

# A Simpler and Modular Construction of Linkable Ring Signature

Wulu Li  
Onething Technologies Co., Ltd.  
Shenzhen, China  
liwulu@onething.net

Yongcan Wang  
Onething Technologies Co., Ltd.  
Shenzhen, China  
wangyongcan@onething.net

Lei Chen  
Onething Technologies Co., Ltd.  
Shenzhen, China

Xin Lai  
Onething Technologies Co., Ltd.  
Shenzhen, China

Xiao Zhang  
Onething Technologies Co., Ltd.  
Shenzhen, China

Jiajun Xin  
Onething Technologies Co., Ltd.  
Shenzhen, China

## ABSTRACT

Linkable ring signature (LRS) plays a major role in the Monero-type cryptocurrencies, as it provides the anonymity of initiator and the prevention of double spending in transactions. In this paper, we propose SLRS: a simpler and modular construction of linkable ring signature scheme, which only use standard ring signature as component, without any additional one-time signatures or zero-knowledge proofs. SLRS is more efficient than existing schemes in both generation and verification. Moreover, we use SLRS to construct an efficient and compact position-preserving linkable multi-ring signature to support application in Monero-type cryptocurrencies. We also give the security proofs, implementation as well as the performance comparisons between SLRS, Ring-CT and Ring-CT 3.0 in size and efficiency.

## KEYWORDS

linkable ring signature, modular construction, high performance, privacy-preserving blockchains

## 1 INTRODUCTION

Blockchain technology was first proposed by Nakamoto[16] in 2008. It is an application system that combines multiple underlying techniques including P2P networks, distributed data storage, network consensus protocols and cryptographic algorithms. In blockchain theory, privacy-preserving techniques have been developed in this decade to provide potential replacements of traditional blockchain-based cryptocurrencies such as Bitcoin[16] and Ethereum[6]. For the application requirements in various privacy-preserving scenarios such as salary, donation, bidding, taxation, a series of privacy-preserving cryptocurrencies have been proposed during these years such as Confidential Transaction[14], Dash[8], Monero[17, 23] and Zerocash[20], etc. As a representative, Monero has realized anonymous and confidential transactions, which can protect the privacy of identities for both initiators and recipients in transactions, as well as the transaction amount. In Monero system, linkable ring signatures[10, 17, 23] (LRS) are used to hide the identity of initiator, range proofs (Borromean[17], Bulletproofs[5]) are used to hide the transaction amount.

After the introduction of Monero (also known as Ring-CT), some follow-up works on new linkable ring signatures have been proposed, including Ring-CT 2.0[21], Ring-CT 3.0[25], which have more

compact size and better efficiency than Ring-CT when the ring size is large. Nevertheless, when the ring size is small ( $\leq 16$ ), both Ring-CT 2.0 and Ring-CT 3.0 are less competitive than Ring-CT (in Monero system, current ring size is  $n = 11$ ). So it is necessary to construct a more efficient LRS to provide more compact signature size and less computations than Monero in practical parameters ( $n < 1000$ ). Moreover, we also need to rethink the technique to realize linkability in linkable ring signatures, and give a simpler and more efficient method to realize linkability.

## 1.1 Our Contributions

*1.1.1 Simpler and Modular Construction of LRS.* In this paper, we introduce SLRS: a simpler and modular construction of linkable ring signature scheme, which only uses ring signature as component, with a very simple embedding method of key-image to achieve linkability. Here we give a brief introduction of SLRS:

The public parameters are  $(\mathbb{G}, q, g, h)$ , where  $g$  is the generator of  $\mathbb{G}$  and  $h \in \mathbb{G}$  is a random element with its discrete logarithm unknown to anyone. Every user generates his  $(PK, SK)$  by usage of the public parameters. When signing, the user chooses a set of public keys  $L_{PK}$ , then publishes a key-image  $I$  and makes a randomized combination between  $L_{PK}$  and  $I$  to get a new  $L_{RPK}$  for ring signature. Then he runs ring signature algorithm (with  $L_{RPK}$ ) to finish the SLRS signature. The verifier computes  $L_{RPK}$  and checks the validity of ring signature. The verifier also checks whether  $I$  is already in the key-image set in the linkability check.

The main advantages of SLRS are summarized as follows:

1. Compared to Ring-CT and Ring-CT 3.0, the efficiency of SLRS (generation and verification) is greatly improved for all ring sizes, meanwhile, SLRS (with AOS or AOS' [1]) is more compact than Ring-CT 3.0 (linkable version) for ring size  $n \leq 24$ , which makes SLRS a potential replacement in Monero.
2. The construction of SLRS is modular, we can choose any suited elliptic-curve-based ring signature scheme as the component, which means that we can choose the the best suited (fastest or shortest) elliptic-curve-based ring signature to get the linkable ring signature directly.
3. The security of SLRS relies on the hardness of discrete logarithm, DDH assumption and the security of corresponding ring signature, without any additional assumptions.

*1.1.2 Modularity.* In the construction of SLRS, we use ring signature as component with the following conditions need to be met:

1. The ring signature needs to be based on sigma protocol (in the random oracle model), in which the signature can be simulated by programming the random oracle;
2. The ring signature component needs to be based on elliptic curve, with the form of public-private key pair being  $(g^x, x)$ ;
3. The ring signature (also can be seen as an 1-out-of- $n$  proof) needs to have special soundness (introduced in Appendix A).

Moreover, the **modularity** mean that the procedures for key-image embedding and signing are separate and independent, this is the major difference between SLRS and existing schemes. In Ring-CT (MLSAG, CLSAG) or Ring-CT 3.0, the key-image embedding happens in the signing algorithm, whereas in SLRS, the key-image embedding happens in the generation of ring. Actually, in SLRS, the key-image  $I = h^x$ ; in MLSAG and CLSAG,  $I = H_p(g^x)^x$ .

**1.1.3 Position-preserving Multi-ring SLRS.** In Monero-type transactions, we usually need the position-preserving multi-ring LRS in which the position of signing key in each ring remains the same. We give two constructions of position-preserving multi-ring SLRS (MSLRS, MSLRS') by usage of AOS and AOS' separately. Our construction is compact, and is more efficient than existing schemes, such as MLSAG [17], CLSAG[10]. Moreover, in the constructions of MSLRS and MSLRS', the basis element (generator) in each ring is different, which is the major difference from existing schemes.

We take  $m$ -ring signature as example, when  $L_{PK_i} = \{g_i^{x_i,1}, \dots, g_i^{x_i,n}\}$  for  $i = 1, \dots, m$ , MSLRS' can provide linkability in any ring with a compact signature size  $(1, m + n)$ , where  $(\cdot, \cdot)$  refers to number of elements in  $(\mathbb{G}, \mathbb{Z}_q^*)$ . As comparisons, MLSAG has the signature size of  $(1, mn + 1)$  and CLSAG has  $(m, n + 1)$ . The efficiency of MSLRS (MSLRS') is also greatly improved due to the new key-image embedding method and the modular construction.

**1.1.4 High Performance.** A brief description of efficiency performances for generation and verification of SLRS (SLRS') and MSLRS (MSLRS') are shown in Table1, note that  $n$  denotes the ring size (in Monero  $n = 11$ ),  $m$  denotes the number of rings and the size  $(\cdot, \cdot)$  refers to number of elements in  $(\mathbb{G}, \mathbb{Z}_q^*)$ . The detailed comparisons between our works and existing schemes are in section 5.

**Table 1: Overall Performance of Our Schemes**

| Scheme       | $n$  | $m$ | Generation | Verification | Size          |
|--------------|------|-----|------------|--------------|---------------|
| SLRS(AOS)    | 11   | 1   | 1.10ms     | 1.09ms       | $(1, n + 1)$  |
|              | 1024 | 1   | 98.04ms    | 97.19ms      |               |
| SLRS'(AOS')  | 11   | 1   | 0.65ms     | 0.66ms       | $(1, n + 1)$  |
|              | 1024 | 1   | 52.80ms    | 51.70ms      |               |
| MSLRS(AOS)   | 11   | 2   | 1.68ms     | 1.71ms       | $(1, mn + 1)$ |
|              | 1024 | 2   | 154.79ms   | 153.50ms     |               |
| MSLRS'(AOS') | 11   | 2   | 1.09ms     | 1.11ms       | $(1, m + n)$  |
|              | 1024 | 2   | 94.61ms    | 93.58ms      |               |

## 1.2 Related Works

**1.2.1 Ring Signatures.** Ring signature is a special type of signature scheme, in which signer can sign on behalf of a group chosen by himself, while maintaining anonymous within the group. In ring signature, signer selects a list of public keys  $L_{PK} = \{PK_1, \dots, PK_n\}$

as the ring elements, and uses his secret key  $SK_\pi$  to sign, where  $\pi \in \{1, \dots, n\}$ . Verifier cannot determine the signer's identity. Ring signature was first proposed by Rivest, Shamir and Tauman[19] in 2001, they constructed ring signature schemes based on RSA trapdoor permutation and Robin trapdoor function, in the random oracle model. In 2002, Abe *et al.*[1] proposed AOS ring signature, which simultaneously supported discrete logarithm (via Sigma protocol) and RSA trapdoor functions (via hash and sign), also in the random oracle model. In 2006, Bender *et al.*[4] introduced the first ring signature scheme in the standard model, by making use of pairing technique. In 2015, Maxwell *et al.*[15] gave Borromean signature scheme, which is a multi-ring signature based on AOS with signature size reduced from  $mn + m$  to  $mn + 1$ , where  $n$  denotes the ring size and  $m$  denotes the number of rings. There are also constructions with nonlinear signature size, including: accumulator-based ring signature [7] with constant signature size, standard model ring signature scheme with  $O(\sqrt{n})$  signature size, ring signature [11] with  $O(\log n)$  signature size.

