(Linkable) Ring Signature from Hash-Then-One-Way Signature

Xingye Lu
Department of Computing
The Hong Kong Polytechnic University
Hong Kong
xingye.lu@connect.polyu.hk

Man Ho Au *
Department of Computing
The Hong Kong Polytechnic University
Hong Kong
mhaau@polyu.edu.hk

Zhenfei Zhang
Algorand
Boston, USA
zhenfei@algorand.com

***Abstract*—In this paper, we revisit the generic construction of ring signatures from hash-then-one-way type (Type-H) signatures proposed by Abe et al. (AOS) in 2004 from the following aspects. First, we give a proof for the generic construction, in a strengthened security model. Previously, this was only done for concrete instantiations, in a weaker model. Second, we extend AOS's framework to generically construct one-time linkable ring signatures from Type-H signatures and one-time signatures. Lastly, we instantiate the generic construction with an NTRU-based Type-H signature: FALCON and obtain a post-quantum linkable ring signature scheme. Our analysis shows that the resulting linkable signature is more efficient than any existing lattice based solutions for small to moderate number of users.**

I. INTRODUCTION

Introduced by Rivest, Shamir and Tauman in [1], a ring signature allows a signer to endorse a message on behalf of a group of potential signers. It can be regarded as a special type of group signatures without a manager for group management and anonymity revocation. In ring signature schemes, public and secret keys usually are generated by the user; and a group is formed spontaneously by collecting users' public keys. Since there are no revocation mechanisms in ring signatures, a signer can always hides itself within the group.

The strong anonymity provided by ring signature maybe undesirable in some real-word scenarios, such as electronic voting. In e-voting scenario, strong anonymity makes it impossible to discard double votes. Observing this fact, Liu, Wei and Wang [2] proposed the notion of linkable ring signature. In a linkable ring signature scheme, the signer remains anonymous in the group, but two signatures generated by the same signer can be linked. The property, linkability, allows linkable ring signatures to be applicable in various cryptographic constructions, including electronic cash and ad-hoc authentication. At present, linkable ring signature is adapted in cryptocurrency as a significant building block to protect user privacy.

The first ring signature scheme [1] is a generic construction using one-way trapdoor permutation and ideal block cipher as building blocks. Since one-way trapdoor permutation is a relatively rare cryptographic construct, the only known instantiation is based on RSA. In 2004, Abe, Ohkubo and Suzuku (AOS) proposed a new generic construction for ring signatures [3] from digital signatures with specific structures. Specifically, they show how to construct a ring signature scheme from the hash-then-one-way type (Type-H) signatures (such as RSA) and three-move type (Type-T) signatures (discrete log (DL) based schemes such as Schnorr). While in principle similar to the concrete instantiations, security proof for the AOS generic construction is not formally presented.

In 1994, Shor presented a quantum algorithm [4] which can be used to break RSA and DL type digital signatures. Since then, significant effort has been made to develop practical quantum computers. Facing the potential threat from quantum computers, post-quantum cryptography has attracted more and more attentions. Lattice-based cryptography is currently an important approach for post-quantum cryptography. In order to construct post-quantum (linkable) ring signatures, generic constructions such as RST and AOS, with adaptations, have been instantiated to the lattice setting.

The two recent constructions of linkable ring signature [5], [6] are instantiations of the AOS framework based on the lattice-based Type-T signature BLISS [7]. The main difference between these two constructions is the way to achieve linkability. One drawback is that rejection sampling, which will affect the time complexity of signature generation, is needed for the security of the underlying Type-T signature. The recent work from Lu et al. [8] followed a different approach.They presented an adaption of the RST framework which relies on a new cryptographic primitive similar to Chameleon hash instead of a one-way trapdoor permutation. Based on their framework, they present a practical lattice-based linkable ring signature from NTRU lattice. While the size of the signature in the above schemes are linear in ring size, their practical performance compares favourably with signature schemes with logarithmic signature size for small rings. In particular, for a ring size up to 1024, the signature size in Lu et al.'s scheme is still smaller than the logarithmic lattice-based ring signatures. As a reference, the traditional ring signatures employed in current cryptocurrency Monero is about 11.

It is fair to say constructing linkable ring signatures from lattices (even with linear signature size) is non-trivial. Common framework such as AOS does not give concrete security proof; and the RST framework relies on one-way trapdoor permutation of which no lattice-based realisation is known. For the adapted framework from Lu et al. [8], one has to provide construction of their new primitive.

A. Overview

In this paper, we first revisit AOS generic method for ring signatures. While they provide a generic approach to construct ring signatures, in their paper, security proofs are only given to the concrete examples. In other words, if one is to instantiate the AOS generic method from other cryptographic setting, a new security proof is needed. Observing this limitation, we give a security proof for the generic AOS transformation from Type-H signature scheme to ring signature schemes. Moreover, in the original paper, the security of its RSA instantiation is based on the one-wayness of the RSA trapdoor function. In our new proof, we instead rely on the unforgeability of the underlying Type-H signature. Also, different from the unforgeability security model in [3], we use a strengthened model that allows for both a corruption oracle and a signing oracle with adversarially chosen keys.

We then extend AOS framework to its linkable variant. Borrowing the idea from [8], we adopt a one-time signature scheme (Π^{OTS}). We build a generic method of constructing linkable ring signature based on Π^{OTS} and Type-H signature with uniform distributed public key. During the key generation procedure, in addition to the public key and secret key pair $(\mathsf{pk}, \mathsf{sk})$, each signer also generates a pair of public key and secret key $(\mathsf{opk}, \mathsf{osk})$ of a one-time signature. The signer then computes $\mathsf{PK} = \mathsf{pk} \oplus \mathsf{H}(\mathsf{opk})$ for some appropriate hash function $\mathsf{H}(\cdot)$. The new public key is PK and the secret key is $\mathsf{SK} = (\mathsf{sk}, \mathsf{opk}, \mathsf{osk})$.

Suppose a signer with public key PK_π wants to sign a message μ on behalf of a group of signers $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_\ell\}$ ($\pi \in \{1, \cdots, \ell\}$). For each public key PK_i in the group, the signer computes $\mathsf{pk}'_i = \mathsf{PK}_i \oplus \mathsf{H}(\mathsf{opk})$. The signer then obtains a new list of "public keys", $\{\mathsf{pk}'_1, \cdots, \mathsf{pk}'_\ell\}$. Note that, for the signer, pk'_π is equivalent to the original ring signature public key pk_π. The signer then runs the ring signature's signing algorithm with inputs μ, sk_π and $\{\mathsf{pk}'_1, \cdots, \mathsf{pk}'_\ell\}$ to obtain a ring signature σ_R. Next, it signs $\{L_{\mathsf{PK}}, \mu, \sigma_R, \mathsf{opk}_\pi\}$ using the one-time signature scheme. Denote by sig the one-time signature. The linkable ring signature is $\sigma = \{\sigma_R, \mathsf{opk}_\pi, sig\}$.

The verification is similar to the non-linkable version, with an additional step to verify the one-time signature.

Similar to [8], we also instantiate this generic linkable ring signature with NTRU lattice using a NTRU-based Type-H signature scheme: FALCON [9]. Our analysis shows that our scheme gives the best performance in terms of signature sizes. We summarize the results in Table I. Details will be provided in Section V-B.

B. Comparison with Other Lattice-Based (Linkable) Ring Signature Scheme

In this section, we give a brief overview of the difference between this work, Raptor [8] and schemes from [5], [6]. We observe that at a high level, [5], [6] are both instantiations of AOS framework from the lattice-based signature BLISS [7], a Three-move type (Type-T) according to AOS's terminology. There are mainly two drawbacks from the underlying BLISS signature. First of all, for a lattice-based Type-T signature, a technique called rejection sampling is always required to prevent the leakage of secret key from signatures. Thus, ring signatures adopting BLISS require multiple iterations. Secondly, BLISS is vulnerable to side-channel attacks [10] due to the usage of a Gaussian sampler.

TABLE I
COMPARISON OF LATTICE-BASED (LINKABLE) RING SIGNATURE AT SECURITY LEVEL $\lambda = 100$.

