IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 14, 2020, accepted October 24, 2020, date of publication October 28, 2020, date of current version November 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3034333

ARCT: An Efficient Aggregating Ring Confidential Transaction Protocol in Blockchain

JUNKE DUAN, LIZE GU, AND SHIHUI ZHENG
School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
Corresponding author: Junke Duan (duanjunke@bupt.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB080301.

ABSTRACT Anonymous cryptocurrency is a branch of cryptocurrency designed to protect users' privacy. Ring Confidential Transaction (RingCT) protocol is widely used in anonymous cryptocurrencies (e.g. Monero) to hide transaction information. However, its expensive computing and storage costs slow down the performance of the system. In this work, we propose a high-efficient RingCT protocol which is called aggregation ring confidential transaction (ARCT). In our protocol, we build a compact aggregation proof for multiple accounts of the sender, which enables our protocol to linearly shorten the signature size compared to other protocols. In addition, in order to further protect the privacy information, we first implement hiding the amount of the sender's accounts. Our protocol can provide cryptocurrencies with better privacy protection and lower resource cost. Besides, it does not require a trust setting. For a typical n=2 input transaction with the ring size of 128, the signature size of our protocol is 93% less than it used in the original protocol. ARCT build on the techniques of inner product optimization algorithm and cryptographic accumulators. We show that our underlying algorithm satisfies the security under the random oracle model.

INDEX TERMS Blockchain, cryptocurrency, data privacy, ringCT.

I. INTRODUCTION

Blockchain [1] was originally used as the underlying technology of Bitcoin by Satoshi Nakamoto in 2008. Blockchain used to stored transaction ledger. Its security depends on cryptographic tools, i.e., hash function, public-key encryption, and digital signature. A blockchain can be thought of as a distributed ledger, which is maintained by nodes that are distributed, each with the same complete ledger. Blockchain uses consensus algorithms to ensure data consistency between nodes without relying on a trusted third party, which greatly reduces the cost of validation and audit transactions. Since each block in the chain is connected to the previous block through a hash pointer, no one can tamper with the contents of the historical block, ensuring the authenticity of the data on the chain.

Recently, blockchain-based applications have a broad prospect in communication, finance, medical treatment, education, and other fields, and many blockchain-based technologies have been proposed for these applications [33], [34]. However, most applications based on blockchain technology still have privacy problems. Since the user's information recorded by record nodes is public, each participant can access all the transaction data. It leads to a series of privacy protection issues of the blockchain. Although users can hide their identities by changing their account addresses, since every transaction's input are all linked to the previous transaction's output, it is still possible to trace the identities of users by analyzing the records of funds transferred. A study by Princeton University in the United States [18] shows that in the Bitcoin system, if users use Bitcoin to conduct transactions with online merchants that accept digital currency, the merchants can easily associate users' account address with their cookies, to find the real identity of users. [17] indicates that account addresses can be linked to IP addresses, then completely de-anonymizing their owners. Therefore, some means are needed to break the link relationship between the spender and receiver of one transaction, while ensuring that the verifier verifies the transaction's legitimacy properly.

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

A. OUR CONTRIBUTIONS

In this article, we propose an efficient ring confidential transaction protocol called ARCT. Under the requirements of privacy protection of the spender's address and transaction amount, our new design provides further privacy protection and significantly reduces the operation and storage

costs of the algorithm. Our work can be summarized as follows:

1) We present an implementation of a linkable ring signature protocol. The complexity of our protocol is the lowest of its kind. The reason is that it can hide the signer's multiple keys in a ''ring'', rather than one key for one ''ring'' as in other schemes (In a ring signature algorithm, a ''ring'' usually represents a set of keys that includes the signer's key). Furthermore, we use inner product optimization based on [14] to reduce the signature size to logarithmic. Our ring signature protocol is used as part of ARCT to hide user accounts. The verifier cannot determine which public keys belong to the signer. Even so, he can still verify that whether the spender knows the secret keys corresponding to her accounts and determine whether the sum of input amounts is equal to it of output amounts.
2) We design the creation and validation of aggregated key-images. The key-image is usually generated by the user's private information (e.g. user's private key). It is used to prove that the user's amount has been spent. The verifier cannot get the spender's information from the key-image. A typical way [11], [19], [25] is to verify that each key-image in the transaction did not exist in the key-image list. Our new design provides one short witness generated by the spender's key-images. Validation only needs a constant number of group element operations to determine whether they has been used rather than searching the entire list.
3) Our protocol implements that hide the number of spender's accounts. Due to we allow multiple addresses to be hidden in one ring, it will expose more information to the adversary. Because the adversary needs to get the information of other account owners in the ring to determine the sender's address, as the number of hidden addresses increases, the attacker's attack condition will become easier. In our design, the number of spender's accounts is not exposed in both signature or witness of key-images. It makes the security of our protocol stronger.

B. ORGANIZATION

The remainder of this article is organized as follows. Section II briefly introduces the related work. Section III reviews some mathematical assumptions and definitions. Section IV gives an overview of the systems that use our protocols and some definitions of basic primitives. Section V gives a complete implementation of our protocol. Section VI compares the efficiency of our scheme with other schemes. Finally, Section VII takes a short conclusion.

II. RELATED WORK

A. MIXING PROTOCOL

In recent years, there have been some privacy protection protocols for blockchain privacy problems. One of them [2]–[6] is mixing protocol which mixes the tokens of different spenders and redistributes them to the receiving address, so as to break the association between the input address and the output addresses. Mixing protocol can be divided into centralized mixing and decentralized mixing. The former requires a trusted third party to act as an online mixer and swap different users' inputs and outputs. The advantage is that the third party will charge extra service fees, so the centralized mixed coin protocol can resist the witch attack. The disadvantage is that users need to pay extra mixed fees and there is a risk that third parties will steal users' tokens. The latter mixes coins between participants save the extra mixing costs but it is susceptible to Sybil attack.

B. NIZK PROOFS

Zerocoin [7] is a cryptocurrency project for privacy protection, which uses the method of non-interactive zero-knowledge proofs (NIZK) to converts the general tokens into commitments corresponding to their information. This allows verifiers to verify the transactions without any useful information, which to some extent protects the private information of the transaction party. However, the converted amount cannot be split again, which limits the application of Zerocoin. The Ethereum [27] also uses NIZK technology to protect the privacy of users' information in smart contracts. However, these protocols that use NIZK make the use of smart contracts expensive. Based on [7], Zerocash [8] use zk-SNARKs (zero-knowledge succinct non-interactive argument of knowledge) to provide anonymity with formal security proof. For achieving better privacy protection, the transaction amount, and input address in [8] were kept secret. However, it requires a complex trusted setup to result in common reference strings (CRS) which is specific to each application, and possesses trapdoors. For example, In Hawk [28], different smart contracts require different CRS, its generation requires trusted third-party or expensive multi-party computation.

C. CONFIDENTIAL TRANSACTIONS

CryptoNote [10] uses the one-time address to hide the address of the transaction receiver and a linkable ring signature [15] to hide the address of the transaction sender. Based on it, Shen Noether built ring confidential transactions (RingCT) [11] for the currently popular anonymous cryptocurrency Monero, which is one of the largest cryptocurrencies. Upon the enhancement of privacy, a major trade-off is the increase of size for the transaction. As shown in Table 1, compared with other schemes, it protects the user's privacy to the greatest extent.

Although ring confidential transactions provides strong anonymity, it has the problem of excessive volume. The size of the transaction increases linearly with the number of ring members and the spender's account addresses, which greatly increases the storage overhead of the system. Aiming at this problem, [19] put forward RingCT 2.0 which is built upon the well-known Pedersen commitment, accumulator with one-way domain and signature of knowledge.

TABLE 1. Comparison of privacy technologies.

technology	Hidden content			Project
	spender	receiver	Transaction's content	
Mixing	√	√	×	Coin Suffle
Zk-SNARKs	√	×	√	Zcash
Confidential Trsactions	√	√	√	Monero

By accumulating the elements in the ring into a single piece of evidence, it shortens the size of the transaction. To solve the problem of low efficiency caused by the large amount of zero-knowledge proof in [19], [32] proposes a practical privacy protection protocol based on [21].

Confidential transactions [12] hide the transaction amount into a commitment. Although this technique protects the privacy of users, but it also causes some trouble in transaction verifying. The verifier is no longer able to verify the relationship between inputs and outputs, and that all transaction values are positive. Because the amounts in cryptocurrencies are represented by group elements, it means that a negative amount is equal to a huge positive amount. SNARKS [8] can be used to solve this problem, but they require a trusted setup. Early Monero [11] used range proof which is based Borromean ring signature [29]. It is less efficient because it needs to be constructed a binary ring signature for each bit of a 32-bit amount. Based on the techniques of Bootle et al. [30], Benedikt et al. proposed bulletproofs [14], a non-interactive zero-knowledge proof protocol with very short proofs. Without a trusted setup, it optimizes the size of the proofs to a logarithmic relationship with the bits of amounts.

D. KEY-IMAGE

In anonymous cryptocurrency, the spender has to prove that she spent the amount without disclosing the content of the transaction. The natural idea is to construct evidence of the amount. Just like a serial number, the key-image is used to uniquely represent an amount. In Monero [11], it's represented by $\boldsymbol{sk}\cdot\boldsymbol{H_p}(\boldsymbol{pk})$ where the function $\boldsymbol{H_p}$ is a $\mathbb{G} \rightarrow \mathbb{G}$ hash function. The verifier verifies that the current key-image exists in the set of key-images that have been used. If it passes, then put it to the set. Similar approaches are used in [8], [19], [25]. But unlike UTXO [1], elements in the set of key-images cannot be delete because, for users' privacy, the verifier cannot know its corresponding account. It causes the validation time increases linearly with the elements in the set.

III. PRILIMINARIES

A. NOTATIONS

We give some notations we are going to use in later sections in table 2.