**1.2.2 Linkable Ring Signatures.** Linkable ring signature is a variant of ring signature, in which the identity of the signer in a ring signature remains anonymous, but two ring signatures can be linked if they are signed by the same signer. Linkable ring signatures are suitable in many different practical applications such as privacy-preserving cryptocurrency (Monero), e-Voting, cloud data storage security, etc. In Monero, linkability is used to check whether double spending happens. The first linkable ring signature scheme is proposed by Liu *et al.*[13] in 2004, under discrete logarithm assumption, in the random oracle model. Later, Tsang *et al.*[22] and Au *et al.*[2] proposed accumulator-based linkable ring signatures with constant signature size. In 2013, Yuen *et al.*[24] gave a standard model linkable ring signature scheme with  $O(\sqrt{n})$  signature size, from pairing technique. In 2014, Liu *et al.*[12] gave a linkable ring signature with unconditional anonymity, he also gave the formalized security model of linkable ring signature, which we will follow in this paper. In 2015, Back *et al.*[3] proposed an efficient linkable ring signature scheme LSAG, which shortens the signature size of [13]. In 2016, based on work of Fujisaki *et al.*[9], Noether *et al.*[17] gave a linkable multi-ring signature scheme MLSAG, which supports transactions with multiple inputs, and was used by Monero. In 2017, Sun *et al.*[21] proposed Ring-CT 2.0, which is an accumulator-based linkable ring signature with asymptotic smaller signature size than Ring-CT 1.0, but is less competitive when  $n$  is small. In 2019, Yuen *et al.*[25] proposed Ring-CT 3.0, a modified Bulletproof-based 1-out-of- $n$  proof protocol with logarithmic size, which has functionality of (linkable) ring signature. In 2019, Goodell *et al.*[10] proposed CLSAG: a modified multi-ring LRS with better efficiency and compactness than MLSAG, we give the detailed description of CLSAG in the appendix A.

## 1.3 Organization

In section 2 we give some preliminaries; in section 3 we give the construction and security proofs of SLRS; in section 4 we give the construction of MSLRS and MSLRS' for multi-ring application; in section 5 we introduce the implementations, performances and comparisons; in section 6 we give the conclusion.

## 2 PRELIMINARIES

In this paper, we use multiplicative cyclic group  $\mathbb{G}$  to represent elliptic group with prime order  $|\mathbb{G}| = q$ .  $g, h \in \mathbb{G}$  are the generators of  $\mathbb{G}$ , group multiplication is  $g_1 \cdot g_2 = g_1 g_2$  and exponentiation is  $g^a$ .  $\mathbb{Z}_q^* = \mathbb{Z}_q \setminus \{0\}$  is the set of nonzero elements in  $\mathbb{Z}_q$ . We use  $H(\cdot)$  to represent hash function, use  $H_p(\cdot)$  to represent Hash-to-Point, and  $\text{negl}(\cdot)$  to represent negligible functions. For verifiers, 1 is for *accept* and 0 is for *reject*. For adversaries, PPT means probabilistic polynomial time. The DDH assumption means any PPT adversary cannot distinguish  $(g^a, h^a)$  from  $(g^a, h^r)$ , where  $r$  is uniformly sampled from  $\mathbb{Z}_q^*$ . The hardness of discrete logarithm problem means that any PPT adversary cannot compute  $x$  from  $g^x$ . Oracle  $\mathcal{RO}$  refers to the random oracle. The security parameter of this paper is  $\lambda = \lceil \log q \rceil$ , where  $q = |\mathbb{G}|$ .

### 2.1 Ring Signatures

Ring signature scheme usually consists of four algorithms: Setup, KeyGen, Rsign, and Verify.

- $\text{Par} \leftarrow \text{Setup}(\lambda)$  is a probabilistic polynomial time (PPT) algorithm which, on input a security parameter  $\lambda$ , outputs the set of security parameters  $\text{Par}$  which includes  $\lambda$ .
- $(\text{PK}_i, \text{SK}_i) \leftarrow \text{KeyGen}(\text{Par})$  is a PPT algorithm which, on input the security parameters  $\text{Par}$ , outputs a key pair  $(\text{PK}_i, \text{SK}_i)$ .
- $\sigma \leftarrow \text{Rsign}(\text{SK}_\pi, \mu, L_{\text{PK}})$  is a ring signature algorithm which, on input user's secret key  $\text{SK}_\pi$ , a list of users' public keys  $L_{\text{PK}} = \{\text{PK}_1, \dots, \text{PK}_n\}$ , where  $\text{PK}_\pi \in L_{\text{PK}}, \pi \in \{1, \dots, n\}$ , and a message  $\mu$ , outputs a ring signature  $\sigma$ .
- $1/0 \leftarrow \text{Verify}(\mu, \sigma, L_{\text{PK}})$  is a verification algorithm which, on input a message  $\mu$ , a list of users' public keys  $L_{\text{PK}}$  and a ring signature  $\sigma$ , outputs 1 or 0.

The security definition of ring signature contains *unforgeability* and *anonymity*. Before giving their definitions, we consider the following oracles which together model the ability of the adversaries in breaking the security of the schemes, in fact, the adversaries are allowed to query the four oracles below:

- $c \leftarrow \mathcal{RO}(a)$ . *Random oracle*, on input  $a$ , random oracle returns a random value.
- $\text{PK}_i \leftarrow \mathcal{JO}(\perp)$ . *Joining oracle*, on request, adds a new user to the system. It returns the public key  $\text{PK}_i$  of the new user.
- $\text{SK}_i \leftarrow \mathcal{CO}(\text{PK}_i)$ . *Corruption oracle*, on input a public key  $\text{PK}_i$  that is a query output of  $\mathcal{JO}$ , returns the corresponding private key  $\text{SK}_i$ .
- $\sigma \leftarrow \mathcal{SO}(\text{PK}_\pi, \mu, L_{\text{PK}})$ . *Signing oracle*, on input a list of users' public keys  $L_{\text{PK}}$ , the public key of the signer  $\text{PK}_\pi$ , and a message  $\mu$ , returns a valid ring signature  $\sigma$ .

**DEFINITION 1 (UNFORGEABILITY).** *Unforgeability for ring signature schemes is defined in the following game between the simulator  $\mathcal{S}$  and the adversary  $\mathcal{A}$ , simulator  $\mathcal{S}$  runs Setup to provide public parameters for  $\mathcal{A}$ ,  $\mathcal{A}$  is given access to oracles  $\mathcal{RO}, \mathcal{JO}, \mathcal{CO}$  and  $\mathcal{SO}$ .  $\mathcal{A}$  wins the game if he successfully forges a ring signature  $(\sigma^*, L_{\text{PK}}^*, \mu^*)$  satisfying the following:*

1.  $\text{Verify}(\sigma^*, L_{\text{PK}}^*, \mu^*) = 1$ .
2. Every  $\text{PK}_i \in L_{\text{PK}}^*$  is returned by  $\mathcal{A}$  to  $\mathcal{JO}$ .
3. No  $\text{PK}_i \in L_{\text{PK}}^*$  is queried by  $\mathcal{A}$  to  $\mathcal{CO}$ .
4.  $(\mu^*, L_{\text{PK}}^*)$  is not queried by  $\mathcal{A}$  to  $\mathcal{SO}$ .

The advantage of  $\mathcal{A}$  in the forging attack is  $\text{Adv}_{\mathcal{A}}^{\text{forge}} = \Pr[\mathcal{A} \text{ wins}]$ .

A ring signature scheme is *unforgeable* if for any PPT adversary  $\mathcal{A}$ ,  $\text{Adv}_{\mathcal{A}}^{\text{forge}} = \text{negl}(\lambda)$ .

**DEFINITION 2 (ANONYMITY).** *Anonymity for ring signature schemes is defined in the following game between the simulator  $\mathcal{S}$  and the adversary  $\mathcal{A}$ , simulator  $\mathcal{S}$  runs Setup to provide public parameters for  $\mathcal{A}$ ,  $\mathcal{A}$  is given access to oracles  $\mathcal{RO}, \mathcal{JO}$  and  $\mathcal{CO}$ .  $\mathcal{A}$  gives a set of public keys  $L_{\text{PK}} = \{\text{PK}_1, \dots, \text{PK}_n\}$ ,  $\mathcal{S}$  randomly picks  $\pi \in \{1, \dots, n\}$ , computes  $\sigma = \text{Rsign}(\text{SK}_\pi, \mu, L_{\text{PK}})$  and sends  $\sigma$  to  $\mathcal{A}$ , where  $\text{SK}_\pi$  is the corresponding private key of  $\text{PK}_\pi$ , then  $\mathcal{A}$  outputs a guess  $\pi^* \in \{1, \dots, n\}$ .  $\mathcal{A}$  wins the game if he successfully guesses  $\pi^* = \pi$ . The advantage of  $\mathcal{A}$  in the anonymity attack is  $\text{Adv}_{\mathcal{A}}^{\text{anon}} = |\Pr[\pi^* = \pi] - 1/n|$ .*

A ring signature scheme is *anonymous* if for any PPT adversary  $\mathcal{A}$ ,  $\text{Adv}_{\mathcal{A}}^{\text{anon}} = \text{negl}(\lambda)$ .

### 2.2 AOS and AOS'

In the construction of SLRS, we can use any ring signature which satisfies the conditions in section 1.1.2. We can choose AOS (AOS') scheme or other sigma-protocol-based ring signature as component in SLRS. The choice is not restricted, we can choose the most suited ones for different ring sizes in different applications.

AOS and AOS' ring signatures are proposed by Abe *et al.*[1] in 2002, the size and running time of each scheme is linear with the ring size. In this paper, we make use of AOS and AOS' in the construction of SLRS (SLRS') and MSLRS (MSLRS'). The detailed descriptions of AOS and AOS' are in the appendix A.

### 2.3 Linkable Ring Signatures

Compared to ring signature, linkable ring signature has the function of linkability, that is, when two ring signatures are signed by the same signer, they are linked by the algorithm Link:

- $\text{linked/unlinked} \leftarrow \text{Link}((\sigma, \mu, L_{\text{PK}}), (\sigma', \mu', L'_{\text{PK}}))$ : verifier checks the two ring signatures are linked or not.

The security definition of linkable ring signature contains *unforgeability*, *anonymity*, *linkability* and *nonslanderability*. The *unforgeability* is the same as Definition 1, and the *anonymity* is slightly different from Definition 2 with additional requirements that all public keys in  $L_{\text{PK}}$  are returned by  $\mathcal{A}$  to  $\mathcal{JO}$  and all public keys in  $L_{\text{PK}}$  are not queried by  $\mathcal{A}$  to  $\mathcal{CO}$  (if the adversary corrupts some of the public keys, then he can break the anonymity of the scheme by computing the corresponding key-images in advance). In the rest of this paper, we use this modified definition of *anonymity* in SLRS and its security proof.