	[11]	[6]	[5]	[12]	[8]	Our
Signature size growth	logarithm	linear	linear	logarithm	linear	linear
linkability	×	✓	✓	×	✓	✓
Sig size for 2^6 users	37 MB	649 KB	585 KB	930 KB	82.7 KB	82.0 KB
2^8 users	48.1 MB	2.47 MB	2.34 MB	1132 KB	326.5 KB	318.9 KB
2^{10} users	59.1 MB	9.77 MB	9.36 MB	1409 KB	1301.9 KB	1266.6 KB
2^{12} users	70.2 MB	39 MB	37.4 MB	1492 KB	5203.3 KB	5057.5 KB

Raptor [8] is a new generic framework for (linkable) ring signature based on RST framework. Instead of relying on one-way trapdoor permutation as in the RST framework, [8] firstly introduces a new primitive, named CH+, and then uses CH+ to construct (linkable) ring signature scheme. They also showed how to build CH+ from one-way trapdoor functions. Thus, their work allows instantiations from lattice-based one-way trapdoor function (rather than permutation). In this paper, we focus on the AOS framework with Type-H signature. Note that there are already lattice-based Type-H signatures in the literature. Comparing with Type-T signature, Type-H does not require rejection sampling. Comparing with linkable Raptor, we are able to achieve a slightly smaller signature size under the same assumption.

C. Contribution

We summarize our contributions as follows.

- We present a new security proof for the AOS generic construction of ring signatures from Type-H signature schemes. In the original paper, proofs are only given for concrete instantiations.
- Our proof is in a stronger security model which allows corruptions and a signing oracle with adversarially chosen keys. As a side note, we reduce the security of the generic construction to the unforgeability of the underlying Type-H signature (instead of the one-wayness of the underling trapdoor function), which allows generic constructions from any given Type-H digital signatures.
- We give a generic method of constructing linkable ring signature from Type-H signatures with uniformly distributed public key. We also provide the security proofs for the generic construction based on the security of underlying Type-H signature scheme.
- We instantiate the generic linkable ring signature from NTRU lattice and obtain a post-quantum and efficient linkable ring signature scheme. Our scheme has the shortest signature size when the ring size is reasonably small (i.e., less than 1024).

D. Related Work

We refer readers to [8] for a more in-depth survey on the development of (linkable) ring signatures. Here we briefly

review ring signatures in the lattice setting. While the generic construction from Brakerski and Kalai [13] can be instantiated from lattice, the result is in a weaker security model. Melchor et al. [14] constructs a lattice-based ring signature based on the signature scheme of Lyubashevsky [15]. Libert et al. [11] gives the first log-size lattice-based construction, but the constant is rather large. Another log-size scheme was recently proposed by Esgin et al. [12], which can be thought of as the lattice counterpart of the log-size scheme from [16].

II. Preliminary

A. Notation

Elements in $\mathbb{Z}_q$ are represented by integers in $[-\frac{q}{2}, \frac{q}{2})$. For a ring $\mathcal{R}$ we define $\mathcal{R}_q$ to be the quotient ring $\mathbb{Z}_q[x]/(x^n+1)$ with n being a power of 2 and q being a prime satisfying $q = k \cdot n + 1$ for some integer k. Elements in $\mathcal{R}_q$ are denoted by lower-case bold letters (e.g. $\mathbf{x}$).

For distribution D, $x \leftarrow D$ means sampling x according to distribution D. $\|\mathbf{v}\|_1$ is the ℓ_1 norm of vector $\mathbf{v}$ and $\|\mathbf{v}\|$ is the ℓ_2 norm of $\mathbf{v}$.

The continuous normal distribution over $\mathbb{R}^n$ centered at $\mathbf{v}$ with standard deviation σ is defined as $\rho^n_{\mathbf{v},\sigma}(\mathbf{x}) = (\frac{1}{\sqrt{2\pi\sigma^2}})^n e^{\frac{-\|\mathbf{x}-\mathbf{v}\|^2}{2\sigma^2}}$. For simplicity, when $\mathbf{v}$ is the zero vector, we use $\rho^n_\sigma(\mathbf{x})$.

The discrete normal distribution over $\mathbb{Z}^n$ centered at $\mathbf{v} \in \mathbb{Z}^n$ with standard deviation σ is defined as $D^n_{\mathbf{v},\sigma}(\mathbf{x}) = \frac{\rho^n_{\mathbf{v},\sigma}(\mathbf{x})}{\rho^n_{\mathbf{v},\sigma}(\mathbb{Z}^n)}$.

B. Ring Signatures

We are going to give the syntax and security model for ring signatures in this section.

1) Syntax: A ring signature scheme usually consists of four algorithms (**Setup**, **KeyGen**, **Signing**, **Verification**):

- **Setup**$(1^\lambda) \rightarrow$ param: On input security parameter 1^λ, this algorithm generates system parameter param. We assume param is an implicit input to all the algorithms below.
- **KeyGen**$\rightarrow (\mathsf{sk}, \mathsf{pk})$: This key generation algorithm generates a private signing key sk and a public verification key pk.
- **Signing**$(\mathsf{sk}, \mu, L_{\mathsf{pk}}) \rightarrow \sigma$: On input message μ, a list of user public keys L_{pk}, and signing key sk corresponding to one of the public keys in L_{pk}, the signing algorithm outputs a ring signature σ on μ and L_{pk}.
- **Verification**$(\mu, \sigma, L_{\mathsf{pk}}) \rightarrow$ $accept/reject$: On input message μ, signature σ and a list of user public keys L_{pk}, the verification algorithm outputs $accept$ if σ is a valid signature for μ and L_{pk}; $reject$, otherwise.

Correctness: the scheme is correct if signatures generated according to above specification are always accepted during verification.

2) Security Notions: A secure ring signature scheme should be unforgeable and anonymous. Before going into the details, we first introduce the following oracles which can be used by adversaries in breaking the security of ring signature schemes:

- *Registration Oracle* $\mathcal{RO}(\perp) \rightarrow \mathsf{pk}_i$: Upon request, $\mathcal{RO}$ generates a new user and returns the public key of the new user.
- *Corruption Oracle* $\mathcal{CO}(\mathsf{pk}) \rightarrow \mathsf{sk}$: On input user public key pk that is a query result of $\mathcal{RO}$, $\mathcal{CO}$ returns the corresponding secret key, sk.
- *Signing Oracle* $\mathcal{SO}(\mu, L_{\mathsf{pk}}, \mathsf{pk}_\pi) \rightarrow \sigma$: On input list of user public keys L_{pk}, message μ and the public key of the signer $\mathsf{pk}_\pi \in L_{\mathsf{pk}}$, $\mathcal{SO}$ returns a valid signature σ on μ and L_{pk}.

a) Unforgeability: The unforgeability of a ring signature scheme is defined via the following game, denoted by $\mathsf{Game}_{\mathsf{forge}}$, between adversary $\mathcal{A}$ and challenger $\mathcal{C}$.

- $Setup$. $\mathcal{C}$ runs **Setup**$(1^\lambda) \rightarrow$ param and sends param to $\mathcal{A}$.
- $Query$. $\mathcal{A}$ may query $\mathcal{RO}$, $\mathcal{CO}$ and $\mathcal{SO}$ for a polynomial bounded number of times in an adaptive manner.
- $Output$.Finally, $\mathcal{A}$ outputs a tuple $(\mu^*, \sigma^*, L^*_{\mathsf{pk}})$.

$\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{forge}}$ if: **Verification**$(\mu^*, \sigma^*, L^*_{\mathsf{pk}}) = accept$; $(\mu^*, L^*_{\mathsf{pk}})$ has not been queried to $\mathcal{SO}$; and no public key in L^*_{pk} has been queried to $\mathcal{CO}$.

The advantage of $\mathcal{A}$, denoted by $\mathbf{adv}^{\mathsf{forge}}_{\mathcal{A}}$, is defined by the probability that $\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{forge}}$:

$$\mathbf{adv}^{\mathsf{forge}}_{\mathcal{A}} = \Pr[\mathcal{A} \text{ wins } \mathsf{Game}_{\mathsf{forge}}]$$

Definition 1 (Unforgeability): A ring signature scheme (**KeyGen**, **Signing**, **Verification**) is said to be unforgeable if for any polynomial-time adversary $\mathcal{A}$, $\mathbf{adv}^{\mathsf{forge}}_{\mathcal{A}}$ is negligible.

b) Anonymity: For a ring signature scheme, this notion captures that it is impossible for an adversary to identify the actual signer of a ring signature with probability better than random guessing. The anonymity of a ring signature scheme can be defined by the following game, denoted by $\mathsf{Game}_{\mathsf{anon}}$, between adversary $\mathcal{A}$ and challenger $\mathcal{C}$:

- $Setup$. $\mathcal{C}$ runs **Setup**$(1^\lambda) \rightarrow$ param and sends param to $\mathcal{A}$.
- $Query$. $\mathcal{A}$ may query $\mathcal{RO}$ and $\mathcal{CO}$ in an adaptive manner.
- $Challenge$. $\mathcal{A}$ picks a list of user public keys $L_{\mathsf{pk}} = \{\mathsf{pk}_1, \mathsf{pk}_2, \cdots, \mathsf{pk}_n\}$, a message μ and sends (L_{pk}, μ) to $\mathcal{C}$. $\mathcal{C}$ randomly picks $\pi \in \{1, \cdots, n\}$ and runs **Signing**$(\mathsf{sk}_\pi, \mu, L_{\mathsf{pk}}) \rightarrow \sigma$. $\mathcal{C}$ sends σ to $\mathcal{A}$.
- $Output$. $\mathcal{A}$ outputs a guess $\pi^* \in \{1, \cdots, n\}$.

