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. 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 (≤ 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 $(\mathsf{PK}, \mathsf{SK})$ 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;

Conference'17, July 2017, Washington, DC, USA
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 $L_{\mathsf{PK}_i} = \{g_i^{x_{i,1}}, \cdots, g_i^{x_{i,n}}\}$ for $i = 1, \cdots, m$, 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, MSLAG 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}=$\{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$ as the ring elements, and uses his secret key SK_π to sign, where $\pi \in \{1, \cdots, n\}$. 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 $negl(\cdot)$ 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.

- $\mathsf{Par} \leftarrow \mathsf{Setup}(\lambda)$ is a probabilistic polynomial time (PPT) algorithm which, on input a security parameter λ, outputs the set of security parameters Par which includes λ.
- $(\mathsf{PK}_i, \mathsf{SK}_i) \leftarrow \mathsf{KeyGen}(\mathsf{Par})$ is a PPT algorithm which, on input the security parameters Par, outputs a key pair $(\mathsf{PK}_i, \mathsf{SK}_i)$.
- $\sigma \leftarrow \mathsf{Rsign}(\mathsf{SK}_\pi, \mu, L_{\mathsf{PK}})$ is a ring signature algorithm which, on input user's secret key SK_π, a list of users' public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$, where $\mathsf{PK}_\pi \in L_{\mathsf{PK}}, \pi \in \{1, \cdots, n\}$, and a message μ, outputs a ring signature σ.
- $1/0 \leftarrow \mathsf{Verify}(\mu, \sigma, L_{\mathsf{PK}})$ is a verification algorithm which, on input a message μ, a list of users' public keys L_{PK} and a ring signature σ, 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.
- $\mathsf{PK}_i \leftarrow \mathcal{JO}(\perp)$. *Joining oracle*, on request, adds a new user to the system. It returns the public key PK_i of the new user.
- $\mathsf{SK}_i \leftarrow \mathcal{CO}(\mathsf{PK}_i)$. *Corruption oracle*, on input a public key PK_i that is a query output of $\mathcal{JO}$, returns the corresponding private key SK_i.
- $\sigma \leftarrow \mathcal{SO}(\mathsf{PK}_\pi, \mu, L_{\mathsf{PK}})$. *Signing oracle*, on input a list of users' public keys L_{PK}, the public key of the signer PK_π, and a message μ, returns a valid ring signature σ.

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 $(\sigma^*, L_{PK}^*, \mu^*)$ satisfying the following:

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

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

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

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 $L_{PK} = \{PK_1, \cdots, PK_n\}$, $\mathcal{S}$ randomly picks $\pi \in \{1, \cdots, n\}$, computes $\sigma = Rsign(SK_\pi, \mu, L_{PK})$ and sends σ to $\mathcal{A}$, where SK_π is the corresponding private key of PK_π, then $\mathcal{A}$ outputs a guess $\pi^* \in \{1, \cdots, n\}$. $\mathcal{A}$ wins the game if he successfully guesses $\pi^* = \pi$. The advantage of $\mathcal{A}$ in the anonymity attack is $\mathrm{Adv}_{\mathcal{A}}^{anon} = |\Pr[\pi^* = \pi] - 1/n|$.

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

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:

- $linked/unlinked \leftarrow \mathsf{Link}((\sigma, \mu, L_{\mathsf{PK}}), (\sigma', \mu', L'_{\mathsf{PK}}))$: 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_{PK} are returned by $\mathcal{A}$ to $\mathcal{JO}$ and all public keys in L_{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 $(\sigma_i, L_{PK}^i, \mu_i), i = 1, \cdots, k$, satisfying the following:

1. *All σ_is are not returned by $\mathcal{A}$ to $\mathcal{SO}$.*
2. *All L_{PK}^i are returned by $\mathcal{A}$ to $\mathcal{JO}$.*
3. *Verify$(\sigma_i, L_{PK}^i, \mu_i) = 1, i = 1, \cdots, k$.*
4. *$\mathcal{A}$ queried $\mathcal{CO}$ less than k times.*
5. *Link$((\sigma_i, L_{PK}^i, \mu_i), (\sigma_j, L_{PK}^j, \mu_j)) = unlinked$ for $i \neq j$ and $i, j \in \{1, \cdots, k\}$.*

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

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

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 μ and a public key $PK_\pi \in L_{PK}$ to $\mathcal{S}$, $\mathcal{S}$ returns the corresponding signature $\sigma \leftarrow Rsign(SK_\pi, L_{PK}, \mu)$ to $\mathcal{A}$. $\mathcal{A}$ wins the game if he successfully outputs a ring signature $(\sigma^*, L_{PK}^*, \mu^*)$, satisfying the following:

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

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

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

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 ($\mathsf{PK} = g^s, \mathsf{SK} = s$) 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: $\{(\mathsf{PK}_i, c_i = g^{x_i} h^{a_i})\}_{i=1,\cdots,n-1}$, along with her input UTXO ($\mathsf{PK}_A = g^s, c_A = g^{x_A} h^{a_A}$), to generate a set of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_A, \mathsf{PK}_1, \cdots, \mathsf{PK}_{n-1}\}$ (randomized order), Alice also generates the output UTXO ($\mathsf{PK}_B, c_B = g^{x_B} h^{a_B}$), 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})

$$L_v = \{c_A c_B^{-1}, c_1 c_B^{-1}, \cdots, c_{n-1} c_B^{-1}\}$$

$$= \{g^{x_A - x_B}, g^{x_1 - x_B} h^{a_1 - a_B}, \cdots, g^{x_{n-1} - x_B} h^{a_{n-1} - a_B}\}.$$

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.