Here we give the definition of *linkability* and *nonslanderability*:

**DEFINITION 3 (LINKABILITY).** *Linkability for linkable ring signature schemes is defined in the following game between the simulator  $\mathcal{S}$  and the adversary  $\mathcal{A}$ , simulator  $\mathcal{S}$  runs Setup to provide public parameters for  $\mathcal{A}$ ,  $\mathcal{A}$  is given access to oracles  $\mathcal{RO}, \mathcal{JO}, \mathcal{CO}$  and  $\mathcal{SO}$ .  $\mathcal{A}$  wins the game if he successfully forges  $k$  ring signatures  $(\sigma_i, L_{\text{PK}}^i, \mu_i), i = 1, \dots, k$ , satisfying the following:*

1. All  $\sigma_i$ s are not returned by  $\mathcal{A}$  to  $\mathcal{SO}$ .
2. All  $L_{\text{PK}}^i$  are returned by  $\mathcal{A}$  to  $\mathcal{JO}$ .
3.  $\text{Verify}(\sigma_i, L_{\text{PK}}^i, \mu_i) = 1, i = 1, \dots, k$ .
4.  $\mathcal{A}$  queried  $\mathcal{CO}$  less than  $k$  times.

5.  $\text{Link}((\sigma_i, L_{PK}^i, \mu_i), (\sigma_j, L_{PK}^j, \mu_j)) = \text{unlinked}$  for  $i \neq j$  and  $i, j \in \{1, \dots, k\}$ .

The advantage of  $\mathcal{A}$  in the link attack is  $\text{Adv}_{\mathcal{A}}^{\text{link}} = \Pr[\mathcal{A} \text{ wins}]$ .

A linkable ring signature scheme is linkable if for any PPT adversary  $\mathcal{A}$ ,  $\text{Adv}_{\mathcal{A}}^{\text{link}} = \text{negl}(\lambda)$ .

The nonslanderability of a linkable ring signature scheme is that  $\mathcal{A}$  cannot slander other honest users by generating a signature linked with signatures from honest users:

**DEFINITION 4 (NONSLANDERABILITY).** Nonslanderability for linkable ring signature schemes is defined in the following game between the simulator  $\mathcal{S}$  and the adversary  $\mathcal{A}$ , simulator  $\mathcal{S}$  runs Setup to provide public parameters for  $\mathcal{A}$ ,  $\mathcal{A}$  is given access to oracles  $\mathcal{RO}$ ,  $\mathcal{JO}$ ,  $\mathcal{CO}$  and  $\mathcal{SO}$ .  $\mathcal{A}$  gives a list of public keys  $L_{PK}$ , a message  $\mu$  and a public key  $PK_{\pi} \in L_{PK}$  to  $\mathcal{S}$ ,  $\mathcal{S}$  returns the corresponding signature  $\sigma \leftarrow \text{Rsign}(SK_{\pi}, L_{PK}, \mu)$  to  $\mathcal{A}$ .  $\mathcal{A}$  wins the game if he successfully outputs a ring signature  $(\sigma^*, L_{PK}^*, \mu^*)$ , satisfying the following:

1.  $\text{Verify}(\sigma^*, L_{PK}^*, \mu^*) = 1$ .
2.  $PK_{\pi}$  is not queried by  $\mathcal{A}$  to  $\mathcal{CO}$ .
3.  $PK_{\pi}$  is not queried by  $\mathcal{A}$  as input to  $\mathcal{SO}$ .
4.  $\text{Link}((\sigma, L_{PK}, \mu), (\sigma^*, L_{PK}^*, \mu^*)) = \text{linked}$ .

The advantage of  $\mathcal{A}$  in slandering attack is  $\text{Adv}_{\mathcal{A}}^{\text{slander}} = \Pr[\mathcal{A} \text{ wins}]$ .

A linkable ring signature scheme is nonslanderable if for any PPT adversary  $\mathcal{A}$ ,  $\text{Adv}_{\mathcal{A}}^{\text{slander}} = \text{negl}(\lambda)$ .

According to [12], linkability and nonslanderability imply unforgeability:

**LEMMA 5 ([12]).** If a linkable ring signature scheme is linkable and nonslanderable, then it is unforgeable.

## 2.4 Linkable Multi-ring Signature in Monero

In Monero system, every UTXO (unspent transaction output[23]) has its public-private key pair ( $PK = g^s, SK = s$ ) and the value commitment  $c = g^x h^a$ , where  $c$  is Pedersen commitment[18],  $a$  is the hidden value and  $x$  is the blinding element. In a transaction, the initiator Alice chooses  $n - 1$  hiding UTXOs:  $\{(PK_i, c_i = g^{x_i} h^{a_i})\}_{i=1, \dots, n-1}$ , along with her input UTXO ( $PK_A = g^s, c_A = g^{x_A} h^{a_A}$ ), to generate a set of public keys  $L_{PK} = \{PK_A, PK_1, \dots, PK_{n-1}\}$  (randomized order), Alice also generates the output UTXO ( $PK_B, c_B = g^{x_B} h^{a_B}$ ), where the input value equals the output value  $a_A = a_B$ . Then Alice computes another ring of commitments (same order as in  $L_{PK}$ )

$$\begin{aligned} L_v &= \{c_A c_B^{-1}, c_1 c_B^{-1}, \dots, c_{n-1} c_B^{-1}\} \\ &= \{g^{x_A - x_B}, g^{x_1 - x_B} h^{a_1 - a_B}, \dots, g^{x_{n-1} - x_B} h^{a_{n-1} - a_B}\}. \end{aligned}$$

Alice uses linkable 2-ring signature to sign the transaction by  $L_{PK}$  and  $L_v$ , with the same position of signing key in each ring (to ensure that the public key and commitment are from the same UTXO), we call it the position-preserving linkable multi-ring signature. In this paper, we use SLRS to construct an efficient and compact MSLRS (MSLRS') for application in Monero-type cryptocurrency.

## 3 SIMPLER LINKABLE RING SIGNATURE

In this section we give the construction and security proofs of SLRS, in section 3.1 we introduce the modular construction of SLRS;

in section 3.2 we give the proof of correctness and proofs of security, including anonymity, unforgeability, linkability and nonslanderability in the random oracle model.

### 3.1 Construction

In our construction of SLRS, we use ring signature (AOS for SLRS, AOS' for SLRS') as the ring signature component. Actually, we assume the ring signature component satisfies the conditions in section 1.1.2, which makes SLRS secure in the random oracle model. We give the introduction of SLRS in the following:

$\text{Par} \leftarrow \text{SLRS.Setup}(\lambda)$ :

1. System chooses an elliptic curve  $\mathbb{G}$  with prime order  $q$  and a generator  $g \in \mathbb{G}$ , system samples another generator  $h \in \mathbb{G}$  whose discrete logarithm is unknown to anyone, system outputs  $(\mathbb{G}, q, g, h)$  as the public parameters.

$(PK, SK) \leftarrow \text{SLRS.KeyGen}(\text{Par})$ :

1. According to the public parameters  $(\mathbb{G}, q, g, h)$ , user Alice samples  $x \in \mathbb{Z}_q^*$  as her secret key, then computes  $PK = g^x$ ;
2. Alice outputs  $PK = g^x$ , and retains  $SK = x$ .

$\sigma \leftarrow \text{SLRS.Sign}(SK_{\pi}, \mu, L_{PK})$ :

1. For a message  $\mu$ , Alice chooses another  $n - 1$  users, together with her own public key, to generate a list of public keys  $L_{PK} = \{PK_1, \dots, PK_n\}$ , where Alice's  $PK = PK_{\pi} \in L_{PK}$ ,  $\pi \in \{1, \dots, n\}$ ;
2. Alice outputs the key-image  $I = h^{x_{\pi}}$ , then computes  $e = H(L_{PK}, I, \mu)$ ;
3. Alice computes the public key set for ring signature  $L_{RPK} = \{PK_1 \cdot I^e, \dots, PK_n \cdot I^e\} = \{g^{x_1} h^{e x_{\pi}}, \dots, g^{x_n} h^{e x_{\pi}}\}$ ;
4. Alice runs the ring signature and get  $\tau \leftarrow \text{Rsign}(SK, \mu, L_{RPK}, I)$  by usage of  $L_{RPK}$  and  $SK = x_{\pi}$ , outputs  $\tau$  (use generator  $gh^{e^*}$ );
5. Alice outputs  $\sigma = (\tau, \mu, L_{PK}, I)$  as the SLRS outputs.

$1/0 \leftarrow \text{SLRS.Verify}(\tau, \mu, L_{PK}, I)$ :

1. Verifier computes  $e^* = H(L_{PK}, I, \mu)$ ;
2. Verifier computes  $L_{RPK}^* = \{PK_1 \cdot I^{e^*}, \dots, PK_n \cdot I^{e^*}\}$ ;
3. Verifier checks the validity of ring signature  $\tau$  with ring  $L_{RPK}^*$  (use generator  $gh^{e^*}$ );
4. If all passed then outputs 1, otherwise outputs 0.

$\text{linked/unlinked} \leftarrow \text{SLRS.Link}(\sigma, \sigma')$ :

1. For two valid SLRS signatures  $\sigma = (\tau, \mu, L_{PK}, I)$  and  $\sigma' = (\tau', \mu', L_{PK}', I')$ , if  $I = I'$  then verifier outputs *linked*, otherwise outputs *unlinked*.

### Algorithm 1: SLRS

## 3.2 Correctness and Security

### 3.2.1 Correctness.

**THEOREM 6 (CORRECTNESS OF SLRS).** For an honest user Alice in SLRS, she can complete the linkable ring signature successfully, and the behavior of double signing (double spending) will be detected while the identity of Alice maintaining anonymous.

**PROOF.** In SLRS, for Alice's public key  $PK = PK_{\pi} = g^{x_{\pi}}$ , she can compute  $I = h^{x_{\pi}}$  and  $e = H(L_{PK}, I, \mu)$ , then she can compute  $L_{RPK} = \{g^{x_1} h^{e x_{\pi}}, \dots, g^{x_n} h^{e x_{\pi}}\}$ . Since  $g^{x_{\pi}} h^{e x_{\pi}} = (gh^e)^{x_{\pi}}$ , then Alice can use her secret key  $SK_{\pi} = x_{\pi}$  to generate the ring signature  $\tau$  using  $gh^e$  as the generator.