$\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{anon}}$ if $\pi^* = \pi$. The advantage of $\mathcal{A}$ is defined as

$$\mathbf{adv}^{\mathsf{anon}}_{\mathcal{A}} = |\Pr[\pi^* = \pi] - \frac{1}{n}|.$$

Definition 2 (Anonymity): A ring signature scheme (**KeyGen**, **Signing**, **Verification**) is said to be anonymous (resp. unconditionally anonymous) if for any polynomial-time (resp. unbounded) adversary $\mathcal{A}$, $\mathbf{adv}^{\mathsf{anon}}_{\mathcal{A}}$ is negligible.

C. Linkable Ring Signatures

In this section, we are going to give the syntax and security requirements of linkable ring signatures.

1) Syntax: A linkable ring signature scheme consists of five algorithms, namely, (**Setup**, **KeyGen**, **Signing**, **Verification**, **Link**). The first four are the same with ring signature. For **Link** algorithm:

Link (σ_1, σ_2, μ_1, μ_2, $L_{\mathsf{pk}}^{(1)}$, $L_{\mathsf{pk}}^{(2)}$)$\rightarrow$ $linked/unlinked$: This algorithm takes two tuples $(\sigma_1, \mu_1, L_{\mathsf{pk}}^{(1)})$, $(\sigma_2, \mu_2, L_{\mathsf{pk}}^{(2)})$ as input, output *linked* or *unlinked*.

Correctness. We say the linkable ring signature scheme is correct if signatures generated according to above specification will always be accepted during verification. Furthermore, two signatures generated according to the above specification from the same signer will be linked.

2) Security Notions: We require a linkable ring signature should be unforgeable, anonymous, linkable and nonslanderable. Same as the security notions for ring signatures, there are also $\mathcal{RO}$, $\mathcal{CO}$ and $\mathcal{SO}$ oracles accessible by adversary. We emphasize that due to the property of one-time linkability, each signer can only be queried at most once as an actual signer in $\mathcal{SO}$.

The security definition of unforgeability for linkable ring signatures remains the same as in section II-B2. The definitions of anonymity, linkability and nonslanderability are adopted from Liu et al. [17].

a) Anonymity: For one-time linkable ring signature, it is required that it is impossible for adversary to identify the actual signer from a ring signature with probability better than random guessing.The anonymity of a linkable ring signature scheme can be defined by the following game, $\mathsf{Game}_{\mathsf{anon}}^{*}$, held between adversary $\mathcal{A}$ and challenger $\mathcal{C}$. The difference between $\mathsf{Game}_{\mathsf{anon}}^{*}$ and $\mathsf{Game}_{\mathsf{anon}}$ is that, in $\mathsf{Game}_{\mathsf{anon}}^{*}$, $\mathcal{A}$ is not allowed to obtain signatures on the possible signers in the challenged ring as it will lead to a trivial attack through linking.

- $Setup$. $\mathcal{C}$ runs **Setup** with security parameter 1^{λ} and sends the system parameter param to $\mathcal{A}$.
- $Query$. $\mathcal{A}$ may query $\mathcal{RO}$ in an adaptive manner.
- $Challenge$. $\mathcal{A}$ picks a list of user public keys $L_{\mathsf{pk}} = \{\mathsf{pk}_1, \mathsf{pk}_2, \cdots, \mathsf{pk}_n\}$ and a message μ. All public keys in L_{pk} should be query outputs of $\mathcal{RO}$. $\mathcal{A}$ sends (L_{pk}, μ) to $\mathcal{C}$. $\mathcal{C}$ randomly picks $\pi \in \{1, \cdots, n\}$ and runs **Signing**(sk_π, μ, L_{pk})$\rightarrow \sigma$. $\mathcal{C}$ sends σ to $\mathcal{A}$.
- $Output$. $\mathcal{A}$ outputs a guess $\pi^* \in \{1, \cdots, n\}$.

$\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{anon}}^{*}$ if $\pi^* = \pi$. The advantage of $\mathcal{A}$ is defined as

$$\mathbf{adv}_{\mathcal{A}}^{\mathsf{anon}} = |\Pr[\pi^* = \pi] - \frac{1}{n}|.$$

Definition 3 (Anonymity): A linkable ring signature scheme is said to be anonymous (resp. unconditionally anonymous) if for any polynomial-time adversary (resp. unbounded adversary) $\mathcal{A}$, $\mathbf{adv}_{\mathcal{A}}^{\mathsf{anon}}$ is negligible.

b) Linkability: This notion captures that only two signatures generated by a same signer can be linked. We use the following game, $\mathsf{Game}_{\mathsf{link}}$, between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ to define linkability:

- $Setup$. $\mathcal{C}$ runs Setup and gives $\mathcal{A}$ system parameter param.
- $Query$. $\mathcal{A}$ is given access to $\mathcal{RO}$, $\mathcal{CO}$, $\mathcal{SO}$ and may query the oracles in an adaptive manner.
- $Output$. $\mathcal{A}$ outputs k sets, $\{L_{\mathsf{pk}}^{(i)}, \mu_i, \sigma_i\}$ for $i \in [1, \cdots, k]$, where $L_{\mathsf{pk}}^{(i)}$ is a list of public keys, μ_i is message, σ_i is signature.

$\mathcal{A}$ wins the game if: all σ_is are not query output of $\mathcal{SO}$; all public keys in $L_{\mathsf{pk}}^{(i)}$ are from $\mathcal{RO}$; **Verification**(μ_i, σ_i,$L_{\mathsf{pk}}^{(i)}$) = *Accept* ; $\mathcal{A}$ queried $\mathcal{CO}$ less than k times; and **Link**(σ_i, σ_j, μ_i, μ_j, $L_{\mathsf{pk}}^{(i)}$, $L_{\mathsf{pk}}^{(j)}$) = *unlinked* for $i, j \in [1, \cdots, k]$ and $i \neq j$.

The advantage of $\mathcal{A}$ is defined by the probability $\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{link}}$:

$$\mathbf{adv}_{\mathcal{A}}^{\mathsf{link}} = \Pr[\mathcal{A} \text{ wins } \mathsf{Game}_{\mathsf{link}}]$$

Definition 4 (Linkability): A linkable ring signature scheme is linkable if for any polynomial-time adversary $\mathcal{A}$, $\mathbf{adv}_{\mathcal{A}}^{\mathsf{link}}$ is negligible.

c) Nonslanderability: The nonslanderability requires that a signer cannot frame other honest signers by generating a signature linked to the honest signer. We use the following game, $\mathsf{Game}_{\mathsf{slander}}$, to define the nonslanderability of a linkable ring signature scheme:

- $Setup$. The challenger $\mathcal{C}$ runs **Setup** and gives $\mathcal{A}$ system parameter param.
- $Query$. The adversary $\mathcal{A}$ is given access to $\mathcal{RO}$, $\mathcal{CO}$, $\mathcal{SO}$ and may query the oracles in an adaptive manner.
- $Challenge$. $\mathcal{A}$ gives $\mathcal{C}$ a list of public keys L_{pk}, a message μ and a public key $\mathsf{pk}_\pi \in L_{\mathsf{pk}}$. $\mathcal{C}$ runs **Signing**($\mathsf{sk}, \mu, L_{\mathsf{pk}}$) and returns the corresponding signature σ to $\mathcal{A}$. $\mathcal{A}$ can continue to issue oracle queries.
- $Output$. $\mathcal{A}$ outputs a list of public keys L_{pk}^*, message μ^*, and a signature σ^*.

$\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{slander}}$ if the following holds: **Verification**(μ^*, σ^*,L_{pk}^*) = *accept*; pk_π is not queried by $\mathcal{A}$ to $\mathcal{CO}$; pk_π is not queried by $\mathcal{A}$ as signer to $\mathcal{SO}$; and **Link**(σ, σ^*, μ, μ^*) = *linked*.

The advantage of $\mathcal{A}$ is defined as:

$$\mathbf{adv}_{\mathcal{A}}^{\mathsf{slander}} = \Pr[\mathcal{A} \text{ wins } \mathsf{Game}_{\mathsf{slander}}]$$

Definition 5 (Nonslanderability): A linkable ring signature scheme is nonsladerable if for any polynomial-time adversary $\mathcal{A}$, $\mathbf{adv}_{\mathcal{A}}^{\mathsf{slander}}$ is negligible.

