# Concise RingCT Protocol Based on Linkable Threshold Ring Signature

Junke Duan, Shihui Zheng, Wei Wang, Licheng Wang, Xiaoya Hu, and Lize Gu

***Abstract*—Ring Confidential Transactions (RingCT) is a typical privacy-preserving protocol for blockchain, which is used for the most popular anonymous cryptocurrency Monero in recent years. RingCT provides the user's identity anonymity based on the linkable ring signature. At the cost of that, the transaction size is increased linearly to the involved users. In this article, we aim to overcome this inefficient aspect of RingCT by introducing the linkable threshold ring signature (LTRS). We first propose a construction of threshold ring signatures for homomorphic cryptosystems, and present an efficient instantiation based on the intractability assumption of the discrete logarithm problem. Based on this framework, an efficient LTRS scheme and a novel construction of the RingCT protocol are presented. Our proposed RingCT protocol enables multiple payers to co-construct an anonymous transaction without revealing their secret account keys, and it is more concise under multiple input accounts. For a transaction with a ring size of 100 and the input accounts number of 64, the communication overhead is about 4% of the original RingCT protocol.**

***Index Terms*—ring signature, linkable threshold ring signature, anonymity, RingCT, blockchain, cryptocurrency.**

## I. Introduction

BLOCKCHAIN is a distributed public ledger. It was initially proposed as the underlying storage technology of Bitcoin [1]. Due to its features, such as decentrality and immutability, blockchain has received more attention in recent years. However, since transactions in blockchain need to be publicly verified and stored, this leads to the disclosure of users' identities, transaction contents, and other private information. As a result, developing privacy-preserving technologies for blockchain has become a key research focus in recent years. RingCT [2], zk-SNARKs [3], and Coin Shuffle [4] are representative privacy-preserving protocols for blockchain. They apply to the well-known anonymous cryptocurrencies Monero, Zcash, and Dash, respectively. This paper focuses on the RingCT protocol and its underlying ring signature scheme.

Ring signature (RS) is a signature scheme with anonymity, proposed by Rivest et al. [5]. It enables a signer to form a temporary group of members, known as a "ring", to sign a message. The verifier can verify that the signer is one of the ring's members, but has no way of knowing which one the signer is. Existing instantiations of RS include: RSA [6], zero-knowledge proof [7], [8], bilinear pair [9]–[11], lattice [12]–[15], etc. RS can be applied to anonymous identity authentication [16], electronic voting [17], and anonymous reporting [18]. However, the perfect anonymity of RS makes it impossible for the verifier to distinguish whether the same secret key signs two signatures. This limits the use of RS in e-cash or anonymous cryptocurrencies. To solve this problem, Liu et al. proposed the linkable ring signature (LRS) [19], which allows the verifier to link two signatures signed using the same secret key. This property ensures that the signer can only use his secret key for a one-time signature. LRS is used in RingCT protocol to implement anonymous transactions with the prevention of double-spending attack[1].

To anonymize the user's identity in the transaction, the additional communication overhead added by the RingCT protocol is significant. In Bitcoin, the average transaction size is 250B, while a typical RingCT transaction has a transaction size of around 2200B, nearly nine times the former. There are some works to optimize the signature size in RingCT. Sun et al. proposed RingCT 2.0 protocol based on the cryptographic accumulator to reduce the ring size [21]. However, it requires a trusted setup, which is unsuitable for the decentralized blockchain. Jia et al. proposed a privacy-protecting payment protocol PBT [22], which is based on one-to-many proofs and a special multi-signature. This protocol offers practical performance superior to RingCT 2.0 and does not require a trusted setup. Yuen et al. proposed RingCT 3.0 protocol to further reduce the signature size by combining the inner product argument [23]. Goodell et al. proposed a concise LRS scheme for RingCT [24], which reduces the signature size by about 25% compared to [2]. However, the transaction size in these improved protocols grows linearly with the number of input accounts. To construct a more concise RingCT, we inspire to hide multiple accounts within the same ring, thereby ensuring that the number of rings does not depend on the number of real input accounts.

To this end, we introduce the threshold ring signature (TRS), which is a threshold variant of RS proposed by Bresson et al. [25]. A $t$-out-of-$n$ TRS scheme allows $t$ signers to sign on behalf of a ring of size $n$, but does not expose the identity of these signers. The instantiations of the TRS scheme include: zero-knowledge proof [26], [27], bilinear pair [28], and so on. Linkable threshold ring signature (LTRS)

This work was supported by the National Key Research and Development Program of China (2022YFB2702703), the National Natural Science Foundation of China (NSFC) (62272040, 62232002), and BUPT Excellent Ph.D. Students Foundation CX2022136.

Junke Duan, Shihui Zheng, Wei Wang, Xiaoya Hu, and Lize Gu are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: {duanjunke,shihuizh,weiwangscsc,huxiaoya,glzisc}@bupt.edu.cn)

Licheng Wang is with the School of Cyberspace Science and Technology of Beijing Institute of Technology Beijing 100081, China (e-mail: lcwang@bit.edu.cn)

[1]An attack on anonymous cryptocurrency in which the attacker reuses the same token [20]

TABLE I
COMPARISON OF THE OVERHEAD

| Type | Ref. | Computations | | Signature/Transaction Size |
|---|---|---|---|---|
| | | Sign | Verify | |
| **TRS** | [25] | $nlogn2^t \cdot E$ | $dlogn2^t \cdot E$ | $logn2^t(n+t)\|\mathbb{Z}_p\|$ |
| | [26] | $(2n-t) \cdot E$ | $2n \cdot E$ | $(2n-t+1)\|\mathbb{Z}_p\|$ |
| | [27] | $3n \cdot E$ | $n \cdot E + (n+1) \cdot P$ | $(n-t+1)\|\mathbb{Z}_p\| + (n+1)\|\mathbb{G}_p\|$ |
| | [28] | $(3n+3t+1) \cdot E$ | $E + (3n+3) \cdot P$ | $(2n+2)\|\mathbb{G}_p\|$ |
| | **Ours** | $2n \cdot E$ | $(2n+1) \cdot E$ | $(n+1)\|\mathbb{Z}_p\|$ |
| **LTRS** | [29] | $(6n-2t) \cdot E$ | $6n \cdot E$ | $(3n-t+2)\|\mathbb{Z}_p\| + n\|\mathbb{G}_p\|$ |
| | [30] | $(9n+4t) \cdot E$ | $10n \cdot E$ | $(4n-t+1)\|\mathbb{Z}_p\| + 3n\|\mathbb{G}_p\|$ |
| | [31] | $((8\sqrt{n}+16)t) \cdot E + t \cdot P$ | $2t \cdot E + (8\sqrt{n}+9)t \cdot P$ | $t(6\sqrt{n}+8)\|\mathbb{G}_p\|$ |
| | **Ours** | $(3n+3t-2) \cdot E$ | $(3n+t) \cdot E$ | $(n+1)\|\mathbb{Z}_p\| + t\|\mathbb{G}_p\|$ |
| **RingCT** | [2] | $(n(4t+2)-2t-1) \cdot E$ | $n(4t+2) \cdot E$ | $(n(t+1)+2)\|\mathbb{Z}_p\| + (n+1)t\|\mathbb{G}_p\|$ |
| | [24] | $(6n+3t-3) \cdot E$ | $(6n+2t) \cdot E$ | $(2n+2)\|\mathbb{Z}_p\| + (n+1)t\|\mathbb{G}_p\|$ |
| | **Ours** | $(5n+3t+2) \cdot E$ | $(5n+t+1) \cdot E$ | $(2n+2)\|\mathbb{Z}_p\| + (n+t)\|\mathbb{G}_p\|$ |

"$E$" and "$P$" refer to the exponentiation and pairing operations respectively. "$|\mathbb{G}_p|$" and "$|\mathbb{Z}_p|$" refer to the length of elements in $\mathbb{G}_p$ and $\mathbb{Z}_p$ respectively. "$n$" and "$t$" refer to the ring size and number of signers secret keys respectively.

is a linkable variant of LRS. Existing LTRS schemes [29]–[31] are respectively built based on the corresponding TRS schemes [26]–[28]. In addition, Aranha et al. proposed a TRS scheme with extendability [32]. Haque et al. proposed a generic construction of TRS with logarithmic signature size [33]. Unfortunately, existing (L)TRS schemes are unsuitable for the RingCT protocol. The reason is that the verification step in RingCT needs to compute the sum of all the input accounts, which requires the verifier to know where the input accounts are in the ring. This is contrary to existing (L)TRS schemes' attribute to hide multiple signers' identities. To resolve this contradiction, we consider constructing an (L)TRS with a new design idea so that it is suitable for improving the RingCT protocol.

## A. Our Contributions

We first propose a new construction of TRS, which is suitable for all homomorphic cryptosystems. Based on it, we propose efficient instantiations of TRS and LTRS under discrete logarithm (DL) assumption, and then design the threshold RingCT protocol. Specifically, our contributions are as follows:

- **Construction of TRS for homomorphic cryptosystems**. We propose a transformation method to convert a 1-out-of-$n$ RS scheme into a $t$-out-of-$n$ TRS scheme in a homomorphic cryptosystem. We call it "sliding window transformation" (SWT). Based on SWT, we propose a construction of the TRS scheme and give its security proofs.
- **Efficient instantiation of TRS.** We propose an efficient instantiation of the TRS scheme based on SWT. It has faster signing/verifying speed and a smaller signature size than other schemes. We give the security proofs of the scheme. By integrating the sum argument in [34], we further reduce its signature size from linear to logarithmic.
- **Efficient instantiation of LTRS.** We propose a linkable ring signature scheme with double hash challenge (DC-LRS). Based on it and SWT, we propose an efficient LTRS scheme. It is more efficient than other schemes. We give the security proofs of them.
- **Threshold RingCT protocol.** Based on our proposed LTRS scheme, we present a threshold RingCT protocol. Compared with the current RingCT protocol [24], our design can significantly reduce the transaction size with multiple account inputs. For a transaction with a ring size of 100 and the input accounts number of 64, the communication overhead of our protocol is about 4% of [24]. In addition, our protocol enables multiple payers to co-construct the anonymous transaction without revealing their account keys, while other similar protocols cannot.

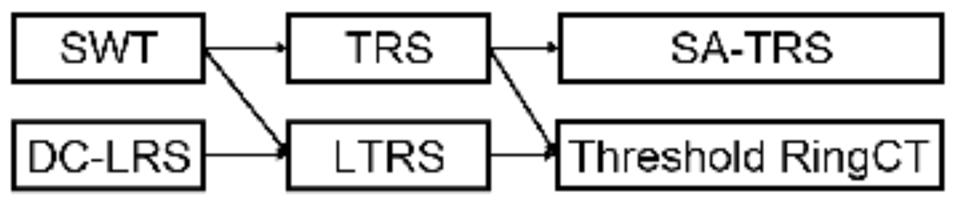


Fig. 1. Relationship between our contributions.

The relationship between our contributions is illustrated in Figure 1, where the schemes at the arrowheads are built upon the schemes at the arrow origins. The comparison of our TRS, LTRS, and RingCT with other schemes in terms of overhead is shown in Table I, and the comparison of signature sizes between our SA-TRS and TRS is presented in Table III.

## B. Outline

The remainder of this paper is organized as follows. We briefly introduce some preliminaries in Section II. We present the formal definitions and security properties of our schemes in Section III. We present the construction of the TRS scheme based on SWT in Section IV. We propose the instantiations and security proofs of our proposed TRS and LTRS schemes in Section V and Section VI, respectively. Section VII shows the design of our proposed threshold RingCT protocol. We experimentally analyze the cost of our proposed schemes in Section VIII. Finally, we make a brief conclusion in Section IX.

## II. Preliminaries

### A. Notations

We list the symbols used in the following sections in Table II. Note that except those in the oracles, the subscripts of all elements are modulo $n$ by default (e.g., $y_{s+t(\text{mod n})}$ is simply denoted as $y_{s+t}$).

TABLE II
Description of Symbols

| Symbol | Description |
|---|---|
| $\mathbb{G}_p$ | Commutative group with the order $p$ |
| $\mathbb{Z}_p$ | Modular $p$ residue class |
| $x, y$ | Secret key and public key |
| $\mathbb{Y}$ | Set with $n$ public keys $y_0, ..., y_{n-1}$ |
| $\mathbb{X}$ | Set with $t$ secret keys $x_s, ..., x_{s+t-1}$ |
| $H_0$/$H_1$ | Hash function which outputs an element in $\mathbb{Z}_p$/$\mathbb{G}_p$ |
| $\mathcal{HO}_0$/$\mathcal{HO}_1$ | Hash oracle whose outputs are elements in $\mathbb{Z}_p$/$\mathbb{G}_p$ |
| $\mathbb{L}_{h_0}$/$\mathbb{L}_{h_1}$ | Set which stores the query-response pairs of $\mathcal{HO}_0$/$\mathcal{HO}_1$ |
| $\mathcal{SO}$ | Signing oracle which inputs $(\mathbb{Y}, m)$, outputs a signature $\sigma$ |
| $\mathbb{L}_s$ | Set which stores the query-response pairs of $\mathcal{SO}$ |

### B. Mathematical Assumptions

***Discrete Logarithm (DL) Assumption.*** *Let $\mathbb{G}_p$ be a group with the generator $g$ and the order $p$. For any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, it holds that*

$$Pr[x \leftarrow \mathcal{A}(g, g^x)] \leq negl(\lambda)$$

*where $g, g^x \in \mathbb{G}_p$.*

***Decisional Diffie-Hellman (DDH) Assumption.*** *Let $\mathbb{G}_p$ be a group with the generator $g$ and the order $p$. For any PPT adversary $\mathcal{A}$ which outputs $1/0$, it holds that*

$$\left| \begin{matrix} Pr[1 \leftarrow \mathcal{A}(g, g^a, g^b, g^{ab})] \\ -Pr[1 \leftarrow \mathcal{A}(g, g^a, g^b, g^c)] \end{matrix} \right| \leq negl(\lambda)$$

*where $g, g^a, g^b, g^c \in \mathbb{G}_p$.*

### C. Key-Recovery under Key-Only Attack

The key-recovery under key-only attack (KR-KOA) [35] is an attack targeting signature schemes or identification protocols. It denotes that an adversary obtains the corresponding secret key with only the public keys of the target scheme. Formally, assuming there is a signature scheme with a setup algorithm $\mathbf{Setup}(\lambda) \rightarrow pp$ and a key generation algorithm $\mathbf{KeyGen}() \rightarrow (y, x)$). A signature scheme is secure against KR-KOA if that

$$Pr\left[ x' = x : \begin{matrix} pp \leftarrow \mathbf{Setup}(\lambda); \\ (y, x) \leftarrow \mathbf{KeyGen}(); \\ x' \leftarrow \mathcal{A}^{\mathcal{HO}}(y) \end{matrix} \right] \leq negl(\lambda).$$

We assume in the following context that all signature schemes satisfy security under KR-KOA.