When double signing occurs, we know from the linkability of SLRS that Alice must have used the key-image  $I = h^{x\pi}$  for twice (proved in *Theorem 10*), then the verifier can detect that double signing occurs and outputs *linked*, at the same time, anyone cannot learn any information about the identity of signer by the anonymity of SLRS (proved in *Theorem 7*).  $\square$

### 3.2.2 Proof of Anonymity.

**THEOREM 7 (ANONYMITY).** *SLRS is anonymous for any PPT adversary  $\mathcal{A}$ , assuming the ring signature component satisfies the conditions in section 1.1.2.*

**PROOF.** Assume  $\mathcal{A}$  is playing the game with  $\mathcal{S}$  in Definition 2,  $\mathcal{A}$  generates a message  $\mu$  and a list of public keys  $L_{PK} = \{PK_1, \dots, PK_n\}$ , where  $PK_i = g^{x_i}$  for  $i = 1, \dots, n$ , and all  $PK_i$ s are returned by  $\mathcal{CO}$ , and  $\mathcal{S}$  knows all  $SK_i = x_i$ .

We consider the following games between  $\mathcal{S}$  and  $\mathcal{A}$ :

- **Game 0.**  $\mathcal{S}$  samples  $\pi \in \{1, \dots, n\}$  uniformly at random, publishes  $I = h^{x\pi}$ , computes  $e = H(L_{PK}, I, \mu)$  and  $L_{RPK} = \{g^{x_1} h^{e x_1}, \dots, g^{x_n} h^{e x_n}\}$ , then generates the ring signature  $\tau = \text{Rsign}(SK, \mu, L_{RPK}, I)$ , and outputs  $\sigma = (\tau, \mu, L_{PK}, I)$  to  $\mathcal{A}$ . When  $\mathcal{A}$  receives  $\sigma$ , he gives a guess  $\pi^* \in \{1, \dots, n\}$ .
- **Game 1.**  $\mathcal{S}$  samples  $\pi \in \{1, \dots, n\}$  and  $r \in \mathbb{Z}_q^*$  uniformly at random, publishes  $I = h^r$ , computes  $e = H(L_{PK}, I, \mu)$  and  $L_{RPK} = \{g^{x_1} h^{e r}, \dots, g^{x_n} h^{e r}\}$ , then generates the ring signature  $\tau = \text{Rsign}(\mu, L_{RPK}, I)$  by programming the random oracle, outputs  $\sigma = (\tau, \mu, L_{PK}, I)$  to  $\mathcal{A}$ . When  $\mathcal{A}$  receives  $\sigma$ , he gives a guess  $\pi^* \in \{1, \dots, n\}$ .

In the two games above, Game 0 is the real game between  $\mathcal{S}$  and  $\mathcal{A}$  in SLRS, and Game 1 is the simulated game in the random oracle model. In Game 1,  $r$  is uniformly sampled by  $\mathcal{S}$ , which is statistical independent from the  $L_{PK}$ , then  $\Pr_{\mathcal{A}}[\pi^* = \pi] = 1/n$ .

Then we only need to prove that Game 0 and Game 1 are computational indistinguishable. In fact, the differences between the two games are generations of  $I$  and  $L_{RPK}$ . According to DDH assumption,  $(g, h, g^{x\pi}, h^{x\pi})$  and  $(g, h, g^{x\pi}, h^r)$  are computational indistinguishable, then  $\mathcal{A}$  cannot distinguish  $h^{x\pi}$  (in Game 0) from  $h^r$  (in Game 1). Then we know  $\mathcal{A}$  cannot distinguish  $\{g^{x_1} h^{e x_1}, \dots, g^{x_n} h^{e x_n}\}$  from  $\{g^{x_1} h^{e r}, \dots, g^{x_n} h^{e r}\}$ , then we know Game 0 and Game 1 are computational indistinguishable, then we finish the anonymity proof of SLRS.  $\square$

### 3.2.3 Proof of Linkability.

**LEMMA 8.** *For any PPT adversary  $\mathcal{A}$ , the probability of  $\mathcal{A}$  to generate  $s, t \in \mathbb{Z}_q^*$  satisfying  $g^s h^t = 1$  (get a nontrivial relationship between  $g$  and  $h$ ) is negligible, under the hardness assumption of discrete logarithm.*

**LEMMA 9.** *For any sigma-protocol-based 1-out-of- $n$  proof (ring signature) with special soundness, if any PPT adversary  $\mathcal{A}$  can generate a valid proof (ring signature) with  $L_{PK}$ , then  $\mathcal{A}$  can extract a valid witness (one secret key from  $L_{PK}$ ) in the random oracle model with nonnegligible advantage. This implies the unforgeability of the corresponding ring signature.*

**PROOF.** It can be easily derived from the special soundness of the sigma-protocol-based 1-out-of- $n$  proof by the rewinding technique (also known as forking lemma).  $\square$

**THEOREM 10 (LINKABILITY).** *SLRS is linkable for any PPT adversary  $\mathcal{A}$ , assuming the ring signature component satisfies the conditions in section 1.1.2.*

**PROOF.** For any PPT adversary  $\mathcal{A}$ , when  $\mathcal{A}$  finished the link game with  $\mathcal{S}$  in Definition 3, we assume that  $\mathcal{A}$  wins the link game with nonnegligible advantage  $\delta$ , that is,  $\mathcal{A}$  returned  $k$  valid SLRS signatures  $\sigma_i = (\tau_i, \mu_i, L_{PK}^i, I_i)$ ,  $i = 1, \dots, k$  ( $\tau_i$ s are the ring signatures), satisfying the following requirements:

1. All  $\sigma_i$ ,  $i = 1, \dots, k$  are not returned by  $\mathcal{SO}$ .
2. All public keys from  $L_{PK}^i$ ,  $i = 1, \dots, k$  are returned by  $\mathcal{CO}$ .
3.  $\text{SLRS.Verify}(\tau_i, L_{PK}^i, \mu_i, I_i) = 1$  for  $i = 1, \dots, k$ .
4.  $\mathcal{A}$  queried  $\mathcal{CO}$  less than  $k$  times.
5.  $\text{SLRS.Link}((\tau_i, L_{PK}^i, \mu_i, I_i), (\tau_j, L_{PK}^j, \mu_j, I_j)) = \text{unlinked}$  for  $i \neq j \in \{1, \dots, k\}$ .

We first prove a statement that, for a list of users' public keys  $L_{PK} = \{PK_1, \dots, PK_n\}$  returned by  $\mathcal{CO}$  with  $PK_i = g^{x_i}$ , any PPT adversary  $\mathcal{A}$  generates a valid SLRS signature  $\sigma \leftarrow \mathcal{SO}$  if and only if he queries the  $\mathcal{CO}$  at least once, except for negligible probability  $\epsilon_0 = \text{negl}(\lambda)$ .

- $\Rightarrow$ . If  $\mathcal{A}$  gets  $SK = x_i$  from  $\mathcal{CO}$ , and then  $\mathcal{A}$  can run the SLRS signature scheme to generate a valid signature  $\sigma = (\tau, \mu, L_{PK}, I)$ .
- $\Leftarrow$ . Assume  $\mathcal{A}$  did not query the  $\mathcal{CO}$  and  $\mathcal{SO}$  for  $L_{PK} = \{PK_1, \dots, PK_n\}$  and finished the SLRS signature over  $L_{PK} = \{PK_1, \dots, PK_n\}$  with nonnegligible probability  $\delta_1$ . We first prove that  $\mathcal{A}$  does not know any of the secret keys in  $L_{PK}$ . Actually, under the hardness of discrete logarithm,  $\mathcal{A}$  cannot compute  $x_i$  from  $PK_i = g^{x_i}$ ,  $i = 1, \dots, n$ , then the probability of  $\mathcal{A}$  obtaining any of  $x_i$  is  $\epsilon_1 = \text{negl}(\lambda)$ . Next, according to the assumption that  $\mathcal{A}$  generates a valid signature  $\sigma = (\tau, \mu, L_{PK}, I)$ , then he must have finished the ring signature  $\tau$  (with generator  $gh^e$ ), where  $e = H(L_{PK}, I, \mu)$ . Without loss of generality, we assume  $I = g^s h^t$  output by  $\mathcal{A}$ , then we have  $L_{RPK} = \{g^{x_1} (g^s h^t)^e, \dots, g^{x_n} (g^s h^t)^e\}$ . Since  $\mathcal{A}$  finished the ring signature  $\tau$  with  $L_{RPK}$  under generator  $gh^e$ , from Lemma 9 we get  $\mathcal{A}$  knows  $RSK = z$  for at least one  $i \in \{1, \dots, n\}$  s.t.  $g^{x_i} (g^s h^t)^e = (gh^e)^z$ , except for negligible probability  $\epsilon_2 = \text{negl}(\lambda)$ . We can also assume that  $e = 0$  happens with negligible probability  $\epsilon_3 = \text{negl}(\lambda)$ . Then  $g^{x_i} (g^s h^t)^e = (gh^e)^z$  means  $\mathcal{A}$  gets a solution for  $g^{x_i - z + es} = h^{e(z-t)}$  with nonnegligible probability  $\delta_1 - \epsilon_1 - \epsilon_2 - \epsilon_3$ , if  $t \neq z$ , then from Lemma 8 we know this contradicts with the hardness of discrete logarithm, so we have  $t = z$ . Then we have  $x_i - t + se = 0$ , if  $s \neq 0$ , then  $e = (t - x_i)s^{-1}$ , which means  $e$  can be pre-computed before  $\mathcal{A}$  runs the hash function (random oracle), which happens with negligible probability. Then we get  $s = 0$ ,  $z = t = x_i$ , which contradicts to the assumptions above. Then we get that  $\mathcal{A}$  generates a valid SLRS signature  $\sigma \leftarrow \mathcal{SO}$  if and only if he queries the  $\mathcal{CO}$  at least once, except for negligible probability.