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:

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

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.

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

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 $\mathsf{PK} = g^x$;
2. Alice outputs $\mathsf{PK} = g^x$, and retains $\mathsf{SK} = x$.

$\sigma \leftarrow \mathsf{SLRS.Sign}(\mathsf{SK}_\pi, \mu, L_{\mathsf{PK}})$:

1. For a message μ, Alice chooses another $n-1$ users, together with her own public key, to generate a list of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$, where Alice's $\mathsf{PK} = \mathsf{PK}_\pi \in L_{\mathsf{PK}}, \pi \in \{1, \cdots, n\}$;
2. Alice outputs the key-image $I = h^{x_\pi}$, then computes $e = H(L_{\mathsf{PK}}, I, \mu)$;
3. Alice computes the public key set for ring signature $L_{\mathsf{RPK}} = \{\mathsf{PK}_1 \cdot I^e, \cdots, \mathsf{PK}_n \cdot I^e\} = \{g^{x_1} h^{e x_\pi}, \cdots, g^{x_n} h^{e x_\pi}\}$;
4. Alice runs the ring signature and gets $\tau \leftarrow \mathsf{Rsign}(\mathsf{SK}, \mu, L_{\mathsf{RPK}}, I)$ by usage of L_{RPK} and $\mathsf{SK} = x_\pi$, outputs τ (use generator gh^e);
5. Alice outputs $\sigma = (\tau, \mu, L_{\mathsf{PK}}, I)$ as the SLRS outputs.

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

1. Verifier computes $e^* = H(L_{\mathsf{PK}}, I, \mu)$;
2. Verifier computes $L_{\mathsf{RPK}}^* = \{\mathsf{PK}_1 \cdot I^{e^*}, \cdots, \mathsf{PK}_n \cdot I^{e^*}\}$;
3. Verifier checks the validity of ring signature τ with ring L_{RPK}^* (use generator gh^{e^*});
4. If all passed then outputs 1, otherwise outputs 0.

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

1. For two valid SLRS signatures $\sigma = (\tau, \mu, L_{\mathsf{PK}}, I)$ and $\sigma' = (\tau', \mu', L'_{\mathsf{PK}}, I')$, 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 $\mathsf{PK} = \mathsf{PK}_\pi = g^{x_\pi}$, she can compute $I = h^{x_\pi}$ and $e = H(L_{\mathsf{PK}}, I, \mu)$, then she can compute $L_{\mathsf{RPK}} = \{g^{x_1} h^{e x_\pi}, \cdots, g^{x_n} h^{e x_\pi}\}$. Since $g^{x_\pi} h^{e x_\pi} = (gh^e)^{x_\pi}$, then Alice can use her secret key $\mathsf{SK}_\pi = x_\pi$ to generate the ring signature τ 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). □

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 μ and a list of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$, where $\mathsf{PK}_i = g^{x_i}$ for $i = 1, \cdots, n$, and all PK_is are returned by $\mathcal{JO}$, and $\mathcal{S}$ knows all $\mathsf{SK}_i = x_i$.

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

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

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 $\mathsf{Pr}_{\mathcal{A}}[\pi^* = \pi] = 1/n$.

Then we only need to prove that Game 0 and Game 1 are computational indistinguishable. If 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_π}, h^r) are computational indistinguishable, then $\mathcal{A}$ cannot distinguish h^{x_π} (in Game 0) from h^r (in Game 1). Then we know $\mathcal{A}$ cannot distinguish $\{g^{x_1}h^{ex_\pi}, \cdots, g^{x_n}h^{ex_\pi}\}$ from $\{g^{x_1}h^{er}, \cdots, g^{x_n}h^{er}\}$, then we know Game 0 and Game 1 are computational indistinguishable, then we finish the anonymity proof of SLRS. □

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). □

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 δ, that is, $\mathcal{A}$ returned k valid SLRS signatures $\sigma_i = (\tau_i, \mu_i, L_{\mathsf{PK}}^i, I_i), i = 1, \cdots, k$ (τ_is are the ring signatures), satisfying the following requirements:

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

We first prove a statement that, for a list of users' public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$ returned by $\mathcal{JO}$ with $\mathsf{PK}_i = g^{x_i}$, any PPT adversary $\mathcal{A}$ generates a valid SLRS signature $\sigma \nleftarrow \mathcal{SO}$ if and only if he queries the $\mathcal{CO}$ at least once, except for negligible probability $\epsilon_0 = negl(\lambda)$.