TABLE 2. Some notations we're going to use.

Notation	Description
$\boldsymbol{negl(\lambda)}$	A negligible function of the security parameter $\boldsymbol{\lambda}$
$\boldsymbol{x \xleftarrow{R} S}$	Choosing a random element $\boldsymbol{x \in S}$ uniformly
$\boldsymbol{a}$	A vector of elements and $\boldsymbol{a_i}$ it the $\boldsymbol{i}$th element of $\boldsymbol{a}$
$\boldsymbol{a \circ b}$	The Hadamard product of $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{Y^a}$	Denote $\sum_{i=1}^{n} \boldsymbol{Y_i}^{\boldsymbol{a_i}}$ where $\boldsymbol{i}$ is the number of elements of both $\boldsymbol{Y}$ and $\boldsymbol{a}$
$\boldsymbol{l^n}$	A vector of length $\boldsymbol{n}$ and the $\boldsymbol{i}$th element equal to $\boldsymbol{l^i}$
$\boldsymbol{\lvert\mathbb{Z}_p\rvert}$	The length of element in $\boldsymbol{\mathbb{Z}_p}$

B. MATHEMATICAL ASSUMPTIONS

Definition 1:(Discrete Logarithm Problem). Let $\mathbb{G}$ be a group where the order of $\mathbb{G}$ is q. There exists no probabilistic polynomial time (PPT) algorithm that can find an integer x such that $h = g^x$, where $g, h \xleftarrow{R} \mathbb{Z}_q$.

Definition 2:(Decisional Diffie-Hellman (DDH) Assumption). Let $\mathbb{G}$ be a group where the order of $\mathbb{G}$ is p. $g \in G$ is the generator of $\mathbb{G}$. There exists no PPT algorithm that is given (g, g^a, g^b, g^c) can determine if $g^c = g^{ab}$ with non-negligible probability, where $a, b, c \xleftarrow{R} \mathbb{Z}_p$.

Definition 3:(Strong RSA Assumption). Let n be an RSA modulus and $x \xleftarrow{R} \mathbb{Z}_n$. There exists no PPT algorithm that can find $e > 1$ and $y \in \mathbb{Z}_n^*$ such that $e^y = x \ mod \ n$.

C. HOMOMORPHIC COMMITMENT

A homomorphic commitment scheme consists of two phases: commitment and validation. In the commitment step, the spender selects a value and constructs a commitment about it, the spender can choose to reveal the value of the commitment in the reveal phase. After that, the verifier can verify that it is indeed the originally committed value. A homomorphic commitment scheme consists of a pair of polynomial algorithms (*Setup*, *Com*).

- $pp \leftarrow Setup(1^\lambda)$: on input a security parameter λ, it outputs public parameters pp for the scheme. which specifies a message space $\mathbb{M}_{pp}$, a randomness space $\mathfrak{R}_{pp}$ and a commitment space $\mathbb{C}_{pp}$.
- $C \leftarrow Com(m, r)$: on input a message $m \in \mathbb{M}_{pp}$ and a randomness $x \in \mathcal{R}_{pp}$, generates a commitment $c \in \mathbb{C}_{pp}$.

From [21], a homomorphic commitment $HCom = (CKGen, Com)$ satisfies the following definitions:

Definition 4:(Hiding). the value of the original message cannot be found by commitment. More precisely, for all PPT adversaries $\mathcal{A}$, it holds that:

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

where challenger $\mathcal{C}$ is a PPT algorithm, $m_0, m_1 \in M$, if $\mathcal{A}$ has exactly 1/2 chance of guessing b, then the $HCom$ has perfect hiding.

Definition 5:(Binding). the same commitment cannot correspond to two different values. More precisely, for all PPT

adversaries $\mathcal{A}$, it holds that:

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

where $m_0, m_1 \in M$, r_0, r_1 are random chosen. If the probability is exactly 0, then the *HCom* has perfectly binding.

Definition 6:(Homomorphism). we assume that the commitment space $\mathbb{C}_{pp}$ is a group of order p. For all $m_0, m_1 \xleftarrow{R} \mathbb{Z}_p$; $r_0, r_1 \xleftarrow{R} \mathbb{Z}_p$, it holds that:

$$Com\left(r_0, m_0\right) + Com\left(r_1, m_1\right) = Com\left(r_0 + r_1, m_0 + m_1\right)$$

D. UNIVERSAL ACCUMULATORS

A cryptographic accumulator is a primitive that produces a short binding commitment to a set of elements. The prover can produce short membership or non-membership proofs for the element in the set. These proofs can be publicly verified against the commitment. Merkle tree [31] is a simple accumulator widely used for membership proofs of transactions in the blockchain. However, it only supports membership proofs, which limits its use in blockchain privacy transactions. The reason is, in the verification phase of a private transaction, the verifier often has to prove that an account is not membership of the account set that has been spent, rather than to prove that it's a member of the account set that has not been spent.

An Accumulator which supports both membership and non-membership proofs is called a universal accumulator. It consists of a tuple of the following polynomial-time algorithms:

- $(pp, aux) \leftarrow \boldsymbol{Gen}(1^{\lambda})$: on input a security parameter λ, it returns system public parameters pp and some auxiliary information, denoted as aux.
- $(A_{t+1}, S_{t+1}) \leftarrow \boldsymbol{Add}(A_t, S_t, x)$: on input a current accumulator A_t, an element x from the odd primes domain, and a set S of accumulated elements, it returns a new accumulator A_{t+1} and a new set S_{t+1}.
- $(A_{t+1}, S_{t+1}) \leftarrow \boldsymbol{Del}(A_t, S_t, x)$: its description is the same as above.
- $u_x \leftarrow \boldsymbol{MemCreate}(A, S, x)$: on input an accumulator $\mathcal{A}$, an element x from the odd primes domain, and a set S of accumulated elements, it returns a witness $u_x = (a, B)$.
- $u_x \leftarrow \boldsymbol{NonMemCreate}(A, S, x)$: its description is the same as above.
- $1/0 \leftarrow \boldsymbol{VerMen}(A, u_x, x)$: on input an accumulator $\mathcal{A}$, an element x from the odd primes domain, and a witness u_x, it returns 1 if the verification passes, else returns 0.
- $1/0 \leftarrow \boldsymbol{VerNonMen}(A, u_x, x)$: its description is the same as above.

Assume all inputs belong to the set $\mathbb{X}_{\lambda}$. A universal accumulator protocol satisfies the following definitions:

Definition 7:(Quasi-Commutative). For an algorithm $f \in \{\boldsymbol{Add}, \boldsymbol{Del}\}$, and for all $x_1, x_2 \in \mathbb{X}_{\lambda}$, $f\left(f\left(A_t, x_1\right), x_2\right) = f\left(f\left(A_t, x_2\right), x_1\right)$.

Definition 8:(Security). A universal accumulator scheme is secure if, for all probabilistic polynomial-time adversary A_{λ},

$$\Pr\left[\begin{array}{l} 1 \leftarrow \boldsymbol{VerMen}\left(A, u_1, x\right); \\ 1 \leftarrow \boldsymbol{VerNonMen}\left(A, u_2, x\right): \\ \left(u_1, u_2\right) \leftarrow A_{\lambda}\left(A, S, x\right), x \in \mathbb{X}_{\lambda} \end{array}\right] = negl(\lambda)$$

It means that it is computationally infeasible to find both a valid membership witness and a valid non-membership witness for any $\boldsymbol{x}$ in $\mathbb{X}_{\boldsymbol{\lambda}}$.

IV. SYSTEM MODEL

In this section, we introduce the system model from two parts. We will first give an overview of a privacy-protected transaction system by using ARCT protocol, then we give some definitions of the algorithms in our protocol.

A. OVERVIEW OF THE SCENARIO

There are three types of participants in our system: spenders, receivers, and verifiers. Informally, a receiver can generate a long-time key pair. For different transactions, the receiver generates different one-time public keys to be his receiving address by using the long-time key pair. When the spender wants to transfer money to the receiver, she constructs a transaction by the following steps:

- Generate a Pedersen commitment for the amounts that she will transfer, then use the commitment and the receiver's one-time public key to construct the receiver's account.
- Use the spender's one-time secret key to scan the UTXO (unspent transaction outputs) to find her accounts. Select some accounts of these so that the input amount is equal to the output amount.
- Generate the content of the transaction which contains the accounts of both parties.
- Ask the verifier (It also called complete node which stores the complete data of blocks) for other users' accounts.
- Construct the witness of the spender's key-images.
- Construct the signature of the transaction by using the accounts collected above.

After constructing the transaction, the spender publishes it to the network by broadcasting. The verifier listens and collects the transactions that are broadcast to the network. When he receives a transaction, he performs the following verification steps:

- Verify the witness to determine whether the spender has spent the same account twice.
- Verify the signature with public information (the content of the transaction) to determine whether spender knows the secret keys corresponding to her accounts and the input amount is equal to the output amount.

Note that during the above validation process, the verifier does not know the identity of the spender and the receiver,

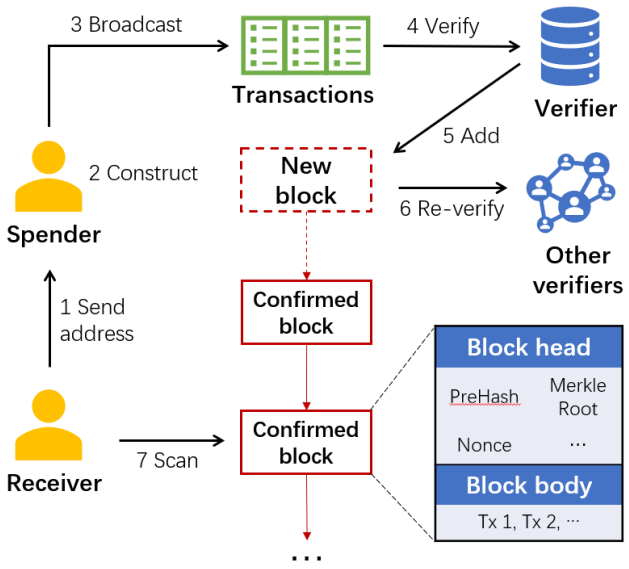


FIGURE 1. an overview of a privacy-protected transaction system by using ARCT protocol.

the transfer amount, and the number of spender's accounts, but that did not prevent him from verifying the deal.