### D. Signature of Knowledge

A zero-knowledge proof [36] protocol enables a prover to prove to a verifier that he knows some knowledge, but does not expose that knowledge. It can be converted to a non-interactive zero-knowledge proof (NIZK) scheme [37] by the Fiat-Shamir heuristic. A NIZK scheme can be converted into a signature of knowledge (SoK) scheme [38] by adding message $m$ to the hash challenge. Formally, assuming $\mathcal{R}$ is a fixed NP-hard relation, $w$ is the witness of the knowledge, and $S$ is a public statement corresponding to $w$. An SoK scheme is used to prove the following NP language:

$$\mathcal{L}(m) = \{S \mid \exists w : (w, S) \in \mathcal{R}\} \tag{1}$$

An SoK scheme contains the setup algorithm $\mathbf{Setup}(\lambda) \rightarrow pp$, the signing algorithm $\mathbf{Sign}(S, w, m) \rightarrow \sigma$, and the verifying algorithm $\mathbf{Verify}(S, \sigma, m) \rightarrow 0/1$. It satisfies the following security properties:

***Completeness***. *The tuple $(S, \sigma, m)$ generated by the honest prover can always be verified as valid.*

***Simulatability***. *The signature $\sigma$ does not reveal any information about the witness $w$.*

***Extractability***. *If a PPT adversary $\mathcal{A}$ can generate a valid signature $\sigma'$, then there is a PPT algorithm* $\mathbf{Ext}$ *which can extract the witness $w$ from $\mathcal{A}$'s output.*

### E. Ring Signature

The ring signature (RS) scheme is an SoK scheme. It is used to prove the following NP language:

$$\mathcal{L}_{RS}(m) = \{\mathbb{Y} \mid \exists x_s, s \in \{0, ..., n-1\} : (x_s, y_s) \in \mathcal{R}\} \tag{2}$$

An RS scheme contains the following algorithms:

$$\mathbf{RS} \begin{cases} \mathbf{Setup}(\lambda) \rightarrow pp & \mathbf{KeyGen}() \rightarrow (y, x) \\ \mathbf{Sign}(\mathbb{Y}, x, m) \rightarrow \sigma & \mathbf{Verify}(\mathbb{Y}, \sigma, m) \rightarrow 0/1 \end{cases}$$

An RS scheme satisfies the following security properties:

***Completeness***. *The signature generated by* $\mathbf{Sign}$ *must be valid.*

***Unforgeability***. *No PPT adversary can forge a valid signature without the secret key $x$.*

***Perfect Anonymity***. *For a $1$-out-of-$n$ ring signature, the probability of guessing the signer's public key is $1/n$.*

A linkable ring signature (LRS) scheme is a special RS scheme. In addition to the completeness and unforgeability mentioned above, it satisfies computational anonymity and linkability, which are described as follows:

***Computational Anonymity***. *For a $1$-out-of-$n$ LRS scheme, the advantage of a PPT adversary in guessing the signer's public key is negligible compared to $1/n$.*

***Linkability***. *Two signatures signed using the same secret key can be identified.*

### F. Homomorphic Commitment

A commitment is a mapping $\mathbb{M} \times \mathbb{R} \rightarrow \mathbb{C}$ where $\mathbb{M}$ is the message space, $\mathbb{R}$ is the random number space, and $\mathbb{C}$ is the commitment space. A homomorphic commitment $HCom$ is a specific commitment scheme, which holds that

$$HCom(m_1, r_1) * HCom(m_2, r_2) = HCom(m_1 \bullet m_2, r_1 \circ r_2)$$

where $m_1, m_2 \in \mathbb{M}$, $r_1, r_2 \in \mathbb{R}$, $*$ is the operation in $\mathbb{C}$, $\bullet$ is the operation in $\mathbb{M}$, and $\circ$ is the operation in $\mathbb{R}$. A homomorphic commitment needs to satisfy the following security properties:

***Hiding.*** *Given a commitment $C$, the probability that a PPT adversary gets the message $m$ of it is negligible.*

***Binding.*** *The probability that one commitment corresponds to two different messages is negligible.*

Pedersen commitment [39] is an instantiation of the homomorphic commitment scheme, which sets $\mathbb{M} = \mathbb{Z}_p, \mathbb{R} = \mathbb{Z}_p^*, \mathbb{C} = \mathbb{G}_p$ with two generators $g, h \in \mathbb{G}_p$. It sets $HCom(m, r) = g^r h^m$.

## III. Definitions and Security Properties

### A. Formal Definitions

Based on the definitions in [25], [29], [40], we give the formal definitions of TRS and LTRS as follows.

TRS scheme is an SoK scheme. It is used to prove that the signer knows the secret keys corresponding to $t$ of $n$ public keys, but does not reveal the $t$ public keys. A $t$-out-of-$n$ TRS scheme consists of the following PPT algorithms:

$\mathbf{Setup}(\lambda) \rightarrow pp$. On input a security parameter $\lambda$, the algorithm outputs the system parameter $pp$ as a common input to other algorithms.

$\mathbf{KeyGen}() \rightarrow (y, x)$. The algorithm outputs a public key $y$ and its corresponding secret key $x$.

$\mathbf{Sign}(\mathbb{Y}, \mathbb{X}, m) \rightarrow \sigma$. This is a possibly interactive procedure. On input a public key set $\mathbb{Y} = \{y_0, ..., y_{n-1}\}$, a secret key set $\mathbb{X} = \{x_s, ..., x_{s+t-1}\}$ which are corresponding to $y_s, ..., y_{s+t-1}$, and a message $m$, the algorithm outputs a signature $\sigma$;

$\mathbf{Verify}(\mathbb{Y}, \sigma, m)$. On input a public key set $\mathbb{Y} = \{y_0, ..., y_{n-1}\}$, a signature $\sigma$, and a message $m$, if the signature is valid, the algorithm outputs 1, otherwise it outputs 0.

LTRS scheme is a modified TRS scheme, which contains the same PPT algorithms as TRS, as well as the following algorithm:

$\mathbf{Link}(\sigma_0, \sigma_1)$. On input two distinct signatures $\sigma_0, \sigma_1$, if there is the same secret key involved in generating both $\sigma_0$ and $\sigma_1$, the algorithm outputs 1, otherwise it outputs 0.

### B. Security Properties

We give the security properties of TRS and LTRS as follows, where $\mathbb{Y}, \mathbb{X}, \mathcal{SO}, \mathcal{HO}$ are defined in Section II-A. Note that for the following five properties, the TRS scheme satisfies the first three properties, and the LTRS scheme satisfies all properties except perfect anonymity.

**Definition 1 (Completeness).** *A TRS/LTRS scheme satisfies completeness if that*

$$Pr\left[\begin{array}{c}\mathbf{Verify}\\(\mathbb{Y},\sigma,\cdot)=1\end{array}:\begin{array}{c}pp\leftarrow\mathbf{Setup}(\lambda);\\\forall i\in\{0,...,n-1\}:\\(y_i,x_i)\leftarrow\mathbf{KeyGen}();\\\sigma\leftarrow\mathbf{Sign}(\mathbb{Y},\mathbb{X},\cdot)\end{array}\right]=1$$

**Definition 2 ($t$-Unforgeability wrt Insider Corruption).** *A TRS/LTRS scheme satisfies $t$-unforgeability wrt insider corruption if for any PPT adversary $\mathcal{A}$, it holds that*

$$Pr\left[\begin{array}{c}\mathbf{Verify}\\(\mathbb{Y},\sigma',\cdot)=1\end{array}:\begin{array}{c}pp\leftarrow\mathbf{Setup}(\lambda);\\\forall i\in\{0,...,n-1\}:\\(y_i,x_i)\leftarrow\mathbf{KeyGen}();\\\sigma'\leftarrow\mathcal{A}^{\mathcal{SO},\mathcal{HO}}\\(pp,\mathbb{Y},\mathbb{X}\backslash\{x_s\},\cdot)\end{array}\right]\leq negl(\lambda)$$

**Definition 3 (Perfect Anonymity wrt Adversarial Keys).** *A TRS scheme satisfies perfect anonymity wrt adversarial keys if for any PPT adversary $\mathcal{A}$, it holds that*

$$Pr\left[b'=b:\begin{array}{c}pp\leftarrow\mathbf{Setup}(\lambda);\\\forall i\in\{0,...,n-1\}:\\(y_i,x_i)\leftarrow\mathbf{KeyGen}();\\\sigma_0\leftarrow\mathbf{Sign}(\mathbb{Y},\mathbb{X},\cdot);\\\sigma_1\leftarrow\mathcal{SO}(\mathbb{Y},\cdot);b\in\{0,1\};\\b'\leftarrow\mathcal{A}^{\mathcal{SO},\mathcal{HO}}(pp,\mathbb{Y},\mathbb{X},\sigma_b,\cdot)\end{array}\right]=\frac{1}{2}$$

**Definition 4 (Computational Anonymity wrt Adversarial Keys).** *An LTRS scheme satisfies computational anonymity if for any PPT adversary $\mathcal{A}$, it holds that*

$$Pr\left[b'=s:\begin{array}{c}pp\leftarrow\mathbf{Setup}(\lambda);\\\forall i\in\{0,...,n-1\}:\\(y_i,x_i)\leftarrow\mathbf{KeyGen}();\\\sigma\leftarrow\mathbf{Sign}(\mathbb{Y},\mathbb{X},\cdot);\\b'\leftarrow\mathcal{A}^{\mathcal{SO},\mathcal{HO}}(pp,\mathbb{Y},\\\mathbb{X}\backslash\{x_s\},\sigma,\cdot);b'\in\{s,s+t\}\end{array}\right]\leq\frac{1}{2}+negl(\lambda)$$

**Definition 5 (Linkability).** *An LTRS scheme satisfies linkability if for any PPT adversary $\mathcal{A}$, it holds that*

$$Pr\left[\begin{array}{c}\mathbf{Verify}\\(\mathbb{Y}',\sigma',\cdot)=1\\\wedge\mathbf{Link}\\(\sigma,\sigma')=0\end{array}:\begin{array}{c}pp\leftarrow\mathbf{Setup}(\lambda);\\\forall i\in\{0,...,n-1\}:\\(y_i,x_i)\leftarrow\mathbf{KeyGen}(),\\(y_i',x_i')\leftarrow\mathbf{KeyGen}();\\\sigma\leftarrow\mathbf{Sign}(\mathbb{Y},\mathbb{X},\cdot);\\\sigma'\leftarrow\mathcal{A}^{\mathcal{SO},\mathcal{HO}}(pp,\mathbb{Y}',\mathbb{X}',\cdot);\\\mathbb{X}\cap\mathbb{X}'\neq\emptyset\end{array}\right]\leq negl(\lambda)$$

## IV. Construction of TRS

### A. Design of SWT

We first introduce the design idea of SWT, which is the cornerstone of our subsequent (L)TRS schemes. Suppose there is an ordered public key set of size $n$. By taking each $t$ adjacent public keys of these $n$ public keys, we get $n$ sub-sets. Further, by computing the "product" of all the elements in each sub-set, we get $n$ "products".

Intuitively, suppose the signer knows the secret keys corresponding to $t$ adjacent public keys. In that case, he also knows the secret key corresponding to the "product" of these public keys. The signer can use this "product" and other $n-1$ "products" to construct a 1-out-of-$n$ ring signature. In other words, if one can construct a valid ring signature in the way described above, it is proved that one knows the secret keys corresponding to $t$ adjacent public keys out of the $n$ public keys.

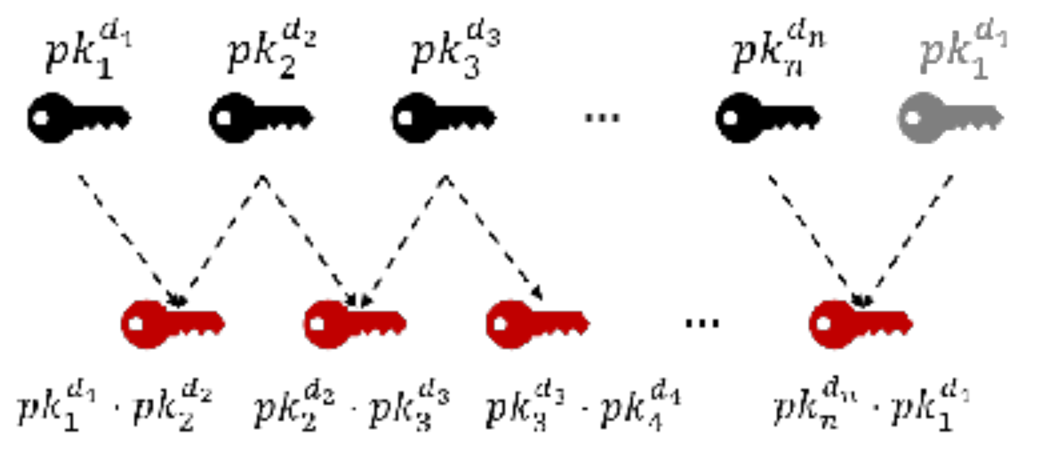

$pk_1^{d_1}$
$pk_2^{d_2}$
$pk_3^{d_3}$
$pk_n^{d_n}$
$pk_1^{d_1}$
$pk_1^{d_1} \cdot pk_2^{d_2}$
$pk_2^{d_2} \cdot pk_3^{d_3}$
$pk_3^{d_3} \cdot pk_4^{d_4}$
$pk_n^{d_n} \cdot pk_1^{d_1}$

Fig. 2. Example of computing a "product" with $t = 2$, where "$\cdot$" denotes the group operation (e.g., modular multiplication), and $d_i$ denotes the hash value of $pk_1||...||pk_n||i, i \in [1, n]$.

However, the above design cannot resist the adversary forging signatures through rogue key attacks. For instance, assuming that an adversary has a key pair $(pk', sk')$ and obtains public keys $pk_2, ..., pk_n$, then he can construct $pk_1 = pk' \cdot pk_2^{-1}$ such that $pk_1 \cdot pk_2 = pk'$. This results in the adversary using $sk'$ and $pk_1, ...pk_n$ to forge a 2-out-of-$n$ TRS.

To counter this weakness, our solution is to introduce distinct hash challenges for each public key, thereby preventing the adversary from constructing a new public key that satisfies a specific relation based on the existing public keys. As illustrated in Figure 2, it first computes $d_i \leftarrow H_0(pk_1||...||pk_n||i)$ for each $i \in [1, n]$, then uses all $pk_i^{d_i}$ as inputs for computing "product". Because the hash challenges are unpredictable and generated from the input of all the public keys involved in the signature, the adversary cannot generate rogue public keys based on a subset of the existing public keys. Specifically, regarding the attack method mentioned in the previous paragraph, the adversary needs to compute $pk_1 = (pk' \cdot pk_2^{-d_2})^{1/d_1}$ such that $pk_1^{d_1} \cdot pk_2^{d_2} = pk'$. However, $d_1, d_2$ are unknown because they are calculated from $pk_1$, therefore the rogue key $pk_1$ cannot be constructed.

We introduce how to construct a TRS using SWT and RS with a simple example. Assume we aim to construct a 2-out-of-4 TRS scheme. The secret and public keys are denoted as $x_i, y_i$ where $y_i = g^{x_i}, i \in \{1, 2, 3, 4\}$. The signer has the secret keys $x_1, x_2$ of $y_1, y_2$. According to SWT, he first computes $d_i = H_0(y_1||...||y_4||i)$ for each $i \in \{1, 2, 3, 4\}$, then he computes $y_{i,j} = y_i^{d_i} \cdot y_j^{d_j}$ where $i, j \in \{1, 2, 3, 4\}$, and computes $x_{1,2} = x_1 d_1 + x_2 d_2$. Then he executes Abe's RS signing algorithm using these precomputed results as inputs. For clarity, We divide the signing steps into the following sub-functions:

- $A(r) : g^r \to R$ is a exponent operation.
- $B(y, r, c) : g^r \cdot y^c \to L$ is a pedersen commitment.
- $R(x, r', c) : r' - x \cdot c \to r$ is a function to reconstruct the commitment $L$.