According to the fourth requirement that the number of times for  $\mathcal{A}$  querying  $\mathcal{CO}$  is  $\leq k - 1$ , and  $\mathcal{A}$  returned  $k$  valid SLRS signatures  $\sigma_i = (\tau_i, \mu_i, L_{PK}^i, I_i)$  for  $i = 1, \dots, k$ , then we know there are two SLRS signatures from the same query of  $\mathcal{CO}$ , saying  $SK = z$  from  $PK = g^z$ , and  $\mathcal{A}$  finished two unlinked valid

SLRS signatures, then there is at least one  $I_i = g^s h^t \neq h^z$  from the two SLRS signatures (otherwise they will be linked). We have  $L_{\text{RPK}} = \{g^{x_1}(g^s h^t)^e, \dots, g^{x_n}(g^s h^t)^e\}$ , since  $\exists j \in \{1, \dots, n\}$  s.t.  $x_j = z$ , and  $\mathcal{A}$  signs with  $\text{PK}_j$ , then we have  $g^{x_j}(g^s h^t)^e = (gh^e)^t g^{z-t+es}$  with  $g^s h^t \neq h^z$ , if  $z - t + es = 0$ , then we have  $s = 0$  and  $z = t$ , otherwise  $e$  will be pre-computed before  $\mathcal{A}$  runs the hash function (random oracle) by  $e = (t - z)s^{-1}$ , which happens with negligible probability  $\epsilon_1 = \text{negl}(\lambda)$ . Then we get  $z - t + es \neq 0$ , and this means  $\mathcal{A}$  can compute  $x$  s.t.  $(gh^e)^x = (gh^e)^t g^{z-t+es}$ , otherwise  $\mathcal{A}$  will break the unforgeability of ring signature, which happens with negligible probability  $\epsilon_2 = \text{negl}(\lambda)$ , however, we know that  $(gh^e)^x = (gh^e)^t g^{z-t+es}$  implies a non-trivial relationship between  $g$  and  $h$ , which happens with nonnegligible probability  $\delta - k\epsilon_0 - \epsilon_1 - \epsilon_2$ , this contradicts to the hardness assumption of discrete logarithm problem, then we finish the linkability proof of SLRS.  $\square$

### 3.2.4 Proof of Nonlanderability.

**THEOREM 11 (NONSLANDERABILITY).** *SLRS is nonlanderable for any PPT adversary  $\mathcal{A}$ , assuming the ring signature component satisfies the conditions in section 1.1.2.*

**PROOF.** For any PPT adversary  $\mathcal{A}$ , when  $\mathcal{A}$  finished the slandering game with  $\mathcal{S}$  in Definition 4,  $\mathcal{A}$  gave a list of public keys  $L_{\text{PK}}$ , a message  $\mu$  and a public key  $\text{PK}_\pi \in L_{\text{PK}}$  to  $\mathcal{S}$ ,  $\mathcal{S}$  returns the SLRS signature  $\sigma = (\tau, \mu, L_{\text{PK}}, I) \leftarrow \text{SLRS.Sign}(\text{SK}_\pi, L_{\text{PK}}, \mu)$  to  $\mathcal{A}$ . We assume that  $\mathcal{A}$  wins the slandering game with nonnegligible advantage  $\delta$ , that is,  $\mathcal{A}$  successfully outputs a SLRS signature  $\sigma^* = (\tau^*, \mu^*, L_{\text{PK}}^*, I^*)$ , satisfying the following:

1.  $\text{SLRS.Verify}(\tau^*, L_{\text{PK}}^*, \mu^*, I^*) = 1$ .
2.  $\text{PK}_\pi$  is not queried by  $\mathcal{A}$  to  $\mathcal{CO}$ .
3.  $\text{PK}_\pi$  is not queried by  $\mathcal{A}$  as input to  $\mathcal{SO}$ .
4.  $\text{SLRS.Link}((\tau, L_{\text{PK}}, \mu, I), (\tau^*, L_{\text{PK}}^*, \mu^*, I^*)) = \text{linked}$ .

From the definition of  $\text{SLRS.Link}$ , we know that  $I^* = I = h^{x_\pi}$ , since  $\text{PK}_\pi = g^{x_\pi}$  was not queried by  $\mathcal{A}$  to  $\mathcal{CO}$  and  $\mathcal{SO}$ , then  $\mathcal{A}$  does not know  $\text{SK} = x_\pi$  except for negligible probability  $\epsilon_0 = \text{negl}(\lambda)$  under the hardness of discrete logarithm problems. We know  $\mathcal{A}$  successfully produced a ring signature  $\tau^*$  with nonnegligible advantage  $\delta - \epsilon_0$ . Again from Lemma 9, according to the unforgeability of ring signature, then we know that  $\mathcal{A}$  knows at least one signing key except for negligible probability  $\epsilon_1 = \text{negl}(\lambda)$ , that is, there exists  $j \in \{1, \dots, n\}$ ,  $\mathcal{A}$  knows  $x$  s.t.  $\text{PK}_j^* \cdot I^e = (gh^e)^x$  with nonnegligible advantage  $\delta - \epsilon_0 - \epsilon_1$ , where  $e = H(L_{\text{PK}}^*, I, \mu)$ . Without loss of generality, we assume  $\text{PK}_j^* = g^s h^t$  output by  $\mathcal{A}$ , then we have  $(g^s h^t) h^{ex_\pi} = (gh^e)^x = (gh^e)^s h^{t+e(x_\pi-s)}$ . Using similar arguments in Theorem 10 (and Lemma 8), if  $t + e(x_\pi - s) = 0$ , then we have  $x_\pi = s$  and  $t = 0$ , otherwise  $e$  will be pre-computed before  $\mathcal{A}$  runs the hash function (random oracle), which happens with negligible probability  $\epsilon_2 = \text{negl}(\lambda)$ . Then  $t + e(x_\pi - s) \neq 0$  and  $\mathcal{A}$  gets a non-trivial relationship between  $g$  and  $h$  with nonnegligible advantage  $\delta - \epsilon_0 - \epsilon_1 - \epsilon_2$ , which contradicts to the hardness of discrete logarithm problem, then we finish the nonlanderability proof of SLRS.  $\square$

According to lemma 5, we get the unforgeability of SLRS:

**COROLLARY 12 (UNFORGEABILITY).** *SLRS is unforgeable for any PPT adversary  $\mathcal{A}$ .*

## 4 MSLRS FOR MULTI-RING APPLICATION

In this section we give two constructions of position-preserving linkable multi-ring signatures by usage of SLRS, named by MSLRS (with AOS) and MSLRS' (with AOS'). The construction is straightforward to realize the functionality of position preserving, which helps MSLRS and MSLRS' suitable in the Monero-type cryptocurrencies. Note that MSLRS' is compact, and is more efficient than CLSAG and MSLRS. Moreover, both of our schemes support different generator in each ring, which is unsupportable in CLSAG. In the following constructions, we take two-ring signatures as example, the linkability is effective for the first ring (similar to Monero).

### 4.1 MSLRS with AOS

Par  $\leftarrow$  MSLRS.Setup( $\lambda$ ):

1. System chooses an elliptic curve  $\mathbb{G}$  with prime order  $q$  and generators  $g_1, g_2 \in \mathbb{G}$ , system samples another generator  $h \in \mathbb{G}$  whose discrete logarithm (to  $g_i, i = 1, 2$ ) is unknown to anyone, system outputs  $(\mathbb{G}, q, g_1, g_2, h)$  as the public parameters.

(PK, SK)  $\leftarrow$  MSLRS.KeyGen(Par):

1. According to the public parameters  $(\mathbb{G}, q, g_1, g_2, h)$ , user Alice samples  $x, y \in \mathbb{Z}_q^*$  as her secret keys, then computes  $g_1^x, g_2^y$ ;
2. Alice outputs  $(\text{PK}, \text{PK}') = (g_1^x, g_2^y)$ , and retains  $(\text{SK}, \text{SK}') = (x, y)$ .

$\sigma \leftarrow$  MSLRS.Sign( $\text{SK}_\pi, \text{SK}'_\pi, \mu, L_{\text{PK}}, L'_{\text{PK}}$ ):

1. For a message  $\mu$ , Alice chooses another  $n - 1$  users, together with her own public keys, to generate two list of public keys  $L_{\text{PK}} = \{\text{PK}_1, \dots, \text{PK}_n\}$  and  $L'_{\text{PK}} = \{\text{PK}'_1, \dots, \text{PK}'_n\}$ , where Alice's  $\text{PK} = \text{PK}_\pi \in L_{\text{PK}}, \text{PK}' = \text{PK}'_\pi \in L'_{\text{PK}}, \pi \in \{1, \dots, n\}$ , which means the position of Alice's public key in each ring is same;
2. Alice outputs the key-image  $I = h^{x_\pi}$ , then computes  $e = H(L_{\text{PK}}, L'_{\text{PK}}, I, \mu)$ ;
3. Alice computes the signature public key set for the first ring

$$L_{\text{RPK}} = \{\text{PK}_1 \cdot I^e, \dots, \text{PK}_n \cdot I^e\} \\ = \{g_1^{x_1} h^{ex_\pi}, \dots, g_1^{x_n} h^{ex_\pi}\} = \{\text{RPK}_i\}_{i=1, \dots, n}$$

4. Alice samples  $r_\pi, t_\pi \in \mathbb{Z}_q^*$  uniformly and computes  $e_{\pi+1} = H((g_1 h^e)^{r_\pi}, g_2^{t_\pi}, L_{\text{RPK}}, L'_{\text{PK}}, I, \mu)$ ;
5. For  $i = \pi + 1, \dots, n, 1, \dots, \pi - 1$ , Alice samples  $z_i, s_i \in \mathbb{Z}_q^*$  uniformly at random, then computes  $e_{i+1} = H((g_1 h^e)^{z_i} / (\text{RPK}_i)^{e_i}, g_2^{s_i} / (\text{PK}'_i)^{e_i}, L_{\text{RPK}}, L'_{\text{PK}}, I, \mu)$ ;
6. Alice computes  $z_\pi = r_\pi + x e_\pi, s_\pi = t_\pi + y e_\pi$  and outputs  $\tau = (e_1; z_1, \dots, z_n; s_1, \dots, s_n)$ ;
7. Alice outputs  $\sigma = (\tau, \mu, L_{\text{PK}}, L'_{\text{PK}}, I)$  as the MSLRS outputs.

1/0  $\leftarrow$  MSLRS.Verify( $\tau, \mu, L_{\text{PK}}, L'_{\text{PK}}, I$ ):

1. Verifier computes  $e^* = H(L_{\text{PK}}, L'_{\text{PK}}, I, \mu)$ ;
2. Verifier computes  $L_{\text{RPK}}^* = \{\text{PK}_1 \cdot I^{e^*}, \dots, \text{PK}_n \cdot I^{e^*}\}$ ;
3. For  $i = 1, \dots, n$ , verifier computes  $e_{i+1}^* = H((g_1 h^{e^*})^{z_i} / (\text{RPK}_i^*)^{e_i^*}, g_2^{s_i} / (\text{PK}'_i)^{e_i^*}, L_{\text{RPK}}, L'_{\text{PK}}, I, \mu)$ , where  $e_1 = e_1^*$ ;
4. Verifier checks  $e_1 \stackrel{?}{=} e_{n+1}^*$ ;
5. If all passed then outputs 1, otherwise outputs 0.

linked/unlinked  $\leftarrow$  MSLRS.Link( $\sigma, \sigma'$ ):

1. For two valid MSLRS signatures  $\sigma_1 = (\tau_1, \mu_1, L_{\text{PK}}^{(1)}, L'_{\text{PK}}^{(1)}, I_1)$  and  $\sigma_2 = (\tau_2, \mu_2, L_{\text{PK}}^{(2)}, L'_{\text{PK}}^{(2)}, I_2)$ , if  $I_1 = I_2$  then verifier outputs linked, otherwise outputs unlinked.