When a transaction is broadcast to the network, multiple verifiers competitively process the transactions. Because the verifiers are distributed in different places, they package different transactions into their blocks. The consensus algorithms, e.g., the Proof of Work ensures ledger consistency between them. Specifically, if a verifier has verified a batch of transactions, before he packages these transactions into blocks, he has to solve a computational puzzle which is to compute a hash value of the block header's information that must satisfy certain conditions (e.g., less than a certain value). It can only be done by trying different random numbers until the conditions are satisfied. When someone figures out a qualified hash value, then his block becomes the newest block of the blockchain. He will broadcast his block to other nodes.

After receiving the block, other verifiers need to verify the block again to ensure the block is compliant. If the verification is successful, other verifiers will add new blocks to their local blockchain, and those who produce the block will be rewarded. Assuming most participants were honest, by using the consensus algorithm, the verifiers could jointly maintain a unified ledger. It's ensuring the security of the system. An overview of the system's work is shown in Figure 1.

B. DEVICES

In this system, participants are divided into light nodes and full nodes according to the storage capacity and computing capacity of their devices. All nodes communicate via P2P protocol. The light node is similar in function to the client, it doesn't have to be online all the time, and it only holds the block heads of the blocks. The light nodes cannot participate in the validation of the transaction and can only create transactions and receive transactions of its own. Some mobile devices and IoT devices can perform the function of light nodes. The full nodes need to stay online and maintain complete blockchain information. They maintain complete routing information to broadcast new blocks, listen and verify transactions. The full node is usually implemented by a server or specialized hardware facility.

C. DEFINITIONS

We give the security models for our ARCT protocol based on [19]. Our protocol consists of a tuple of polynomial time algorithms (***Setup***, ***KeyGen***, ***Mint***, ***AccGen***, ***Spend***, ***Verify***), the syntax of which are described as follows:

- $pp \leftarrow \textbf{\textit{Setup}}\left(1^{\lambda}\right)$: on input a security parameter λ, it outputs the system parameters pp which are used for the later algorithms.
- $(pk, sk) \leftarrow \textbf{\textit{KeyGen}}()$: the algorithm outputs corresponding private-public key pair (pk, sk). Similar to Monero [11], it can be divided into two parts: generating long-term key pairs and one-time key pairs. The former is used for users to receive money transfers and the is used by the sender to construct the transaction.
  $(ltpk, ltsk) \leftarrow \textbf{\textit{LongTermKeyGen}}()$: it outputs a long-term secret key $ltsk$ and a long-term public key $ltpk$.
  $(pk, aux_{ot}) \leftarrow \textbf{\textit{OneTimePKGen}}(ltpk)$: on input a long-term public key $ltpk$, it outputs a one-time key pk and an auxiliary information aux_{ot}.
  $sk \leftarrow \textbf{\textit{OneTimeSKGen}}(pk, ltsk, aux_{ot})$: on input a one-time key pk, a long-term secret key $ltsk$, and an auxiliary information aux_{ot}, it outputs one-time secret key sk.
- $(C, ck) \leftarrow \textbf{\textit{Mint}}(pk, v)$: on input an amount v and a public key pk, the algorithm outputs a commitment C for pk as well as the associated commitment key ck.
- $(act, ask) \leftarrow \textbf{\textit{AccGen}}(sk, pk, C, ck, v)$: on input a commitment C with key ck and value v, a key pair (pk, sk), outputs an account $act = (pk, C)$, the corresponding secret key of which is $ask = (sk, ck, v)$.
- $(tx, sig, img) \leftarrow \textbf{\textit{Spend}}(m, A_\pi, K_\pi, A, V_{out})$: on input a set A_π of accounts with the corresponding set of account secret keys K_π, a set $\mathcal{A}$ of input accounts containing A_π, a set V_{out} of output amounts and some transaction string $m \in \{0, 1\}^*$, the algorithm outputs a transaction tx (containing m, A, A_{out}), a signature sig and an aggregating key-image img.
- $1/0 \leftarrow \textbf{\textit{Verify}}(m, tx, sig, img)$: on input transaction tx, signature sig, the aggregating key-image img and some transaction string m, the algorithm verifies whether a set of accounts with serial numbers S is spent properly for the transaction tx towards addresses A_{out}, and outputs 0/1 when the spending is in/valid.

Definition 9:(Correctness). This property requires that the verifier can correctly verify the legal signature generated by

the signer. it holds that:

$$\Pr\left[\begin{array}{c} Verify\,(tx, sig, img) = 1 : \\ for\ i \in [1, m],\ (pk_i, sk_i) \leftarrow KeyGen\,(1^\lambda), \\ (C_i, ck_i) \leftarrow Mint\,(pk_1, v_i), \\ (act, ask) \leftarrow AccGen\,(pk_i, sk_i, C_i, ck_i, v_i); \\ A_\pi = \{act_i\}_{i\in[1,m]}; K_\pi = \{ask_i\}_{i\in[1,m]}; \\ (tx, sig, img) \leftarrow Spend(m, A_\pi, K_\pi, A, A_{out}) \end{array}\right] = 1$$

Definition 10:(Unforgeability). This property requires that no attacker can generate a valid transaction using only public information. In other words, for all PPT adversaries $\mathcal{A}$, it holds that:

$$\Pr\left[\begin{array}{c} Verify\,(tx, sig, IMG) = 1 : \\ (tx, sig, img) \leftarrow A(m, A_\pi, A, A_{out}) \end{array}\right]$$

Definition 11:(Anonymity). This property requires that no PPT adversaries $\mathcal{A}$ be able to determine the index of the sender's account in the ring. Specifically, it holds that:

$$\Pr[A_\pi \cap A'_\pi \neq \emptyset : A'_\pi \leftarrow A\,(tx, sig, img)\,;\,] \leq negl(\lambda)$$

where A'_π is a set of act, and m, n is the number of the elements in A_π and A.

Definition 12:(Linkability). If two transactions contain same spender's accounts and they are both expended in the corresponding transaction, we say the two transactions are linkable. In other word, for all PPT adversaries $\mathcal{A}$, it holds that:

$$\Pr\left[\begin{array}{c} A_{\pi,1} \cap A_{\pi,2} \neq \emptyset : for i = 1, 2, \\ Verify(tx_i, sig_i, img_i) = 1, \\ (tx_i, sig_i, IMG_i) \leftarrow \\ \mathcal{A}(m_i, A_{\pi,i}, K_{\pi,i}, A_i, A_{out,i}) \end{array}\right] \leq negl(\lambda)$$

where $\{0, 1\} \leftarrow \Sigma$ is a PPT algorithm.

V. OUR RINGCT PROTOCOL

In this section, we introduce the instantiation of our protocol. We will first give the design of our aggregated key-images protocol, then we give the design of our inner-product proof algorithm which is used to hide the sender's public keys. Finally, we combine the algorithms in the first two parts to give the complete implementation of our protocol.

A. AGGREGATED KEY-IMAGES

In this part, we propose the design of our aggregated key-images protocol. Our protocol is based on the batching non-membership proofs in [26]. Informally, it consists of two phases: commitment and validation. In the commitment phase, the prover first computes her key-images by the corresponding key-pairs, then she calculates the aggregation of these key-images and generates a non-membership proof of it. In order to reduce the cost of validation, the proof contains some zero-knowledge proof protocols. In the validation phase, the verifier first executes the verification algorithm, if it passes, then he adds the aggregated key-images to the used set by using the corresponding algorithm.

Formally, our protocol contains a tuple of polynomial time algorithms: ($Setup$, $KeyGen$, Add, $NonMemCreate^*$, $VerNonMem^*$). Note all accumulated values are odd primes. The syntax of which are described as follows:

- $\mathbb{G} \leftarrow \boldsymbol{Setup}(1^\lambda)$: on input a security parameter λ, it outputs a group $\mathbb{G}$ with a generator g and a set of odd primes $\mathbb{X}_{prime}$.
- $x \leftarrow KeyGen()$: it picks a key-image $x \xleftarrow{R} \mathbb{X}_{prime}$.
- $(A_{t+1}, \boldsymbol{S_{t+1}}) \leftarrow \boldsymbol{Add}(A_t, \boldsymbol{S_t}, \boldsymbol{X} = \{x_1, \ldots, x_n\})$: on input a current accumulator A_t, a set $\boldsymbol{X}$ contains n odd prime elements and a set $\boldsymbol{S_t}$ of accumulated elements, if $x \in \boldsymbol{S_t} \cap x \in \boldsymbol{X}$, it returns $(A_t, \boldsymbol{S_t})$, else compute $x^* = \prod_{i=1}^n x_i$, $\boldsymbol{S_{t+1}} = \boldsymbol{S_t} \cup \{x^*\}$, $A_{t+1} = A_t^{x^*}$, return $(A_{t+1}, \boldsymbol{S_{t+1}})$.
- $u_{x^*} \leftarrow \boldsymbol{NonMemCreate^*}(A, \boldsymbol{S}, \boldsymbol{X} = \{x_1, \ldots, x_n\})$: on input an accumulator A, a set X contains n odd prime elements, and a set S of accumulated elements, it performs:

1) Compute $s^* = \prod_{s\in\boldsymbol{S}} s$, $x^* = \prod_{i=1}^n x_i$.
2) Compute $a, b = Bezout(s^*, x^*)$.
3) Compute $V = A^a$, $B = g^b$.
4) Perform a proof of knowledge protocol $\pi_V \leftarrow PoK$: $\{a : V = A^a\}$.
5) Perform a proof of exponent protocol $\pi_g \leftarrow PoE$: $\{x^*, b, g \cdot V^{-1} : x^{*b} = g \cdot V^{-1}\}$.
6) Output $u_x = \{V, B, \pi_V, \pi_g\}$.

