\mathcal{A}'(h_0, h_1 = h_0^{\alpha}) = \alpha \wedge \overline{\mathsf{abort}_c}] + \Pr[\mathcal{A}'(h_0, h_1 = h_0^{\alpha}) = \alpha \wedge \mathsf{abort}_c]\\ &\geq \Pr[\mathcal{A}'(h_0, h_1 = h_0^{\alpha}) = \alpha | \overline{\mathsf{abort}_c}] \cdot \Pr[\overline{\mathsf{abort}_c}]\\ &= \Pr[\mathcal{A}'(h_0, h_1 = h_0^{\alpha}) = \alpha | \overline{\mathsf{abort}_c}, \mathsf{case}_1] \cdot \Pr[\mathsf{case}_1 | \overline{\mathsf{abort}_c}] \cdot \Pr[\overline{\mathsf{abort}_c}] +\\ &\quad \Pr[\mathcal{A}'(h_0, h_1 = h_0^{\alpha}) = \alpha | \overline{\mathsf{abort}_c}, \mathsf{case}_2] \cdot \Pr[\mathsf{case}_2 | \overline{\mathsf{abort}_c}] \cdot \Pr[\overline{\mathsf{abort}_c}]\\ &\geq \big(\Pr[\mathsf{case}_1 | \overline{\mathsf{abort}_c}] + \Pr[s_{j^*} \in \mathcal{S}_{\mathcal{J}} | \overline{\mathsf{abort}_c}, \mathsf{case}_2] \cdot \Pr[\mathsf{case}_2 | \overline{\mathsf{abort}_c}]\big) \cdot \Pr[\overline{\mathsf{abort}_c}]\\ &= \big(1 + (\tfrac{\sum_{i \in \mathcal{J}} \ell_i}{q_{ad}} - 1) \cdot \Pr[\mathsf{case}_2 | \overline{\mathsf{abort}_c}]\big) \cdot \Pr[\overline{\mathsf{abort}_c}]\\ &\geq \big(1 + (\tfrac{\sum_{i \in \mathcal{J}} \ell_i}{q_{ad}} - 1) \cdot \Pr[\mathcal{A} \text{ wins} | \overline{\mathsf{abort}_c}]\big) \cdot \Pr[\overline{\mathsf{abort}_c}]\\ &= \big(1 - \tfrac{q_{co}}{q_{ac}}\big)\big(1 + (\tfrac{\sum_{i \in \mathcal{J}} \ell_i}{q_{ad}} - 1)(\epsilon - \varepsilon)\big).\end{aligned}$$

Now, the proof of Theorem 1 is completed. □

A.2 Proof for Theorem 2

Proof. The proof proceeds via the following games. The first game is exactly the original game, while the last is such a game that completely hides the information about b. In the following, we will denote by Win_i the event that the adversary guesses correctly in Game$_i$.

Game$_0$: This is actually the original game (cf. Definition 5). In more details, the challenger generates public parameters $pp = \big(1^\lambda, \mathsf{desc}, \mathsf{par}, h_0, h_1, u, \tilde{h}\big)$ by running the $\mathsf{Setup}(1^\lambda)$ algorithm, and responds all queries to `AddGen`, `ActGen`, `Spend` and `Corrupt` as in the real game. Whenever receiving the challenge query $(\mathtt{m}, A_{s_0}, A_{s_1}, A, R)$, A_{s_i} being the s_i-th group of accounts of A for $i \in \{0,1\}$, the challenger chooses $b \in \{0,1\}$ uniformly at random and generates the challenge spending as follows. For sake of clarity, we denote all input accounts as $A = \{(pk_i^{(k)}, cn_i^{(k)})\}_{i\in[n],k\in[m]}$, the s_b-th group of A as $A_{s_b} = \{act_{s_b}^{(k)} = (pk_{s_b}^{(k)}, cn_{s_b}^{(k)})\}_{k\in[m]}$ where the corresponding key to account $act_{s_b}^{(k)}$ is $ask_{s_b}^{(k)} = (sk_{s_b}^{(k)}, (r_{s_b}^{(k)}, a_{s_b}^{(k)}))$, and the set of output addresses as $R = \{pk_{out,j}\}_{j\in[t]}$. Precisely, the challenge spending (tx^*, π^*, S^*) is computed as below:

1. Generate output account $act_{out,j}^{(b)} = (pk_{out,j}, cn_{out,j}^{(b)})$ for all address $pk_{out,j} \in R$ and add $act_{out,j}^{(b)}$ to $A_R^{(b)}$, where $cn_{out,j}^{(b)} = h_0^{r_{out,j}^{(b)}} h_1^{a_{out,j}^{(b)}}$ and the input balance $\{a_{s_b}^{(k)}\}$ and output balance $\{a_{out,j}^{(b)}\}$ satisfy that $\sum_{k=1}^{m} a_{s_b}^{(k)} = \sum_{j=1}^{t} a_{out,j}^{(b)}$.
2. Compute $\widetilde{pk}_i^{(b)} = \prod_{k=1}^{m} pk_i^{(k)} \cdot \prod_{k=1}^{m} cn_i^{(k)} / \prod_{j=1}^{t} cn_{out,j}^{(b)}$ for all $i \in [n]$, the accumulated values $v_k = \mathsf{ACC.Eval}(\mathsf{desc}, \{pk_i^{(k)} \cdot u^i\})$ for all $k \in [m]$ and $v_{m+1}^{(b)} = \mathsf{ACC.Eval}(\mathsf{desc}, \{\widetilde{pk}_i^{(b)} \cdot u^i\})$, as well as the serial numbers $s_k^{(b)} = \tilde{h}^{sk_{s_b}^{(k)}}$ for all $k \in [m]$.
3. Compute $(w_{s_b}^{(k)}, z_{s_b}^{(k)}, sk_{s_b}^{(k)})$ for each $k \in [m]$, where $w_{s_b}^{(k)} = \mathsf{ACC.Wit}(\mathsf{desc}, \{pk_i^{(k)} \cdot u^i | i \neq s_b\})$ and $z_{s_b}^{(k)} \doteq pk_{s_b}^{(k)} \cdot u^{s_b}$. Also, compute $(w_{s_b}^{(m+1)}, z_{s_b}^{(m+1)}, \widetilde{sk}_{s_b}^{(b)})$, where $w_{s_b}^{(m+1)} = \mathsf{ACC.Wit}(\mathsf{desc}, \{\widetilde{pk}_i^{(b)} \cdot u^i | i \neq s_b\})$, $z_{s_b}^{(m+1)} \doteq \widetilde{pk}_{s_b}^{(b)} \cdot u^{s_b}$ and $\widetilde{sk}_{s_b}^{(b)} = \sum_{k=1}^{m} sk_{s_b}^{(k)} + \sum_{k=1}^{m} r_{s_b}^{(k)} - \sum_{j=1}^{t} r_{out,j}^{(b)}$. Then, generate a signature of knowledge π^* on $tx^* = (\mathtt{m}, A, A_R^{(b)})$ as:

$$\mathrm{SoK}\left\{ (\{w_k, z_k, x_k\}_{k=1}^{m+1}, \gamma) : \begin{array}{c} f(w_{m+1}, z_{m+1}) = v_{m+1} \wedge z_{m+1} = h_0^{x_{m+1}} u^\gamma \wedge \\ f(w_1, z_1) = v_1 \wedge z_1 = h_0^{x_1} u^\gamma \wedge s_1 = \tilde{h}^{x_1} \wedge \\ \vdots \\ f(w_m, z_m) = v_m \wedge z_m = h_0^{x_m} u^\gamma \wedge s_m = \tilde{h}^{x_m} \end{array} \right\}(tx^*),$$

   and return the challenge spending (tx^*, π^*, S^*), where $S^* = (s_1^{(b)}, s_2^{(b)}, \cdots, s_m^{(b)})$.

Finally, the adversary outputs its guess b'. Obviously, from the definition of anonymity we have that

$$\Pr[\mathrm{Win}_0] = \Pr[b' = b].$$

Game$_1$: This game is the same as Game$_0$ except that par and all signatures of knowledge are generated by calling the simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimSign})$ of SoK. That is, $(\mathsf{par}, td) \leftarrow \mathsf{SimGen}(1^\lambda)$ and all SoKs w.r.t. `Spend` queries and challenge query are produced without using any witness. More precisely, the challenge query $(\mathtt{m}, A_{s_0}, A_{s_1}, A, R)$ and the admissible spending queries $(\mathtt{m}, A_s, A, R)$ s.t. $A_s \cap A_{s_b} = \emptyset$ are simulated as follows.