**Algorithm 2: MSLRS**

Note that the key-image works for the linkability of the first ring  $L_{PK}$ , we use a multi-ring generalization of AOS in the construction to make sure the same position of the corresponding signing key in each ring. Meanwhile, the generator in each ring ( $g_1 h^e$  and  $g_2$ ) is different. The signature size (with  $I$ ) of  $m$ -ring MSLRS is  $(1, mn + 1)$ , where  $(\cdot, \cdot)$  refers to number of elements in  $(\mathbb{G}, \mathbb{Z}_q^*)$ .

**4.2 MSLRS' with AOS'**

Par  $\leftarrow$  MSLRS'.Setup( $\lambda$ ):

- System chooses an elliptic curve  $\mathbb{G}$  with prime order  $q$  and generators  $g_1, g_2 \in \mathbb{G}$ , system samples another generator  $h \in \mathbb{G}$  whose discrete logarithm (to  $g_i, i = 1, 2$ ) is unknown to anyone, system outputs  $(\mathbb{G}, q, g_1, g_2, h)$  as the public parameters.

$(PK, SK) \leftarrow$  MSLRS'.KeyGen(Par):

- According to the public parameters  $(\mathbb{G}, q, g_1, g_2, h)$ , user Alice samples  $x, y \in \mathbb{Z}_q^*$  as her secret keys, then computes  $g_1^x, g_2^y$ ;
- Alice outputs  $(PK, PK') = (g_1^x, g_2^y)$ .

$\sigma \leftarrow$  MSLRS'.Sign( $SK_\pi, SK'_\pi, \mu, L_{PK}, L'_{PK}$ ):

- For a message  $\mu$ , Alice chooses another  $n - 1$  users, together with her own public keys, to generate two list of public keys  $L_{PK} = \{PK_1, \dots, PK_n\}$  and  $L'_{PK} = \{PK'_1, \dots, PK'_n\}$ , where Alice's PK =  $PK_\pi \in L_{PK}, PK' = PK'_\pi \in L'_{PK}, \pi \in \{1, \dots, n\}$ , which means the position of Alice's public key in each ring is same;
- Alice outputs the key-image  $I = h^{x\pi}$ , then computes  $e = H(L_{PK}, L'_{PK}, I, \mu)$ ;
- Alice computes the signature public key set for the first ring
 
$$L_{RPK} = \{PK_1 \cdot I^e, \dots, PK_n \cdot I^e\}$$

$$= \{g_1^{x_1} h^{e x_\pi}, \dots, g_1^{x_n} h^{e x_\pi}\} = \{RPK_i\}_{i=1, \dots, n};$$
- Alice samples  $c_1, \dots, c_{\pi-1}, c_{\pi+1}, \dots, c_n \in \mathbb{Z}_q^*$  uniformly, then Alice samples  $\alpha, \beta \in \mathbb{Z}_q^*$  uniformly, computes  $R_1 = (g_1 h^e)^\alpha \prod_{i \neq \pi} (RPK_i)^{c_i}$  and  $R_2 = g_2^\beta \prod_{i \neq \pi} (PK'_i)^{c_i}$ ;
- Alice computes  $c = H(R_1, R_2, L_{RPK}, L'_{PK}, I, \mu)$ ;
- Alice computes  $c_\pi = c - \sum_{i \neq \pi} c_i$ , then computes  $z_1 = \alpha - c_\pi x$  and  $z_2 = \beta - c_\pi y$ , outputs  $\tau = (z_1, z_2; c_1, \dots, c_n)$ ;
- Alice outputs  $\sigma = (\tau, \mu, L_{PK}, L'_{PK}, I)$  as the MSLRS' outputs.

$1/0 \leftarrow$  MSLRS'.Verify( $\tau, \mu, L_{PK}, L'_{PK}, I$ ):

- Verifier computes  $e^* = H(L_{PK}, L'_{PK}, I, \mu)$ ;
- Verifier computes  $L_{RPK}^* = \{PK_1 \cdot I^{e^*}, \dots, PK_n \cdot I^{e^*}\}$ ;
- Verifier computes  $R_1^* = (g_1 h^{e^*})^{z_1} \prod_{i=1}^n (RPK_i^*)^{c_i}$  and  $R_2^* = g_2^{z_2} \prod_{i=1}^n (PK'_i)^{c_i}$ ;
- Verifier checks  $\sum_{i=1}^n c_i \stackrel{?}{=} H(R_1^*, R_2^*, L_{RPK}^*, L'_{PK}, I, \mu)$ ;
- If all passed then outputs 1, otherwise outputs 0.

$linked/unlinked \leftarrow$  MSLRS'.Link( $\sigma, \sigma'$ ):

- For two valid MSLRS' signatures  $\sigma_1 = (\tau_1, \mu_1, L_{PK}^{(1)}, L'_{PK}^{(1)}, I_1)$  and  $\sigma_2 = (\tau_2, \mu_2, L_{PK}^{(2)}, L'_{PK}^{(2)}, I_2)$ , if  $I_1 = I_2$  then verifier outputs *linked*, otherwise outputs *unlinked*.

**Algorithm 3: MSLRS'**

Similar to MSLRS, the key-image works for the linkability of the first ring  $L_{PK}$ , and the generator in each ring ( $g_1 h^e$  and  $g_2$ ) is different. The signature size (with  $I$ ) of  $m$ -ring MSLRS' is  $(1, m + n)$ , which is a compact scheme as CLSAG (with size  $(m, n + 1)$ [10]), where  $(\cdot, \cdot)$  refers to number of elements in  $(\mathbb{G}, \mathbb{Z}_q^*)$ .

**5 IMPLEMENTATION AND PERFORMANCE**

We implement our works, including SLRS (SLRS'), MSLRS (MSLRS'), as well as the existing schemes, such as MLSAG, CLSAG and Ring-CT 3.0 in Golang, use Ed25519 curve and Ristretto library. We use SHA256 as the hash function. All experiments are conducted on a desktop with 64-bit Win 10 system and 16GB RAM. The processor is Intel(R) Core(TM) i7-8700 CPU @ 3.2 GHz with 6 cores.

We compare the size and efficiency (generation and verification) of each scheme for single ring (in 5.1) and double rings (in 5.2) respectively, the implementations follow the original algorithms directly, without any multi-threading parallel acceleration. We select Ring-CT 3.0 with linkability in the comparison, which has the same functionality as MLSAG and SLRS.

Note that the SLRS' (with AOS') and Ring-CT 3.0 are suited for multi-threading parallel acceleration, as the exponentiations can be done in parallel during generation and verification in each scheme. We also give implementations of SLRS' and Ring-CT 3.0 under multi-threading parallel acceleration with significant improvements in efficiency, we give the detailed comparison in 5.3.

**5.1 Performance of Single Ring****Table 2: Performance of Single Ring**

| Scheme      | $n$  | Generation | Verification | Size                |
|-------------|------|------------|--------------|---------------------|
| SLRS(AOS)   | 11   | 1.10ms     | 1.09ms       | $(1, n + 1)$        |
|             | 32   | 3.10ms     | 3.08ms       |                     |
|             | 128  | 12.31ms    | 12.17ms      |                     |
|             | 1024 | 98.04ms    | 97.19ms      |                     |
| SLRS'(AOS') | 11   | 0.65ms     | 0.66ms       | $(1, n + 1)$        |
|             | 32   | 1.73ms     | 1.71ms       |                     |
|             | 128  | 6.67ms     | 6.54ms       |                     |
|             | 1024 | 52.80ms    | 51.70ms      |                     |
| Ring-CT 3.0 | 11   | 6.18ms     | 3.17ms       | $(2 \log n + 9, 7)$ |
|             | 32   | 13.85ms    | 6.22ms       |                     |
|             | 128  | 55.01ms    | 22.65ms      |                     |
|             | 1024 | 434.60ms   | 172.12ms     |                     |
| LSAG        | 11   | 1.64ms     | 1.66ms       | $(1, n + 1)$        |
|             | 32   | 4.85ms     | 4.86ms       |                     |
|             | 128  | 19.53ms    | 19.39ms      |                     |
|             | 1024 | 156.39ms   | 155.22ms     |                     |

The detailed performance results are summarized in Table2, where  $(\cdot, \cdot)$  refers to number of elements in  $(\mathbb{G}, \mathbb{Z}_q^*)$ . Note that LSAG is the single ring version of MLSAG and Ring-CT 3.0 is the linkable version with key-image. The comparison of generation time is in Figure1 and the comparison of verification time is in Figure4 (in Appendix B). All implementations use no parallel accelerations.

**5.2 Performance of Double Rings**

The detailed performance results of double rings ( $m = 2$ ) are summarized in Table3. We compare the generation time (in Figure2) and verification time (in Figure5). All implementations use no parallel accelerations.

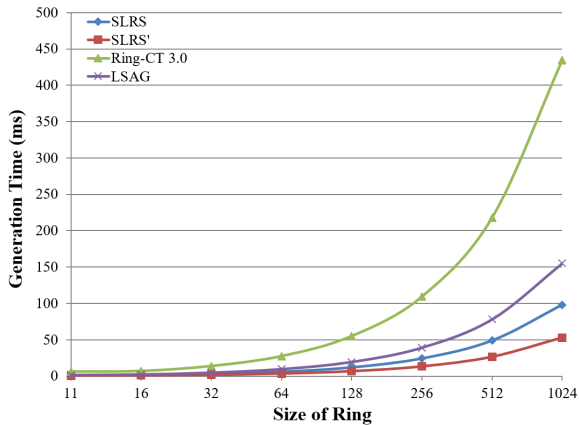


Figure 1: Generation Time of Single Ring.