The instantiations of PoK and PoE are introduced in appendix B.

- $1/0 \leftarrow \boldsymbol{VerNonMem^*}(A, u_x = \{V, B, \pi_V\})$: on input an accumulator A and a witness u_x, it performs the verification algorithms of PoK and PoE. If they are both true, return 1, else return 0.

In this way, the verification process only needs a constant number of group element multiplication, so it can reduce the complexity of validation from n to 1. We will give a detailed comparison in section VII.

B. INNER-PRODUCT PROOF

In this part, we propose a zero-knowledge proof to prove that prover's public keys are in a group. Our starting point is a binary vector $\boldsymbol{a_L} = (a_1, \ldots, a_n)$, where the indexes of element "1" of $\boldsymbol{a_L}$ are represent spender's public keys indexes in a public-key group. For checking that the elements of $\boldsymbol{a_L}$ contain only 1 and 0, we define $\boldsymbol{a_R} = \boldsymbol{a_L} - 1^n$, and construct a zero-knowledge proof for the following conditions:

$$\boldsymbol{a_L} \circ \boldsymbol{a_R} = 0^n \quad (1)$$

$$\boldsymbol{a_L} - \boldsymbol{a_R} = 1^n \quad (2)$$

Then we pick $y \xleftarrow{R} \mathbb{Z}_p$ so that $\boldsymbol{b} = 0^n$ is equals to $< \boldsymbol{b}, y^n >= 0$ with an overwhelming probability, so we convert previous equations to:

$$< \boldsymbol{a_L}, \boldsymbol{a_R} \circ y^n > = 0 \quad (3)$$

$$< \boldsymbol{a_L} - 1 - \boldsymbol{a_R}, y^n > = 0 \quad (4)$$

Pick $z \xleftarrow{R} \mathbb{Z}_p$ so that we can use these above equations as a linear combination of z. we have:

$$z < \boldsymbol{a_L} - 1 - \boldsymbol{a_R}, y^n > + < \boldsymbol{a_L}, \boldsymbol{a_R} \circ y^n >= 0 \quad (5)$$

By converting the two inner products of $\boldsymbol{a_L}$ and $\boldsymbol{a_R}$ to one inner product, we have:

$$< \boldsymbol{a_L} - z \cdot 1^n, (\boldsymbol{a_R} + z \cdot 1^n) \circ y^n >= \left(z - z^2\right) < 1^n, y^n >$$

For simplicity, we set:

$$\boldsymbol{L} = \boldsymbol{a_L} - z \cdot 1^n \quad (6)$$
$$\boldsymbol{R} = (\boldsymbol{a_R} + z \cdot 1^n) \circ y^n \quad (7)$$
$$\delta = \left(z - z^2\right) < 1^n, y^n > \quad (8)$$

Note that δ is made up of common random parameters, the verifier can computer δ first, then accepts $\boldsymbol{L}$ and $\boldsymbol{R}$ from the prover and verifies that $< \boldsymbol{L}, \boldsymbol{R} >= \delta$. If it's true, the verifier believes that $\boldsymbol{a_L}$ is a binary vector.

Now it can verify whether $\boldsymbol{a_L}$ is a binary vector by verify an inner product. But the above proof process exposes the value of $\boldsymbol{a_L}$ and $\boldsymbol{a_R}$, so we set two random vectors $\boldsymbol{s_L}$ and $\boldsymbol{s_R}$ to hide $\boldsymbol{a_L}$ and $\boldsymbol{a_R}$, as follows:

$$\boldsymbol{L_x} = \left(\boldsymbol{a_L} - z \cdot 1^n\right) + \boldsymbol{s_L} \cdot x \quad (9)$$
$$\boldsymbol{R_x} = \left(\boldsymbol{a_R} + z \cdot 1^n + \boldsymbol{s_R} \cdot x\right) \circ y^n \quad (10)$$

This prevents the verifier from getting information about $\boldsymbol{a_L}$ from $\boldsymbol{L_x}$ or $\boldsymbol{R_x}$. These two equations and their inner product can be viewed as polynomials of x. The constant terms of $< \boldsymbol{L_x}, \boldsymbol{R_x} >$ is equals to δ.

Assume a group $\mathbb{G}$ with order p and generators $g, h \xleftarrow{R} \mathbb{G}$. we give a set of public keys $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$ where $Y_i = g^{x_i}$, x_i is the corresponding private key. A vector $\boldsymbol{h}$ which consists of n elements that are logarithmically unknown to each other and the random numbers $\alpha, \rho \in \mathbb{Z}_p$. The prover computes that:

$$A = h^{\alpha} \boldsymbol{Y}^{\boldsymbol{a_L}} \boldsymbol{h}^{\boldsymbol{a_R}} \quad (11)$$
$$S = h^{\rho} \boldsymbol{Y}^{\boldsymbol{s_L}} \boldsymbol{h}^{\boldsymbol{s_R}} \quad (12)$$

The spender sends $\mathcal{A}$ and S to the verifier. It can prove that $\boldsymbol{L}$ and $\boldsymbol{R}$ is well-formed by verifying that:

$$h^{\mu} \boldsymbol{Y}^{\boldsymbol{L}} \boldsymbol{h}'^{\boldsymbol{R}} = A \cdot S^x \cdot \boldsymbol{Y}^{-z \cdot 1^n} \cdot \boldsymbol{h}^{z \cdot y^n} \quad (13)$$

where $\boldsymbol{h}' = (h'_1, \ldots, h'_n)$ and for $i = 1, \ldots, n$, $h'_i = h_i^{y^{-i+1}}$.

By the above process, we get an interactive zero-knowledge proof about hiding user's public keys in a group. By applying the Fiat-Shamir heuristic, we can convert them into non-interactive proofs.

The size of the above proof increases linearly with the number of group members. Further, we use the inner-product optimization algorithm with low communication complexity proposed by [30]. It can prove the knowledge that the prover knows two vectors of Pedersen commitment that satisfy an inner product relation. More precisely, it is an efficient proof for the following relation:

$$\left\{ \begin{array}{l} P = \boldsymbol{g}^{\boldsymbol{a}} \boldsymbol{h}^{\boldsymbol{b}} \wedge c =< \boldsymbol{a}, \boldsymbol{b} >: \\ \boldsymbol{h} \in \mathbb{G}^n, P \in G, c \in \mathbb{Z}_p; \boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^n \end{array} \right\}$$

For simply, we denote a tuple of algorithms $(\boldsymbol{CreateImpIP}, \boldsymbol{VerImpIP})$ to represent its process. we can use it to reduce the proof size to $\boldsymbol{logn}$. The instantiation of it is in appendix C.

C. ARCT PROTOCOL

In this part, we present our new RingCT protocol under the formalized syntax mentioned in Section V. Our scheme is designed as follows:

In the setup step, the system needs to generate some parameters as common variables, formally, it is the following polynomial time algorithm:

$\boldsymbol{Setup}\left(1^{\lambda}\right)$: on input a security parameter λ, it picks a group $\mathbb{G}$ with prime order p and a set of odd primes $\mathbb{X}_{prime}$, and some generators $g, h, h_v \in \mathbb{G}$. The hash functions which be used include $\boldsymbol{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p, \boldsymbol{H_G} : \{0,1\}^* \rightarrow \mathbb{G}, \boldsymbol{H_{prime}} : \{0,1\}^* \rightarrow \mathbb{X}_{prime}$. Compute $\boldsymbol{h} = \{h_1, \ldots, h_n\}$ where $h_1 = \boldsymbol{H_G}(h)$ and $h_i = \boldsymbol{H_G}\left(h_{i-1}\right), i \in \{2, n\}$.

The receiver needs to generate a long-term key pair. Then he sends the long-term public key to the spender. The spender uses it and a random value to generate a one-time public key and some auxiliary information. When the transaction has been recorded in the blockchain, the receiver can use the long-term secret key and the auxiliary information to generate the one-time secret key. The receiver has access to the account only if he has the one-time secret key. In the process of it, because the sender does not have the long-term secret key of the receiver, so she can't spend this account. For others, they couldn't determine whom this one-time key belongs to. Formally, the algorithm $\boldsymbol{KeyGen}$ contains the following polynomial-time algorithms:

$\boldsymbol{LongTermKeyGen}()$: it picks a long term secret key $ltsk = (ltsk_1, ltsk_2) \xleftarrow{R} \mathbb{Z}_p$ and computes long term public key $ltpk = (g^{ltsk_1}, g^{ltsk_2})$.

$\boldsymbol{OneTimePKGen}(ltpk)$: on input a long term public key $ltpk = (ltsk_1, ltsk_2)$, it picks $(r \xleftarrow{R} \mathbb{Z}_p$ and computes one-time public key $pk = g^{ltsk_1} \cdot g^{H(g^{ltsk_2 r})}$, it outputs pk and the auxiliary information $R = g^r$.

$\boldsymbol{OneTimeSKGen}(pk, ltsk, aux_{ot})$: on input a one-time key pk, a long term secret key $ltsk = (ltsk_1, ltsk_2)$, and an auxiliary information R, if $pk = g^{ltsk_1} \cdot g^{H(R^{ltsk_2})}$, it outputs $sk = ltsk_1 + H(R^{ltsk_2})$.