For a spending query $(\mathtt{m}, A_s, A, R)$, the challenger computes the statement $(v_1, v_2, \cdots, v_{m+1}, s_1, s_2, \cdots, s_m)$ together with the corresponding witness $(\{w_k, z_k, x_k\}_{k=1}^{m+1}, s)$, and then computes $\pi = \mathsf{Sim}(tx, (v_1, v_2, \cdots, v_{m+1}, s_1, s_2, \cdots, s_m))$ without using the witness. Similarly, when receiving a challenge query $(\mathtt{m}, A_{s_0}, A_{s_1}, A, R)$, the challenger computes the associated statement and witness as in the previous game, but generates the SoK as $\pi^* = \mathsf{Sim}(tx^*, (v_1, v_2, \cdots, v_{m+1}^{(b)}, s_1^{(b)}, s_2^{(b)}, \cdots, s_m^{(b)}))$.

Clearly, by the SimExt-security of SoK we get that

$$|\Pr[\mathrm{Win}_1] - \Pr[\mathrm{Win}_0]| \leq negl(\lambda).$$

Game$_2$: The difference of this game from Game$_1$ is that during the challenge phase the output coin w.r.t. address $pk_{out,j} \in R$ is chosen uniformly at random. More specifically, in this game we set $h_1 = h_0^{\alpha}$ for $\alpha \leftarrow \mathbb{Z}_q$, and compute each output coin $cn_{out,j}$ for address $pk_{out,j} \in R$ by picking $\hat{r}_j \in \mathbb{Z}_q$ uniformly at random and setting $cn_{out,j} = h_0^{\hat{r}_j}$, which implicity sets the corresponding key to $cn_{out,j}$ as $ck_{out,j} = (\hat{r}_j - \alpha \cdot a_{out,j}^{(b)},\ a_{out,j}^{(b)})$. In this case, $\widetilde{pk}_i = \prod_{k=1}^{m} pk_i^{(k)} \cdot \prod_{k=1}^{m} cn_i^{(k)} / \prod_{j=1}^{t} cn_{out,j}$ for all $i \in [n]$ is completely independent of b, and so is $v_{m+1} = \mathsf{ACC.Eval}(\mathsf{desc}, \{\widetilde{pk}_i \cdot u^i\})$. Thus, by the perfectly hiding property of HCom_P, we get that

$$\Pr[\mathrm{Win}_2] = \Pr[\mathrm{Win}_1].$$

Game$_3$: This game differs from the previous only in the generation of serial numbers. In this game, the serial numbers for the challenge query are chosen uniformly at random. More specifically, $s_k \leftarrow \mathbb{G}_q$ for each $k \in [m]$, instead of $s_k = \tilde{h}^{sk_{s_b}^{(k)}}$. Recall that no accounts in A_{s_b} is corrupted and no spending queries $(\mathtt{m}, A_s, A, R)$ s.t. $A_s \cap A_{s_b} \neq \emptyset$ is permitted, so the serial numbers $S^* = (s_1, s_2, \cdots, s_m)$ are fresh to the adversary and they are uniformly distributed from $\mathcal{A}$'s view. Moreover, the simulated SoK $\pi^* = \mathsf{Sim}(tx*, (v_1, v_2, \cdots, v_{m+1}, s_1, s_2, \cdots, s_m))$ is completely independent of b. Therefore, the simulated spending (tx^*, π^*, S^*) in this game leaks no information about b and thus $\Pr[b' = b] = 1/2$.

Lemma 1. *Under the DDH assumption, Game$_2$ is computationally indistinguishable from Game$_3$. More precisely, for all $\lambda \in \mathbb{N}$ and PPT distinguishers $\mathcal{D}$, it holds that*

$$|\Pr[\mathrm{Win}_3] - \Pr[\mathrm{Win}_2]| \leq 2mq_{ac} \cdot Adv_{\mathcal{D}}^{\mathrm{DDH}}(\lambda),$$

where m denotes the number of spent accounts in each transaction.

Proof. To prove the above lemma, we first introduce a sequence of games, denoted by Game$_{2,i}$ for $i \in \{0, 1, \cdots, m\}$. In more details, these games are defined as follows.

Game$_{2,i}$: it is almost identical to Game$_2$ except that the first i serial numbers (w.r.t. challenge query) are chosen uniformly at random, instead of computing as $s_k = \tilde{h}^{sk_{s_b}^{(k)}}$. Obviously, for the case $i = 0$ it is exactly Game$_2$, and for $i = m$ it is the same as Game$_3$. Thus, to the end we only need to show that every two consecutive games Game$_{2,i}$ and Game$_{2,i+1}$ are computationally indistinguishable. Next, we present a reduction of this to the standard DDH assumption.