- $\Rightarrow$. If $\mathcal{A}$ gets $\mathsf{SK} = x_i$ from $\mathcal{CO}$, and then $\mathcal{A}$ can run the SLRS signature scheme to generate a valid signature $\sigma = (\tau, \mu, L_{\mathsf{PK}}, I)$.
- $\Leftarrow$. Assume $\mathcal{A}$ did not query the $\mathcal{CO}$ and $\mathcal{SO}$ for $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$ and finished the SLRS signature over $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$ with nonnegligible probability δ_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 $\mathsf{PK}_i = g^{x_i}, i = 1, \cdots, n$, then the probability of $\mathcal{A}$ obtaining any of x_i is $\epsilon_1 = negl(\lambda)$.
  Next, according to the assumption that $\mathcal{A}$ generates a valid signature $\sigma = (\tau, \mu, L_{\mathsf{PK}}, I)$, then he must have finished the ring signature τ (with generator gh^e), where $e = H(L_{\mathsf{PK}}, I, \mu)$. Without loss of generality, we assume $I = g^s h^t$ output by $\mathcal{A}$, then we have $L_{\mathsf{RPK}} = \{g^{x_1}(g^s h^t)^e, \cdots, g^{x_n}(g^s h^t)^e\}$. Since $\mathcal{A}$ finished the ring signature τ with L_{RPK} under generator gh^e, from Lemma 9 we get $\mathcal{A}$ knows $\mathsf{RSK} = z$ for at least one $i \in \{1, \cdots, n\}$ $s.t.$ $g^{x_i}(g^s h^t)^e = (gh^e)^z$, except for negligible probability $\epsilon_2 = negl(\lambda)$. We can also assume that $e = 0$ happens with negligible probability $\epsilon_3 = negl(\lambda)$. 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 $\sigma \nleftarrow \mathcal{SO}$ 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 $\sigma_i = (\tau_i, \mu_i, L_{\mathsf{PK}}^i, I_i)$ for $i = 1, \cdots, k$, then we know there are two SLRS signatures from the same query of $\mathcal{CO}$, saying $\mathsf{SK} = z$ from $\mathsf{PK} = g^z$, 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 $L_{\mathsf{RPK}} = \{g^{x_1}(g^s h^t)^e, \cdots, g^{x_n}(g^s h^t)^e\}$, since $\exists j \in \{1, \cdots, n\}$ $s.t.$ $x_j = z$, and $\mathcal{A}$ signs with 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 $\epsilon_1 = negl(\lambda)$. 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 $\epsilon_2 = negl(\lambda)$, 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. □

3.2.4 Proof of Nonslanderability.

Theorem 11 (Nonslanderability). SLRS is nonslanderable 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_{PK}, a message μ and a public key $\mathsf{PK}_\pi \in L_{\mathsf{PK}}$ to $\mathcal{S}$, $\mathcal{S}$ returns the SLRS signature $\sigma = (\tau, \mu, L_{\mathsf{PK}}, I) \leftarrow \mathsf{SLRS.Sign}(\mathsf{SK}_\pi, L_{\mathsf{PK}}, \mu)$ to $\mathcal{A}$. We assume that $\mathcal{A}$ wins the slandering game with nonnegligible advantage δ, that is, $\mathcal{A}$ successfully outputs a SLRS signature $\sigma^* = (\tau^*, \mu^*, L^*_{\mathsf{PK}}, I^*)$, satisfying the following:

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

From the definition of SLRS.Link, we know that $I^* = I = h^{x_\pi}$, since $\mathsf{PK}_\pi = g^{x_\pi}$ was not queried by $\mathcal{A}$ to $\mathcal{CO}$ and $\mathcal{SO}$, then $\mathcal{A}$ does not know $\mathsf{SK} = x_\pi$ except for negligible probability $\epsilon_0 = negl(\lambda)$ under the hardness of discrete logarithm problems. We know $\mathcal{A}$ successfully produced a ring signature τ^* 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 $\epsilon_1 = negl(\lambda)$, that is, there exists $j \in \{1, \cdots, n\}$, $\mathcal{A}$ knows $x\ s.t.\ \mathsf{PK}^*_j \cdot I^e = (gh^e)^x$ with nonnegligible advantage $\delta - \epsilon_0 - \epsilon_1$, where $e = H(L^*_{\mathsf{PK}}, I, \mu)$. Without loss of generality, we assume $\mathsf{PK}^*_j = g^s h^t$ 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 $\epsilon_2 = negl(\lambda)$. 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 nonslanderability proof of SLRS. □

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. 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.

4.1 MSLRS with AOS

$\mathsf{Par} \leftarrow \mathsf{MSLRS.Setup}(\lambda)$:

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.

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

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 $(\mathsf{PK}, \mathsf{PK}') = (g_1^x, g_2^y)$, and retains $(\mathsf{SK}, \mathsf{SK}') = (x, y)$.

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

1. For a message μ, Alice chooses another $n-1$ users, together with her own public keys, to generate two list of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$ and $L'_{\mathsf{PK}} = \{\mathsf{PK}'_1, \cdots, \mathsf{PK}'_n\}$, where Alice's $\mathsf{PK} = \mathsf{PK}_\pi \in L_{\mathsf{PK}}, \mathsf{PK}' = \mathsf{PK}'_\pi \in L'_{\mathsf{PK}}, \pi \in \{1, \cdots, n\}$, 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 $e = H(L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I, \mu)$;
3. Alice computes the signature public key set for the first ring
$$L_{\mathsf{RPK}} = \{\mathsf{PK}_1 \cdot I^e, \cdots, \mathsf{PK}_n \cdot I^e\}$$
$$= \{g_1^{x_1} h^{ex_\pi}, \cdots, g_1^{x_n} h^{ex_\pi}\} = \{\mathsf{RPK}_i\}_{i=1,\cdots,n};$$
4. Alice samples $r_\pi, t_\pi \in \mathbb{Z}^*_q$ uniformly and computes $e_{\pi+1} = H((g_1 h^e)^{r_\pi}, g_2^{t_\pi}, L_{\mathsf{RPK}}, L'_{\mathsf{PK}}, I, \mu)$;
5. For $i = \pi+1, \cdots, n, 1, \cdots, \pi-1$, Alice samples $z_i, s_i \in \mathbb{Z}^*_q$ uniformly at random, then computes $e_{i+1} = H((g_1 h^e)^{z_i}/(\mathsf{RPK}_i)^{e_i}, g_2^{s_i}/(\mathsf{PK}'_i)^{e_i}, L_{\mathsf{RPK}}, L'_{\mathsf{PK}}, I, \mu)$;
6. Alice computes $z_\pi = r_\pi + xe_\pi, s_\pi = t_\pi + ye_\pi$ and outputs $\tau = (e_1; z_1, \cdots, z_n; s_1, \cdots, s_n)$;
7. Alice outputs $\sigma = (\tau, \mu, L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I)$ as the MSLRS outputs.