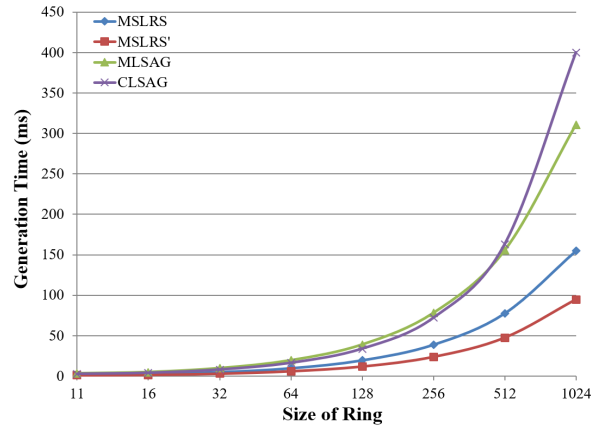


Figure 2: Generation Time of Double Rings.

Table 3: Performance of Double Rings

| Scheme       | $n$  | Generation | Verification | Size          |
|--------------|------|------------|--------------|---------------|
| MSLRS(AOS)   | 11   | 1.67ms     | 1.71ms       | $(1, mn + 1)$ |
|              | 16   | 4.83ms     | 4.84ms       |               |
|              | 32   | 19.46ms    | 19.20ms      |               |
|              | 128  | 154.79ms   | 153.50ms     |               |
|              | 1024 | 400.47ms   | 400.48ms     |               |
| MSLRS'(AOS') | 11   | 1.09ms     | 1.11ms       | $(1, m + n)$  |
|              | 32   | 3.02ms     | 3.04ms       |               |
|              | 128  | 11.88ms    | 11.83ms      |               |
|              | 1024 | 94.61ms    | 93.58ms      |               |
|              | 1024 | 94.61ms    | 93.58ms      |               |
| MLSAG        | 11   | 3.26ms     | 3.31ms       | $(1, mn + 1)$ |
|              | 32   | 9.63ms     | 9.45ms       |               |
|              | 128  | 38.79ms    | 38.60ms      |               |
|              | 1024 | 310.80ms   | 308.35ms     |               |
|              | 1024 | 310.80ms   | 308.35ms     |               |
| CLSAG        | 11   | 2.72ms     | 2.80ms       | $(m, n + 1)$  |
|              | 32   | 7.99ms     | 8.05ms       |               |
|              | 128  | 33.76ms    | 33.61ms      |               |
|              | 1024 | 400.47ms   | 400.48ms     |               |
|              | 1024 | 400.47ms   | 400.48ms     |               |

Table 4: Performance under Parallel Acceleration

| Scheme      | $n$  | Generation | Verification | Size                |
|-------------|------|------------|--------------|---------------------|
| SLRS'(AOS') | 11   | 0.32ms     | 0.26ms       | $(1, n + 1)$        |
|             | 16   | 0.39ms     | 0.36ms       |                     |
|             | 32   | 0.61ms     | 0.56ms       |                     |
|             | 64   | 1.09ms     | 1.02ms       |                     |
|             | 128  | 2.00ms     | 1.88ms       |                     |
|             | 256  | 3.79ms     | 3.58ms       |                     |
|             | 512  | 7.38ms     | 6.92ms       |                     |
|             | 1024 | 14.63ms    | 13.81ms      |                     |
| Ring-CT 3.0 | 11   | 2.98ms     | 1.30ms       | $(2 \log n + 9, 7)$ |
|             | 16   | 3.24ms     | 1.40ms       |                     |
|             | 32   | 5.26ms     | 2.00ms       |                     |
|             | 64   | 8.17ms     | 3.09ms       |                     |
|             | 128  | 11.90ms    | 4.76ms       |                     |
|             | 256  | 19.75ms    | 8.78ms       |                     |
|             | 512  | 36.60ms    | 16.76ms      |                     |
|             | 1024 | 69.56ms    | 32.97ms      |                     |

### 5.3 Performance under Parallel Acceleration

Both SLRS' and Ring-CT 3.0 are suitable for multi-threading parallel acceleration to reduce the time of generation and verification for 3-5 times. In this subsection we give the detailed performance results and comparison between SLRS' and Ring-CT 3.0 under parallel acceleration in Table4 and Figure3.

## 6 CONCLUSION

In this paper, we give a simpler and modular construction of linkable ring signature scheme (SLRS) by modifying the key-image generation and embedding algorithm, and using ring signature as component, without any additional one-time signatures or zero-knowledge proofs, which reduces the signature size, shortens the time for generation and verification. Our construction is modular,

one can choose any suited elliptic-curve-based ring signature (satisfying the conditions in section 1.1.2) as component to realize linkability. Moreover, our construction can be generalized to position-preserving linkable multi-ring signature (MSLRS, MSLRS') to adapt to the application of cryptocurrencies. At last, both SLRS' and MSLRS' can be adapted to parallel acceleration to further improve the performance.

## REFERENCES

- [1] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 2002. 1-out-of-n signatures from a variety of keys. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 415–432.
- [2] Man Ho Au, Sherman SM Chow, Willy Susilo, and Patrick P Tsang. 2006. Short linkable ring signatures revisited. In *European Public Key Infrastructure Workshop*. Springer, 101–115.
- [3] Adam Back. 2015. Ring signature efficiency. *Bitcointalk* (accessed 1 May 2015) (2015). <https://bitcointalk.org/index.php>.
- [4] Adam Bender, Jonathan Katz, and Ruggero Morselli. 2006. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography Conference*. Springer, 60–79.



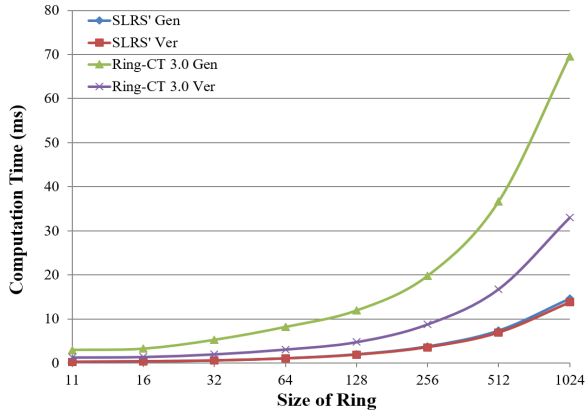


Figure 3: Computation Time under Parallel Acceleration.

- [5] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 315–334.
- [6] Vitalik Buterin. 2014. A next-generation smart contract and decentralized application platform. (2014). [https://cryptorating.eu/whitepapers/Ethereum/Ethereum\\_white\\_paper.pdf](https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf).
- [7] Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup. 2004. Anonymous identification in ad hoc groups. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 609–626.
- [8] Evan Duffield and Daniel Diaz. 2015. Dash: A privacycentric cryptocurrency. *GitHub* (2015). <https://github.com/dashpay/dash/wiki/Whitepaper>.
- [9] Eiichiro Fujisaki and Koutarou Suzuki. 2007. Traceable ring signature. In *International Workshop on Public Key Cryptography*. Springer, 181–200.
- [10] Brandon Goodell, Sarang Noether, and RandomRun. 2019. Compact linkable ring signatures and applications. *Cryptology ePrint Archive*, Report 2019/654. <https://eprint.iacr.org/2019/654>.
- [11] Jens Groth and Markulf Kohlweiss. 2015. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 253–280.
- [12] Joseph K Liu, Man Ho Au, Willy Susilo, and Jianying Zhou. 2013. Linkable ring signature with unconditional anonymity. *IEEE Transactions on Knowledge and Data Engineering* 26, 1 (2013), 157–165.
- [13] Joseph K Liu, Victor K Wei, and Duncan S Wong. 2004. Linkable spontaneous anonymous group signature for ad hoc groups. In *Australasian Conference on Information Security and Privacy*. Springer, 325–335.
- [14] Greg Maxwell. 2015. Confidential transactions. (2015). [https://people.xiph.org/~greg/confidential\\_values.txt](https://people.xiph.org/~greg/confidential_values.txt).
- [15] Gregory Maxwell and Andrew Poelstra. 2015. Borromean ring signatures. [https://raw.githubusercontent.com/Blockstream/borromean\\_paper/master/borromean\\_draft\\_0.01\\_34241bb.pdf](https://raw.githubusercontent.com/Blockstream/borromean_paper/master/borromean_draft_0.01_34241bb.pdf).
- [16] Satoshi Nakamoto et al. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). <https://git.dhimmel.com/bitcoin-whitepaper/>.
- [17] Shen Noether, Adam Mackenzie, et al. 2016. Ring confidential transactions. *Ledger* 1 (2016), 1–18.
- [18] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*. Springer, 129–140.
- [19] Ronald L Rivest, Adi Shamir, and Yael Tauman. 2001. How to leak a secret. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 552–565.
- [20] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.
- [21] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. 2017. RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency Monero. In *European Symposium on Research in Computer Security*. Springer, 456–474.
- [22] Patrick P Tsang and Victor K Wei. 2005. Short linkable ring signatures for e-voting, e-cash and attestation. In *International Conference on Information Security Practice and Experience*. Springer, 48–60.
- [23] Nicolas Van Saberhagen. 2013. CryptoNote v 2.0. (2013). <https://cryptonote.org/whitepaper.pdf>.

- [24] Tsz Hon Yuen, Joseph K Liu, Man Ho Au, Willy Susilo, and Jianying Zhou. 2013. Efficient linkable and/or threshold ring signature without random oracles. *Comput. J.* 56, 4 (2013), 407–421.
- [25] Tsz Hon Yuen, Shi-feng Sun, Joseph K Liu, Man Ho Au, Muhammed F Esgin, Qingzhao Zhang, and Dawu Gu. 2019. RingCT 3.0 for Blockchain Confidential Transaction: Shorter Size and Stronger Security. (2019).

## A REMAINING PRELIMINARIES

### A.1 Special Soundness

**DEFINITION 13 ( $k$ -SPECIAL SOUNDNESS).** In sigma protocols with Fiat-Shamir transformation in the random oracle model, for any non-uniform polynomial time adversary  $\mathcal{A}$  who can generate  $k$  valid proofs  $(x, c, e_1, s_1), \dots, (x, c, e_k, s_k)$ , then there exists an extraction algorithm  $Ext$  which can extract a witness  $(x, w) \in R$ , where  $c$  represents the commitment,  $e_i$ s are challenges and  $s_i$ s are responses.