A user's account contains two parts: address and the corresponding amount. The address is represented by the one-time public key. It hides the identity of the user. The amount is represented by Pedersen Commitment [12]. In confidential transactions, Pedersen commitment is used to represent an amount of user. We denote $C = g^r h^v$ where v is used to represent the amount and r is called the blinding factor. Since the value range of v in the actual transfer operation is limited,

if don't set blinding factor r, the attacker can guess the v by the exhaustive method. Once an attacker gets the value of a commitment, then all commitments which hide the same secret value will be revealed. Formally, the following two algorithms describe the process of constructing an account.

Mint(pk, v) : on input an amount v and a public key pk, it picks $ck \xleftarrow{R} \mathbb{Z}_p$ and computes $C = g^{ck}h_v^v$. It returns C and ck.

AccGen(sk, pk, C, ck, v) : on input a commitment $C = g^{ck}h_v^v$ with key ck and value v, a key pair (pk, sk), it outputs an account $act = (pk, C)$, the corresponding secret key of which is $ask = (sk, ck, v)$.

The sender's transfer behavior can be viewed as that to collect the appropriate accounts to satisfy the transaction's input amount equals output amount. She needs to select random numbers for the Pedersen commitments in the output accounts. Then use the transaction's one-time public key to encrypt the random numbers and corresponding values and put them into the content of the transaction. When she has the required input and output accounts ready, she constructs the aggregated key-images and corresponding zero-knowledge proof using the public key in her accounts by using the algorithm in Section VI.A. Then the spender constructs some zero-knowledge proofs of the transaction which are used to prove that:

- The spender has the right to spend the amount in the account, that is, she knows the private keys for her accounts.
- The input amount of a transaction is equal to the output amount.
- The accounts the spender spends correspond to the key-images in the aggregated key-images.

Formally, the algorithm ***Spend*** contains the following polynomial time algorithms:

Spend$(m, A_\pi, K_\pi, A, V_{out})$: on input a set $\boldsymbol{A_\pi} = \{act_{\pi,1}, \ldots, act_{\pi,m}\}$ with the corresponding set of account secret keys K_π, a set $\boldsymbol{A} = \{act_1, \ldots, act_n\}, A_\pi \subseteq A$, a set $\boldsymbol{V_{out}} = \{v_{out,1}, \ldots, v_{out,l}\}$ and some transaction string $m \in \{0,1\}^*$, it performs the following steps:

1) If $\sum_{i=1}^{m} v_{\pi,i} \neq \sum_{i=1}^{l} v_{out,i}$, it returns $\perp$, else it chooses $ck_{out,i} \xleftarrow{R} \mathbb{Z}_q$ for each $v_{out,i}$. Then it computes commitment $C_{out,i} = g^{ck_{out,i}}h_v^{v_{out,i}}, i \in (1, l)$ and generates the set $\boldsymbol{A_{out}} = (act_{out,1}, \ldots, act_{out,l})$ where $act_{out,i} = (pk_{out,i}, C_{out,i})$. The sender adds all $ck_{out,i}, v_{out,i}$ encrypted by pk into the transaction.
2) Generate a range proof of all $v_{out,i}, i \in (1, l)$ by using the technique in [14].
3) Pick $\theta, \varphi \xleftarrow{R} \mathbb{X}_{prime}$ and compute $img = \theta \sum_{i=1}^{m} H_{prime}(pk_{\pi,1})$. Get the set $\boldsymbol{S}$ of img and the accumulator $\mathcal{A}$ from local or other nodes. Execute ***NonMemCreate****.
4) Generate a binary vector $\boldsymbol{a_L} = (a_1, \ldots, a_n)$ where $a_i = 1$ if act_i in A is also a member of A_π. Compute $\boldsymbol{a_R} = a_L - 1^n$ and $u_1 = \boldsymbol{H}(m), u_i = \boldsymbol{H}(u_{i-1}), i \in \{2, n\}$. Pick $\alpha, \rho \xleftarrow{R} \mathbb{Z}_p, \boldsymbol{s_L}, \boldsymbol{s_R} \xleftarrow{R} \mathbb{Z}_p^n$. Then compute:

$$\boldsymbol{Y_1} = \left\{pk_1^{u_1}C_1, \ldots, pk_n^{u_n}C_n\right\} \tag{14}$$

$$\boldsymbol{Y_2} = \{\boldsymbol{H_{prime}}(pk_1), \ldots, \boldsymbol{H_{prime}}(pk_n)\} \tag{15}$$

$$A_1 = h^\alpha \sum_{i=1}^{m} pk_{\pi,i}^{u_{\pi,i}} C_{\pi,i} \tag{16}$$

$$A_2 = h^\beta \boldsymbol{h^{a_R}} \tag{17}$$

$$S_1 = h^\rho \boldsymbol{Y_1^{s_L}} \boldsymbol{h^{s_R}} \tag{18}$$

$$S_2 = \varphi \boldsymbol{Y_2^{s_L}} \tag{19}$$

5) Compute the string:

$$str = m||\boldsymbol{Y_1}||\boldsymbol{Y_2}||A_1||A_2||S_1||S_2 \tag{20}$$

$$y = \boldsymbol{H}(str), w = \boldsymbol{H}(y), z = \boldsymbol{H}(w) \tag{21}$$

$$\mu_1 = \alpha + w\beta + x\rho \tag{22}$$

6) Assume there are two polynomials of a variable x, we have:

$$\boldsymbol{L_x} = a_L - z \cdot 1^n + x \cdot s_L \tag{23}$$

$$\boldsymbol{R_x} = (w \cdot a_L + wz \cdot 1^n + s_R \cdot x) \circ y^n \tag{24}$$

$$t_x = < L_x, R_x > \tag{25}$$

Note that this step doesn't generate the value of x. By these polynomials, it can be confirmed that;

$$t_x = t_o + t_1 x + t_2 x^2 \tag{26}$$

7) Pick $\tau_1, \tau_2 \xleftarrow{R} \mathbb{Z}_p$, and compute $T_1 = g^{t_1}h^{\tau_1}, T_2 = g^{t_2}h^{\tau_2}, x = \boldsymbol{H}(T_1, T_2)$. Then compute $\tau_x = \tau_1 x + \tau_2 x^2, \mu_2 = \alpha + \tau_1, \mu_3 = \sum_{i=1}^{m} sk_{\pi,i}u_{\pi,i} + ck_{\pi,i} - \sum_{i=1}^{l} ck_{out,i} + t_1, \mu_4 = \theta\varphi^x$ and $\boldsymbol{L_x}, \boldsymbol{R_x}$.
8) The algorithm outputs a transaction $tx = (m, \boldsymbol{A}, \boldsymbol{A_{out}})$, a signature:

$$sig = (\mu_1, \mu_2, \mu_3, \mu_4, \tau_x, A_1, A_2, S_1, S_2, \boldsymbol{L_x}, \boldsymbol{R_x}) \tag{27}$$

and an aggregating key-image img.

- $1/0 \leftarrow$ ***Verify***(tx, sig, img): on input a transaction tx, a signature sig and the aggregating key-image img, the algorithm verifies:

1) Get the accumulator $\mathcal{A}$ from local and perform ***VerNonMem****to check the non-membership proof of img.
2) Check the range proof of all $v_{out,i}, i \in (1, l)$.
3) Compute $\boldsymbol{Y_1}, \boldsymbol{Y_2}$ by public parameters, then compute $str = m||\boldsymbol{Y_1}||\boldsymbol{Y_2}||A_1||A_2||S_1||S_2, \delta = w\left(z - z^2\right) < 1^n, y^n >$ and challenges $y = \boldsymbol{H}(str), w = \boldsymbol{H}(y), z = \boldsymbol{H}(w), x = \boldsymbol{H}(T_1, T_2)$. Define $\boldsymbol{h'} = (h'_1, \ldots, h'_n)$ where $h'_i = h_i^{y^{-i+1}}$. Then check that:

$$t_x = < \boldsymbol{L_x}, \boldsymbol{R_x} > \tag{28}$$

$$h^{\tau_x}g^{t_x} = g^\delta \cdot T_1^x \cdot T_2^{x^2} \tag{29}$$

$$h^{\mu_1}\boldsymbol{Y_1^{L_x}}\boldsymbol{h'^{R_x}} = A_1 \cdot A_2^w \cdot S_1^x \cdot \boldsymbol{Y_1^{-z \cdot 1^n}} \cdot \boldsymbol{h'^{z \cdot y^n}} \tag{30}$$

$$h^{\mu_2}g^{\mu_3} = A_1 \cdot T_1 / \prod_{i=1}^{l} C_{out,i} \tag{31}$$

TABLE 3. Compare of RingCT Schemes.

Ref.	Spender (exponentiation)	Verifier (exponentiation)	Communication ($\|\mathbb{Z}_p\|$)
[11]	$2(m+1)n$	$2(m+1)n$	$(m+1)(n+2)$
[19]	$(m+1)$ $\cdot(n+1)$	$(4\lambda+n+4)$ $\cdot(m+1)$	$(7\lambda+12)$ $\cdot(m+1)$
[25]	$6mn+2logmn$ $+7$	$2logmn+8$	$m+2logmn$ $+17$
[32]	$(m+1)\cdot$ $(n+7logn+1)$	$(m+1)\cdot$ $(n+7logn+3)$	$(m+1)\cdot$ $(7logn+3)$
Ours	$2n+3logn+m$ $+8$	$3logn+11$	$m+2logn+9$

$\boldsymbol{n}$: The number of accounts required to construct one signature.
$\boldsymbol{m}$: The number of accounts of the sender.
λ: The number of bits of the group element

$$\mu_4 \boldsymbol{Y_2^{L_x}} = IMG \cdot S_2^x \cdot \boldsymbol{Y_1^{-z\cdot 1^n}} \tag{32}$$

If these equations are all true, perform ***add*** in Section 3.5 to update the state of the accumulator and return 1, otherwise return 0.