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

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

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

1. For two valid MSLRS signatures $\sigma_1 = (\tau_1, \mu_1, L^{(1)}_{\mathsf{PK}}, L'^{(1)}_{\mathsf{PK}}, I_1)$ and $\sigma_2 = (\tau_2, \mu_2, L^{(2)}_{\mathsf{PK}}, L'^{(2)}_{\mathsf{PK}}, I_2)$, 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'

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

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.

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

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 $(\mathsf{PK}, \mathsf{PK}') = (g_1^x, g_2^y)$.

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

1. For a message μ, Alice chooses another $n-1$ users, together with her own public keys, to generate two list of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$ and $L'_{\mathsf{PK}} = \{\mathsf{PK}'_1, \cdots, \mathsf{PK}'_n\}$, where Alice's $\mathsf{PK} = \mathsf{PK}_\pi \in L_{\mathsf{PK}}, \mathsf{PK}' = \mathsf{PK}'_\pi \in L'_{\mathsf{PK}}, \pi \in \{1, \cdots, n\}$, 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 $e = H(L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I, \mu)$;
3. Alice computes the signature public key set for the first ring
$$L_{\mathsf{RPK}} = \{\mathsf{PK}_1 \cdot I^e, \cdots, \mathsf{PK}_n \cdot I^e\}$$
$$= \{g_1^{x_1} h^{e x_\pi}, \cdots, g_1^{x_n} h^{e x_\pi}\} = \{\mathsf{RPK}_i\}_{i=1,\cdots,n};$$
4. Alice samples $c_1, \cdots, c_{\pi-1}, c_{\pi+1}, \cdots, c_n \in \mathbb{Z}_q^*$ uniformly, then Alice samples $\alpha, \beta \in \mathbb{Z}_q^*$ uniformly, computes $R_1 = (g_1 h^e)^\alpha \prod_{i\neq\pi} (\mathsf{RPK}_i)^{c_i}$ and $R_2 = g_2^\beta \prod_{i\neq\pi} (\mathsf{PK}'_i)^{c_i}$;
5. Alice computes $c = H(R_1, R_2, L_{\mathsf{RPK}}, L'_{\mathsf{PK}}, I, \mu)$;
6. 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 $\tau = (z_1, z_2; c_1, \cdots, c_n)$;
7. Alice outputs $\sigma = (\tau, \mu, L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I)$ as the MSLRS' outputs.

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

1. Verifier computes $e^* = H(L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I, \mu)$;
2. Verifier computes $L^*_{\mathsf{RPK}} = \{\mathsf{PK}_1 \cdot I^{e^*}, \cdots, \mathsf{PK}_n \cdot I^{e^*}\}$;
3. Verifier computes $R_1^* = (g_1 h^{e^*})^{z_1} \prod_{i=1}^n (\mathsf{RPK}_i^*)^{c_i}$ and $R_2^* = g_2^{z_2} \prod_{i=1}^n (\mathsf{PK}'_i)^{c_i}$;
4. Verifier checks $\sum_{i=1}^n c_i \stackrel{?}{=} H(R_1^*, R_2^*, L^*_{\mathsf{RPK}}, L'_{\mathsf{PK}}, I, \mu)$;
5. If all passed then outputs 1, otherwise outputs 0.

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

1. For two valid MSLRS' signatures $\sigma_1 = (\tau_1, \mu_1, L^{(1)}_{\mathsf{PK}}, L'^{(1)}_{\mathsf{PK}}, I_1)$ and $\sigma_2 = (\tau_2, \mu_2, L^{(2)}_{\mathsf{PK}}, L'^{(2)}_{\mathsf{PK}}, I_2)$, 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.

It should be emphasized that in this section we only give the performances and comparisons of the linkable ring signature schemes mentioned above, without any consideration of the adaptability in the Monero system. In fact, as discussed in [23], the key-image $I = h^x$ with fixed base h will not be deployed safely in Monero system due to the one-time public key generation algorithm $PK_U = g^{H(PK_v^r)} PK_s$, where (PK_v, PK_s) is the public key of user, and PK_U is the one-time public key of the new UTXO. So both SLRS (SLRS') and MSLRS (MSLRS') cannot be used directly into the Monero system. We will modify both SLRS and MSLRS by adding new key-image with randomized base, to realize the adaptability in Monero system in the appendix C. The detailed implementation and performance of the modified schemes will appeared in the full version of this paper.

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.