III. AOS Ring Signature Revisited

Recall that AOS's generic method builds ring signature schemes from hash-then-one-way type (Type-H) signature schemes. We first review the concept of (Type-H) digital signatures. Specifically, in a Type-H signature, pk and sk, created from key generation algorithm $\mathcal{G}_{sig}$, are associated with a one-way trapdoor function and its trapdoor respectively. Let F be a trapdoor one-way function and I be its inverse function. For any c from appropriate domain, compute $e = F_{\mathsf{pk}}(c)$ is easy. However, given any e', one cannot find the preimage c' such that $e' = F_{\mathsf{pk}}(c')$ in polynomial time without trapdoor. Secret key sk can be considered as the trapdoor which allows one to efficiently compute one of the preimages of e'. In signing algorithm **Sign**, $\mathsf{H} : \{0,1\}^* \rightarrow \Delta$ is a hash function that hashes message μ and auxiliary information *aux* and I is the inverse function. Domain Δ is supposed to be an abelian group. A Type-H digital signature has the following structure.

Sign(μ, sk)

1 : $c = \mathsf{H}(\mu, aux)$
2 : $s = I_{\mathsf{sk}}(c)$
3 : Return $\sigma = (s, aux)$

Verify$(\sigma, \mu, \mathsf{pk})$

1 : $\sigma \xrightarrow{parsing} (s, aux)$
2 : $c = \mathsf{H}(\mu, aux)$
3 : $e = F_{\mathsf{pk}}(s)$
4 : Return 1 if $c = e$.Otherwise, 0.

A. Ring Signatures from Type-H Digital Signatures

Let $\mathsf{H}_i : \{0,1\}^* \rightarrow \Delta_i$ be a hash function where Δ_i is an abelian group. For $a, b \in \Delta_i$, let $a + b$ denote the group operation and $a - b$ be the group operation with inverse of b. Δ_i depends on pk_i in user public key list L_{pk}.

- **Setup**$(1^\lambda) \rightarrow$ param: On input security parameter 1^λ, this algorithm generates system parameter param which includes hash function H. We assume param is an implicit input to all the algorithms listed below.
- **KeyGen**$\rightarrow (\mathsf{sk}, \mathsf{pk})$: This key generation algorithm generates the key pairs using the key generation function of a signature scheme of his choice $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{G}^{sig}(\mathsf{param})$.
- **Signing**$(\mathsf{sk}_\pi, \mu, L_{\mathsf{pk}}) \rightarrow \sigma$: On input message μ, a list of user public keys $L_{\mathsf{pk}} = \{\mathsf{pk}_1, \cdots, \mathsf{pk}_\ell\}$, and signing key sk_π. the signing algorithm runs as follow:
  $\mathbf{G-1}$ (Initialization): Compute $e_\pi = \beta$, $\beta \leftarrow_\$ \Delta_\pi$. Then compute $c_{\pi+1} = \mathsf{H}_{\pi+1}(L_{\mathsf{pk}}, \mu, e_\pi)$.
  $\mathbf{G-2}$ (Forward Sequence): For $i = \pi + 1, \cdots, \ell$,1,$\cdots,\pi - 1$, compute $e_i = c_i + F_i(s_i, \mathsf{pk}_i)$,where s_i is randomly chosen. Then compute $c_{i+1} = \mathsf{H}_{i+1}(L_{\mathsf{pk}}, \mu, e_i)$.
  $\mathbf{G-3}$ (Forming the Ring): $s_\pi = I_\pi(\beta - c_\pi, sk_\pi)$. Output signature $\sigma = \{c_1, s_1, \cdots, s_\ell\}$ for message μ and public key list L_{pk}.
- **Verification**$(\mu, \sigma, L_{\mathsf{pk}}) \rightarrow$ $accept/reject$: On input message μ, signature σ and list of user public keys $L_{\mathsf{pk}} = \{\mathsf{pk}_1, \cdots, \mathsf{pk}_n\}$, the verification algorithm runs as follow: For $i = 1, \cdots, \ell$, compute $e_i = c_i + F_i(s_i, \mathsf{pk}_i)$ and then computes $c_{i+1} = \mathsf{H}_{i+1}(L_{\mathsf{pk}}, \mu, e_i)$ if $i \neq \ell$. Accept if $c_1 = \mathsf{H}_1(L_{\mathsf{pk}}, \mu, e_\ell)$. Otherwise, reject.

B. Security Analysis

In this section, we prove that the above generic construction is unconditional anonymous and is unforgeable if the underlying signature scheme is unforgeable.

Theorem 1 (Anonymity): AOS ring signature scheme is unconditional anonymous.

Due to space limitation, we roughly sketch the proof below. $\mathcal{RO}$ can be perfectly simulated with properly generated keys. The challenge signature will be created by programming the random oracle without using the corresponding signing key. Specifically, the challenge signature will be of the form $\sigma = \{c_1, s_1, \cdots, s_\ell\}$. In the actual signing algorithm, s_π is generated by $I_\pi(\beta - c_\pi, \mathsf{sk}_\pi)$ while in the simulation, all s_i are sampled according to the output distribution o I_i, and that $e_i = c_i + F_i(s_i, \mathsf{pk}_i)$, $c_{i+1} = \mathsf{H}_{i+1}(L_{\mathsf{pk}}, \mu, e_i)$, with $\mathsf{H}_1(L_{\mathsf{pk}}, \mu, e_\ell)$ programmed to be c_1 in the random oracle model. It is straightforward to prove that the distribution of the simulated signature is the same as a real signature, and that it is independent of the signer's key. Therefore, the probability for adversary to make a successful guess is no more than $\frac{1}{\ell}$, meaning that the scheme is unconditionally anonymous.

Theorem 2 (Unforgeability): AOS framework is unforgeable in random oracle model if the underlying Type-H signature scheme is unforgeable.

Proof 1: Assume there is an adversary $\mathcal{A}$ who can successfully forge a ring signature with probability δ by making at most q_r queries to $\mathcal{RO}$ oracle, q_c queries to $\mathcal{CO}$ oracle, q_s queries to $\mathcal{SO}$ oracle, and q_h queries to all the random oracles H_i. Then we can construct a simulator $\mathcal{S}$ who can break the unforgeability of the underlying Type-H signature scheme with a non-negligible probability.

$\mathcal{S}$ is given a Type-H signature scheme public key pk_c, it is asked to output σ_c such that σ_c is a valid forgery for pk_c. In order to use $\mathcal{A}$ to solve this problem instance, the simulator $\mathcal{S}$ needs to simulate the challenger $\mathcal{C}$ and oracles to play $\mathsf{Game}_{\mathsf{forge}}$ with $\mathcal{A}$. $\mathcal{S}$ runs as follow:
Setup. Simulator $\mathcal{S}$ picks hash functions H_i: $\{0,1\}^* \rightarrow \Delta_i$. H_is will be modelled as random oracles. Assume $\mathcal{A}$ queries random oracles in the form of $Q = (k, L_{\mathsf{pk}}, \mu, e_{k-1})$ where k is a index in L_{pk}. $\mathcal{S}$ returns $\mathsf{H}_k(L_{\mathsf{pk}}, \mu, e_{k-1})$ to $\mathcal{A}$.

$\mathcal{S}$ then randomly picks message μ_c and queries the challenger from the underlying forge game for its hash value (or uses the underlying signature's hash function to compute the hash value). $\mathcal{S}$ receives hash value h'_c.
Oracle Simulation. $\mathcal{S}$ simulates the oracles as follow:

- $\mathcal{RO}(\perp)$: Assume the adversary $\mathcal{A}$ can only queries $\mathcal{RO}$ q_r times ($q_r \geq 1$). $\mathcal{S}$ randomly picks an index $\mathcal{I} \in [1, \cdots, q_r]$. For index $\mathcal{I}$, $\mathcal{S}$ assigns pk_c to index $\mathcal{I}$ as the public key. For other indexes, $\mathcal{S}$ generates the public key and secret key according to **KeyGen**. Upon the jth query, $\mathcal{S}$ returns the corresponding public key.
- $\mathcal{CO}(\mathsf{pk})$: On input a public key pk returned by $\mathcal{RO}$ oracle, $\mathcal{S}$ first checks whether it corresponds to the index $\mathcal{I}$. If yes, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ returns the corresponding secret key sk.
- $\mathcal{SO}(\mu, L_{\mathsf{pk}}, \mathsf{pk}_\pi)$: When $\mathcal{A}$ queries $\mathcal{SO}$ on message μ, a list of public keys $L_{\mathsf{pk}} = \{\mathsf{pk}_1, \cdots, \mathsf{pk}_\ell\}$ and the public key for the signer pk_π where $\mathsf{pk}_\pi \in L_{\mathsf{pk}}$, $\mathcal{S}$ simulates $\mathcal{SO}$ as follow:
  - If $\mathsf{pk}_\pi \neq \mathsf{pk}_\mathcal{I}$, $\mathcal{S}$ runs **Signing**$(\mathsf{sk}_\pi, \mu, L_{\mathsf{pk}})\mathcal{S}$ returns the signature σ to $\mathcal{A}$;
  - If $\mathsf{pk}_\pi = \mathsf{pk}_\mathcal{I}$, $\mathcal{S}$ randomly choose c_1 from its range. For $i \in [1, \cdots, \ell]$ and pk_i, $\mathcal{S}$ samples s_i and computes $e_i = c_i + F_i(s_i, \mathsf{pk}_i)$, $c_{i+1} = \mathsf{H}_{i+1}(L_{\mathsf{pk}}, \mu, e_i)$. $\mathcal{S}$ then programs random oracle as $\mathsf{H}_1(L_{\mathsf{pk}}, \mu, e_\ell) = c_1$. $\sigma = \{c_1, s_1, \cdots, s_\ell\}$.
- *Random Oracle Q*: At beginning of the simulation, $\mathcal{S}$ randomly picks $v, u \leftarrow_\$ [1, \cdots, q_h]$ ($1 \leq v \leq u \leq q_h$). During simulation, if $Q_v = (k+1, L_{\mathsf{pk}}, \mu, e_k)$, $Q_u = (k, L_{\mathsf{pk}}, \mu, e_{k-1})$. $\mathcal{S}$ programs $\mathsf{H}_k(L_{\mathsf{pk}}, \mu, e_{k-1}) = e_k - h'_c$. For other query, if a query input that has already been programmed, $\mathcal{S}$ returns the corresponding output. Otherwise, the output of the random oracle will be randomly sampled from its range. $\mathcal{S}$ will record all

the queries to the random oracle in a table, in case same query is issued twice.

Output. Finally, $\mathcal{A}$ will output a forgery $(\mu^*, \sigma^*, L^*_{\mathsf{pk}})$ with probability δ such that $\mathbf{Verification}(\mu^*, \sigma^*, L^*_{\mathsf{pk}}) = accept$; $(\mu^*, L^*_{\mathsf{pk}})$ has not been queried by $\mathcal{A}$ for signature; and no public key in L^*_{pk} has been input to $\mathcal{CO}$. If $\mathcal{A}$ wants to successfully forge such a ring signature, $\mathcal{A}$ must close a gap in $e^*_t - c^*_t$ for some $\mathsf{pk}_t \in L^*_{\mathsf{pk}}$ by first querying $Q^*_1 = (t+1, L^*_{\mathsf{pk}}, \mu^*, e^*_t)$, then querying $Q^*_2 = (t, L^*_{\mathsf{pk}}, \mu^*, e^*_{t-1})$. The probability for $\mathcal{S}$ successfully guessing $Q_v = Q^*_1$ and $Q_u = Q^*_1$ during random oracle simulation should be at least $\frac{1}{{q_h}^2}$. The probability for $\mathsf{pk}_t = \mathsf{pk}_{\mathcal{I}}$ should be at least $\frac{1}{q_r}$. $\mathcal{S}$ then have $e^*_t - c^*_t = F_t(\mathsf{pk}_t, s^*_t) = F_{\mathcal{I}}(\mathsf{pk}_{\mathcal{I}}, s^*_t)$. Since $Q^*_1 = Q_v$ and $Q^*_2 = Q_u$, we have $e^*_t - c^*_t = h'_c$ where h'_c is the hash output of μ_c. $\mathcal{S}$ outputs (μ_c, s^*_t) as a forgery.The probability for $\mathcal{S}$ to output such a forgery is at least $\frac{\delta}{{q_h}^2 \cdot q_r}$.

IV. Our Generic Ring Signature with Linkability

In this section, we give our generic construction of linkable ring signatures. Our generic construction mainly has two building blocks, namely, a Type-H signature scheme and a one-time signature scheme Π^{OTS}.

Same as AOS ring signature scheme, we have $\mathsf{H}_i : \{0,1\}^* \rightarrow \Delta_i$ being a hash function where range Δ_i is an abelian group. For $a, b \in \Delta_i$, let $a + b$ denote the group operation and $a - b$ be the group operation with inverse of b. Δ_i depends on pk_i in user public key list L_{pk}. Also, the distribution of public key pk of the underlying Type-H signature scheme should be uniformly distributed in its possible range. We emphasize that, for RSA and one-way trapdoor functions in lattice [18], [9], their public keys are all uniformly distributed.

- **Setup**$(1^\lambda) \rightarrow$ param: On input security parameter 1^λ, this algorithm generates system parameter param which includes a hash function H. We assume param is an implicit input to all the algorithms listed below. It also selects a one-time signature scheme $\Pi^{OTS} = \{\mathsf{OKeygen}, \mathsf{OSign}, \mathsf{OVer}\}$
- **KeyGen**$\rightarrow (\mathsf{sk}, \mathsf{pk})$: This key generation algorithm generates the key pairs using the key generation function of a signature scheme of his choice $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{G}^{sig}(\mathsf{param})$. It also generates a pair of Π^{OTS} public key and secret key $(\mathsf{opk}, \mathsf{osk}) \leftarrow \mathsf{OKeygen}(1^\lambda)$ and computes $\mathsf{mk} = \mathsf{H}^*(\mathsf{opk})$. It sets public key $\mathsf{PK} = \mathsf{pk} \oplus \mathsf{mk}$ and secret key $\mathsf{SK} = \{\mathsf{sk}, \mathsf{opk}, \mathsf{osk}\}$.
- **Signing**$(\mathsf{SK}_\pi, \mu, L_{\mathsf{PK}}) \rightarrow \sigma$: On input message μ, a list of user public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_\ell\}$, and signing key $\mathsf{SK}_\pi = \{\mathsf{sk}_\pi, \mathsf{opk}_\pi, \mathsf{osk}_\pi\}$. For $i \in [1, \cdots, \ell]$, compute $\mathsf{pk}_i = \mathsf{PK}_i \oplus \mathsf{mk}_\pi$ where $\mathsf{mk}_\pi = \mathsf{H}^*(\mathsf{opk}_\pi)$. Signer obtains a new public key list $L_{\mathsf{pk}} = \{\mathsf{pk}_1, \cdots, \mathsf{pk}_\ell\}$. The signing algorithm runs as follow:
  $\mathbf{G-1}$ (Initialization): Compute $e_\pi = \beta$ where $\beta \leftarrow_\$ \Delta_\pi$. Then compute $c_{\pi+1} = \mathsf{H}_{\pi+1}(L_{\mathsf{pk}}, \mu, e_\pi)$.
  $\mathbf{G-2}$ (Forward Sequence): For $i = \pi + 1, \cdots, \ell, 1, \cdots, \pi - 1$, compute $e_i = c_i + F_i(s_i, \mathsf{pk}_i)$ where s_i is randomly chosen. Then compute $c_{i+1} = \mathsf{H}_{i+1}(L_{\mathsf{pk}}, \mu, e_i)$.
  $\mathbf{G-3}$ (Forming the Ring): $s_\pi = I_\pi(\beta - c_\pi, sk_\pi)$ Compute one-time signature $sig = \mathsf{OSign}(\mathsf{osk}_\pi; (c_1, s_1, \cdots, s_\ell, L_{\mathsf{PK}}, \mathsf{opk}_\pi))$. Output signature $\sigma = \{c_1, s_1, \cdots, s_\ell, sig, \mathsf{opk}_\pi\}$ for message μ and public key list L_{PK}.
- **Verification**$(\mu, \sigma, L_{\mathsf{PK}}) \rightarrow accept/reject$: On input message μ, signature σ and list of user public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_n\}$. Parse σ to $\{c_1, s_1, \cdots, s_\ell, sig, \mathsf{opk}\}$. For $i \in [1, \cdots, \ell]$, compute $\mathsf{pk}_i = \mathsf{PK}_i \oplus \mathsf{mk}_\pi$ where $\mathsf{mk}_\pi = \mathsf{H}^*(\mathsf{opk})$. The verification algorithm runs as follow: for $i = 1, \cdots, \ell$, compute $e_i = c_i + F_i(s_i, \mathsf{pk}_i)$ and then computes $c_{i+1} = \mathsf{H}_{i+1}(L_{\mathsf{pk}}, \mu, e_i)$ if $i \neq \ell$. Continue if $c_1 = \mathsf{H}_1(L_{\mathsf{pk}}, \mu, e_\ell)$. Otherwise, reject.
  Check whether $\mathsf{OVer}(\mathsf{opk}; (c_1, s_1, \cdots, s_\ell, L_{\mathsf{PK}}, \mathsf{opk})) = 1$. If not,output reject. If all pass, output accept.
- **Link**$(\sigma_1, \sigma_2, \mu_1, \mu_2, L^{(1)}_{\mathsf{PK}}, L^{(2)}_{\mathsf{PK}}, \rightarrow$ *linked/unlinked*: On input two message signature pairs (μ_1, σ_1) and (μ_2, σ_2), this algorithm first checks the validity of signatures σ_1 and σ_2. If **Verification**$(\mu_1, \sigma_1, L^{(1)}_{\mathsf{PK}}) \rightarrow$ *accept* and **Verification**$(\mu_2, \sigma_2, L^{(2)}_{\mathsf{PK}}) \rightarrow$ *accept*, it parses σ_1=$\{c^{(1)}_1, s^{(1)}_1, \cdots, s^{(1)}_\ell, \mathsf{opk}_1, sig_1\}$ and $\sigma_2 = \{c^{(2)}_1, s^{(2)}_1, \cdots, s^{(2)}_\ell, \mathsf{opk}_2, sig_2\}$. The algorithm outputs *linked* if $\mathsf{opk}_1 = \mathsf{opk}_2$. Otherwise, output *unlinked*.