### A.2 AOS Ring Signature

We give the introduction of AOS ring signature[1] in the following:

- $Par \leftarrow Setup(\lambda)$ : system chooses an elliptic curve  $\mathbb{G}$  and a generator  $g$  as the public parameters.
- $(PK_\pi, SK_\pi) \leftarrow KeyGen(Par)$ : according to the public parameters, user  $P_\pi$  samples  $x \in \mathbb{Z}_q^*$  uniformly at random, computes  $g^x$  and sets  $(PK_\pi, SK_\pi) = (g^x, x)$ .
- $\sigma \leftarrow Rsign(SK_\pi, \mu, L_{PK})$ : when user  $P_\pi$  generates a ring signature for message  $\mu$ , he chooses another  $n - 1$  users' public keys, together with his own  $PK_\pi$  to obtain a set of public keys  $L_{PK} = \{PK_1, \dots, PK_n\}$ , where  $PK_\pi \in L_{PK}$  and  $\pi \in \{1, \dots, n\}$ , then he does as follows:
  1.  $P_\pi$  samples  $r_\pi \in \mathbb{Z}_q^*$  uniformly at random, then computes  $c_{\pi+1} = H(g^{r_\pi}, L_{PK}, \mu)$ ;
  2. For  $i = \pi + 1, \dots, n, 1, \dots, \pi - 1$ ,  $P_\pi$  samples  $z_i \in \mathbb{Z}_q^*$  uniformly and computes  $c_{i+1} = H(g^{z_i} / (PK_i)^{c_i}, L_{PK}, \mu)$ ;
  3.  $P_\pi$  computes  $z_\pi = r_\pi + xc_\pi$ ;
  4. Output the ring signature  $\sigma = (c_1; z_1, \dots, z_n)$ .
- $1/0 \leftarrow Verify(\mu, \sigma, L_{PK})$ : for a ring signature  $(\mu, L_{PK}, \sigma)$ , for  $i = 1, \dots, n$  the verifier computes

$$c_{i+1}^* = H(g^{z_i} / (PK_i)^{c_i^*}, L_{PK}, \mu)$$

where  $c_1 = c_1^*$ , then checks  $c_1 \stackrel{?}{=} c_{n+1}^*$ , if all passed then outputs 1, otherwise outputs 0.

### A.3 AOS' Ring Signature

AOS' is introduced in the Appendix of [1] with better efficiency than AOS.

- $Par \leftarrow Setup'(\lambda)$ : system chooses an elliptic curve  $\mathbb{G}$  and a generator  $g$  as the public parameters.
- $(PK_\pi, SK_\pi) \leftarrow KeyGen'(Par)$ : according to the public parameters, user  $P_\pi$  samples  $x \in \mathbb{Z}_q^*$  uniformly at random, computes  $g^x$  and sets  $(PK_\pi, SK_\pi) = (g^x, x)$ .
- $\sigma \leftarrow Rsign'(SK_\pi, \mu, L_{PK})$ : when user  $P_\pi$  generates a ring signature for message  $\mu$ , he chooses another  $n - 1$  users' public keys, together with his own  $PK_\pi$  to obtain a set of public keys  $L_{PK} = \{PK_1, \dots, PK_n\}$ , where  $PK_\pi \in L_{PK}$  and  $\pi \in \{1, \dots, n\}$ , then he does as follows:

1. For  $i = 1, \dots, \pi - 1, \pi + 1, \dots, n$ ,  $P_\pi$  samples  $\alpha, c_i \in \mathbb{Z}_q^*$  uniformly at random, then computes  $R = g^\alpha \prod_{i \neq \pi} \text{PK}_i^{c_i}$  and  $c = H(R, L_{\text{PK}}, \mu)$ ;
  2.  $P_\pi$  computes  $c_\pi = c - \sum_{i \neq \pi} c_i$ ;
  3.  $P_\pi$  computes  $z = \alpha - x c_\pi$ ;
  4. Output the ring signature  $\sigma = (z; c_1, \dots, c_n)$ .
- $1/0 \leftarrow \text{Verify}'(\mu, \sigma, L_{\text{PK}})$ : for a ring signature  $(\mu, L_{\text{PK}}, \sigma)$ , the verifier computes  $R^* = g^z \prod_{i=1}^n \text{PK}_i^{c_i}$ , then checks  $\sum_{i=1}^n c_i \stackrel{?}{=} H(R^*, L_{\text{PK}}, \mu)$ , if all passed then outputs 1, otherwise outputs 0.

The AOS (AOS') ring signature schemes are unforgeable and anonymous in the random oracle model.

#### A.4 CLSAG Ring Signature

CLSAG is introduced in [10] to provide a more compact and efficient linkable multi-ring signature than MLSAG in Monero. We give the introduction of 2-ring signature as example:

- $\text{Par} \leftarrow \text{Setup}(\lambda)$ : system chooses an elliptic curve  $\mathbb{G}$  and generators  $g \in \mathbb{G}$ , outputs  $(\mathbb{G}, q, g)$  as the public parameters.
- $(\text{PK}, \text{SK}) \leftarrow \text{KeyGen}(\text{Par})$ :
  1. According to the public parameters  $(\mathbb{G}, q, g)$ , for  $d = 2$  (two rings), user Alice samples  $x, y \in \mathbb{Z}_q^*$  as her secret keys, computes  $\text{PK}_A = g^x, \text{PK}'_A = g^y$ .
- $\sigma \leftarrow \text{Rsign}(\text{SK}_{\text{Alice}}, \mu, L_{\text{PK}})$ :
  1. For a message  $\mu$ , Alice chooses another  $n-1$  users, together with her own public keys, to generate two lists of public keys:  
 $L_{\text{PK}} = \{\text{PK}_1, \dots, \text{PK}_n\}, L'_{\text{PK}} = \{\text{PK}'_1, \dots, \text{PK}'_n\}$ . Where Alice's  $\text{PK}_{\text{Alice}} = (\text{PK}_A, \text{PK}'_A) = (\text{PK}_i, \text{PK}'_i) \in L_{\text{PK}}$ ;
  2. Alice computes  $I = H_p(\text{PK}_A)^x, D = H_p(\text{PK}_A)^y$ , and  $e_1 = H(I, D, L_{\text{PK}}, 1), e_2 = H(I, D, L_{\text{PK}}, 2)$ ;
  3. Alice samples  $\alpha \in \mathbb{Z}_q^*$  uniformly at random, computes

$$L_i = g^\alpha, R_i = H_p(\text{PK}_A)^\alpha,$$

then computes  $c_{i+1} = H(L_i, R_i, \mu, L_{\text{PK}}, I, D)$ . Then for  $j = i+1, \dots, i-1$ , Alice samples  $s_j \in \mathbb{Z}_q^*$  and computes

$$L_j = g^{s_j} (\text{PK}_j^{e_1} (\text{PK}'_j)^{e_2})^{c_j},$$

$$R_j = H_p(\text{PK}_j)^{s_j} (I^{e_1} D^{e_2})^{c_j}.$$

Then computes  $c_{j+1} = H(L_j, R_j, \mu, L_{\text{PK}}, I, D)$ ;

4. Alice computes  $s_i = \alpha - c_i(e_1 x + e_2 y)$ ;
  5. Alice outputs  $\sigma = (c_1, s_1, \dots, s_n, I, D, L_{\text{PK}})$ .
- $1/0 \leftarrow \text{Verify}(c_1, s_1, \dots, s_n, I, D, L_{\text{PK}})$ :
1. Verifier computes  $e_1^* = H(I, D, L_{\text{PK}}, 1), e_2^* = H(I, D, L_{\text{PK}}, 2)$ ;
  2. For  $j = 1, \dots, n$ , verifier computes

$$L_j^* = g^{s_j} (\text{PK}_j^{e_1^*} (\text{PK}'_j)^{e_2^*})^{c_j^*},$$

$$R_j^* = H_p(\text{PK}_j)^{s_j} (I^{e_1^*} D^{e_2^*})^{c_j^*}.$$

Where  $c_1 = c_1^*$ . Then computes

$$c_{j+1}^* = H(L_j^*, R_j^*, \mu, L_{\text{PK}}, I, D);$$

3. Verifier checks whether  $c_{n+1}^* \stackrel{?}{=} c_1$ ;
4. If all passed then outputs 1, otherwise outputs 0.

- $\text{linked/unlinked} \leftarrow \text{Link}(\sigma, \sigma')$ : For two CLSAG signatures  $\sigma = (c_1, s_1, \dots, s_n, I, D, L_{\text{PK}})$  and  $\sigma' = (c'_1, s'_1, \dots, s'_n, I', D', L'_{\text{PK}})$ , if  $I = I'$  then verifier outputs *linked*, otherwise outputs *unlinked*.

Where  $H_p(\cdot)$  refers to Hash-to-Point, similar to MLSAG. Note that CLSAG only support the same generators in all rings.

## B REMAINING PERFORMANCE COMPARISONS

The verification time comparisons for single ring and double rings are given in this subsection. Verification time of single ring is shown in Figure4, and verification time of double rings is shown in Figure5. From the comparisons we can conclude that our schemes ((M)SLRS, (M)SLRS') are more efficient than existing schemes.

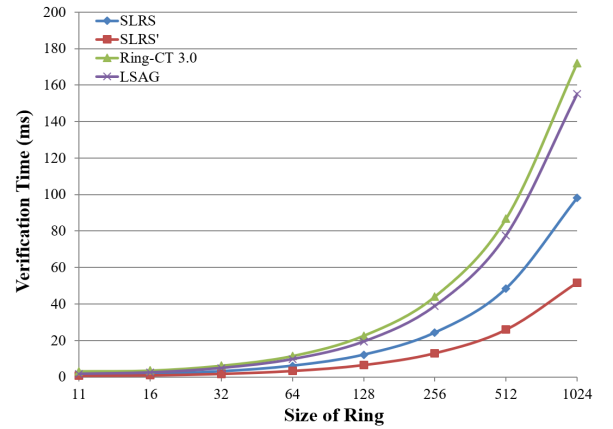


Figure 4: Verification Time of Single Ring.

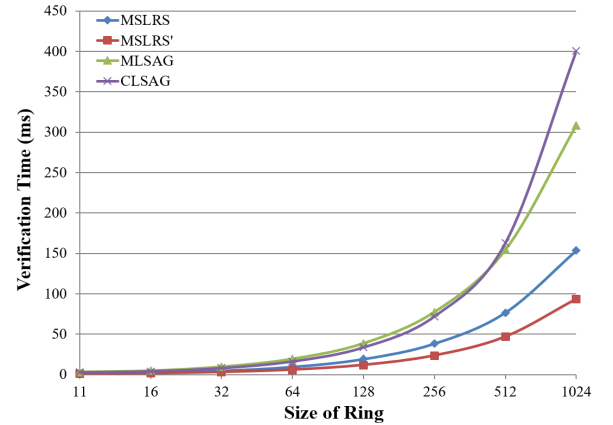


Figure 5: Verification Time of Double Rings.