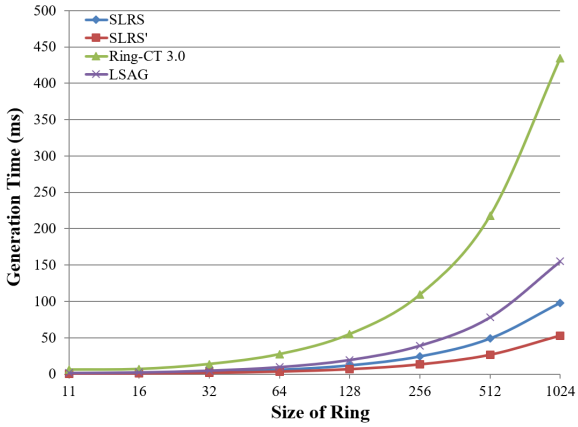

Figure 1: Generation Time of Single Ring.

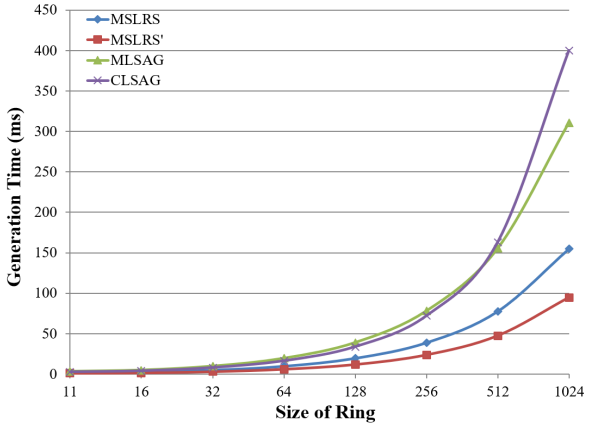

Figure 2: Generation Time of Double Rings.

5.2 Performance of Double Rings

Table 3: Performance of Double Rings

Scheme	n	Generation	Verification	Size
MSLRS(AOS)	11	1.67ms	1.71ms	$(1, mn+1)$
	32	4.83ms	4.84ms	
	128	19.46ms	19.20ms	
	1024	154.79ms	153.50ms	
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	
MLSAG	11	3.26ms	3.31ms	$(1, mn+1)$
	32	9.63ms	9.45ms	
	128	38.79ms	38.60ms	
	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	

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.

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'). At last, both SLRS' and MSLRS' can be adapted to parallel acceleration to further improve the performance.

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	

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.

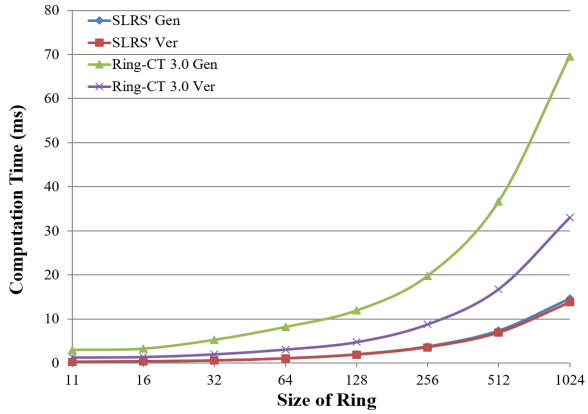

Figure 3: Computation Time under Parallel Acceleration.

[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.

[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.

[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.

[15] Gregory Maxwell and Andrew Poelstra. 2015. Borromean ring signatures. 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 $(x, c, e_1, s_1), \cdots, (x, c, e_k, s_k)$, then there exists an extraction algorithm Ext which can extract a witness $(x, w) \in R$, where c represents the commitment, e_is are challenges and s_is are responses.

A.2 AOS Ring Signature

We give the introduction of AOS ring signature[1] in the following: here we introduce the generalized AOS ring signature for the generator in each position is different (g_i is the generator in the i-th position for $i = 1, \cdots, n$).

- $\mathsf{Par} \leftarrow \mathsf{Setup}(\lambda)$: system chooses an elliptic curve $\mathbb{G}$ and a generator $g_1, \cdots, g_n$ as the public parameters.
- $(\mathsf{PK}_\pi, \mathsf{SK}_\pi) \leftarrow \mathsf{KeyGen}(\mathsf{Par})$: according to the public parameters, user P_π samples $x \in \mathbb{Z}_q^*$ uniformly at random, computes g_π^x and sets $(\mathsf{PK}_\pi, \mathsf{SK}_\pi) = (g_\pi^x, x)$.
- $\sigma \leftarrow \mathsf{Rsign}(\mathsf{SK}_\pi, \mu, L_{\mathsf{PK}})$: when user P_π generates a ring signature for message μ, he chooses another $n-1$ users' public keys, together with his own PK_π to obtain a set of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\} = \{g_1^{x_1}, \cdots, g_n^{x_n}\}$, where $\mathsf{PK}_\pi \in L_{\mathsf{PK}}$ and $\pi \in \{1, \cdots, n\}$, then he does as follows:
  1. P_π samples $r_\pi \in \mathbb{Z}_q^*$ uniformly at random, then computes $c_{\pi+1} = H(g_\pi^{r_\pi}, L_{\mathsf{PK}}, \mu)$;
  2. For $i = \pi+1, \cdots, n, 1, \cdots, \pi-1$, P_π samples $z_i \in \mathbb{Z}_q^*$ uniformly and computes $c_{i+1} = H(g_i^{z_i}/(\mathsf{PK}_i)^{c_i}, L_{\mathsf{PK}}, \mu)$;
  3. P_π computes $z_\pi = r_\pi + xc_\pi$;
  4. Output the ring signature $\sigma = (c_1; z_1, \cdots, z_n)$.
- $1/0 \leftarrow \mathsf{Verify}(\mu, \sigma, L_{\mathsf{PK}})$: for a ring signature $(\mu, L_{\mathsf{PK}}, \sigma)$, for $i = 1, \cdots, n$ the verifier computes

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

  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.