The steps of signing is shown as follows:

(1) The signer chooses a random number $r'$ and computes the challenge $c_{2,3} \leftarrow H_0(A(r'_{1,2})||m)$, where $m$ is the message.
(2) The signer picks $r_{2,3}$ randomly and computes the challenge $c_{3,4} \leftarrow H_0(B(y_{2,3}, r_{2,3}, c_{2,3})||m)$, then he computes $c_{4,1}, c_{1,2}$ similarly.
(3) The signer computes $r_{1,2} \leftarrow R(x_{1,2}, r'_{1,2}, c_{1,2})$ and outputs a signature $\sigma = (r_{1,2}, ..., r_{4,1}, c_{1,2})$.

When verifying the signature $\sigma$, the verifier first computes $y_{1,2}, ..., y_{4,1}$ like the signer, then he computes $c'_{2,3} \leftarrow H_0(B(y_{1,2}, r_{1,2}, c_{1,2})||m)$ and computes $c'_{3,4}, c'_{4,1}, c'_{1,2}$ similarly. If $c'_{1,2} = c_{1,2}$, then the signature is valid.

In addition to the aforementioned examples, our construction is supported by a wide range of existing cryptosystems. These include well-established cryptographic schemes like RSA, Elgamal, and elliptic curves. The compatibility of our construction with such diverse cryptosystems underscores its versatility and potential for integration into existing systems.

## *B. Construction of TRS*

Formally, we construct our TRS scheme as follows, building upon the intuitive description provided earlier. Given two algebraic systems $(\mathbb{A}_1, +)$ and $(\mathbb{A}_2, \cdot)$, let $f : \mathbb{A}_1 \to \mathbb{A}_2$ be a homomorphism with the following cryptographic properties:
**One-wayness**. For a given $x$, it is easy to compute $f(x)$. On the contrary, it is hard to compute $x$ for given $f(x)$.
**Trapdoor**. Knowing a trapdoor $T$ makes it easy to compute $x$ for given $f(x)$.
Then we can build our construction of the TRS scheme in Algorithm 1.

**Algorithm 1** Construction of TRS

```
 1: $\mathbf{Setup}(\lambda)$ Return $pp \leftarrow \mathbf{RS}.\mathbf{Setup}(\lambda)$;
 2: $\mathbf{KeyGen}()$ Return $(y, x) \leftarrow \mathbf{RS}.\mathbf{KeyGen}()$;
 3: $\mathbf{Sign}(\mathbb{Y}, \mathbb{X}, m)$
 4:     For $i$ in range $[0, n-1]$:
 5:         Compute $d_i \leftarrow H_0(y_0||...||y_{n-1}||i)$;
 6:         Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k^{d_k}$;
 7:     Set $\mathbb{Y}' \leftarrow \{\mathbf{y}_0, ..., \mathbf{y}_{n-1}\}$; Compute $\mathbf{x}_s \leftarrow \sum_{k=s}^{s+t-1} x_k \cdot c_d$;
 8:     Return $\sigma \leftarrow \mathbf{RS}.\mathbf{Sign}(\mathbb{Y}', \mathbf{x}_s, m)$;
 9: $\mathbf{Verify}(\mathbb{Y}, \sigma, m)$
10:     For $i$ in range $[0, n-1]$:
11:         Compute $d_i \leftarrow H_0(y_0||...||y_{n-1}||i)$;
12:         Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k^{d_k}$;
13:     Set $\mathbb{Y}' \leftarrow \{\mathbf{y}_0, ..., \mathbf{y}_{n-1}\}$; Return $\mathbf{RS}.\mathbf{Verify}(\mathbb{Y}', \sigma, m)$;
```

The operations in line 5 of Algorithm 1 require $t$ secret keys. If they belong to the same signer, then Algorithm 1 is executed as usual. A typical scenario of this condition is the cryptocurrency based on the unspent transactions outputs model (e.g., Bitcoin, Monero, Zcash), where a user has multiple one-time public keys to avoid trace analysis [41]. The other case is that the $t$ secret keys come from different signers. In this case, preserving each user's secret key is necessary. One method is shown as follows, where $P_0, ..., P_{t-1}$ are denoted as $t$ signers with secret keys $x_0, ..., x_{t-1}$, respectively.

(1) $P_0$ computes $z_0 \leftarrow x_0 + r_0$ where $r_0$ is a random number. $P_0$ sends $z_0$ to $P_1$. For each $i \in [1, t-1]$, $P_i$ chooses random number $r_i$, computes $z_i \leftarrow z_{i-1} + x_i + r_i$ and sends $z_i$ to $P_{i+1 (\text{mod } t)}$.
(2) $P_0$ computes $z'_0 \leftarrow z_{t-1} - r_0$ and sends $z'_0$ to $P_1$. For each $i \in [1, t-1]$, $P_i$ computes $z'_i \leftarrow z'_{i-1} - r_i$ and sends $z_i$ to $P_{i+1 (\text{mod } t)}$.
(3) $P_0$ gets $z'_{t-1} = \sum_{i=0}^{t-1} x_i$.

Note that in Step (1), since the signers add random numbers to their own secret keys, the secret key of each signer will not be disclosed, even if other malicious signers collude. In Step

(2), it requires collusion of at least $t-1$ malicious signers to steal the secret key of the remaining signer.

Another method is that each signer constructs his signature shardings. All signature shardings can be combined to generate a complete signature. It requires only two rounds of interaction, but needs to be designed based on the specific underlying RS scheme. We present an instantiation of this method in Section VI-C.

### C. Security Proofs

**Theorem 1.** *The proposed TRS scheme satisfies t-unforgeability wrt insider corruption if the underlying RS scheme is unforgeable and secure against KR-KOA.*

*Proof.* Assume that the simulator $\mathcal{S}$ is given a public key $y^*$. We will prove that if $\mathcal{A}$ can break the $t$-unforgeability wrt insider corruption of our scheme, then $\mathcal{S}$ can break security against KR-KOA or unforgeability of RS by invoking $\mathcal{A}$.
**Setup.** $\mathcal{S}$ runs $\mathbf{Setup}(\lambda) \rightarrow pp$. For each $i \in [1, n-1]$, $\mathcal{S}$ runs $\mathbf{KeyGen}() \rightarrow (y_i, x_i)$. $\mathcal{S}$ sets $y_0 \leftarrow y^*$, $\mathbb{Y} \leftarrow \{y_0, ..., y_{n-1}\}$ and $\mathbb{X}^* \leftarrow \{x_1, ..., x_{t-1}\}$, and sends $(pp, \mathbb{Y}, \mathbb{X}^*)$ to $\mathcal{A}$.
**Query.** $\mathcal{A}$ can adaptively query the oracles $\mathcal{SO}, \mathcal{HO}_0$ respectively. If $\mathbb{Y}_k$ in the $k$-th query to $\mathcal{SO}$ is equal to $\mathbb{Y}$, the game aborts, else $\mathcal{SO}$ returns a valid signature $\sigma_k$ to $\mathcal{A}$. For the $k$-th query $m_k$ to $\mathcal{HO}_0$, it returns a hash value $c_k$ to $\mathcal{A}$.
**Challenge.** $\mathcal{A}$ generates a signature $\sigma'$ and sends $(\sigma', m)$ to $\mathcal{S}$. If $\mathbf{Verify}(\mathbb{Y}, \sigma', m) = 1$, then $\mathcal{A}$ wins the game.

Given that $\mathbb{Y}' = \{\mathbf{y}_0, ..., \mathbf{y}_{n-1}\}$ where $\mathbf{y}_i = \prod_{k=i}^{i+t-1} y_k^{d_k}$ for each $i \in [0, n-1]$, if $\mathcal{A}$ wins, then there are two cases as follows:
**Case 0.** $\mathcal{A}$ has already obtained the secret key corresponding to one of the public keys in $\mathbb{Y}'$, allowing it to execute $\mathbf{RS}.\mathbf{Sign}$ to generate a valid signature $\sigma'$. $\mathcal{A}$ has a $1/n$ probability of obtaining the secret key corresponding to $\mathbf{y}_0$. Without loss of generality, we assume $t=2$, then it has $\mathbf{y}_0 = y^{*d_0} y_1^{d_1}$. According to the extractability of SoK, there is an extractor that can extract the secret key $\mathbf{x}'$ used in $\mathbf{RS}.\mathbf{Sign}$. Then $\mathcal{S}$ can get the secret key $x^* = \frac{\mathbf{x}' - x_1 d_1}{d_0}$. This is contrary to the security against KR-KOA of RS.
**Case 1.** $\mathcal{A}$ did not obtain the secret key corresponding to any of the public keys in $\mathbb{Y}'$. In other words, $\mathcal{A}$ can construct the signature $\sigma'$ using only $\mathcal{A}$ and $m$ such that $\mathbf{RS}.\mathbf{Verify}(\mathbb{Y}', \sigma', m) = 1$. It is contrary to the unforgeability of RS. □

**Theorem 2.** *The proposed TRS scheme satisfies perfect anonymity wrt adversarial keys if the underlying RS scheme satisfies perfect anonymity.*

*Proof.* We will prove that if $\mathcal{A}$ can break perfect anonymity wrt adversarial keys of our scheme, then there is a simulator $\mathcal{S}$, which can break perfect anonymity of RS by invoking $\mathcal{A}$.
**Setup.** $\mathcal{S}$ generates the system parameter $pp$ by running $\mathbf{Setup}(\lambda)$, generates $(y_i, x_i)$ for each $i \in \{0, ..., n-1\}$, and sets $\mathbb{Y} = \{y_0, ..., y_{n-1}\}, \mathbb{X} = \{x_s, ..., x_{s+t-1}\}$. $\mathcal{S}$ sends $(pp, \mathbb{Y}, \mathbb{X})$ to $\mathcal{A}$. Then $\mathcal{S}$ flips a coin to choose $b \in \{0, 1\}$. If $b = 0$, $\mathcal{S}$ returns $\sigma_0 \leftarrow \mathbf{Sign}(\mathbb{Y}, \mathbb{X}, m)$ to $\mathcal{A}$; if $b = 1$, $\mathcal{S}$ returns $\sigma_1 \leftarrow \mathcal{SO}(\mathbb{Y}, m)$ to $\mathcal{A}$.
**Query.** $\mathcal{A}$ can adaptively query the oracles $\mathcal{SO}, \mathcal{HO}_0$ respectively. For the $k$-th query $(\mathbb{Y}_k, \cdot)$ to $\mathcal{SO}$, it returns a valid signature $\sigma_k$ to $\mathcal{A}$. For the $k$-th query $m_k$ to $\mathcal{HO}_0$, it returns a hash value $c_k$ to $\mathcal{A}$.
**Guess.** $\mathcal{A}$ returns a guess $b'$ to $\mathcal{S}$. If the probability that $b' = b$ is greater than 1/2, then $\mathcal{A}$ wins the game.

If $\mathcal{A}$ wins, then for the signatures $\sigma_0$ and $\sigma_1$ of $(\mathbb{Y}', m)$ where $\mathbb{Y}'$ is computed as line 4-5 in Algorithm 1, $\mathcal{S}$ can use $b'$ as the guess to break the perfect anonymity of RS. □

The anonymity of our TRS scheme is equivalent to the anonymity of its underlying RS scheme, where the number of possible combinations of the signers' public keys is $n$.

## V. Instantiation of TRS

In this section, we first give an efficient instantiation of TRS and its security proofs. Then we give an optimization function of our instantiation to further reduce the signature size to logarithmic.

### A. Instantiation

Based on SWT and the DualRing (DR) RS scheme [34], we propose the instantiation of TRS in Algorithm 2. It is used to prove the following NP language:

$$\mathcal{L}_{TRS}(m) = \left\{ \mathbb{Y} \middle| \begin{array}{c} \exists \mathbb{X} = \{x_s, ..., x_{s+t-1}\} : \\ \forall x_i \in \mathbb{X}, g^{x_i} \in \mathbb{Y} \end{array} \right\} \quad (3)$$

**Algorithm 2** Instantiation of TRS

```
$\mathbf{Setup}(\lambda)$
    Initialize $(\mathbb{G}_p, g) \leftarrow \lambda$, $H_0 : \{0,1\}^* \rightarrow \mathbb{Z}_p$;
    Return $pp \leftarrow (\mathbb{G}_p, g, H_0, H_1)$;
$\mathbf{KeyGen}()$
    Choose $x \in \mathbb{Z}_p^*$; Compute $y = g^x$; Return $(y, x)$;
$\mathbf{Sign}(\mathbb{Y}, \mathbb{X}, m)$
    For each $i$ in range $[0, n-1]$:
        Compute $d_i \leftarrow H_0(y_0||...||y_{n-1}||i)$;
        Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k^{d_k}$;
    Choose $z \in \mathbb{Z}_p^*$; Compute $Z \leftarrow g^z$;
    For each $i$ in range $[s+1, s+n-1]$: Choose $c_i \in \mathbb{Z}_p^*$;
    Compute $c \leftarrow H_0(Z \cdot \prod_{i \neq s} \mathbf{y}_i^{c_i} ||m)$, $c_s \leftarrow c - \sum_{i \neq s} c_i$;
    Compute $r \leftarrow z - c_s \cdot \sum_{k=s}^{s+t-1} x_k \cdot d_k$;
    Return $\sigma \leftarrow (r, c_0, ..., c_{n-1})$;
$\mathbf{Verify}(\mathbb{Y}, \sigma, m)$
    Parse $\sigma = (r, c_0, ..., c_{n-1})$;
    For each $i$ in range $[0, n-1]$:
        Compute $d_i \leftarrow H_0(y_0||...||y_{n-1}||i)$;
        Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k^{d_k}$;
    If $\sum_{i=0}^{n-1} c_i = H_0(g^r \cdot \prod_{i=0}^{n-1} \mathbf{y}_i^{c_i} ||m)$, then return 1;
    Else return 0;
```

### B. Security Proofs of TRS

We give the security proofs of the above scheme as follows, where the instantiation of the signing oracle $\mathcal{SO}_C$ is shown in Algorithm 3.