A. Security Analysis

Theorem 3 (Linkability): The linkable ring signature is linkable in random oracle model if the underlying Type-H signature scheme and Π^{OTS} are unforgeable.

Proof 2: Let $\mathcal{A}$ be an adversary who can successfully forge a linkable ring signature with probability δ by making at most q_r queries to $\mathcal{RO}$ oracle, q_c queries to $\mathcal{CO}$ oracle, q_s queries to $\mathcal{SO}$ oracle, and q^*_h queries to random oracle H^*, q_h queries to all the random oracles H_i. We show how to construct simulator $\mathcal{S}$ who can break unforgeability of the underlying Type-H signature scheme with a non-negligible probability.

Given public key pk_c of a Type-H signature scheme, $\mathcal{S}$'s task is to output a forged signature σ_c on any message of its choice. In order to use $\mathcal{A}$ to solve this problem instance, the simulator $\mathcal{S}$ needs to simulate the challenger $\mathcal{C}$ and oracles to play $\mathsf{Game}_{\mathsf{forge}}$ with $\mathcal{A}$. $\mathcal{S}$ runs as follow: *Setup*. $\mathcal{S}$ picks hash functions H^*, H_is and sets as system parameter. H^*, H_is will be modeled as random oracle. For H_i, $\mathcal{A}$ queries it in the form of $Q(k, L_{\mathsf{PK}}, \mu, e_{k-1})$. $\mathcal{S}$ returns $\mathsf{H}_k(L_{\mathsf{PK}}, \mu, e_{k-1})$ to $\mathcal{A}$.For H^*, $\mathcal{S}$ picks $\{h^*_1, h^*_2, \cdots, h^*_{p^*}\} \leftarrow_\$ \Delta^*_i$ as the q^*_h responses of random oracle H^*.$\mathcal{S}$ then random picks a message μ_c and queries the challenger from the forge game of the underlying signature scheme for its hash value (or compute the hash value by the given hash function). $\mathcal{S}$ receives hash value h'_c.
Oracle Simulation. $\mathcal{S}$ simulates the oracles as follow:

- $\mathcal{RO}(\perp)$: Assume adversary $\mathcal{A}$ can only queries $\mathcal{RO}$ q_r times ($q_r \geq 1$). $\mathcal{A}$ random picks an index $\mathcal{I} \leftarrow_\$ [1, \cdots, q_r]$. For index $\mathcal{I}$, $\mathcal{S}$ sets $\mathsf{PK}_{\mathcal{I}} = \mathsf{pk}_c \oplus h^*_{\mathcal{Q}}$ where $h^*_{\mathcal{Q}} \leftarrow_\$ \{h^*_1, h^*_2, \cdots, h^*_{p^*}\}$. For other index, $\mathcal{S}$ samples PK uniformly random from its possible range. Upon the jth query, $\mathcal{S}$ returns the corresponding public key.
- $\mathcal{CO}(\mathsf{PK})$: On input a public key PK returned by $\mathcal{RO}$ oracle, $\mathcal{S}$ first checks whether it corresponds to index

$\mathcal{I}$. If yes, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ runs $\mathsf{OKeygen}(1^\lambda) \rightarrow (\mathsf{opk}, \mathsf{osk})$. $\mathcal{S}$ runs $\mathcal{G}^{sig}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$. $\mathcal{S}$ returns $(\mathsf{sk}, \mathsf{opk}, \mathsf{osk})$ as secret key and programs $\mathsf{H}^*(\mathsf{opk}) = \mathsf{PK} \oplus \mathsf{pk}$.

- $\mathcal{SO}(\mu, L_{\mathsf{PK}}, \mathsf{PK}_\pi)$: When $\mathcal{A}$ queries $\mathcal{SO}$ on message μ, a list of public keys $L_{\mathsf{PK}} = \{\mathsf{PK}_1, \cdots, \mathsf{PK}_\ell\}$ and the public key for the signer PK_π where $\mathsf{PK}_\pi \in L_{\mathsf{PK}}$, $\mathcal{S}$ simulates $\mathcal{SO}$ as follow:
  1). If PK_π has been queried to $\mathcal{CO}$,$\mathcal{S}$ runs **Signing**$(\mathsf{sk}_\pi, \mu, L_{\mathsf{PK}})$ where the output of the random oracle H_i will be . $\mathcal{S}$ returns the signature σ to $\mathcal{A}$;
  2). If $\mathsf{PK}_\pi = \mathsf{PK}_\mathcal{I}$, $\mathcal{S}$ runs $\mathsf{OKeygen}(1^\lambda) \rightarrow (\mathsf{opk}_\pi, \mathsf{osk}_\pi)$ and randomly samples a hash value to $\mathsf{H}^*(\mathsf{opk}_\pi)$. $\mathcal{S}$ computes $\mathsf{pk}_i = \mathsf{H}^*(\mathsf{opk}_\pi) \oplus \mathsf{PK}_i$ for $i \in [1, \cdots, \ell]$. $\mathcal{S}$ randomly choose c_1 from its range. For $i \in [1, \cdots, \ell]$, $\mathcal{S}$ samples s_i and computes $e_i = c_i + F_i(s_i, \mathsf{pk}_i)$, $c_{i+1} = \mathsf{H}_{i+1}(L_{\mathsf{pk}}, \mu, e_i)$. $\mathcal{S}$ then programs random oracle as $\mathsf{H}_1(L_{\mathsf{pk}}, \mu, e_\ell) = c_1$. $\mathcal{S}$ also computes one-time signature $sig = \mathsf{OSign}(\mathsf{osk}_\pi;\ s_1,\ \cdots,\ s_\ell, L_{\mathsf{PK}}, \mathsf{opk})$. $\mathcal{S}$ returns signature $\sigma = \{\ s_1,\ \cdots,\ s_\ell,\ \mathsf{opk}_\pi,\ sig\}$;
  3). for other PK, $\mathcal{S}$ runs $\mathsf{OKeygen}(1^\lambda) \rightarrow (\mathsf{opk}, \mathsf{osk})$. $\mathcal{S}$ runs $\mathcal{G}^{sig}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$. $\mathcal{S}$ returns $(\mathsf{sk}, \mathsf{opk}, \mathsf{osk})$ as secret key and programs $\mathsf{H}^*(\mathsf{opk}) = \mathsf{PK} \oplus \mathsf{pk}$. $\mathcal{S}$ runs **Signing**$(\mathsf{sk}_\pi, \mu, L_{\mathsf{PK}})$ where the output of the random oracle will be programmed as the first $h_t^i \in \{h_1^{(i)}, h_2^{(i)}, \cdots, h_{p_i}^{(i)}\}$ that has not been used yet. $\mathcal{S}$ returns the signature σ to $\mathcal{A}$
- *Random Oracle* H^*: For query input that has already been programmed, $\mathcal{S}$ returns the corresponding output. Otherwise, the output of the random oracle will be the first $h_i^* \in \{h_1^*, h_2^*, \cdots, h_{p^*}^*\}$ that has not been used yet. $\mathcal{S}$ will record all the queries to the random oracle in a table, in case same query is issued twice.
- *Random Oracle* Q: At beginning of the simulation, $\mathcal{S}$ randomly picks $v, u \leftarrow_\$ [1, \cdots, q_h]$ $(1 \le v \le u \le q_h)$. During simulation, if $Q_v = (k+1, L_{\mathsf{PK}}, \mu, e_k)$, $Q_u = (k, L_{\mathsf{PK}}, \mu, e_{k-1})$. $\mathcal{S}$ programs $\mathsf{H}_k(L_{\mathsf{PK}}, \mu, e_{k-1}) = e_k - h_c'$. For other query, if a query input that has already been programmed, $\mathcal{S}$ returns the corresponding output. Otherwise, the output of the random oracle will be randomly sampled from its range. $\mathcal{S}$ will record all the queries to the random oracle in a table, in case same query is issued twice.