- $\mathsf{Par} \leftarrow \mathsf{Setup'}(\lambda)$: system chooses an elliptic curve $\mathbb{G}$ and a generator g as the public parameters.
- $(\mathsf{PK}_\pi, \mathsf{SK}_\pi) \leftarrow \mathsf{KeyGen'}(\mathsf{Par})$: according to the public parameters, user P_π samples $x \in \mathbb{Z}_q^*$ uniformly at random, computes g^x and sets $(\mathsf{PK}_\pi, \mathsf{SK}_\pi) = (g^x, x)$.

– $\sigma \leftarrow$ Rsign'$(\mathsf{SK}_\pi, \mu, L_{\mathsf{PK}})$: when user P_π generates a ring signature for message μ, he chooses another $n-1$ users' public keys, together with his own PK_π to obtain a set of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$, where $\mathsf{PK}_\pi \in L_{\mathsf{PK}}$ and $\pi \in \{1, \cdots, n\}$, then he does as follows:
  1. For $i = 1, \cdots, \pi-1, \pi+1, \cdots, n$, P_π samples $\alpha, c_i \in \mathbb{Z}_q^*$ uniformly at random, then computes $R = g^\alpha \prod_{i \neq \pi} \mathsf{PK}_i^{c_i}$ and $c = H(R, L_{\mathsf{PK}}, \mu)$;
  2. P_π computes $c_\pi = c - \sum_{i \neq \pi} c_i$;
  3. P_π computes $z = \alpha - x c_\pi$;
  4. Output the ring signature $\sigma = (z; c_1, \cdots, c_n)$.
– $1/0 \leftarrow$ Verify'$(\mu, \sigma, L_{\mathsf{PK}})$: for a ring signature $(\mu, L_{\mathsf{PK}}, \sigma)$, the verifier computes $R^* = g^z \prod_{i=1}^n \mathsf{PK}_i^{c_i}$, then checks $\sum_{i=1}^n c_i \stackrel{?}{=} H(R^*, L_{\mathsf{PK}}, \mu)$, 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:

– $\mathsf{Par} \leftarrow \mathsf{Setup}(\lambda)$: system chooses an elliptic curve $\mathbb{G}$ and generators $g \in \mathbb{G}$, outputs $(\mathbb{G}, q, g)$ as the public parameters.
– $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\mathsf{Par})$:
  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 $\mathsf{PK}_A = g^x, \mathsf{PK}'_A = g^y$.
– $\sigma \leftarrow \mathsf{Rsign}(\mathsf{SK}_{Alice}, \mu, L_{\mathsf{PK}})$:
  1. For a message μ, Alice chooses another $n-1$ users, together with her own public keys, to generate two lists of public keys:
     $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}, L'_{\mathsf{PK}} = \{\mathsf{PK}'_1, \cdots, \mathsf{PK}'_n\}$. Where Alice's $\mathsf{PK}_{Alice} = (\mathsf{PK}_A, \mathsf{PK}'_A) = (\mathsf{PK}_i, \mathsf{PK}'_i) \in L_{\mathsf{PK}}$;
  2. Alice computes $I = H_p(\mathsf{PK}_A)^x$, $D = H_p(\mathsf{PK}_A)^y$, and $e_1 = H(I, D, L_{\mathsf{PK}}, 1)$, $e_2 = H(I, D, L_{\mathsf{PK}}, 2)$;
  3. Alice samples $\alpha \in \mathbb{Z}_q^*$ uniformly at random, computes
  $$L_i = g^\alpha, R_i = H_p(\mathsf{PK}_A)^\alpha,$$
  then computes $c_{i+1} = H(L_i, R_i, \mu, L_{\mathsf{PK}}, I, D)$. Then for $j = i+1, \cdots, i-1$, Alice samples $s_j \in \mathbb{Z}_q^*$ and computes
  $$L_j = g^{s_j} (\mathsf{PK}_j^{e_1} (\mathsf{PK}'_j)^{e_2})^{c_j},$$
  $$R_j = H_p(\mathsf{PK}_j)^{s_j} (I^{e_1} D^{e_2})^{c_j}.$$
  Then computes $c_{j+1} = H(L_j, R_j, \mu, L_{\mathsf{PK}}, I, D)$;
  4. Alice computes $s_i = \alpha - c_i(e_1 x + e_2 y)$;
  5. Alice outputs $\sigma = (c_1, s_1, \cdots, s_n, I, D, L_{\mathsf{PK}})$.
– $1/0 \leftarrow \mathsf{Verify}(c_1, s_1, \cdots, s_n, I, D, L_{\mathsf{PK}})$:
  1. Verifier computes $e_1^* = H(I, D, L_{\mathsf{PK}}, 1)$, $e_2^* = H(I, D, L_{\mathsf{PK}}, 2)$;
  2. For $j = 1, \cdots, n$, verifier computes
  $$L_j^* = g^{s_j} (\mathsf{PK}_j^{e_1^*} (\mathsf{PK}'_j)^{e_2^*})^{c_j^*},$$
  $$R_j^* = H_p(\mathsf{PK}_j)^{s_j} (I^{e_1^*} D^{e_2^*})^{c_j^*}.$$
  Where $c_1 = c_1^*$. Then computes
  $$c_{j+1}^* = H(L_j^*, R_j^*, \mu, L_{\mathsf{PK}}, I, D);$$
  3. Verifier checks whether $c_{n+1}^* \stackrel{?}{=} c_1$;
  4. If all passed then outputs 1, otherwise outputs 0.
– $linked/unlinked \leftarrow \mathsf{Link}(\sigma, \sigma')$: For two CLSAG signatures $\sigma = (c_1, s_1, \cdots, s_n, I, D, L_{\mathsf{PK}})$ and $\sigma' = (c'_1, s'_1, \cdots, s'_n, I', D', L'_{\mathsf{PK}})$, 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.