**Theorem 3.** *The proposed TRS scheme satisfies t-unforgeability wrt insider corruption if the DL assumption holds.*

*Proof.* Assume that there is a simulator $\mathcal{S}$ which is given a DL problem instance $(g, g^\alpha)$. We will prove that if a

**Algorithm 3** Oracle $\mathcal{SO}_C(\mathbb{Y}, m)$

```
1: For each $((\mathbb{Y}_i, m_i), \sigma_i)$ in $\mathbb{L}_s$: if $\mathbb{Y}_i = \mathbb{Y} \wedge m_i = m$, return $\sigma_i$;
2: For each $i$ in range $[0, n-1]$:
3:     Compute $d_i \leftarrow \mathcal{HO}_0(y_0||...||y_{n-1}||i)$;
4:     Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k^{d_k}$;
5: Choose $r, c_0, ..., c_{n-1} \in \mathbb{Z}_p^*$ randomly;
6: Set $\mathbb{L}_{h_0} \leftarrow \mathbb{L}_{h_0} \cup \{(g^r \cdot \prod_{k=0}^{n-1} \mathbf{y}_k^{c_k}||m, \sum_{k=0}^{n-1} c_k)\}$;
7: Set $\sigma \leftarrow (r, c_0, ..., c_{n-1}), \mathbb{L}_s \leftarrow \mathbb{L}_s \cup \{((\mathbb{Y}, m), \sigma)\}$; Return $\sigma$;
```

PPT adversary $\mathcal{A}$ can break the $t$-unforgeability wrt insider corruption of our scheme with a non-negligible probability, then $\mathcal{S}$ can resolve the DL problem instance by invoking $\mathcal{A}$.
**Setup.** $\mathcal{S}$ runs $\mathbf{Setup}(\lambda) \to pp$ and chooses $s \in \{0, ..., n-1\}$ randomly. For each $i \in \{0, ..., n-1\}$, if $i = s$, then $\mathcal{S}$ sets $y_i \leftarrow g^\alpha$, otherwise $\mathcal{S}$ computes $(y_i, x_i) \leftarrow \mathbf{KeyGen}()$. $\mathcal{S}$ sets $\mathbb{Y} \leftarrow \{y_0, ..., y_{n-1}\}, \mathbb{X}^* \leftarrow \{x_{s+1}, ..., x_{s+t-1}\}$ and sends $\mathbb{Y}, \mathbb{X}^*$ to $\mathcal{A}$.
**Query.** $\mathcal{A}$ can adaptively query the oracles $\mathcal{SO}_C, \mathcal{HO}_0$ respectively. For $\mathcal{SO}_C$, if the $k$-th query $\mathbb{Y}_k = \mathbb{Y}$, then the game aborts, otherwise $\mathcal{SO}_C$ returns a valid signature $\sigma_k$ to $\mathcal{A}$. For the $k$-th query $m_k$ to $\mathcal{HO}_0$, it returns a hash value $c_k$ to $\mathcal{A}$.
**Forgery.** $\mathcal{A}$ generates a signature $\sigma' = (r', c'_0, ..., c'_{n-1})$ and sends it to $\mathcal{S}$. If $\mathbf{Verify}(\mathbb{Y}, \sigma', \cdot) = 1$, then $\mathcal{A}$ wins the game.

If $\mathcal{A}$ wins, then it has $1/n$ probability that $r' = z' - c'_s(\alpha d_s + \sum_{i=s+1}^{s+t-1} x_i d_i)$. According to the forking lemma, $\mathcal{S}$ rewinds the game and invokes $\mathcal{A}$ to generate another valid signature $\sigma'' = (r'', c''_0 ..., c''_{n-1})$ such that $r'' = z' - c''_s(\alpha d_s + \sum_{i=s+1}^{s+t-1} x_i d_i)$ where $r'' \neq r', c''_s \neq c'_s$. Then $\mathcal{S}$ can get the discrete logarithm $\alpha$ of $g^\alpha$ by computing that:

$$\alpha = \frac{r'' - r'}{(c'_s - c''_s)d_s} - \frac{x_{s+1}d_{s+1} + ... + x_{s+t-1}d_{s+t-1}}{d_s}.$$

It is contrary to the DL assumption. □

**Theorem 4.** *The proposed TRS scheme satisfies perfect anonymity wrt adversarial keys if the simulation is indistinguishable from reality.*

*Proof.* We will prove the theorem by constructing a simulator $\mathcal{S}$, which generates a valid signature using only the public key set. The adversary $\mathcal{A}$ cannot distinguish it from reality.
**Setup.** $\mathcal{S}$ generates the system parameter $pp$ by running $\mathbf{Setup}(\lambda)$, generates $(y_i, x_i)$ for each $i \in \{0, n-1\}$, sets $\mathbb{Y} = \{y_0, ..., y_{n-1}\}, \mathbb{X} = \{x_s, ..., x_{s+t-1}\}$, and sends $(pp, \mathbb{Y}, \mathbb{X})$ to $\mathcal{A}$. Then $\mathcal{S}$ flips a coin to choose $b \in \{0, 1\}$. If $b = 0$, then $\mathcal{S}$ runs $\mathbf{Sign}(\mathbb{Y}, \mathbb{X}, \cdot)$ and returns signature $\sigma_0$ to $\mathcal{A}$, else $\mathcal{S}$ runs $\mathcal{SO}_C(\mathbb{Y}, \cdot)$ and returns signature $\sigma_1$ to $\mathcal{A}$.
**Query.** $\mathcal{A}$ can adaptively query the oracles $\mathcal{SO}_C, \mathcal{HO}_0$ respectively. For the $k$-th query $(\mathbb{Y}_k, \cdot)$ to $\mathcal{SO}_C$, it returns a valid signature $\sigma_k$ to $\mathcal{A}$. For the $k$-th query $m_k$ to $\mathcal{HO}_0$, it returns a hash value $c_k$ to $\mathcal{A}$.
**Guess.** $\mathcal{A}$ returns a guess $b'$ to $\mathcal{S}$. If the probability that $b' = b$ is greater than 1/2, then $\mathcal{A}$ wins the game.

Since the simulator has been successfully constructed, and the signatures generated by both **Sign** and $\mathcal{SO}$ are valid, the two signatures are indistinguishable to the adversary $\mathcal{A}$. □

## C. Logarithmic Optimization

By introducing the sum argument (SA) scheme [34], we can further reduce the signature size of Algorithm 2 to logarithmic. Compared to Algorithm 2, this optimization offers advantages in scenarios where it is necessary to ensure a small signature size and limited network bandwidth, such as in mobile devices and sensor networks.

The SA scheme is a variant of the well-known inner product argument [42]. It is used to prove the following NP statement:

$$\mathcal{L}_{SA} = \left\{ (\mathbb{Y}, P, c) \middle| \begin{array}{c} \exists \mathbb{C} = \{c_0, ..., c_{n-1}\} : \\ P = \prod_{i=0}^{n-1} y_i^{c_i}, c = \sum_{i=0}^{n-1} c_i \end{array} \right\} \quad (4)$$

where $P \in \mathbb{G}_p$ and $c \in \mathbb{Z}_p$.

The SA scheme contains two algorithms $\mathbf{Prove}(\mathbb{Y}, P, c, \mathbb{C}) \to \pi$ and $\mathbf{Verify}(\mathbb{Y}, P, c, \pi) \to 1/0$, where the size of the proof $\pi$ is $2logn|\mathbb{G}_p| + 2|\mathbb{Z}_p|$. The details of these instantiations are in [34]. It can be seen in Algorithm 2 that the signature output by Algorithm **Sign** contains the random numbers $c_0, ..., c_{n-1}$. Also, in Algorithm **Verify**, $\prod_{i=0}^{n-1} y_i^{c_i}$ and $\sum_{i=0}^{n-1} c_i$ need to be calculated. This matches the zero-knowledge statement proved by SA. Therefore, we can use the **Prove** in SA to construct the statement $P$ and proof $\pi$ of $c_0, ..., c_{n-1}$. Formally, we give the instantiation of TRS optimized by SA in Algorithm 4.

**Algorithm 4** SA-TRS

```
1: $\mathbf{Setup}$ and $\mathbf{KeyGen}$ are the same as in Algorithm 2.
2: $\mathbf{Sign}(\mathbb{Y}, \mathbb{X}, m)$
3:     Run TRS.Sign$(\mathbb{Y}, \mathbb{X}, m) \to (r, c_0, ..., c_{n-1})$;
4:     Get $(\mathbf{y}_0, ..., \mathbf{y}_{n-1})$ generated in TRS.Sign;
5:     Compute $P \leftarrow \prod_{i=0}^{n-1} \mathbf{y}_i^{c_i}, c \leftarrow \sum_{i=0}^{n-1} c_i$;
6:     Set $\mathbb{C} \leftarrow \{c_0, ..., c_{n-1}\}, \mathbb{Y}' \leftarrow \{\mathbf{y}_0, ..., \mathbf{y}_{n-1}\}$;
7:     Run SA.$\mathbf{Prove}(\mathbb{Y}', P, c, \mathbb{C}) \to \pi$;
8:     Return $\sigma \leftarrow (r, P \cdot g^r, \pi)$;
9: $\mathbf{Verify}(\mathbb{Y}, \sigma, m)$
10:    Parse $\sigma = (r, Q, \pi)$;
11:    For each $i$ in range $[0, n-1]$:
12:        Compute $d_i \leftarrow H_0(y_0||...||y_{n-1}||i)$;
13:        Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k^{d_k}$;
14:    Set $\mathbb{Y}' \leftarrow \{\mathbf{y}_0, ..., \mathbf{y}_{n-1}\}$;
15:    Compute $c \leftarrow H_0(Q||m), P \leftarrow Q \cdot g^{-r}$;
16:    Return SA.$\mathbf{Verify}(\mathbb{Y}', P, c, \pi)$;
```

## D. Security Proofs of SA-TRS

**Theorem 5.** *SA-TRS satisfies unforgeability wrt insider corruption if the underlying TRS satisfies unforgeability wrt insider corruption.*

*Proof.* We will prove that if a PPT adversary $\mathcal{A}$ can break the unforgeability wrt insider corruption of SA-TRS, then there is a simulator $\mathcal{S}$ which can break the unforgeability wrt insider corruption of the underlying TRS by invoking $\mathcal{A}$.
**Setup.** It is the same as the setup phase in the proof of Theorem 3.
**Query.** When $\mathcal{A}$ sends the $k$-th query $(\mathbb{Y}_k, \cdot)$ to the signing oracle, if $\mathbb{Y}_k = \mathbb{Y}$, then the game aborts, otherwise $\mathcal{S}$ runs $\mathcal{SO}_C(\mathbb{Y}_k, \cdot) \to (r_k, c_{0_k}, ..., c_{n-1_k})$, then $\mathcal{S}$ runs line 2-7 in Algorithm 4 and returns $\sigma_k$ to $\mathcal{A}$.
**Forgery.** $\mathcal{A}$ generates a signature $\sigma' = (r', Q', \pi')$ and sends it to $\mathcal{S}$. If $\mathbf{Verify}(\mathbb{Y}, \sigma', \cdot) = 1$, then $\mathcal{A}$ wins the game.

TABLE III
COMPARISON OF SIGNATURE SIZE BETWEEN DR-BASED TRS AND SA-TRS (KB)

| Scheme \ Ring Size | 10 | 20 | 40 | 80 | 160 |
|---|---|---|---|---|---|
| DR-based TRS | 0.375 | 0.688 | 1.313 | 2.563 | 5.063 |
| SA-TRS | 0.281 | 0.319 | 0.357 | 0.394 | 0.432 |

If $\mathcal{A}$ wins, by the statistical witness-extended emulation of SA [34], $\mathcal{S}$ can run an emulator $\mathcal{E}$ to get $(r'_0, c'_0, ..., c'_{n-1})$. Then $\mathcal{S}$ runs line 4-6 in Algorithm 4 to generate $\mathbb{Y}'$. Finally, $\mathcal{S}$ returns a signature $(r', c'_0, ..., c'_{n-1})$ of the challenge $(\mathbb{Y}', \cdot)$ to break the unforgeability wrt insider corruption of TRS. □

**Theorem 6.** *SA-TRS satisfies perfect anonymity wrt adversarial keys if the underlying TRS satisfies perfect anonymity wrt adversarial keys.*

*Proof.* We will prove that if a PPT adversary $\mathcal{A}$ can break the perfect anonymity wrt adversarial corruption of SA-TRS, then there is a simulator $\mathcal{S}$ which can break the perfect anonymity wrt adversarial corruption of the TRS by invoking $\mathcal{A}$.
**Setup.** Assuming that $\mathcal{S}$ receives a challenge $(r, c_0, ..., c_{n-1})$ of $(\mathbb{Y}, \cdot)$. $\mathcal{S}$ runs lines 3-7 of Algorithm 4 and returns $\sigma = (r, Q, \pi)$ to $\mathcal{A}$.
The **Query** and **Guess** phases are similar to those in the proof of Theorem 4. We will not repeat them here.

If $\mathcal{A}$ wins the game, then $\mathcal{S}$ returns $b'$ generated by $\mathcal{A}$ as a response to the challenge to break the perfect anonymity wrt adversarial keys of TRS. □

The signature sizes of DR-based TRS and SA-TRS are compared as shown in Table III. We set the element size in $\mathbb{Z}_p$ to 256 bits and that in $\mathbb{G}_p$ to 512 bits, and set the threshold to half the ring size. As the ring size increases, the signature size of the former increases linearly, while that of the latter increases logarithmically.

## VI. INSTANTIATION OF LTRS

In this section, we introduce the design of our double-challenge linkable ring signature (DC-LRS). Combining it with SWT, we give the instantiation and security proofs of our LTRS scheme.

### A. Design of DC-LRS

The design of LRS was initially inspired by the undeniable signature scheme [43]. For a public key $y = g^x$ with secret key $x$, the signature of the message $m$ is denoted as $\sigma = H_1(m)^x$. In the first LRS scheme proposed by Liu et al. [19], it is proposed to realize linkability through the "serial number" of the signature, which is denoted as $sn = H_1(y_i)^{x_i}$, where $i \in \{0, ..., n-1\}$. By constructing an SoK of the following NP language, a signer binds his secret key to the serial number.

$$\mathcal{L}_{LRS}(m) = \left\{ (\mathbb{Y}, sn) \middle| \begin{array}{c} \exists i \in \{1, ..., n\}: \\ log_g y_i = log_{H(y_i)} sn_i \end{array} \right\} \tag{5}$$

If the signer uses the same secret key to sign different messages, the signature can be identified by judging whether the serial number in the two signatures is same.

However, SWT cannot be directly combined with the LRS scheme to construct a LTRS scheme. To illustrate this problem, we first review an example of a linkable schnorr signature scheme in Figure 3. It is used to prove the NP language

$$\mathcal{L}_{LSS} = \{(y, sn) | x : y = g^x \wedge sn = H(y)^x\}(m) \tag{6}$$

Suppose we replace the public key in the scheme with the product of multiple public keys, such as $y_1 \cdot y_2$, it satisfies that $y_1 \cdot y_2 = g^{x_1 + x_2}$ but there is no PPT algorithm $F$ such that $H_1(y_1)^{x_1} \cdot H_1(y_2)^{x_2} = F(x_1 + x_2)$. The reason is that the operation between the serial numbers of different public keys is not homomorphic.

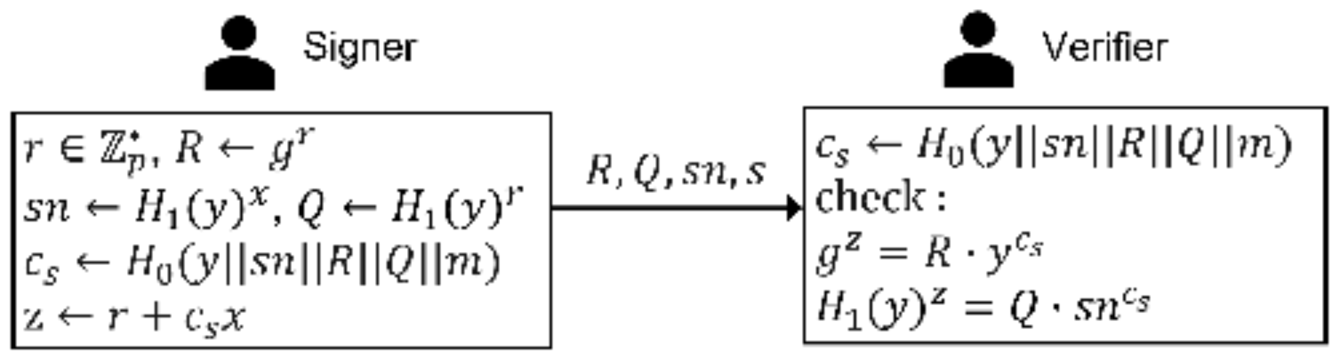

Signer
Verifier
$r \in \mathbb{Z}_p^*, R \leftarrow g^r$
$sn \leftarrow H_1(y)^x, Q \leftarrow H_1(y)^r$
$c_s \leftarrow H_0(y||sn||R||Q||m)$
$z \leftarrow r + c_s x$
$R, Q, sn, s$
$c_s \leftarrow H_0(y||sn||R||Q||m)$
check :
$g^z = R \cdot y^{c_s}$
$H_1(y)^z = Q \cdot sn^{c_s}$

Fig. 3. Linkable schnorr siganture.