Output. Adversary $\mathcal{A}$ outputs k sets $\{L_{\mathsf{PK}}^{(i)},\ \mu_i,\ \sigma_i\}$ for $i \in [1, \cdots, k]$. These k sets should satisfy that **Verification**$(\mu_i, \sigma_i, L_{\mathsf{PK}}^{(i)}) = accept$; $\mathcal{A}$ queried $\mathcal{CO}$ less than k times; and **Link**$(\sigma_i, \sigma_j, \mu_i, \mu_j, L_{\mathsf{PK}}^{(i)}, L_{\mathsf{PK}}^{(j)}) = unlinked$ for $i \neq j$ and $i, j \in [1, \cdots, k]$. Since $\mathcal{A}$ is allowed query $\mathcal{CO}$ less than k times. At least one of the output signatures should be generated from the opk that $\mathcal{A}$ does not obtain from $\mathcal{CO}$ or $\mathcal{SO}$. If any of the opk from the returned signatures are obtained from $\mathcal{SO}$, $\mathcal{A}$ breaks the unforgeability of Π^{OTS}. Assume σ_j, $j \in \{1, \cdots, k\}$ is not produced by the opk obtaining from $\mathcal{CO}$ or $\mathcal{SO}$. $\mathsf{H}^*(opk_j) \neq h_\mathcal{Q}^*$, abort.

The probability for $\mathsf{H}^*(opk_j) = h_\mathcal{Q}^*$ is no less than $\frac{1}{q_h^*}$. In the following we use $(\mu^*, \sigma^*, L_{\mathsf{PK}}^*)$ to denote $(\mu^j, \sigma^j, L_{\mathsf{PK}}^{(j)})$. If $\mathcal{A}$ wants to successfully forge such a linkable ring signature, $\mathcal{A}$ must close a gap in $e_t^* - c_t^*$ by first querying $Q_1^* = (t+1, L_{\mathsf{PK}}^*, \mu^*, e_t^*)$, then querying $Q_2^* = (t, L_{\mathsf{PK}}^*, \mu^*, e_{t-1}^*)$. The probability for $\mathcal{S}$ successfully guessing $Q_v = Q_1^*$ and $Q_u = Q_1^*$ during random oracle simulation should be at least $\frac{1}{{q_h}^2}$. The probability for $\mathsf{PK}_t = \mathsf{PK}_\mathcal{I}$ should be at least $\frac{1}{q_r}$. Since $\mathsf{H}^*(opk^*) = h_\mathcal{Q}^*$, we have $\mathsf{H}^*(opk^*) \oplus \mathsf{PK}_\mathcal{I} = \mathsf{pk}_c$. $\mathcal{S}$ then have $e_t^* - c_t^* = F_t(\mathsf{pk}_c, s_t^*)$. Since $Q_1^* = Q_v$ and $Q_2^* = Q_u$, we have $e_t^* - c_t^* = h_c'$ where h_c' is the hash output of μ_c. $\mathcal{S}$ outputs (μ_c, s_t^*) as a forgery.The probability for $\mathcal{S}$ to output such a forgery is at least $\frac{\delta}{{q_h}^2 \cdot q_h^* \cdot q_r}$.

Due to page limitation, we omit the proof of the other properties, namely, unforgeability, anonymity and nonslanderability, which will be available in the full version of the paper.

V. Instantiations from NTRU

In this section, we are going to instantiate the linkable ring signature to NTRU lattice using a NTRU based Type-H signature scheme: Falcon. In 2008, Gentry, Peikert and Vaikuntanathan [18] construct a one-way trapdoor function using 'hard' basis and 'good' basis of a lattice and apply it to construct a lattice-based Type-H signature schemes. Prest at al. [19], [9] then use NTRU lattices [20] to instantiate the GPV construction. The corresponding NTRU-based Type-H signature scheme is named Falcon [9]. Falcon is a candidate lattice-based signature scheme to the NIST post-quantum standardization process[1].

A. Linkable Ring Signature from Falcon

Falcon signature scheme is a Type-H signature scheme. Thus, we can apply the generic method of transforming Type-H signature scheme to linkable ring signature scheme to Falcon.

Setup$(1^\lambda) \rightarrow \mathsf{param}$: On input security parameter 1^λ, this algorithm chooses H^*, $\mathsf{H}_i : \{0,1\}^* \rightarrow R_q$, $D_\mathbf{b}$, $D_\mathbf{r}$ and η.

KeyGen$\rightarrow (\mathsf{sk}, \mathsf{pk})$: This algorithm firstly generates

- $(\mathbf{a}, \mathbf{f}, \mathbf{g}, \bar{\mathbf{f}}, \bar{\mathbf{g}}) \leftarrow$ Falcon.KeyGen(param), and
- $(\mathbf{a}_{ots}, \mathbf{f}_{ots}, \mathbf{g}_{ots}, \bar{\mathbf{f}}_{ots}, \bar{\mathbf{g}}_{ots}) \leftarrow$ Falcon.KeyGen(param)

Then it sets $\mathbf{a}' := \mathbf{a} + \mathsf{H}^*(\mathbf{a}_{ots}) \bmod q$. The public key $\mathsf{pk} = \{\mathbf{a}'\}$ and secret key $\mathsf{sk} = \{\mathsf{sk}^0 = \{\mathbf{f}, \mathbf{g}, \bar{\mathbf{f}}, \bar{\mathbf{g}}\}, \mathsf{sk}^1 = \{\mathbf{f}_{ots}, \mathbf{g}_{ots}, \bar{\mathbf{f}}_{ots}, \bar{\mathbf{g}}_{ots}\}, \mathbf{a}_{ots}\}$.

Signing$(\mathsf{sk}_\pi, \mu, L_{\mathsf{pk}}, \mathsf{param}) \rightarrow \sigma$: On input message μ, list of user public keys $L_{\mathsf{pk}} = \{\mathsf{pk}_1, \cdots, \mathsf{pk}_\ell\}$, and signing key $\mathsf{sk}_\pi = \{\mathsf{sk}_\pi^0 = \{\mathbf{f}_\pi, \mathbf{g}_\pi, \bar{\mathbf{f}}_\pi, \bar{\mathbf{g}}_\pi\}, \mathsf{sk}_\pi^1 = \{\mathbf{f}_{ots}, \mathbf{g}_{ots}, \bar{\mathbf{f}}_{ots}, \bar{\mathbf{g}}_{ots}\ \}, \mathbf{a}_{ots}\}$ of $\mathsf{pk}_\pi = \{\mathbf{a}_\pi'\}$, and the system parameter param, the signing algorithm runs as follow:

1) for $i \in [1, \cdots, \ell]$, compute $\mathbf{a}_i = \mathbf{a}_i' - \mathsf{H}^*(\mathbf{a}_{ots}) \bmod q$. Signer then obtains a new list $L = \{\mathbf{a}_1, \cdots, \mathbf{a}_\ell\}$.
2) it then randomly samples a polynomial $\mathbf{e}_\pi \leftarrow_\$ R_q$ and computes $\mathbf{c}_{\pi+i} = \mathsf{H}_{\pi+1}(L, \mu, \mathbf{e}_\pi)$;
3) for $i = \pi+1, \cdots, \ell, 1, \cdots, \pi-1$, compute $\mathbf{e}_i = \mathbf{c}_i + \mathbf{x}_{i,0} + \mathbf{a}_i \mathbf{x}_{i,1}$ where $\mathbf{x}_i = (\mathbf{x}_{i,0}, \mathbf{x}_{i,1}) \leftarrow \mathcal{D}_{\mathcal{R},\eta}^2$ and $\mathbf{c}_{i+1} = \mathsf{H}_{i+1}(L, \mu, \mathbf{e}_i)$;

[1] https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization

4) compute $(\mathbf{x}_{\pi,0}, \mathbf{x}_{\pi,1}) = \text{FALCON.sign}(\mathsf{sk}_\pi^0; \mathbf{e}_\pi - \mathbf{c}_\pi)$ such that $\mathbf{x}_{\pi,0} + \mathbf{a}_\pi \mathbf{x}_{\pi,1} = \mathbf{e}_\pi - \mathbf{c}_\pi$;
5) it then generates a signature *sig* on $\mu' = \{\{\mathbf{x}_i\}_{i=1}^{i}, \mathbf{c}_1, L_{\mathsf{pk}}, \mathbf{a}_{ots}\}$ by computing *sig*=$\text{FALCON.sign}(\mathsf{sk}_\pi^1; \mu')$.