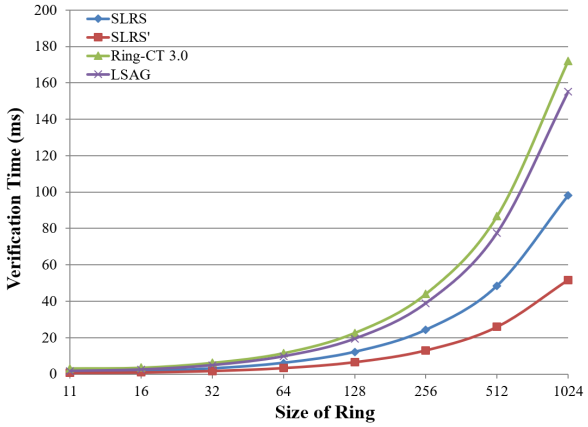

Figure 4: Verification Time of Single Ring.

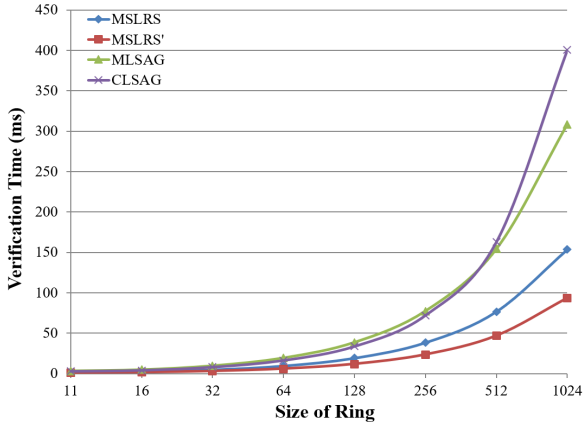

Figure 5: Verification Time of Double Rings.

C MONERO ADAPTABILITY

In Monero system, the public key of every user contains a view key $\mathsf{PK}_v = g^{x_v}$ and a spending key $\mathsf{PK}_s = g^{x_s}$, and the one-time public key generation algorithm of each UTXO is $\mathsf{PK}_U = g^{H(\mathsf{PK}_v^r)}\mathsf{PK}_s$ with the corresponding secret key $\mathsf{SK}_U = H(\mathsf{PK}_v^r) + x_s$. Then if $\mathsf{PK}_U = g^{H(\mathsf{PK}_v^r)}\mathsf{PK}_s$ and $\mathsf{PK}'_U = g^{H(\mathsf{PK}_v^{r'})}\mathsf{PK}_s$ for two UTXOs sent between one initiator and one recipient, then $I = h^{H(\mathsf{PK}_v^r)+x_s}$ and $I' = h^{H(\mathsf{PK}_v^{r'})+x_s}$ with $I/I' = h^{H(\mathsf{PK}_v^r)-H(\mathsf{PK}_v^{r'})}$ known by the initiator, which means that the system is not secure.

So the key-image with fixed base needs to be repaired to achieve adaptability in Monero system. In this section we introduce SLRSM and MSLRSM to fill this gap.

C.1 SLRS For Monero

In the construction of SLRSM, we use similar construction as in SLRS, except for the key-image generation method.

$\mathsf{Par} \leftarrow \mathsf{SLRSM.Setup}(\lambda)$:

1. System chooses an elliptic curve $\mathbb{G}$ with prime order q and a generator $g \in \mathbb{G}$, system outputs $(\mathbb{G}, q, g)$ as the public parameters.

$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{SLRSM.KeyGen}(\mathsf{Par})$:

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

$\sigma \leftarrow \mathsf{SLRSM.Sign}(\mathsf{SK}_\pi, \mu, L_{\mathsf{PK}})$:

1. For a message μ, Alice chooses another $n-1$ users, together with her own public key, to generate a list of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$, where Alice's $\mathsf{PK} = \mathsf{PK}_\pi \in L_{\mathsf{PK}}, \pi \in \{1, \cdots, n\}$;
2. Alice computes $h_i = H_p(\mathsf{PK}_i)$ for $i = 1, \cdots, n$, then computes the key-image $I = h_\pi^{x_\pi}$, and computes $e = H(L_{\mathsf{PK}}, I, \mu)$;
3. Alice computes the public key set for ring signature $L_{\mathsf{RPK}} = \{\mathsf{PK}_1 \cdot I^e, \cdots, \mathsf{PK}_n \cdot I^e\} = \{g^{x_1} h_\pi^{ex_\pi}, \cdots, g^{x_n} h_\pi^{ex_\pi}\}$;
4. Alice runs AOS ring signature and gets $\tau \leftarrow \mathsf{Rsign}(\mathsf{SK}, \mu, L_{\mathsf{RPK}}, I)$ by usage of L_{RPK} and $\mathsf{SK} = x_\pi$, outputs τ (use generator gh_i^e in the i-th position);
5. Alice outputs $\sigma = (\tau, \mu, L_{\mathsf{PK}}, I)$ as the SLRSM outputs.