To solve this problem, we consider separating the generation of hash challenges in the scheme from commitments $R$ and $Q$. Specifically, for the above example, we changed the equations checked in the verification step to $g^{z \cdot c_s} = R \cdot y^{c \cdot c_s}$ and $H_1(y)^{z \cdot c_s} = Q \cdot sn^{c \cdot c_s}$, where $c = H_0(y||sn||m)$. For the same $R = g^r$, $n$ different public keys $y_1, ..., y_n$ and $Q_1 = H_1(y_1)^r, ..., Q_n = H_1(y_n)^r$, signers with corresponding secret keys $x_1, ..., x_n$ can construct $z_i = r/c_s + c \cdot x_i$ respectively to satisfy the verifications $g^{z_i \cdot c_s} = R \cdot y_i^{c \cdot c_s}$ and $H_1(y)^{z_i \cdot c_s} = Q_i \cdot sn_i^{c \cdot c_s}$. In the above construction, it can be seen that each $z_i$ can be used to prove that each serial number $H_1(y_i)^{x_i}$ is honestly generated, thereby achieving linkability. Moreover, the design of LRS does not require the signer to pass $z_i$ in the signature (see Algorithm 5 for details), thereby preventing the disclosure of the information of signers' public keys. The instantiation of the DC-LRS scheme is shown in Algorithm 5.

**Algorithm 5** Instantiation of DC-LRS
```
 1: **Setup** and **KeyGen** are the same as in Algorithm 2.
 2: **Sign**$(\mathbb{Y}, x_s, m)$
 3:     Compute $sn \leftarrow H_1(y_s)^{x_s}, c = H_0(\mathbb{Y}||sn||m)$;
 4:     Choose $z \leftarrow \mathbb{Z}_p^*$ randomly;
 5:     Compute $c_{s+1} \leftarrow H_0((g \cdot H_1(y_s))^z||m)$;
 6:     For each $i$ in range $[s+1, s+n-1]$:
 7:         Choose $r_i \in \mathbb{Z}_p^*$;
 8:         Compute $c_{i+1} \leftarrow H_0((g \cdot H_1(y_i))^{r_i/c_i} \cdot (y_i \cdot sn)^c||m)$;
 9:     Compute $r_s \leftarrow c_s(z - cx_s)$;
10:     Return $\sigma \leftarrow (r_0, ..., r_{n-1}, c_0, sn)$;
11: **Verify**$(\mathbb{Y}, \sigma, m)$
12:     Parse $\sigma = (r_0, ..., r_{n-1}, c_0, sn)$;
13:     Compute $c'_1 \leftarrow H_0((g \cdot H_1(y_0))^{r_0/c_0} \cdot (y_0 \cdot sn)^c||m)$;
14:     For each $i$ in range $[1, n-1]$:
15:         Compute $c'_{i+1} \leftarrow H_0((g \cdot H_1(y_i))^{r_i/c'_i} \cdot (y_i \cdot sn)^c||m)$;
16:     If $c'_0 = c_0$, then return 1, else return 0;
```

### B. Security Proofs of DC-LRS

We give the security proofs of DC-LRS scheme as follows, where the instantiation of $\mathcal{SO}_{DC}$ is shown in Algorithm 6.

**Algorithm 6** Oracle $\mathcal{SO}_{DC}(\mathbb{Y}, m)$

```
1: For each $((\mathbb{Y}_i, m_i), \sigma_i)$ in $\mathbb{L}_s$: if $\mathbb{Y}_i = \mathbb{Y} \wedge m_i = m$, return $\sigma_i$;
2: Compute $c = \mathcal{HO}_0(\mathbb{Y}||sn||m)$;
3: Choose $r_0, ..., r_{n-1}, c_0 \in \mathbb{Z}_p^*$, $sn \in \mathbb{G}_p$ randomly;
4: For each $i$ in range $[0, n-2]$:
5:     Compute $c_{i+1} \leftarrow \mathcal{HO}_0((g \cdot H_1(y_i))^{r_i/c_i} \cdot (y_i \cdot sn)^c||m)$;
6: Set $\mathbb{L}_{h_0} \leftarrow \mathbb{L}_{h_0} \cup \left\{ \begin{pmatrix} (g \cdot H_1(y_{n-1}))^{r_{n-1}/c_{n-1}} \\ \cdot (y_{n-1} \cdot sn)^c||m, c_0 \end{pmatrix} \right\}$;
7: Set $\sigma \leftarrow (r_0, ..., r_{n-1}, c_0)$, $\mathbb{L}_s \leftarrow \mathbb{L}_s \cup \{((\mathbb{Y}, m), \sigma)\}$;Return $\sigma$;
```

**Theorem 7.** *The proposed DC-LRS scheme satisfies unforgeability if the DL assumption holds.*

*Proof.* We will prove that if a PPT adversary $\mathcal{A}$ can break the unforgeability of our scheme with a non-negligible probability, then a simulator $\mathcal{S}$ which is given a DL problem instance $(g, g^\alpha)$ can resolve the instance by invoking $\mathcal{A}$.
**Setup.** $\mathcal{S}$ generates the system parameter $pp$ by running $\mathbf{Setup}(\lambda)$. $\mathcal{S}$ chooses $s \in \{0, ..., n-1\}$ randomly. For each $i \in \{0, ..., n-1\}$, if $i = s$, $\mathcal{S}$ sets $y_i \leftarrow g^\alpha$, otherwise $\mathcal{S}$ runs $\mathbf{KeyGen}() \rightarrow (y_i, x_i)$. $\mathcal{S}$ sends $\mathbb{Y} = \{y_0, ..., y_{n-1}\}$ to $\mathcal{A}$.
**Query.** $\mathcal{A}$ can adaptively query the oracles $\mathcal{SO}_{DC}, \mathcal{HO}_0, \mathcal{HO}_1$ respectively. For $\mathcal{SO}_{DC}$, if the $k$-th query $\mathbb{Y}_k = \mathbb{Y}$, then the game aborts, else it returns a valid signature $\sigma_k$ to $\mathcal{A}$. For the $k$-th query $m_k$ to $\mathcal{HO}_0$ or $\mathcal{HO}_1$, it returns a hash value $c_k$.
**Forgery.** $\mathcal{A}$ generates a signature $\sigma' = (r'_0, ..., r'_{n-1}, c'_0)$ and sends it to $\mathcal{S}$. If $\mathbf{Verify}(\mathbb{Y}, \sigma', \cdot) = 1$, then $\mathcal{A}$ wins the game.

If $\mathcal{A}$ wins, then it has $1/n$ probability that $r'_s = c'_s(z' - c'\alpha)$. According to the forking lemma, $\mathcal{S}$ rewinds the game and invokes $\mathcal{A}$ to generate another valid signature $\sigma'' = (r'_0, ..., r'_{n-1}, c'_0)$ such that $r''_s = c''_s(z' - c''\alpha)$ where $r''_s \neq r'_s, c''_s \neq c'_s, c'' \neq c'$. Then $\mathcal{S}$ can get the discrete logarithm $\alpha$ of $g^\alpha$ by computing that:

$$\alpha = (r'_s/c'_s - r''_s/c''_s)/(c'' - c')$$

It is contrary to the DL assumption. ☐

**Theorem 8.** *The proposed DC-LRS scheme satisfies computational anonymity if the DDH assumption holds.*

*Proof.* Assuming that the simulator $\mathcal{S}$ receives a DDH problem instance $(g, g^\alpha, g^\beta, Z)$. We will prove that if $\mathcal{A}$ can break the computational anonymity with a non-negligible advantage $\epsilon$, then $\mathcal{S}$ can resolve this DDH instance by invoking $\mathcal{A}$.
**Setup.** $\mathcal{S}$ generates the system parameter $pp = (\mathbb{G}_p, g)$, sets $y_s = g^\alpha$ and generates other key pairs by $\mathbf{KeyGen}$ to form $\mathbb{Y} = \{y_0, ..., y_{s-1}, y_{s+1}, ..., y_{n-1}\}$. $\mathcal{S}$ sets $\mathbb{L}_{h_1} \leftarrow \mathbb{L}_{h_1} \cup \{(y_s, g^\beta)\}$, and runs $\mathcal{SO}_{DC}(\mathbb{Y}, \cdot) \rightarrow \sigma$ with modifying line 3 of Algorithm 6 to "Choose $r_0, ..., r_{n-1}, c_0 \in \mathbb{Z}_p^*$ randomly, compute $sn = Z$". $\mathcal{S}$ sends $(\mathbb{Y}, \sigma)$ to $\mathcal{A}$.
**Query.** The oracles $\mathcal{SO}_{DC}, \mathcal{HO}_0, \mathcal{HO}_1$ always respond to the queries from $\mathcal{A}$ normally.
**Guess.** $\mathcal{A}$ outputs a guess $b' \in \{0, ..., n-1\}$. If $\mathcal{A}$ guesses $b' = s$ with advantage $\epsilon$, then $\mathcal{A}$ wins the game.

If $Z = g^{\alpha\beta}$, then it has $sn_s = \mathcal{SO}_1(y_s)^{x_s}$. The simulation is indistinguishable from reality. Thus $\mathcal{A}$ has probability $1/2 + \epsilon$ of guessing correctly.

If $Z \neq g^{\alpha\beta}$, then it has $sn_s \neq \mathcal{HO}_1(y_s)^{x_s}$, so $sn_s$ is independent of $y_s$. Thus $\mathcal{A}$ only has a probability $1/2$ of guessing the message correctly.

Therefore, the probability that $\mathcal{A}$ solves the DDH problem is $\frac{1}{2} + \frac{1}{2}\epsilon$. It is contrary to the DDH assumption. ☐

**Theorem 9.** *The proposed DC-LRS scheme satisfies linkability if the DL assumption holds.*

*Proof.* Assume that there is a simulator $\mathcal{S}$ which is given a DL problem instance $(g, g^\alpha)$. We will prove that if a PPT adversary $\mathcal{A}$ can break the linkability of our scheme with a non-negligible probability, then $\mathcal{S}$ can resolve the DL problem instance by invoking $\mathcal{A}$.
**Setup.** $\mathcal{S}$ generates the system parameter $pp$ by running $\mathbf{Setup}(\lambda)$. $\mathcal{S}$ chooses $s \in \{0, ..., n-1\}$ randomly. $\mathcal{S}$ runs $\mathbf{KeyGen}$ to generate some key pairs and forms $\mathbb{Y} = \{y_0, ..., y_{n-1}\}, \mathbb{Y}' = \{y'_0, ..., y'_{s-1}, y_s, y'_{s+1}, ..., y'_{n-1}\}, x_s$, where $\forall i \in \{0, n-1\}, i \neq s, y_i \neq y'_i$. $\mathcal{S}$ sets $\mathbb{L}_{h_1} \leftarrow \mathbb{L}_{h_1} \cup \{(y_s, g^\alpha)\}$ and sends $(\mathbb{Y}, \mathbb{Y}', x_s)$ to $\mathcal{A}$.
**Query.** The oracles always respond to the queries from $\mathcal{A}$ normally.
**Forgery.** $\mathcal{A}$ returns a signature $\sigma' = (r'_0, ..., r'_{n-1}, c'_0, sn')$ to $\mathcal{S}$. If $\mathbf{Verify}(\mathbb{Y}', \sigma', \cdot) = 1$, $\mathbf{Link}(\sigma, \sigma') = 0$, then $\mathcal{A}$ wins the game.

If $\mathcal{A}$ wins, according to $\mathbf{Link}(\sigma, \sigma') = 0$, it has $sn'_s \neq sn_s = \mathcal{HO}_1(y_s)^{x_s}$. Might as well denote $sn' = \mathcal{HO}_1(y_s)^{x_s}$, where $x'_s \neq x_s$. According to $\mathbf{Verify}(\mathbb{Y}', \sigma', \cdot) = 1$, it has

$$(g \cdot \mathcal{HO}_1(y_s))^{r'_s/c'_s} = (g \cdot \mathcal{HO}_1(y_s))^{z'_s}/(y_s \cdot sn'_s)^{c'}$$

Because $\mathcal{HO}(y_s) = g^\alpha$, it has:

$$(1+\alpha)(r'_s/c'_s) = (1+\alpha)z'_s - (x_s + \alpha x'_s)c'$$

$$\alpha = \frac{z'_s - c' \cdot x_s - r'_s/c'_s}{r'_s/c'_s - z'_s + c' \cdot x'_s}$$

Therefore, $\mathcal{S}$ can get the discrete logarithm $\alpha$ of $g^\alpha$, which is contrary to the DL assumption. ☐

## C. Instantiation of LTRS

Based on DC-LRS and SWT, we give the instantiation of the LTRS scheme as shown in Algorithm 7. It is used to prove the following NP language, where $\mathbb{SN}$ is a set of serial numbers.

$$\mathcal{L}_{LTRS}(m) = \left\{ \begin{matrix} (\mathbb{Y}, \\ \mathbb{SN}) \end{matrix} \middle| \begin{matrix} \exists \mathbb{X} = \{x_s, ..., x_{s+t-1}\} : \forall x_i \in \mathbb{X}, \\ g^{x_i} \in \mathbb{Y} \wedge H_1(y_i)^{x_i} \in \mathbb{SN} \end{matrix} \right\} \tag{7}$$

In order to make each signer collaborate to construct a signature without disclosing their secret keys, the **Sign** algorithm of Algorithm 7 can be performed as follows, where $P_s, ..., P_{s+t-1}$ denotes to the $t$ signers.

(1) On line 3 of Algorithm 7, each signer $P_i$ computes $sn_i$ and sends it to $P_s$, where $i = s, ..., s+t-1$;
(2) $P_s$ runs the statements on line 4 and the first two statements on line 5. Then $P_0$ sends $c, z_s$ to $P_1, ..., P_{t-1}$;
(3) Each signer $P_i$ computes $C_i \leftarrow (g/H(y_i))^{z_s + c \cdot x_i}$ and sends it to $P_s$, where $i = s, ..., s+t-1$;
(4) $P_s$ computes $c_{s+1} \leftarrow H_0(\prod_{k=s}^{s+t-1} \frac{C_k}{(g \cdot H(y_i))^{c \cdot x_s}})||m)$. Then he runs the rest of the statements in **Sign**.