Suppose that there exists an adversary $\mathcal{A}$ that can distinguish Game$_{2,i}$ and Game$_{2,i+1}$ with non-negligible probability, then we can design an efficient distinguisher $\mathcal{D}$ to solve the DDH problem.

Given a random DDH instance (g, g^a, g^b, g^c) for $a, b, c \leftarrow \mathbb{Z}_q$, the distinguisher $\mathcal{D}$ generates desc and par by calling $\mathsf{ACC.Gen}(1^\lambda)$ and $\mathsf{Gen}(1^\lambda)$ respectively, sets $h_0 = g$ and $\tilde{h} = g^b$, and then picks random generators $h_1, u \in \mathbb{G}_q$. After that, it outputs parameters $pp = \left(1^\lambda, \mathsf{desc}, \mathsf{par}, h_0, h_1, u, \tilde{h}\right)$. Without loss of generality, we assume that $\mathcal{A}$ makes at most q_{ad}, q_{ac} and q_{co} queries to $\mathtt{AddGen}$, $\mathtt{ActGen}$ and $\mathtt{Corrupt}$ respectively. $\mathcal{D}$ first randomly picks $j^* \in [q_{ad}]$, and chooses uniformly at random $x_i \in \mathbb{Z}_q$ for all $i \in [q_{ad}]$, then $\mathcal{D}$ sets the queried addresses as

$$pk_i = \begin{cases} h_0^{x_i}, & i \neq j^* \\ (g^a)^{x_i}, & i = j^* \end{cases}$$

where the corresponding secret key for pk_i is implicitly set as $sk_i = x_i$ for all $i \in [q_{ad}] \backslash \{j^*\}$ and $sk_{j^*} = a \cdot x_{j^*}$. After that, $\mathcal{D}$ simulates all oracles in the following way:

– $\mathtt{AddGen}(i)$: for the i-th query, simply returns address pk_i.

- $\mathtt{ActGen}(i, a_i)$: on input index i and amount a_i, randomly chooses $r_i \in \mathbb{Z}_q$ and sets $c_i = h_0^{r_i} h_1^{a_i}$, then returns $(cn_i, ck_i) = (c_i, (r_i, a_i))$ and adds i and $act_i = (pk_i, cn_i)$ to initially empty lists $\mathcal{I}$ and $\mathcal{G}$, respectively.
- $\mathtt{Spend}(\mathtt{m}, A_s, A, R)$: on input transaction string $\mathtt{m}$, input accounts A containing $A_s \subset \mathcal{G}$ and output addresses R, $\mathcal{D}$ generates (tx, π, S) as in Game$_3$ regardless of whether $act_{j^*} \in A_s$ or not. Recall that signature of knowledge π is generated currently by invoking the simulator Sim of SoK.
- $\mathtt{Corrupt}(i)$: on input query $i \in \mathcal{I}$, $\mathcal{D}$ aborts if $i = j^*$, otherwise returns x_i.

When receiving challenge query $(\mathtt{m}, A_{s_0}, A_{s_1}, A, R)$ s.t. $A_{s_i} \in A$ and $A_{s_i} \subset \mathcal{G}$, $\mathcal{D}$ checks if $act_{j^*} \in A_{s_i}$ for $i \in \{0, 1\}$. If not, $\mathcal{D}$ halts and outputs a random guess. Else, it picks $b \leftarrow \{0, 1\}$ and returns a random bit if $act_{j^*} \notin A_{s_b} \vee act_{j^*} \neq act_{s_b}^{(i+1)}$, otherwise uses A_{s_b} to answer this query as follows:

1. For each $pk_{out,j} \in R$, chooses $cn_{out,j}$ uniformly at random and sets $\widetilde{pk}_i = \prod_{k=1}^{m} pk_i^{(k)} \cdot \prod_{k=1}^{m} cn_i^{(k)} / \prod_{j=1}^{t} cn_{out,j}$ for all $i \in [n]$.
2. Computes $(v_1, v_2, \cdots, v_m, v_{m+1})$ and sets $s_1, s_2, \cdots, s_i \leftarrow \mathbb{G}_q$, $s_{i+1} = (g^c)^{x_{j^*}}$ and $s_k = \tilde{h}^{x_k}$ for all $k \in \{i+2, i+3, \cdots, m\}$, then simulates $\pi^* = \mathsf{Sim}(tx*, (v_1,\ v_2, \cdots, v_{m+1}, s_1, s_2, \cdots, s_m))$ and returns (tx^*, π^*, S^*).