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

1. Verifier computes $e^* = H(L_{\mathsf{PK}}, I, \mu)$ and computes $h_i = H_p(\mathsf{PK}_i)$ for $i = 1, \cdots, n$;
2. Verifier computes $L^*_{\mathsf{RPK}} = \{\mathsf{PK}_1 \cdot I^{e^*}, \cdots, \mathsf{PK}_n \cdot I^{e^*}\}$;
3. Verifier checks the validity of AOS ring signature τ with ring L^*_{RPK} (use generator $gh_i^{e^*}$ in the i-th position);
4. If all passed then outputs 1, otherwise outputs 0.

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

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

Algorithm 4: SLRSM

C.2 MSLRS For Monero

We can use similar method as in MSLRS (multi-ring extension) to construct position-preserving linkable multi-ring signature form SLRSM. Moreover, we can further modify MSLRS to achieve similar functionality and compactness as CLSAG. In the following we give a 2-ring example of MSLRSM:

$\mathsf{Par} \leftarrow \mathsf{MSLRSM.Setup}(\lambda)$:

1. System chooses an elliptic curve $\mathbb{G}$ with prime order q and generators $g \in \mathbb{G}$, system outputs $(\mathbb{G}, q, g)$ as the public parameters.

$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{MSLRSM.KeyGen}(\mathsf{Par})$:

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

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

1. For a message μ, Alice chooses another $n-1$ users, together with her own public keys, to generate two list of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$ and $L'_{\mathsf{PK}} = \{\mathsf{PK}'_1, \cdots, \mathsf{PK}'_n\}$, where Alice's $\mathsf{PK} = \mathsf{PK}_\pi \in L_{\mathsf{PK}}, \mathsf{PK}' = \mathsf{PK}'_\pi \in L'_{\mathsf{PK}}, \pi \in \{1, \cdots, n\}$, which means the position of Alice's public key in each ring is same;
2. Alice computes $h_i = H_p(\mathsf{PK}_i)$ for $i = 1, \cdots, n$, then computes the key-image $I = h_\pi^{x_\pi}, I' = h_\pi^{y_\pi}$, then computes $e_k = H(L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I, I', \mu, k)$ for $k = 1, 2$;
3. Alice computes the public key set for AOS ring signature
$$L_{\mathsf{RPK}} = \{\mathsf{PK}_1 I^{e_1} (\mathsf{PK}'_1)^{e_2} (I')^{e_1 e_2}, \cdots, \mathsf{PK}_n I^{e_1} (\mathsf{PK}'_n)^{e_2} (I')^{e_1 e_2}\}$$
$$= \{g^{x_1 + e_2 y_\pi} h_\pi^{e_1 x_\pi + e_1 e_2 y_\pi}, \cdots, g^{x_n + e_2 y_\pi} h_\pi^{e_n x_\pi + e_1 e_2 y_\pi}\}$$
$$= \{\mathsf{RPK}_i\}_{i=1,\cdots,n};$$
4. Alice runs the AOS ring signature scheme, and gets $\tau \leftarrow \mathsf{Rsign}(\mathsf{RSK}_\pi, \mu, L_{\mathsf{RPK}}, I, I')$ by usage of L_{RPK} and $\mathsf{RSK}_\pi = \mathsf{SK}_\pi + e_2 \mathsf{SK}'_\pi = x_\pi + e_2 y_\pi$, outputs τ (use generator $gh_i^{e_1}$ in the i-th position);
5. Alice outputs $\sigma = (\tau, \mu, L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I, I')$ as the MSLRSM outputs.

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

1. Verifier computes $h_i^* = H_p(\mathsf{PK}_i)$ for $i = 1, \cdots, n$, then computes $e_k^* = H(L_{\mathsf{PK}}, L'_{\mathsf{PK}}, I, I', \mu, k)$ for $k = 1, 2$;
2. Verifier computes
$$L_{\mathsf{RPK}} = \{\mathsf{PK}_1 I^{e_1^*} (\mathsf{PK}'_1)^{e_2^*} (I')^{e_1^* e_2^*}, \cdots, \mathsf{PK}_n I^{e_1^*} (\mathsf{PK}'_n)^{e_2^*} (I')^{e_1^* e_2^*}\}$$
3. Verifier checks the validity of AOS ring signature τ with ring L^*_{RPK} (use generator $g(h_i^*)^{e_1^*}$ in the i-th position);
4. If all passed then outputs 1, otherwise outputs 0.

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

1. For two valid MSLRSM signatures $\sigma_1 = (\tau_1, \mu_1, L^{(1)}_{\mathsf{PK}}, L'^{(1)}_{\mathsf{PK}}, I_1, I'_1)$ and $\sigma_2 = (\tau_2, \mu_2, L^{(2)}_{\mathsf{PK}}, L'^{(2)}_{\mathsf{PK}}, I_2, I'_2)$, if $I_1 = I_2$ then verifier outputs *linked*, otherwise outputs *unlinked*.

Algorithm 5: MSLRSM