**Algorithm 7** Instantiation of LTRS

```
1: $\textbf{Setup}(\lambda)$ and $\textbf{KeyGen}()$ are the same as in Algorithm 2.
2: $\textbf{Sign}(\mathbb{Y}, \mathbb{X}, m)$
3:     Initialize $\mathbb{SN} \leftarrow \emptyset$; For each $i$ in range $[s, s+t-1]$: Compute $sn_i \leftarrow H_1(y_i)^{x_i}$, set $\mathbb{SN} \leftarrow \mathbb{SN} \cup \{sn_i\}$;
4:     Compute $\mathbf{sn} \leftarrow \prod_{k=0}^{t-1} sn_{s+k}^{H_0(k||\mathbb{SN})}$; For each $i$ in range $[0, n-1]$: Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k, \mathbf{h}_i \leftarrow \prod_{k=i}^{i+t-1} H_1(y_k)^{H_0(k-i||\mathbb{SN})}$;
5:     Compute $c \leftarrow H_0(\mathbb{Y}||\mathbb{SN}||m)$; Choose $z_s \in \mathbb{Z}_p^*$; For each $i$ in range $[s+1, s+t-1]$: Compute $z_i \leftarrow z_s + c(x_i - x_s)$;
6:     Compute $c_{s+1} \leftarrow H_0(\prod_{k=s}^{s+t-1}(g \cdot H_1(y_k)^{H_0(k-i||\mathbb{SN})})^{z_k}||m)$;
7:     For each $i$ in range $[s+1, s+n-1]$: Choose $r_i \in \mathbb{Z}_p^*$, compute $c_{i+1} \leftarrow H_0((g^t \cdot \mathbf{h}_i)^{r_i/c_i} \cdot (\mathbf{y}_i \cdot \mathbf{sn})^c||m)$;
8:     Compute $r_s \leftarrow c_s(z_s - c \cdot x_s)$; Return $\sigma \leftarrow (r_0, ..., r_{n-1}, \mathbb{SN}, c_0)$;
9: $\textbf{Verify}(\mathbb{Y}, \sigma, m)$
10:    Parse $\sigma = (r_0, ..., r_{n-1}, \mathbb{SN} = \{sn_0, ..., sn_{t-1}\}, c_0)$; Compute $\mathbf{sn} \leftarrow \prod_{k=0}^{t-1} sn_k^{H_0(k||\mathbb{SN})}, c = H_0(\mathbb{Y}||\mathbb{SN}||m)$;
11:    For each $i$ in range $[0, n-1]$: Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k, \mathbf{h}_i \leftarrow \prod_{k=i}^{i+t-1} H_1(y_k)^{H_0(k-i||\mathbb{SN})}$;
12:    Compute $c'_1 \leftarrow H_0((g^t \cdot \mathbf{h}_0)^{r_0/c_0} \cdot (\mathbf{y}_0 \cdot \mathbf{sn})^c||m)$; For each $i$ in range $[1, n-1]$: Compute $c'_{i+1} \leftarrow H_0((g^t \cdot \mathbf{h}_i)^{r_i/c'_i} \cdot (\mathbf{y} \cdot \mathbf{sn})^c||m)$;
13:    If $c'_0 \neq c_0$, then return 0, else return 1;
14: $\textbf{Link}(\sigma, \sigma')$
15:    If there is $sn$ in $\sigma$, $sn'$ in $\sigma'$ and $sn = sn'$, then return 1, else return 0;
```

**Algorithm 8** Oracle $\mathcal{SO}_L(\mathbb{Y}, m)$

```
1: For each $((\mathbb{Y}_i, m_i), \sigma_i)$ in $\mathbb{L}_s$: if $\mathbb{Y}_i = \mathbb{Y} \wedge m_i = m$, return $\sigma_i$;
2: Parse $\mathbb{Y} \rightarrow \{y_0, ..., y_{n-1}\}$;
3: For each $i$ in range $[0, t-1]$:
4:     Choose $sn_i \in \mathbb{G}_p^*$ randomly; Set $\mathbb{SN} \leftarrow \mathbb{SN} \cup \{sn_i\}$;
5: Compute $\mathbf{sn} \leftarrow \prod_{k=0}^{t-1} sn_k$;
6: Compute $c \leftarrow \mathcal{HO}_0(\mathbb{Y}||\mathbb{SN}||m)$; Choose $c_0$ randomly;
7: For each $i$ in range $[0, n-1]$:
8:     Choose $r_i \in \mathbb{Z}_p^*$ randomly; Compute $\mathbf{y}_i \leftarrow \prod_{k=i}^{i+t-1} y_k$;
9:     Compute $\mathbf{h}_i \leftarrow \prod_{k=i}^{i+t-1} \mathcal{HO}_1(y_k)^{\mathcal{HO}_0(k-i||\mathbb{SN})}$;
10:    Compute $C_i \leftarrow (g^t/\mathbf{h}_i)^{r_i/c_i} \cdot (\mathbf{y}_i/\mathbf{sn})^c$;
11:    Compute $c_{i+1} \leftarrow \mathcal{HO}_0(C_i||m)$;
12: Set $\mathbb{L}_{h_0} \leftarrow \mathbb{L}_{h_0} \backslash \{(C_{n-1}, c_n)\} \cup \{(C_{n-1}, c_0)\}$;
13: Set $\sigma \leftarrow (r_0, ..., r_{n-1}, \mathbb{SN}, c_0)$;
14: Set $\mathbb{L}_s \leftarrow \mathbb{L}_s \cup \{((\mathbb{Y}, m), \sigma)\}$; Return $\sigma$;
```

## *D. Security Proofs of LTRS*

We give the security proofs of the above instantiation as follows. We first give the instantiation of the signing oracle $\mathcal{SO}_L$ in Algorithm 8.

**Theorem 10.** *The proposed LTRS scheme satisfies t-unforgeability wrt insider corruption if the DL assumption holds.*

*Proof.* The steps of the game are the same as those of Theorem 3, except that the signing oracle $\mathcal{SO}_C$ is replaced by $\mathcal{SO}_L$.

If $\mathcal{A}$ wins the game, then it has 1/n probability that $r'_s = c'_s(r' - c'\alpha)$. According to the forking lemma, $\mathcal{S}$ rewinds the game and invokes $\mathcal{A}$ to generate another valid signature $\sigma'' = (r''_0, ..., r''_{n-1}, \cdot, c''_0)$ such that $r''_s = c''_s(r' - c''\alpha)$ where $r'' \neq r', c''_s \neq c'_s, c'' \neq c'$. Then $\mathcal{S}$ can get the discrete logarithm $\alpha$ of $g^\alpha$ by computing that $\alpha = \frac{r'_s/c'_s - r''_s/c''_s}{c'' - c'}$. It is contrary to the DL assumption. □

**Theorem 11.** *The proposed LTRS scheme satisfies computational anonymity wrt adversarial keys if the DDH assumption holds.*

*Proof.* The steps of the game are similar to those of Theorem 8, except that the **Setup** phase is changed as follows:
**Setup.** $\mathcal{S}$ generates the system parameter $pp = (\mathbb{G}_p, g)$, sets $y_s = g^\alpha$ and generates other key pairs by $\textbf{KeyGen}$ to form $\mathbb{Y} = \{y_0, ..., y_{n-1}\}, \mathbb{X}^* = \{x_{s+1}, ..., s_{s+t-1}\}$. $\mathcal{S}$ sets $\mathbb{L}_{h_1} \leftarrow \mathbb{L}_{h_1} \cup \{(y_s, g^\beta)\}$, and runs $\mathcal{SO}_L(\mathbb{Y}, \cdot) \rightarrow \sigma$ with modifying line 5 of Algorithm 8 to "If $i = 0$, then compute $sn_i \leftarrow Z$, otherwise compute $sn_i \leftarrow \mathcal{HO}_1(y_{s+i})^{x_{s+i}}$". $\mathcal{S}$ sends $(\mathbb{Y}, \mathbb{X}^*, \sigma)$ to $\mathcal{A}$.

□

**Theorem 12.** *The proposed LTRS scheme satisfies linkability if the DL assumption holds.*

*Proof.* The steps of the game are similar to those of Theorem 9, except that the **Setup** phase is changed as follows:
**Setup.** $\mathcal{S}$ generates the system parameter $pp$ by running $\textbf{Setup}(\lambda)$. $\mathcal{S}$ chooses $s \in \{0, ..., n-1\}$ randomly. $\mathcal{S}$ runs $\textbf{KeyGen}$ to generate some key pairs and forms $\mathbb{Y} = \{y_0, ..., y_{n-1}\}, \mathbb{X} = \{x_s, ..., x_{s+t-1}\}, \mathbb{Y}' = \{y'_0, ..., y'_{s-1}, y_s, y'_{s+1}, ..., y'_{n-1}\}, \mathbb{X}' = \{x_s, x'_{s+1} ..., x'_{s+t-1}\}, \forall i \in \{s+1, s+t-1\}, x_i \neq x'_i$. $\mathcal{S}$ sets $\mathbb{L}_h \leftarrow \mathbb{L}_h \cup \{(y_s, g^\alpha)\}$, runs $\textbf{Sign}(\mathbb{Y}, \mathbb{X}, \cdot) \rightarrow \sigma$ and sends $(\mathbb{Y}, \mathbb{X}, \mathbb{Y}', \mathbb{X}')$ to $\mathcal{A}$.

□

## VII. THRESHOLD RINGCT PROTOCOL

In this section, we first describe the workflow of our threshold RingCT protocol, then we give the design and security analysis of it. Finally, we presented the advantages of our design compared to the original RingCT protocol [2].

### *A. Overview*

We assume that in a peer-to-peer network, there are three participants: the payer, the payee, and the miner. They interact with each other through the following steps, as depicted in Figure 4.

(1) The payee generates his receiving address through an address derivation algorithm, and then sends the receiving address to the payer.
(2) The payer constructs an anonymous transaction. It hides the transaction amount through the signature algorithm in our TRS scheme, and hides the addresses of the payer and payee through the signature algorithm in our LTRS scheme. The payer broadcasts the transaction to the miners.
(3) When a miner receives a transaction, he first verifies the validity of the transaction. Specifically, he verifies that the

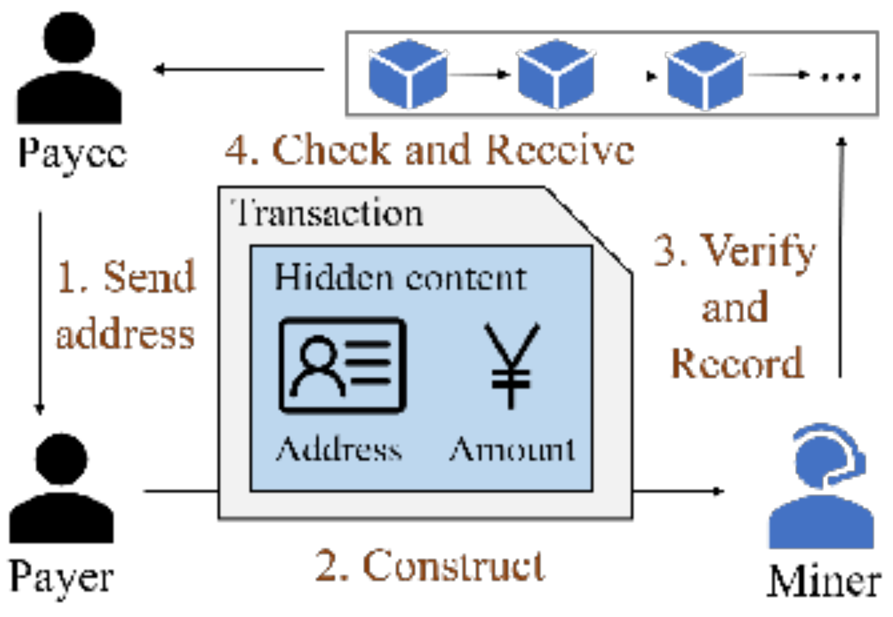


Fig. 4. Workfow of threshold RingCT protocol.

input amount of the transaction equals the output amount through our TRS scheme's verification algorithm, and that the payer has the authority to spend this amount in the input account through our LTRS scheme's verification algorithm. The miner verifies these properties without knowing the hidden content of the transaction. If the transaction is valid, the miner packages the transaction into a candidate block, and works with other miners to add the block to the blockchain through its consensus mechanism.

(4) The payee confirms whether the transfer is completed by checking the transaction records stored in the blockchain related to their receiving address through the receiving algorithm in our protocol.

## B. Design of Threshold RingCT

We denote $lpk = (pk_A, pk_B)$ as a long-term public key with the long-term secret key $lsk = (sk_A, sk_B)$. $y$ is a one-time public key with the one-time secret key $x$. $C$ is a Pedersen commitment of a random number $a$ and an amount $v$. We denote $act = (y, C)$ as an account with the secret account key $sak = (x, a, v)$. The party that owns $sak$ can construct an anonymous transaction to spend the amount $v$ of the account $act$. $\mathbb{A} = \{act_0, ..., act_{n-1}\}$ is an account set and $\mathbb{S} = \{sak_s, ..., sak_{s+t-1}\}$ is an secret account key set. The threshold RingCT protocol contains the following PPT algorithms:

$\textbf{Setup}(\lambda) \rightarrow pp$. On input a security parameter $\lambda$, the algorithm outputs the system parameter $pp$.

$\textbf{LKGen}() \rightarrow (lpk, lsk)$. The algorithm outputs a long-term public key $lpk$ and its corrsponding long-term secret key $lsk$.

$\textbf{OpkGen}(lpk) \rightarrow (y, R)$. On input a long-term public key $lpk$, it outputs a one-time public key $y$ and a label $R$.

$\textbf{OskExt}(y, lpk, lsk, R) \rightarrow x$. On input a one-time public key $y$, a long-term key pair $lpk, lsk$ and a label $R$, the algorithm outputs a one-time secret key $x$.

$\textbf{Spend}(\mathbb{A}, \mathbb{S}, lpk, m) \rightarrow (tx, \sigma)$. On input an account set $\mathbb{A}$, an secret account key set $\mathbb{S}$, a long-term public key $lpk$, and some auxiliary information $m \in \{0,1\}^*$, the algorithm outputs a transaction $tx$ and a signature $\sigma$.

$\textbf{Verify}(tx, \sigma) \rightarrow 0/1$. On input a transaction $tx$ and a signature $\sigma$, the algorithm outputs $0/1$ if the proof is in/valid.

$\textbf{Link}(\sigma, \sigma') \rightarrow 0/1$. On input two signature $\sigma$ and $\sigma'$, it outputs $0/1$ if the two signatures are un/linked.

$\textbf{Receive}(tx, lpk, lsk) \rightarrow sak$. On input a transaction $tx$, and the long-term key pair $lpk, lsk$, the algorithm outputs an secret account key $sak$.

We give the instantiation of the proposed threshold RingCT protocol in Algorithm 9, where $\textbf{PKE}.\textbf{Enc}(m, y), \textbf{PKE}.\textbf{Dec}(ct, x)$ are public key encryption and decryption algorithms with message $m$, public key $y$, ciphertext $ct$, and secret key $x$. For the case of multiple payers, they need to cooperate to generate signature $\sigma_{LTRS}$ in the **Spend** algorithm, which we have covered in Section VI-C.

## C. Security Properties and Proofs

We present the security properties of threshold RingCT and the security proofs of our instantiation as follows:

**Unforgeability.** No PPT adversary $\mathcal{A}$ can spend amount that belongs to an account without the secret account key.

*Proof.* If $\mathcal{A}$ can break the unforgeability of the proposed threshold RingCT, then there is an adversary $\mathcal{B}$ that can break the unforgeability of the underlying LTRS.