We give a brief explanation of the verification process of the above equations. Equation (29) is used to verify that $\boldsymbol{L_x}$ and $\boldsymbol{R_x}$ are honestly generated. Equation (30) is used to verify that A_1 is honestly generated. Equation (31) is used to verify that the spender knows the secret keys corresponding to her accounts. Because her secret keys are involved in the creation of μ_3. It also verifies that the input amount equals to the output amount. Because there are no h_v's terms in the equation (31), it verifies whether the h_v's term in A_1 is equal to it in C_{out}. Finally, the equation (32) is used to verify the key-images correspond to the spender's account.

By using the technique in Section 3.4, the size of $\boldsymbol{L_x}, \boldsymbol{R_x}$ in signature *sig* can be optimized from n to $logn$. For intuition, we keep the original content of the signature before optimization. The correlation instantiations of the optimization algorithms are in appendix C.

VI. EFFICIENCY ANALYSIS

In this section, we briefly compare the efficiency of our protocol with others. The simulations are conducted on an Intel i5-8265U 1.3GHz, 8GB machine with Windows 10. We use the PyNacl python library for Ed25519 in our implementation. The length of the element in $\mathbb{Z}_p$ is represented by 33 bytes, and it in $\mathbb{G}$ is represented by 32 bytes. For making the communication cost intuitive, we unify the units in the interaction into the length of the element in $\mathbb{Z}_p$. We only care about exponential operations on group elements because other operations are much less expensive than that (In the simulation, an exponential operation on group elements takes about 1ms, an addition operation on it takes about 4μs, and a general hash function takes about 0.1μs). We give a comparison of the calculation and communication cost of the whole protocol in Table 3.

The linkable ring signature used in the scheme [11], [19], [25], [32] needs to construct a ring signature for each account of the sender, so their costs of construction and validation increase linearly with the input accounts of the spender. Because our protocol allows multiple user accounts to be hidden in the same ring, the costs of them do not change as the sender's account increases. For communication cost, Scheme [19] use the one-way accumulator

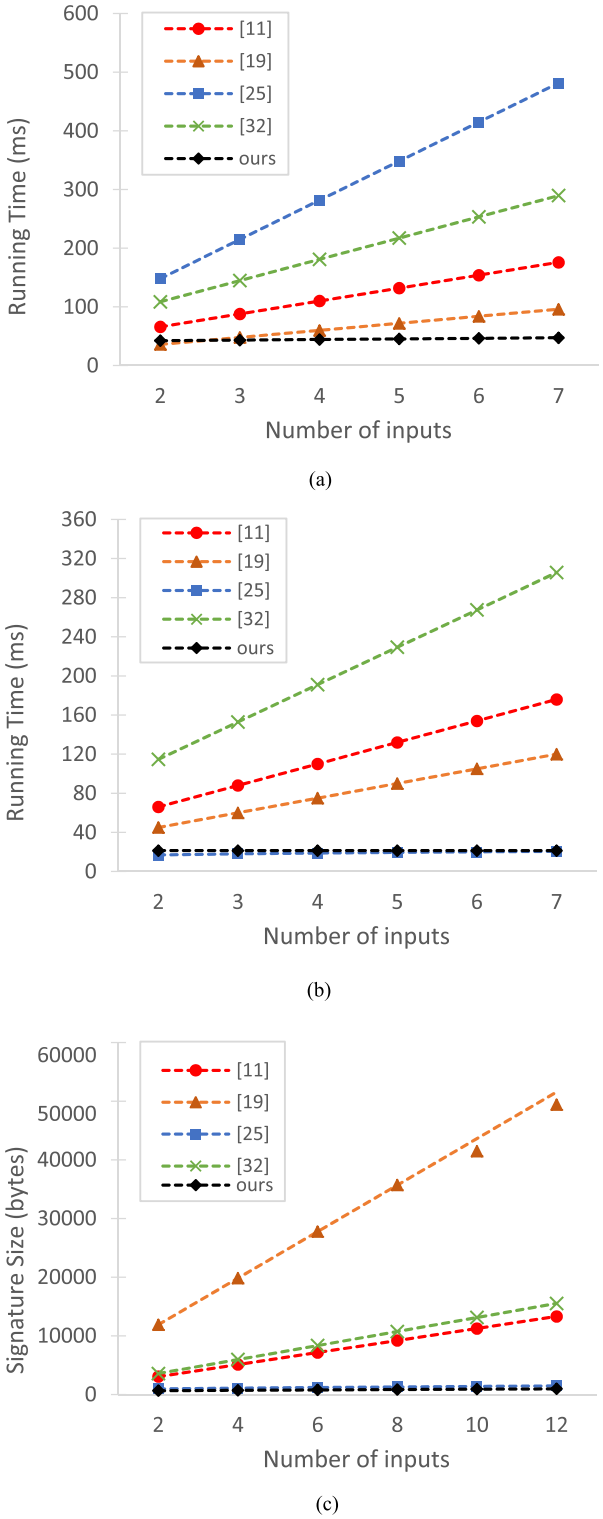


FIGURE 2. A. Signing time's comparison under increased inputs
B. Verifying time's comparison under increased inputs
C. Signature size's comparison under increased inputs.

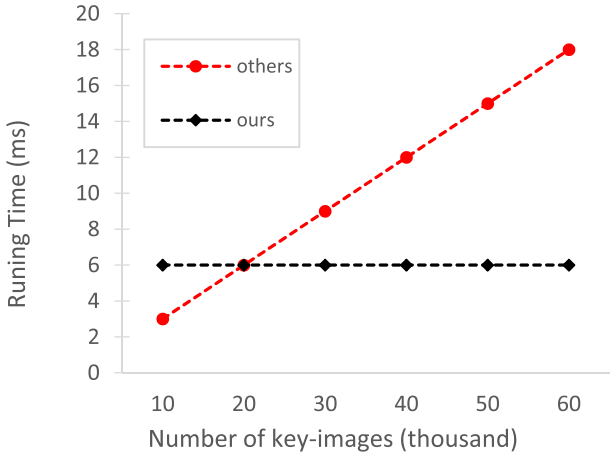


FIGURE 3. Verifying time's comparison under increased key-images.

to compress the size of each group of signatures, whose communication complexity is about $\boldsymbol{O(m)}$. But the knowledge signature it constructs is associated with the length $\boldsymbol{\lambda}$ of the group element. When $\boldsymbol{n < \lambda}$, the communication complexity is still higher than that of [11]. [32] reduce the number of groups to logarithm based on sigma protocol and muti-signature. [25] also reduces the signature size to logarithm by using the technique in [30], but the signature sizes of them are still related to $\boldsymbol{m}$. In our scheme, the above complexity only increases as $\boldsymbol{n}$ multiplies the logarithm. For a 2-inputs and a ring size of 128 transaction, our interaction cost of ringCT is approximately 0.8kB. which is 93% less than it in Monero (12kB).

The efficiency analysis is divided into two parts. First, we compared the cost of construction, validation, and communication between our protocol and others under the same ring size (ring size of 11) and increased inputs. Second, we compare the cost of construction and validation of aggregated key-images. The comparison can be seen in Figure 2 and Figure 3.

From Figure 2.A, we can see that when the spender constructs the signature, in addition to our protocol, the running times of protocols [11], [19], [25], [32] are increase linearly with the input, with protocol [25] increasing the fastest. The advantage of our protocol grows with the inputs.

From Figure 2.B, we can see that when verifying the signature, the running times of protocols [11], [19], [32] are increase linearly with the input. For fewer inputs, the running time of the protocol [25] is slightly lower than that of our protocol. However, as the number of inputs increases, it will eventually be higher than that of our protocol. The comparison between signature size from Figure 2.C is similar to Figure 2.B.

From the above comparison, we can see that our protocols running time is lower than the protocol [11], [19], [25], [32] in the constructing signature phase. For verifying time and signature size, our protocol's Our protocol is more efficient than [11], [19], [32] and closer to [25]. With more spender's accounts, we have a more obvious advantage.

For key-images' verification, other schemes all traverse the set to see if there are equal elements, so its running time increases linearly as you increase the elements of the set. Every time a transaction passes validation, some new key-images are added to the set. In our protocol, verifying them needs just about six exponential operations on group elements. It doesn't change as the increasing of elements in the set. In our simulation, an exponential operation of group element costs about 1ms, and an operation of access and compare costs about 0.3μs. We assume that 10k transactions are recorded per day and each transaction has 1 key-image added to the set. When the system executes for one month, the execution time of our algorithm will be reduced to 1/15 of that of the traditional method.

VII. CONCLUSION

In this work, we design a high-efficient ring confidential transaction protocol to protect users' privacy information in the blockchain. We define and implement the underlying cryptographic primitives of our protocol, and describes the application of the algorithm in anonymous cryptocurrency. Compared with other RingCT protocols, our scheme has lower computing and storage costs, especially when the transaction involves multiple transfer's sub-accounts. Our protocol is based on the inner product optimization algorithm and cryptographic accumulator. We prove the security of the underlying primitives under the random oracle model.

In future work, we expect that put our algorithm into more applications beyond what was discussed, also, we hope to further optimize its performance.

APPENDIX

A. SECURITY PROOFS

Our proofs are based on random oracle model [15] and adaptive chosen-message attack [13]. Some parameters like g, p have been defined in the previous article, and omitted from notations. For simply, we prove for the case that spender has one account act_π. Other cases are similar and not be discussed.

Formally, we give the security proofs of our protocol by defining a security game played between an adversary $\mathcal{A}$ and a simulator $\mathcal{M}$. Assume that both $\mathcal{A}$ and $\mathcal{M}$ receive their respective problems. They perform the following steps:

Setup: The simulator $\mathcal{M}$ prepares some public and secret information, such as public keys and secret keys. The adversary $\mathcal{A}$ has access to get public information.

Query: The adversary $\mathcal{A}$ can adaptively query some hash values or signatures to the simulator $\mathcal{M}$. $\mathcal{A}$ would not query the same message twice.