Eventually, the adversary $\mathcal{A}$ output its guess b', and $\mathcal{D}$ outputs 1 if $b' = b$.

For easy analysis, we denote by E_1 the event that $(act_{j^*} \in A_{s_0}) \vee (act_{j^*} \in A_{s_1})$ and E_2 the event that $act_{j^*} \in A_{s_b} \wedge act_{j^*} = act_{s_b}^{(i+1)}$. Conditioned on E_1 and E_2, $\mathcal{D}$ perfectly simulates Game$_{2,i}$ if (g, g^a, g^b, g^c) is a valid DDH instance, in which case it holds that $s_{i+1} = (g^c)^{\cdot x_{j^*}} = (g^{ab})^{x_{j^*}} = \tilde{h}^{sk_{j^*}} = \tilde{h}^{sk_{s_b}^{(i+1)}}$. Otherwise, $\mathcal{D}$ perfectly simulates Game$_{2,i+1}$, in which case we have $s_{i+1} = (g^c)^{x_{j^*}}$ that is uniformly distributed from $\mathcal{A}$'s view. Thus, it is easy to get from the above discussion that

$$\begin{aligned} & |\Pr[\mathrm{Win}_{2,i}] - \Pr[\mathrm{Win}_{2,i+1}]| \\ &= |\Pr[\mathcal{D}(g, g^a, g^b, g^{ab}) = 1 | \mathsf{E}_1 \wedge \mathsf{E}_2] - \Pr[\mathcal{D}(g, g^a, g^b, g^c) = 1 | \mathsf{E}_1 \wedge \mathsf{E}_2]| \\ &= \tfrac{1}{\Pr[\mathsf{E}_1 \wedge \mathsf{E}_2]} |\Pr[\mathcal{D}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{D}(g, g^a, g^b, g^c) = 1]| \\ &= 2q_{ac} \cdot Adv_{\mathcal{D}}^{\mathrm{DDH}}(1^\lambda), \end{aligned}$$

where m is the number of spent accounts in each transaction.

At last, from the above analysis we get that

$$|\Pr[b' = b] - 1/2| \leq 2mq_{ac} \cdot Adv_{\mathcal{D}}^{\mathrm{DDH}}(1^\lambda) + negl(\lambda).$$

Hence, under the DDH assumption and securities of building blocks, our protocol is anonymous in the random oracle model. □

A.3 Proof for Theorem 3

Proof. The proof of this theorem is similar to that of Theorem 1. We can embed a random DL instance into a simulated game, such that if there exists an adversary that can slander an honest spending, then it can be used to design an efficient algorithm to solve the DL problem with non-negligible probability. For completeness, we present the details as follows.

Suppose for contradiction that there is an efficient adversary $\mathcal{A}$ that can break the non-slanderability of our RingCT protocol with non-negligible probability ϵ, then we can use $\mathcal{A}$ as a subroutine to design an efficient algorithm $\mathcal{A}'$ to solve the DL problem in $\mathbb{G}_q$.

Given a random DL instance $(h_0, h_1 = h_0^\alpha)$, the algorithm $\mathcal{A}'$ produces desc and par by running the algorithms $\mathsf{ACC.Gen}(1^\lambda)$ and $\mathsf{Gen}(1^\lambda)$ respectively. It then chooses uniformly at random $\beta \in \mathbb{Z}_q$, computes $\tilde{h} = h_0^\beta$ and sets $pp = (\mathsf{desc}, \mathsf{par}, h_0, h_1, u, \tilde{h})$. Without loss of generality, we assume that $\mathcal{A}$ makes at most q_{ad}, q_{ac} and q_{co} queries to $\mathtt{AddGen}$, $\mathtt{ActGen}$

and `Corrupt`, respectively. First, $\mathcal{A}'$ randomly picks $j^* \in [q_{ad}]$ and chooses uniformly at random $x_i \in \mathbb{Z}_q$ for all $i \in [q_{ad}]$, and then sets the queried addresses as

$$pk_i = \begin{cases} h_0^{x_i}, & i \neq j^* \\ h_1^{x_i}, & i = j^* \end{cases}$$

where the associated secret key with pk_i is implicitly set as $sk_i = x_i$ for all $i \in [q_{ad}] \backslash \{j^*\}$ and $sk_{j^*} = \alpha \cdot x_{j^*}$. After that, $\mathcal{A}'$ simulates all oracles in the following way:

– `AddGen`(i): for the i-th query, directly returns address pk_i.
– `ActGen`(i, a_i): on input index i and an amount a_i, randomly chooses $r_i \in \mathbb{Z}_q$ and sets $c_i = h_0^{r_i} h_1^{a_i}$. Then, sends back $(cn_i, ck_i) = (c_i, (r_i, a_i))$ and adds i, $act_i = (pk_i, cn_i)$ and $s_i = \tilde{h}^{sk_i}$ to lists $\mathcal{I}$, $\mathcal{G}$ and $\mathcal{S}$, respectively.
– `Spend`$(\mathtt{m}, A_s, A, R)$: on input a spending query $(\mathtt{m}, A_s, A, R)$ s.t. $A_s \subset \mathcal{G}$, $\mathcal{A}'$ generates (tx, π, S) as in real RingCT protocol with the corresponding account keys K_s if $act_{j^*} \notin A_s$. Otherwise, $\mathcal{A}'$ generates π by invoking the simulator Sim of SoK with the appropriately accumulated values and corresponding serial numbers (to A_s) as input. By the SimExt security of SoK, the simulated spending without witness is computationally indistinguishable from the real.
– `Corrupt`(i): on input index $i \in \mathcal{I}$, $\mathcal{A}'$ aborts if $i = j^*$. Otherwise, it returns x_i and adds s_i to corruption list $\mathcal{C}$.

Finally, $\mathcal{A}$ outputs $\big((\hat{tx}, \hat{\pi}, \hat{S}), (tx^*, \pi^*, S^*)\big)$. Suppose that $\mathcal{A}$ succeeds in this simulated game, meaning that the output of $\mathcal{A}$ satisfies that (1) $\mathsf{Verify}(tx^*, \pi^*, S^*) = 1$; (2) $(tx^*, \pi^*, S^*) \notin \mathcal{T}$; (3) $\hat{S} \cap \mathcal{C} = \emptyset$ and $\hat{S} \cap S^* \neq \emptyset$. Next, we show how to use $\mathcal{A}$ to solve the DL problem in $\mathbb{G}_q$. Precisely, the discrete logarithm α of h_1 to base h_0 can be computed in the following way: check if $j^* \in \hat{S} \cap S^*$, simply outputs a random guess if it does not hold; otherwise, we can use trapdoor td to extract a valid witness w.r.t. (tx^*, π^*, S^*) as before, where the witness contains sk_{j^*} satisfying $pk_{j^*} = h_0^{sk_{j^*}} = h_0^{\alpha \cdot x_{j^*}}$. Then, the discrete logarithm α of h_1 to h_0 can be computed as $\alpha = sk_{j^*}/x_{j^*}$.

At last, we briefly analyze the probability of $\mathcal{A}'$ solving the DL problem. For clarity, we denote by abort_c the event that $\mathcal{A}'$ aborts in the `Corrupt` phase, and $\overline{\mathsf{abort}_c}$ the complementary event. Obviously, if abort_c does not happen, the simulated game is the same as the real from the point of $\mathcal{A}$'s view, except that the spending queries $(\mathtt{m}, A_s, A, R)$ s.t. $act_{j^*} \in A_s$ are answered by calling the simulator of SoK. Thus, $\mathcal{A}$ wins this game with probability at least $\epsilon - \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$. In this case, if $j^* \in \hat{S} \cap S^*$ holds, $\mathcal{A}'$ can successfully compute the discrete logarithm α of h_1 to h_0. Given the above, we get that the probability that $\mathcal{A}'$ succeeds to solve the discrete logarithm of h_1 to h_0 is

$$\begin{aligned}
&\Pr[\mathcal{A}'(h_0, h_1 = h_0^{\alpha}) = \alpha] \\
&\geq \Pr[\mathcal{A} \text{ wins} \wedge j^* \in \hat{S} \cap S^*] \\
&\geq \Pr[\mathcal{A} \text{ wins} \wedge j^* \in \hat{S} \cap S^* | \overline{\mathsf{abort}_c}] \cdot \Pr[\overline{\mathsf{abort}_c}] \\
&= \Pr[\mathcal{A} \text{ wins} | \overline{\mathsf{abort}_c}] \cdot \Pr[j^* \in \hat{S} \cap S^* | \overline{\mathsf{abort}_c}] \cdot \Pr[\overline{\mathsf{abort}_c}] \\
&\geq \tfrac{\epsilon - \varepsilon(\lambda)}{q_{ad}}.
\end{aligned}$$

Hence, under the DL assumption the proposed protocol is non-slanderable in the random oracle. □