For a given challenge$(\mathbb{Y} = \{y_0, ..., y_{n-1}\}, \mathbb{X}^* = \{x_{n+1}, ..., x_{n+t-1}\})$, $\mathcal{B}$ generates other required variables, and uses them together with $(\mathbb{Y}, \mathbb{X}^*)$ to form $(\mathbb{A}, \mathbb{S}^*)$. Then $\mathcal{B}$ sends $(\mathbb{A}, \mathbb{S}^*)$ to $\mathcal{A}$. If $\mathcal{A}$ can forge a valid pair $(tx', \sigma' = (\sigma'_{LTRS}, \cdot))$ of spending $t$ accounts, then it has $\textbf{LTRS}.\textbf{Verify}(\mathbb{Y}, \sigma'_{LTRS}, tx') = 1$. $\mathcal{B}$ can use $\sigma'_{LTRS}$ as a valid forgery of message $tx'$, thus breaking the unforgeability of the underlying LTRS. □

**Balance.** No PPT adversary $\mathcal{A}$ can construct a transaction in which the input amount is not equal to the output amount.

*Proof.* If $\mathcal{A}$ can break the balance of our protocol, then there is an adversary $\mathcal{B}$ that can break the DL assumption.

For a given DL problem instance $(g, g^\alpha)$, $\mathcal{B}$ sets the system parameter $pp \leftarrow (\cdot, g, g^\alpha, \cdot, \cdot)$. Then $\mathcal{B}$ forms $a, v$ and a pair $(\mathbb{A}, \mathbb{S} = (\cdot, a_s, v_s), ..., (\cdot, a_{s+t-1}, v_{s+t-1}))$ in the required form which satisfy $v \neq \sum_{k=s}^{s+t-1} v_k$. Then $\mathcal{B}$ sends $(a, v, \mathbb{A}, \mathbb{S})$ to $\mathcal{A}$. If $\mathcal{A}$ can construct a valid pair $(tx' = (\mathbb{A}, (\cdot, g^a g^{\alpha \cdot v}), \cdot, \cdot, \cdot), \sigma' = (\cdot, \sigma'_{RS} = (c'_0, r'_0, ..., r'_{n-1})))$, then $\mathcal{B}$ rewinds the game and invokes $\mathcal{A}$ to generate another valid pair where $\sigma''_{RS} = (c''_0, r''_0, ..., r''_{n-1}), c'_0 \neq c''_0, r'_i \neq r''_i$ for $i \in \{0, ..., n-1\}$. $\mathcal{B}$ can get $\alpha$ by computing $\alpha = \frac{r'_s - r''_s + \tilde{a}(c'_s - c''_s)}{\tilde{v}(c''_s - c'_s)}$, where $\tilde{a} = (\sum_{k=s}^{s+t-1} a_k) - a, \tilde{v} = (\sum_{k=s}^{s+t-1} v_k) - v$. It is contrary to the DL assumption. □

**Computational Anonymity.** Give t-1 accounts and their secret account keys to a PPT adversary $\mathcal{A}$. For a transaction that spends the t-1 accounts and an additional account, $\mathcal{A}$ cannot distinguish the addition account from the input account set.

*Proof.* If $\mathcal{A}$ can break the computational anonymity of our protocol, then there is an adversary $\mathcal{B}$ that can break the computational anonymity of the underlying LTRS.

For a given challenge $(\mathbb{Y}, \mathbb{X}^*, \sigma_{LTRS}, \cdot)$, $\mathcal{B}$ generates $sak = (x, a, v)$. For $k = s, ..., s+t-1$, $\mathcal{B}$ chooses $a_k, v_k \in \mathbb{Z}_p^*$ and computes $C_k = g^{a_k} h^{v_k}$ where $v = \sum_{k=s}^{s+t-1} v_k$. For $k = 0, ..., s-1, s+t, ..., n-1$, $\mathcal{B}$ chooses $C_k \in \mathbb{G}_p$ randomly. Then $\mathcal{B}$ generates $\mathbb{C}$ by performing line 15 of Algorithm 9

**Algorithm 9** Instantiation of Threshold RingCT

```
1: **Setup**$(\lambda)$
2:     Initialize $(\mathbb{G}_p, g, h) \leftarrow \lambda, H_0 : \{0,1\}^* \rightarrow \mathbb{Z}_p, H_1 : \{0,1\}^* \rightarrow \mathbb{G}_p$; Return $pp \leftarrow (\mathbb{G}_p, g, h, H_0, H_1)$;
3: **LKGen**()
4:     Choose $sk_A, sk_B \in \mathbb{Z}_p^*$ randomly; Compute $pk_A \leftarrow g^{sk_A}, pk_B \leftarrow g^{sk_B}$; Return $(lpk \leftarrow (pk_A, pk_B), lsk \leftarrow (sk_A, sk_B))$;
5: **OpkGen**$(lpk)$
6:     Parse $lpk = (pk_A, pk_B)$; Choose $r \in \mathbb{Z}_p^*$ randomly; Compute $R \leftarrow g^r$; Compute $y \leftarrow g^{H_0(pk_A^r)} \cdot pk_B$; Return $(y, R)$;
7: **OskExt**$(y, lpk, lsk, R)$
8:     Parse $lpk = (pk_A, pk_B), lsk = (sk_A, sk_B)$; If $y = g^{H_0(R^{sk_A})} \cdot pk_B$, then return $x \leftarrow H_0(R^{sk_A}) + sk_B$, else return 0;
9: **Spend**$(\mathbb{A}, \mathbb{S}, lpk, m)$
10:    Parse $lpk = (pk_A, pk_B), \mathbb{A} = \{act_0, ..., act_{n-1}\}, \mathbb{S} = \{sak_s, ..., sak_{s+t-1}\}$;
11:    For each $i$ in range $[0, n-1]$: Parse $act_i = (y_i, C_i)$; For each $i$ in range $[s, s+t-1]$: Parse $sak_i = (x_i, a_i, v_i)$;
12:    Set $\mathbb{Y} \leftarrow \{y_0, ..., y_{n-1}\}, \mathbb{X} \leftarrow \{x_s, ..., x_{s+t-1}\}$; Compute $v = \sum_{k=s}^{s+t-1} v_k$;
13:    Run **OpkGen**$(lpk) \rightarrow (y, R, r)$; Choose $a \in \mathbb{Z}_p^*$ randomly; Compute $C \leftarrow g^a h^v$; Set $act \leftarrow (y, C)$;
14:    Run **PKE.Enc**$((a, v), y) \rightarrow ct$; Set $tx \leftarrow (\mathbb{A}, act, R, ct, m)$; Compute $\tilde{a} \leftarrow \sum_{k=s}^{s+t-1} a_k - a$;
15:    For each $i$ in range $[0, n-1]$: Compute $\mathbf{C}_i \leftarrow (\prod_{k=i}^{i+t-1} C_k)/C$; Set $\mathbb{C} \leftarrow \{\mathbf{C}_0, ..., \mathbf{C}_{n-1}\}$;
16:    Run **LTRS.Sign**$(\mathbb{Y}, \mathbb{X}, tx) \rightarrow \sigma_{LTRS}$; Run **RS.Sign**$(\mathbb{C}, \tilde{a}, tx) \rightarrow \sigma_{RS}$; Return $(tx, \sigma \leftarrow (\sigma_{LTRS}, \sigma_{RS}))$;
17: **Verify**$(tx, \sigma)$
18:    Parse $tx = (\mathbb{A} = \{act_0 = (y_0, C_0), ..., act_{n-1} = (y_{n-1}, C_{n-1})\}, act = (y, C), R, m)$; Set $\mathbb{Y} \leftarrow \{y_0, ..., y_{n-1}\}$;
19:    For each $i$ in range $[0, n-1]$: Compute $\mathbf{C}_i \leftarrow (\prod_{k=i}^{i+t-1} C_k)/C$; Set $\mathbb{C} \leftarrow \{\mathbf{C}_0, ..., \mathbf{C}_{n-1}\}$;
20:    If **LTRS.Verify**$(\mathbb{Y}, \sigma_{LTRS}, tx) = 1 \wedge$ **RS.Verify**$(\mathbb{C}, \sigma_{RS}, tx) = 1$, then return 1, else return 0;
21: **Link**$(\sigma, \sigma')$
22:    Parse $\sigma = (\sigma_{LTRS}, \sigma_{RS}), \sigma' = (\sigma'_{LTRS}, \sigma'_{RS})$; Return **LTRS.Link**$(\sigma_{LTRS}, \sigma'_{LTRS})$;
23: **Receive**$(tx, lpk, lsk)$
24:    Parse $tx = (\mathbb{A}, act = (y, C), R, ct, m)$; Run **OskExt**$(y, lpk, lsk, R) \rightarrow x$;
25:    If $x \neq 0$, then run **PKE.Dec**$(ct, x) \rightarrow (a, v)$, return $sak \leftarrow (x, a, v)$, else return 0;
```

and runs $\mathbf{RS}.\mathbf{Sign}(\mathbb{C}, (\sum_{k=s}^{s+t-1} a_k) - a, \cdot) \rightarrow \sigma_{RS}$. $\mathcal{B}$ sends $(\sigma_{LTRS}, \sigma_{RS})$ to $\mathcal{A}$. Finally, $\mathcal{A}$ returns a guess $b'$ to $\mathcal{B}$, and $\mathcal{B}$ returns $b'$ to break the computational anonymity of the underlying LTRS. □

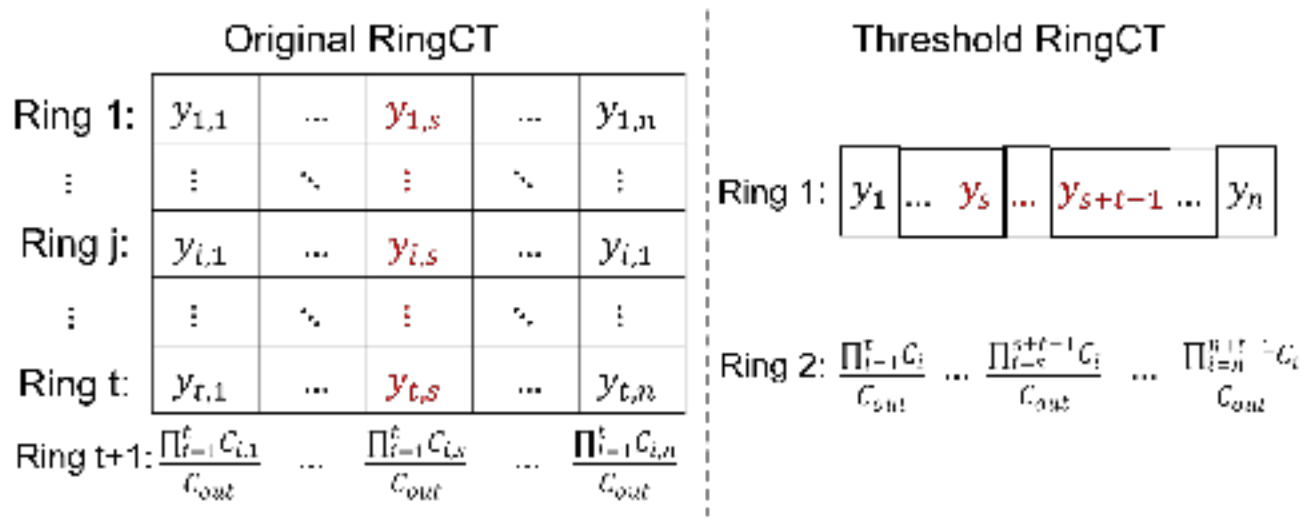
**Linkability.** Two transactions that spend the amount of the same account will be linked.

*Proof.* Assume that $\mathcal{A}$ outputs two valid pairs $(tx', \sigma')$ that cost the amount in the same account, and $\mathbf{Link}(\sigma', \sigma'') = 0$. Then it has $\mathbf{LTRS}.\mathbf{Link}(\sigma'_{\mathbf{LTRS}}, \sigma''_{\mathbf{LTRS}}) = \mathbf{0}$. $\mathcal{B}$ can returns $(\sigma'_{LTRS}, \sigma''_{LTRS})$ of the challenge $(\mathbb{Y}', \mathbb{X}', \mathbb{Y}'', \mathbb{X}'')$ to break the linkability of the underlying LTRS, where $(\mathbb{Y}', \mathbb{X}')$ are from $tx'$ and $(\mathbb{Y}'', \mathbb{X}'')$ are from $tx''$. □

## D. Comparison with Original Protocol

The comparison of the transaction's signature in the original RingCT protocol [24] and our proposal is shown in Figure 5. In [24], for $t$ input accounts, additional $t(n-1)$ auxiliary accounts are needed to construct the signature. Our protocol only needs $n-t$ auxiliary accounts.

In some cases, the anonymity of our scheme is better than that of CLSAG in some cases. The reason is that CLSAG requires the payer's accounts to be in the same index of each ring (that is, the same column in the matrix in Figure 5). Assuming the adversary identifies $t'$ of $t$ input accounts for the transaction ($t' < t$), the other $t - t'$ accounts are exposed. For our scheme, the adversary only guesses the other accounts with the probability of $\frac{1}{t-t'+1}$.



Original RingCT
Threshold RingCT
Ring 1:
Ring j:
Ring t:
Ring t+1:
$y_{1,1}$ ... $y_{1,s}$ ... $y_{1,n}$
$y_{i,1}$ ... $y_{i,s}$ ... $y_{i,1}$
$y_{t,1}$ ... $y_{t,s}$ ... $y_{t,n}$
$\frac{\prod_{i=1}^{t} C_{i,1}}{C_{out}}$ ... $\frac{\prod_{i=1}^{t} C_{i,s}}{C_{out}}$ ... $\frac{\prod_{i=1}^{t} C_{i,n}}{C_{out}}$
Ring 1: $y_1$ ... $y_s$ ... $y_{s+t-1}$ ... $y_n$
Ring 2: $\frac{\prod_{i=1}^{t} C_i}{C_{out}}$ ... $\frac{\prod_{i=s}^{s+t-1} C_i}{C_{out}}$ ... $\frac{\prod C_i}{C_{out}}$

Fig. 5. Comparison of the construction of the transaction signature in the original protocol and our proposal.

However, the requirement for adjacent real input accounts could lead to a loss of anonymity in our protocol in some cases. For example, assuming there are two users who are respectively the owners of the $i$-th and $j$-th accounts, where $j > i$ and the distance $j - i + 1 <= t$. Then they can jointly determine that the $j - i + 1$ accounts between them are not real input accounts. Setting the number of input accounts $t = 2$ per ring can avoid this issue while achieving a communication overhead reduction of about 31% compared to the original RingCT. In application scenarios requiring higher transaction efficiency, anonymity can be appropriately compromised by setting a higher $t$.

## VIII. Experiment

The simulation experiment is performed on a laptop with a 1.80GHz Intel Core i5-8265U CPU, 8GB memory, and Ubuntu 16.04 operating system. We construct implementations of group operations based on Short Weierstrass elliptic curve parameters provided in the golang library.