Challenge: The adversary $\mathcal{A}$ generates an output to her problem. If it passes the validation, then the simulator $\mathcal{M}$ gets the solution to his problem.

Proof of Theorem 10 (Unforgeability): Suppose there is a PPT adversary $\mathcal{A}$ which has public information and a PPT simulator $\mathcal{M}$ which is given a discrete logarithm problem (g, g^a) and need to find a.

Setup: The simulator $\mathcal{M}$ prepares some public-secret key pairs from corresponding domains. Then $\mathcal{M}$ sets $Y^* = g^a$ with unknown corresponding secret key. $\mathcal{M}$ also uses these public keys and some randomly generated Pedersen commitments to make up the accounts.

Query: If the adversary $\mathcal{A}$ queries the $\boldsymbol{H}$ oracle, $\boldsymbol{H_G}$ oracle and $\boldsymbol{H_{prime}}$ oracle, the simulator $\mathcal{M}$ returns random values from their respective domains. When $\mathcal{A}$ queries the ***spend*** oracle with input $(m, act_\pi, \boldsymbol{A}, Y_{out})$, if $\mathcal{M}$ knows the secret keys corresponding to act_π, it executes the ***spend*** algorithm honestly and outputs the valid signature. Else if $Y_\pi = Y^*$, $\mathcal{M}$ chooses $\beta \xleftarrow{R} \mathbb{Z}_p$ and sets $\mu_3 = \beta u_\pi + ck_\pi - \sum_{i=1}^{l} ck_{out,i} + t_1$. The other parameters $(\mu_1, \mu_2, \rho, \tau_1, \tau_2, \theta, \varphi)$ are chosen randomly from their respective domains. $\mathcal{M}$ sets the values of (A_1, A_2, S_1) and the challenges (w, x, y, z) to satisfy the equation (3). Then $\mathcal{M}$ honestly computes other parameters and outputs a valid signature.

Challenge: The adversary $\mathcal{A}$ outputs (tx, sig, img) to the simulator $\mathcal{M}$. If g^a does not belongs to tx, $\mathcal{M}$ returns failure and exits. If $verify\,(tx, sig, img) = 1$, then based on the forking lemma [BCC16], $\mathcal{M}$ rewinds the $\boldsymbol{H}$ oracle which used to outputs $u_\pi = \boldsymbol{H}(u_{\pi-1})$, then outputs $u'_\pi = \boldsymbol{H}(u_{\pi-1})$. By computing with $u_\pi, u'_\pi, \mu_3, \mu'_3$, we have:

$$\frac{\mu_3 - \mu'_3}{u_\pi - u'_\pi} = \frac{\left(\frac{au_\pi + ck_\pi - \sum_{i=1}^{l} ck_{out,i} + t_1}{-au'_\pi - ck_\pi + \sum_{i=1}^{l} ck_{out,i} - t_1}\right)}{u_\pi - u'_\pi} = a$$

Therefore $\mathcal{M}$ gets an instantiation to solve discrete logarithm with probability $1/n$, which contradicts the discrete logarithm assumption.

Proof of Theorem 11 (Anonymity): Assume that a PPT adversary $\mathcal{A}$ is given (tx, sig, img). Note that tx only contains the input and output accounts $\boldsymbol{A}, \boldsymbol{A_{out}}$ without spender's information. The key-image img is a composite number multiplied by some prime number. Finding spender's information from it is equivalent to the solving prime factorization problem. Therefore, we analyze the situation in which $\mathcal{A}$ finds information from the signature.

Suppose there is a PPT simulator $\mathcal{M}$ which is given a DDH problem (g, g^a, g^b, g^c) and need to determine if $g^c = g^{ab}$.

Setup: The simulator $\mathcal{M}$ prepares some public-secret key pairs from corresponding domains. Then $\mathcal{M}$ sets $Y^* = g^a$ with unknown corresponding secret key. $\mathcal{M}$ also uses these public keys and some randomly generated Pedersen commitments to make up the accounts.

Query: If the adversary $\mathcal{A}$ queries the $\boldsymbol{H}$ oracle, $\boldsymbol{H_G}$ oracle and $\boldsymbol{H_{prime}}$ oracle, the simulator $\mathcal{M}$ returns random values from their respective domains, then $\mathcal{M}$ stores the queries about $\boldsymbol{H}$ oracle into a set $\boldsymbol{Q_H}$. When $\mathcal{A}$ queries the ***spend*** oracle with input $(m, act_\pi, \boldsymbol{A}, Y_{out})$, if $\mathcal{M}$ knows the secret keys corresponding to act_π, it executes the ***spend*** algorithm honestly and outputs the valid signature, then $\mathcal{M}$ stores the queries about $\boldsymbol{H}$ oracle into $\boldsymbol{Q_H}$. Else if $Y_\pi = Y^*$, $\mathcal{M}$ declares failure and exits.

Challenge: The adversary $\mathcal{A}$ gives an input $(m, \boldsymbol{A}, \boldsymbol{Y_{out}})$ to the simulator $\mathcal{M}$. If Y^* does not belongs to $\boldsymbol{A}$ or $m \in \boldsymbol{Q_H}$, $\mathcal{M}$ declares failure and exits. Else in the signature phase, $\mathcal{M}$ sets $A_1 = h^\alpha g^c C_\pi$ where C_π is corresponding to $Y_\pi = Y^*$. $\mathcal{M}$ can set the other elements and challenges adaptively from their respective domains in order to satisfy the verification.

If the PPT adversary $\mathcal{A}$ can distinguish the index π of Y_π, then the simulator $\mathcal{M}$ can determine if $g^c = g^{ab}$ so that $\mathcal{M}$ gets an instantiation to solve the DDH problem. It contradicts the DDH assumption.

Proof of Theorem 12 (Linkability): Assume there is a PPT adversary $\mathcal{A}$ which can construct two valid transactions using same Y_π and a PPT simulator $\mathcal{M}$ which is given a strong RSA problem g and need to find d and e such that $d^e = g$.

Setup: The simulator $\mathcal{M}$ prepares some public-secret key pairs from corresponding domains. $\mathcal{M}$ also uses these public keys and some randomly generated Pedersen commitments to make up the accounts.

Query: If the adversary $\mathcal{A}$ queries the $\boldsymbol{H}$ oracle, $\boldsymbol{H_G}$ oracle and $\boldsymbol{H_{prime}}$ oracle, the simulator $\mathcal{M}$ returns random values from their respective domains. When $\mathcal{A}$ queries the ***spend*** oracle with input $(m, act_\pi, \boldsymbol{A}, Y_{out})$, $\mathcal{M}$ executes the ***spend*** algorithm honestly and outputs the valid signature. $\mathcal{A}$ can also get the private keys corresponding the public key which generated by $\mathcal{M}$.

Challenge: Suppose $\mathcal{A}$ uses the public Y_π corresponding the secret key x to construct two transactions. We ignore the other parts of the signature except for img because they are irrelevant to determine linkability. Suppose $\mathcal{M}$ sets the values current state of accumulator and the product of key-images as (A_t, y_t). $\mathcal{A}$ computes $img = \theta x$, then performs ***NonMemCreate*** and outputs $u_x = (a, B = g^b)$. $\mathcal{M}$ honestly performs ***VerNonMem*** and updates the state $(A_{t+1} = A_t^{\theta x}, y_{t+1} = y_t \cdot \theta x)$.

Finally, $\mathcal{A}$ generates $img' = \theta' x$ and performs ***NonMemCreate*** to output $u'_x = (a', B' = g^{b'})$. If u'_x passes the verification algorithm, then we have:

$$\begin{aligned} A_{t+1}^{a'} B'^{\theta' x} &= g \\ g^{y_t \theta x a'} g^{b' \theta' x} &= g \\ (g^x)^{y_t \theta a' + b' \theta'} &= g \end{aligned}$$

Let $d = g^x, e = y_t \theta a' + b' \theta'$, then $\mathcal{M}$ gets an instantiation to solve strong RSA problem, which contradicts the strong RSA assumption.

B. BATCH NON-MEMBERSHIP PROOFS

Assume that x^* is the product of the prover's elements, y^* is the product of the elements which are added into the accumulator. A_t is the accumulator. By the following algorithm, the computational complexity of the validation process can be reduced to a constant.

NonMemCreate(x^*, y^*, A_t) :

$$
\begin{aligned}
(a, b) &= Bezout(y^*, x^*) \\
A &= A_t^a \\
B &= g^b \\
g_H &= \boldsymbol{H_G}(A_t, A) \\
z_a &= g_H^a \\
l_a &= \boldsymbol{H_{prime}}(A_t, A, z_a) \\
\alpha &= \boldsymbol{H}(A_t, A, z_a, l_a) \\
q_a &= \lfloor a/l \rfloor \\
r_a &= a \bmod l_a \\
Q_a &= \left(A_t \cdot g_H^{\alpha}\right)^{q_a} \\
\pi_A &= \{z_a, Q_a, r_a\} \\
l_b &= \boldsymbol{H_{prime}}(B, x^*, g \cdot A^{-1}) \\
q_b &= \left\lfloor x^*/l_b \right\rfloor \\
Q_b &= B^{q_b}
\end{aligned}
$$

return $u_{x^*} = \{A, B, \pi_A, Q_b\}$

$\boldsymbol{VerNonMem}(A_t, u_{x^*}, x^*)$:

$$
\begin{aligned}
g_H &= \boldsymbol{H_G}(A_t, A) \\
l_a &= \boldsymbol{H_{prime}}(A_t, A, z_a) \\
l_b &= \boldsymbol{H_{prime}}(B, x^*, g \cdot A^{-1}) \\
\alpha &= \boldsymbol{H}(A_t, A, z_a, l_a) \\
r_b &= x^* mod\ l_b
\end{aligned}
$$

if $Q_a^{l_a}(A_t g^{\alpha})^{r_a} = A z_a^{\alpha} \wedge Q_b^{l_b} B^{r_b} = g \cdot A^{-1}$, return 1, else return 0.