The linkable ring signature of μ and L_{pk} is $\sigma = \{\{\{\mathbf{x}_i\}_{i=1}^{\ell}, \mathbf{c}_1, \mathbf{a}_{ots}, sig\}$.

Verification$(\mu, \sigma, L_{\mathsf{pk}}) \rightarrow accept/reject$: On input message μ, signature σ and a list of user public keys L_{pk}, the verification algorithm performs as follows:

1) parses $\sigma = \{\mathbf{x}_1, \cdots, \mathbf{x}_\ell, \mathbf{c}_1, \mathbf{a}_{ots}, sig\}$;
2) For $i \in [1, \cdots, \ell]$, compute $\mathbf{a}_i = \mathbf{a}'_i - \mathsf{H}^*(\mathbf{a}_{ots}) \bmod q$;
3) checks whether for all $i \in [1, \cdots, \ell]$, $\|\mathbf{x}_i\|_\infty \leq \beta$; outputs $reject$ if not;
4) for all $i \in [1, \cdots, \ell]$, computes $\mathbf{y}_i = \mathbf{x}_{i,0} + \mathbf{a}_i \mathbf{x}_{i,1}$, $\mathbf{e}_i = \mathbf{c}_i + \mathbf{y}_i$. Then compute $\mathbf{c}_{i+1} = \mathsf{H}_{i+1}(L, \mu, \mathbf{e}_i)$ if $i \neq \ell$. Continue if $\mathbf{c}_1 = \mathsf{H}_1(L, \mu, \mathbf{e}_\ell)$. Otherwise, reject.
5) verify whether sig is a FALCON signature for $(\{\mathbf{x}_i\}_{i=1}^{\ell}, \mathbf{c}_1, \{\mathbf{a}'_i\}_{i=1}^{\ell}, \mathbf{a}_{ots})$ with public key $\mathbf{a}_{ots}$; outputs $reject$ if fails.
6) outputs $accept$.

Link$(\sigma_1, \sigma_2, \mu_1, \mu_2, L_{\mathsf{PK}}^{(1)}, L_{\mathsf{PK}}^{(2)}, \rightarrow$ *linked/unlinked*: On input two message signature pairs (μ_1, σ_1) and (μ_2, σ_2), this algorithm first checks the validity of signatures σ_1 and σ_2. If $\mathsf{Vf}(\mu_1, \sigma_1, L_{\mathsf{PK}}^{(1)}) \rightarrow$ *accept* and $\mathsf{Vf}(\mu_2, \sigma_2, L_{\mathsf{PK}}^{(2)}) \rightarrow$ *accept*, it parses $\sigma_1 = \{\{\{\mathbf{x}_i^{(1)}\}_{i=1}^{\ell}, \mathbf{c}^{(1)}, \mathbf{a}_{ots}^1, sig_1\}$ and $\sigma_2 = \{\{\{\mathbf{x}_i^{(2)}\}_{i=1}^{\ell}, \mathbf{c}^{(2)}, \mathbf{a}_{ots}^2, sig_2\}$. The algorithm outputs *linked* if $\mathbf{a}_{ots}^1 = \mathbf{a}_{ots}^2$. Otherwise, output *unlinked*.

Note that in this implementation we use additions and subtractions over the $\mathcal{R}_q$ instead of bit-wise XOR operations. Under the random oracle model $\mathcal{H}_1(\mathbf{a}_{ots})$ will output a random ring element. This creates a perfect one-time mask that assures $\mathbf{a}'$ is indistinguishable from random.

B. Efficiency Analysis

Here we give some estimated performance of instantiating generic construction with FALCON-512. The estimated linkable signature size is around $617 \times 2(\ell + 1) + 2 * 897 \approx 1.23(\ell + 1) + 2 * 0.897$ kilo bytes, where ℓ is the number of users in a signature. For a signature of FALCON-512, the size is around 2×617 bytes. Besides, 897 bytes is the size of a ring element in FALCON-512.

We give comparison with some other (linkable) ring signature scheme in Table I. Comparing with other linear-size linkable ring signature, we have the smallest signature size. Comparing with scheme with logarithmic signature size, our linkable ring signature scheme has a better performance with a small ring (ring size $\leq 2^{10}$). Thus, our linkable ring signature is practical and can be applied in scenarios with small groups.

VI. CONCLUSION

We presented a new generic construction of linkable ring signatures from Type-H digital signatures and proved its security in the random oracle model. We instantiated our framework with NTRU lattice. The resulting scheme outperforms state-of-the-art post-quantum constructions when the ring size is reasonably small.

REFERENCES

[1] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565.

[2] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," in *Information Security and Privacy*, H. Wang, J. Pieprzyk, and V. Varadharajan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 325–335.

[3] M. Abe, M. Ohkubo, and K. Suzuki, "1-out-of-n signatures from a variety of keys," in *Advances in Cryptology — ASIACRYPT 2002*, Y. Zheng, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 415–432.

[4] P. W. Shor, "Polynominal time algorithms for discrete logarithms and factoring on a quantum computer," in *Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings*, 1994, p. 289. [Online]. Available: https://doi.org/10.1007/3-540-58691-1_68

[5] C. Baum, H. Lin, and S. Oechsner, "Towards practical lattice-based one-time linkable ring signatures," in *Information and Communications Security*, D. Naccache, S. Xu, S. Qing, P. Samarati, G. Blanc, R. Lu, Z. Zhang, and A. Meddahi, Eds. Cham: Springer International Publishing, 2018, pp. 303–322.

[6] W. A. Alberto Torres, R. Steinfeld, A. Sakzad, J. K. Liu, V. Kuchta, N. Bhattacharjee, M. H. Au, and J. Cheng, "Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice ringct v1.0)," in *Information Security and Privacy*, W. Susilo and G. Yang, Eds. Cham: Springer International Publishing, 2018, pp. 558–576.

[7] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 40–56.

[8] X. Lu, M. H. Au, and Z. Zhang, "Raptor: A practical lattice-based (linkable) ring signature," Cryptology ePrint Archive, Report 2018/857, 2018, https://eprint.iacr.org/2018/857.

[9] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU." [Online]. Available: http://www.di.ens.fr/~prest/Publications/falcon.pdf

[10] T. Espitau, P.-A. Fouque, B. Gerard, and M. Tibouchi, "Side-channel attacks on bliss lattice-based signatures – exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers," Cryptology ePrint Archive, Report 2017/505, 2017, https://eprint.iacr.org/2017/505.

[11] B. Libert, S. Ling, K. Nguyen, and H. Wang, "Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 1–31.

[12] M. F. Esgin, R. Steinfeld, A. Sakzad, J. K. Liu, and D. Liu, "Short lattice-based one-out-of-many proofs and applications to ring signatures," Cryptology ePrint Archive, Report 2018/773, 2018, https://eprint.iacr.org/2018/773.

[13] Z. Brakerski and Y. T. Kalai, "A framework for efficient signatures, ring signatures and identity based encryption in the standard model," Cryptology ePrint Archive, Report 2010/086, 2010, https://eprint.iacr.org/2010/086.

[14] C. Aguilar Melchor, S. Bettaieb, X. Boyen, L. Fousse, and P. Gaborit, "Adapting lyubashevsky's signature schemes to the ring signature setting," in *Progress in Cryptology – AFRICACRYPT 2013*, A. Youssef, A. Nitaj, and A. E. Hassanien, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–25.

[15] V. Lyubashevsky, "Lattice signatures without trapdoors," in *Advances in Cryptology – EUROCRYPT 2012*, D. Pointcheval and T. Johansson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 738–755.

[16] J. Groth and M. Kohlweiss, "One-out-of-many proofs: Or how to leak a secret and spend a coin," in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 253–280.

[17] J. K. Liu, M. H. Au, W. Susilo, and J. Zhou, "Linkable ring signature with unconditional anonymity," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 157–165, Jan 2014.

[18] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Proceedings of the 40th*

annual ACM symposium on Theory of computing, ser. STOC '08. New York, NY, USA: ACM, 2008, p. 197–206.

[19] L. Ducas, V. Lyubashevsky, and T. Prest, "Efficient identity-based encryption over ntru lattices," in *Advances in Cryptology – ASIACRYPT 2014*, P. Sarkar and T. Iwata, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 22–41.

[20] J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem," in *Algorithmic Number Theory*, J. P. Buhler, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 267–288.