Figure 6 shows the comparison of the signing/verifying time between our proposed TRS scheme and other TRS schemes. We set the condition that the ring size increases from 10 to 100 and the threshold to half the ring size. For each case, we repeated the signing and verifying algorithm 100 times and took their average execution time. It can be seen that our scheme is similar in terms of signing and verifying time to [26] and faster than [27], [28]. As shown in Figure 7, we compare the proposed LTRS scheme and other LTRS schemes. The experiment set is the same as that of the experiment in Figure

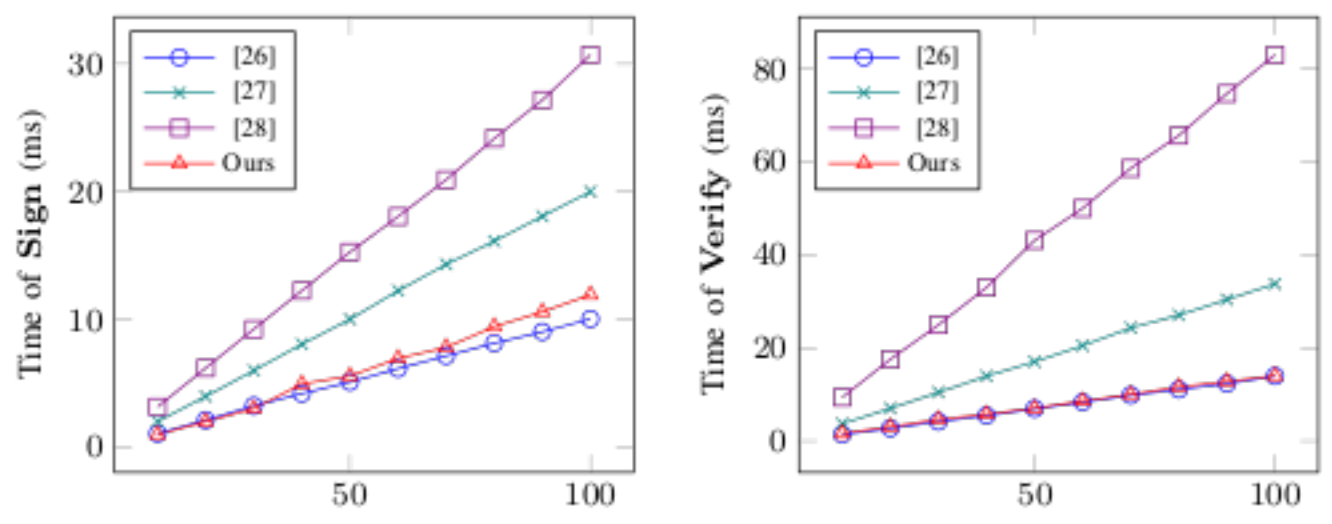


Fig. 6. Efficiency comparison of TRS schemes with increasing ring size.

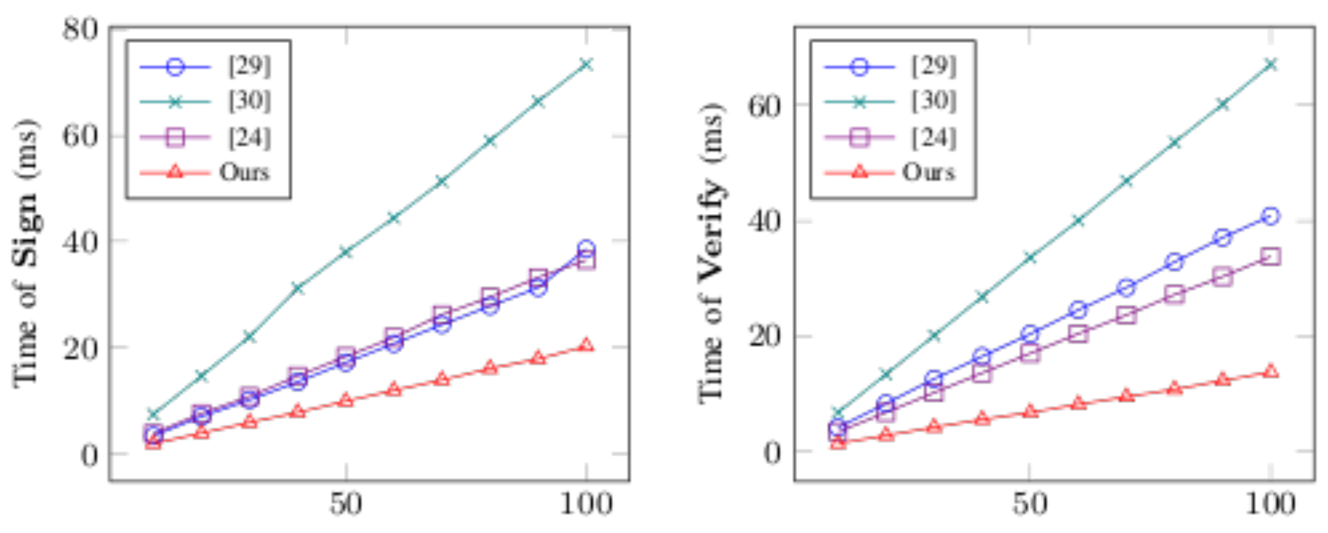


Fig. 7. Efficiency comparison of LTRS schemes with increasing ring size.

TABLE IV
COMPARISON OF COMMUNICATION OVERHEAD UNDER DIFFERENT THRESHOLDS (KB).

| T | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|
| [2] | 15.77 | 28.36 | 53.54 | 103.88 | 204.58 | 405.98 |
| [23] | 8.00 | 15.86 | 31.90 | 63.77 | 127.50 | 254.97 |
| [24] | 9.52 | 15.86 | 28.54 | 53.88 | 104.58 | 205.98 |
| **Ours** | **6.39** | **6.45** | **6.58** | **6.83** | **7.33** | **8.33** |

6. It can be seen that our scheme is faster than the schemes [24], [29], [30] in the signing and verifying time.

We compare the communication overhead of our threshold RingCT protocol with that of other RingCT protocols in Table IV. We mainly consider the main parts of the transaction, including input accounts, output accounts, and the signature. We refer to the set in Monero, which uses Ed25519 elliptic curve public key signature system with 257 bits public key and 256 bits secret key. We set the ring size to 100 and increment the threshold from 2 to 64. As shown in Table IV, when the threshold is set to 2, the communication overhead of our protocol is approximately 67% of that in [24]. As the threshold increases, the advantage of our improvement in communication overhead becomes more significant. For a transaction where $n = 100$ and $t = 64$, the communication overhead of our protocol is about 4% of that in [24].

## IX. Conclusion

In this work, we first proposed a construction of TRS scheme for homomorphic cryptosystems. Based on it, we presented the efficient instantiations of TRS and LTRS under the DL assumption. Compared to previous work, our instantiations have less signature/verification overhead and smaller signature sizes. Based on our proposed LTRS, we proposed a threshold RingCT protocol, which is more efficient than others and allows multiple payers to co-construct an anonymous transaction.

## Acknowledgment

We would like to express our sincere gratitude to the anonymous reviewers for their valuable comments and suggestions, which have significantly contributed to enhancing the security of our scheme.

## References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.
[2] S. Noether and A. Mackenzie, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, 2016.
[3] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 459–474.
[4] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *ESORICS 2014, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, vol. 8713. Springer, 2014, pp. 345–364.
[5] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *ASIACRYPT 2001, Gold Coast, Australia, December 9-13, 2001, Proceedings*, vol. 2248. Springer, 2001, pp. 552–565.
[6] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup, "Anonymous identification in ad hoc groups," in *EUROCRYPT 2004, Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, C. Cachin and J. Camenisch, Eds., vol. 3027. Springer, 2004, pp. 609–626.
[7] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, and C. Petit, "Short accountable ring signatures based on DDH," in *ESORICS 2015, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*, vol. 9326. Springer, 2015, pp. 243–265.
[8] G. Malavolta and D. Schröder, "Efficient ring signatures in the standard model," in *ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, vol. 10625. Springer, 2017, pp. 128–157.
[9] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *EUROCRYPT 2003, Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, vol. 2656. Springer, 2003, pp. 416–432.
[10] S. S. M. Chow, V. K. Wei, J. K. Liu, and T. H. Yuen, "Ring signatures without random oracles," in *ASIACCS 2006, Taipei, Taiwan, March 21-24, 2006*. ACM, 2006, pp. 297–302.
[11] F. Zhang and K. Kim, "Id-based blind signature and ring signature from pairings," in *ASIACRYPT 2002, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, vol. 2501. Springer, 2002, pp. 533–547.
[12] W. Beullens, S. Katsumata, and F. Pintore, "Calamari and falafl: Logarithmic (linkable) ring signatures from isogenies and lattices," in *ASIACRYPT 2020, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, vol. 12492. Springer, 2020, pp. 464–492.
[13] Z. Liu, K. Nguyen, G. Yang, H. Wang, and D. S. Wong, "A lattice-based linkable ring signature supporting stealth addresses," in *ESORICS 2019, Luxembourg, September 23-27, 2019, Proceedings, Part I*, vol. 11735. Springer, 2019, pp. 726–746.
[14] X. Lu, M. H. Au, and Z. Zhang, "Raptor: A practical lattice-based (linkable) ring signature," in *ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, vol. 11464. Springer, 2019, pp. 110–130.
[15] M. F. Esgin, R. Steinfeld, A. Sakzad, J. K. Liu, and D. Liu, "Short lattice-based one-out-of-many proofs and applications to ring signatures," in *ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, vol. 11464. Springer, 2019, pp. 67–88.
[16] E. F. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *CCS 2004, Washington, DC, USA, October 25-29, 2004*. ACM, 2004, pp. 132–145.
[17] S. Patachi and C. Schürmann, "Eos a universal verifiable and coercion resistant voting protocol," in *E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, vol. 10615. Springer, 2017, pp. 210–227.
[18] M. Abe, M. Ohkubo, and K. Suzuki, "1-out-of-n signatures from a variety of keys," in *ASIACRYPT 2002, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, vol. 2501. Springer, 2002, pp. 415–432.
[19] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," in *ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, vol. 3108. Springer, 2004, pp. 325–335.

[20] T. Nishide and K. Sakurai, "Security of offline anonymous electronic cash systems against insider attacks by untrusted authorities revisited," in *INCoS 2011, Fukuoka, Japan, November 30 - Dec. 2, 2011*. IEEE Computer Society, 2011, pp. 656–661.

[21] S. 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 *ESORICS 2017, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, vol. 10493. Springer, 2017, pp. 456–474.

[22] Y. Jia, S. Sun, Y. Zhang, Q. Zhang, N. Ding, Z. Liu, J. K. Liu, and D. Gu, "${\sf PBT}$pbt: A new privacy-preserving payment protocol for blockchain transactions," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 1, pp. 647–662, 2022.

[23] T. H. Yuen, S. 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 *FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, vol. 12059. Springer, 2020, pp. 464–483.

[24] B. Goodell, S. Noether, and A. Blue, "Concise linkable ring signatures and forgery against adversarial keys," *Cryptology ePrint Archive*, 2019.

[25] E. Bresson, J. Stern, and M. Szydlo, "Threshold ring signatures and applications to ad-hoc groups," in *CRYPTO 2002, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, vol. 2442. Springer, 2002, pp. 465–480.

[26] J. K. Liu, V. K. Wei, and D. S. Wong, "A separable threshold ring signature scheme," in *ICISC 2003, Seoul, Korea, November 27-28, 2003, Revised Papers*, vol. 2971. Springer, 2003, pp. 12–26.

[27] S. S. M. Chow, L. C. K. Hui, and S. Yiu, "Identity based threshold ring signature," in *ICISC 2004, Seoul, Korea, December 2-3, 2004, Revised Selected Papers*, vol. 3506. Springer, 2004, pp. 218–232.

[28] T. H. Yuen, J. K. Liu, M. H. Au, W. Susilo, and J. Zhou, "Threshold ring signature without random oracles," in *ASIACCS 2011, Hong Kong, China, March 22-24, 2011*. ACM, 2011, pp. 261–267.

[29] 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 *INDOCRYPT 2004, Chennai, India, December 20-22, 2004, Proceedings*, vol. 3348. Springer, 2004, pp. 384–398.

[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, Malacca, Malaysia, October 13-15, 2010. Proceedings*, vol. 6402. Springer, 2010, pp. 166–183.

[31] 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.*, vol. 56, no. 4, pp. 407–421, 2013.

[32] D. F. Aranha, M. Hall-Andersen, A. Nitulescu, E. Pagnin, and S. Yakoubov, "Count me in! extendability for threshold ring signatures," in *PKC 2022, Virtual Event, March 8-11, 2022, Proceedings, Part II*, vol. 13178. Springer, 2022, pp. 379–406.

[33] A. Haque, S. Krenn, D. Slamanig, and C. Striecks, "Logarithmic-size (linkable) threshold ring signatures in the plain model," in *PKC 2022, Virtual Event, March 8-11, 2022, Proceedings, Part II*, vol. 13178. Springer, 2022, pp. 437–467.

[34] T. H. Yuen, M. F. Esgin, J. K. Liu, M. H. Au, and Z. Ding, "Dualring: Generic construction of ring signatures with efficient instantiations," in *CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, vol. 12825. Springer, 2021, pp. 251–281.

[35] E. Kiltz, D. Masny, and J. Pan, "Optimal security proofs for signatures from identification schemes," in *CRYPTO 2016, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, vol. 9815. Springer, 2016, pp. 33–61.

[36] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *J. Cryptol.*, vol. 1, no. 2, pp. 77–94, 1988.

[37] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *CRYPTO '91, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, vol. 576. Springer, 1991, pp. 433–444.

[38] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *CRYPTO 2006, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, vol. 4117. Springer, 2006, pp. 78–96.

[39] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO '91, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, vol. 576. Springer, 1991, pp. 129–140.

[40] A. Haque and A. Scafuro, "Threshold ring signatures: New definitions and post-quantum security," in *PKC 2020, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, vol. 12111. Springer, 2020, pp. 423–452.

[41] D. Ermilov, M. Panov, and Y. Yanovich, "Automatic bitcoin address clustering," in *ICMLA 2017, Cancun, Mexico, December 18-21, 2017*. IEEE, 2017, pp. 461–466.

[42] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *EUROCRYPT 2016, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, vol. 9666. Springer, 2016, pp. 327–357.

[43] D. Chaum, "Zero-knowledge undeniable signatures," in *EUROCRYPT '90, Aarhus, Denmark, May 21-24, 1990, Proceedings*, vol. 473. Springer, 1990, pp. 458–464.

**Junke Duan** received the M.S. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2021. He is currently pursuing the Ph.D. degree in cyber security with the Beijing University of Posts and Telecommunications. His current research interests include applied cryptography and blockchain technology.

**Shihui Zheng** received the Ph.D. degree from Shandong University, China, in 2006. From 2006 to 2008, she was a Post-Doctoral Researcher in the School of Information Engineering at Beijing University of Posts and Telecommunications (BUPT), China. In 2008, she joined the School of Cyberspace Security & National Engineering Laboratory for Disaster Backup and Recovery at BUPT. Her current research interest is cryptographic scheme design.

**Wei Wang** received the M.S. degree from Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology, Jinan, China, in 2021. He is currently pursuing the Ph.D. degree in cyber security with the Beijing University of Posts and Telecommunications. His current research interests include applied cryptography and blockchain technology.

**Licheng Wang** received the B.S. degree in computer science from Northwest Normal University, China, in 1995, the M.S. degree in mathematics from Nanjing University, China, in 2001, and the Ph.D. degree in cryptography from Shanghai Jiao Tong University, China, in 2007. He is currently a professor with the Beijing Institute of Technology. His current research interests include modern cryptography, network security, and trust management.

**Xiaoya Hu** received the B.S. degree in Information and Computing Sciences from Shandong University of Science and Technology, Qingdao, China, in 2017. She received a M.S. degree Computer Technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2020. She is currently working towards her PhD degree in Cyberspace Security in Beijing University of Posts and Telecommunications, Beijing, China. Her current research fields include security and privacy in blockchain and applied cryptography.

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