The first half of the judgment is used to verify the knowledge $A = A_t^a$. The second half is used to verify the equation $Bezout\,(y^*, x^*) = 1$, in other words, to verify that x^* and y^* are each other's prime numbers. The above validation operation only performs 6 exponential operations on group elements.

C. IMPROVED INNER PRODUCT ALGORITHM

Denote that $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^{\boldsymbol{n}}, \boldsymbol{g}, \boldsymbol{h} \in \mathbb{G}^{\boldsymbol{n}}, u, P \in \mathbb{G}, \boldsymbol{S} = \emptyset$, the

$$\boldsymbol{CreateImpIP}\,(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{g}, \boldsymbol{h}, u, P, n, \boldsymbol{S}) :$$

while $n > 1$:

$$
\begin{aligned}
n' &= n'/2 \\
L &= g_{[n':]}^{\boldsymbol{a}_{[:n']}} h_{[:n']}^{\boldsymbol{b}_{[n':]}} u^{<\boldsymbol{a}_{[:n']}, \boldsymbol{b}_{[n':]}>} \\
R &= g_{[:n']}^{\boldsymbol{a}_{[n':]}} h_{[n':]}^{\boldsymbol{b}_{[:n']}} u^{<\boldsymbol{a}_{[n':]}, \boldsymbol{b}_{[:n']}>} \\
x &= \boldsymbol{H}(L, R) \\
\boldsymbol{g}' &= \boldsymbol{g}_{[:n']}^{x^{-1}} \circ \boldsymbol{g}_{[n':]}^{x} \\
\boldsymbol{h}' &= \boldsymbol{h}_{[:n']}^{x} \circ \boldsymbol{h}_{[n':]}^{x^{-1}} \\
P' &= L^{x^2} P R^{x^2} \\
\boldsymbol{a}' &= \boldsymbol{a}_{[:n']} \cdot x + \boldsymbol{a}_{[n':]} \cdot x^{-1} \\
\boldsymbol{b}' &= \boldsymbol{b}_{[:n']} \cdot x^{-1} + \boldsymbol{b}_{[n':]} \cdot x \\
\boldsymbol{S}' &= \boldsymbol{S} \cup \{(L, R)\} \\
&\boldsymbol{CreateImpIP}\left(\boldsymbol{a}', \boldsymbol{b}', \boldsymbol{g}', \boldsymbol{h}', u, P', n', \boldsymbol{S}\right)
\end{aligned}
$$

return $(\boldsymbol{g}, \boldsymbol{h}, a, b, P, \boldsymbol{S})$

$$\boldsymbol{VerImpIP}\,(\boldsymbol{g}, \boldsymbol{h}, a, b, P, \boldsymbol{S}) :$$

for $j \in \{1, logn\}$:

$$
\begin{aligned}
x_j &= \boldsymbol{H}\left(L_j, R_j\right) \\
b\,(i, j) &= \begin{cases} 1 : \text{the } j\text{th bit of } i-1 \text{ is1} \\ -1 : \text{otherwise} \end{cases}
\end{aligned}
$$

for $i \in \{1, n\}$:

$$
\begin{aligned}
s_i &= \prod_{j=1}^{logn} x_j^{b(i,j)} \\
\boldsymbol{s} &= \{s_1, \ldots, s_n\}
\end{aligned}
$$

if $g^{a \cdot s} \cdot h^{b \cdot s^{-1}} \cdot u^{a \cdot b} = P \cdot \prod_{j=1}^{logn} L_j^{x_j^2} \cdot R_j^{x_j^{-2}}$, return 1 else return 0

By the above algorithm, the vector size in each loop can be reduced by half, and then the half vector stays the same structure as before. After going through the *logn* loops, the size of the vector reduces to 1. Through the Fiat-Shamir heuristic, the sender is able to pre-calculate challenge x to achieve non-interactive proofs. For two vectors of length n, it only has to send $2logn + 2$ elements.

REFERENCES

[1] S. Nakamoto. (2008). *Bitcoin: A Peer-To-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for bitcoin with accountable mixes," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Christ Church, Barbados, 2014, pp. 486–504.

[3] G. Maxwell. (2013). *Coinjoin: Bitcoin Privacy For The Real World*. [Online]. Available: https://bitcointalk.org/index.php

[4] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better—How to make bitcoin a better currency," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Kralendijk, Bonaire, 2012, pp. 399–414.

[5] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-resistant mixing for bitcoin," in *Proc. 13th Workshop Privacy Electron. Soc.*, Scottsdale, AZ, USA, 2014, pp. 149–158.

[6] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *Proc. ESORICS*, Wroclaw, Poland, 2014, pp. 345–364.

[7] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed E-cash from bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, May 2013, pp. 397–411.

[8] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, San Jose, CA, USA, May 2014, pp. 459–474.

[9] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, Feb. 1989.

[10] N. Van Saberhagen. (2013). *Cryptonote V2.0*. [Online]. Available: https://cryptonote.org/whitepaper.pdf

[11] S. Noether, A. Mackenzie, and T. M. Research Lab, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, Dec. 2016.

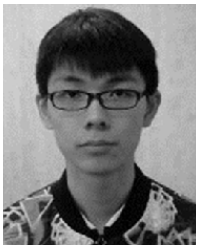
[12] G. Maxwell. (2015). Confidential transactions. [Online]. Available: https://people.xiph.org/greg/confidential values.txt.

[13] J. Herranz and G. Sáez, "Forking lemmas for ring signature schemes," in *Proc. INDOCRYPT*, New Delhi, India, 2003, pp. 266–279.

[14] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proc. IEEE Symp. Secur. Privacy (SP)*, London, U.K., May 2018, pp. 315–334.

[15] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," in *Proc. ACISP*, Sydney, NSW, Australia, 2004, pp. 325–335.

[16] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, London, U.K., 2006, pp. 390–399.
[17] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," in *Proc. IEEE Symp. Secur. Privacy*, San Jose, CA, USA, May 2015,1 pp. 122–134.
[18] S. Goldfeder, H. Kalodner, D. Reisman, and A. Narayanan, "When the cookie meets the blockchain: Privacy risks of Web payments via cryptocurrencies," *PoPETs*, vol. 2018, no. 4, pp. 179–199, Aug. 2018.
[19] 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," in *Proc. ESORICS*, 2017, pp. 456–474.
[20] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. CRYTO*, Santa Barbara, CA, USA, 1986, pp. 186–194.
[21] J. Groth and M. Kohlweiss, "One-out-of-many proofs: Or how to leak a secret and spend a coin," in *Proc. EUROCRYPT*, Sofia, Bulgaria, 2015, pp. 253–280.
[22] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *J. Cryptol.*, vol. 1, no. 2, pp. 77–94, 1988.
[23] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *Advances in Cryptology—CRYPTO'97*. Santa Barbara, CA, USA, 1997, pp. 410–424.
[24] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *Advances in Cryptology—CRYPTO 2006*. Santa Barbara, CA, USA, 2006, pp. 78–96.
[25] T. H. Yuen, S. F. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu, "RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Kota Kinabalu, Malaysia, 2020, pp. 464–483.
[26] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," in *Advances in Cryptology—CRYPTO 2019*. Santa Barbara, CA, USA, 2019, pp. 561–586.
[27] G. Wood. (2014). *Ethereum: A Secure Decentralized Generalized Transaction Ledger*. Ethereum. [Online]. Available: https://ethereum.org/en/whitepaper/
[28] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA, May 2016, pp. 839–858.
[29] M. Abe, M. Ohkubo, and K. Suzuki, "1-out-of-n signatures from a variety of keys," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Queenstown, New Zealand, 2002, pp. 415–432.
[30] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *Advances in Cryptology—EUROCRYPT 2016*. Vienna, Austria, 2016, pp. 327–357.
[31] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology—CRYPTO'87*. Santa Barbara, CA, USA, 1987, pp. 369–378.
[32] J. Yanxue, S. Shifeng, Z. Yuncong, Z. Qingzhao, D. Ning, L. Zhiqiang, L. Joseph, and G. Dawu, "PBT: A new privacy-preserving payment protocol for blockchain transactions," *IEEE Trans. Depend. Sec. Comput.*, early access, May 29, 2020, doi: 10.1109/TDSC.2020.2998682.
[33] E. M. Abou-Nassar, A. M. Iliyasu, P. M. El-Kafrawy, O.-Y. Song, A. K. Bashir, and A. A. A. El-Latif, "DITrust chain: Towards blockchain-based trust models for sustainable healthcare IoT systems," *IEEE Access*, vol. 8, pp. 111223–111238, 2020.
[34] H. Shu, P. Qi, Y. Huang, F. Chen, D. Xie, and L. Sun, "An efficient certificateless aggregate signature scheme for blockchain-based medical cyber physical systems," *Sensors*, vol. 20, no. 5, p. 1521, Mar. 2020.



JUNKE DUAN received the B.S. degree in industry design from the Beijing University of Posts and Telecommunications, China, in 2016, where he is currently pursuing the M.S. degree in cyber security. His current research interests include cryptography, blockchain technology and privacy, and cryptocurrency.

LIZE GU received the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2005. He is currently a Professor with BUPT, where he joined the School of Cyberspace Security and National Engineering Laboratory for Disaster Backup and Recovery. His current research interests include modern cryptography and blockchain technology.

SHIHUI ZHENG received the Ph.D. degree from Shandong University, China, in 2006. From 2006 to 2008, she was a Postdoctoral Researcher with the School of Information Engineering, Beijing University of Posts and Telecommunications (BUPT), China, where she joined the School of Cyberspace Security and National Engineering Laboratory for Disaster Backup and Recovery, in 2008. Her current research interest includes cryptographic schemes design.