Efficient Linkable Ring Signatures: New Framework and Post-Quantum Instantiations

Yuxi Xue[1], Xingye Lu[1✉], Man Ho Au[1], and Chengru Zhang[2]

[1] The Hong Kong Polytechnic University, Hong Kong
yuxi-ivy.xue@connect.polyu.hk
{xing-ye.lu,mhaau}@polyu.edu.hk
[2] The University of Hong Kong, Hong Kong
u3008875@connect.hku.hk

Abstract. In this paper, we introduce a new framework for constructing linkable ring signatures (LRS). Our framework is based purely on signatures of knowledge (SoK) which allows one to issue signatures on behalf of any NP-statement using the corresponding witness. Our framework enjoys the following advantages: (1) the security of the resulting LRS depends only on the security of the underlying SoK; (2) the resulting LRS naturally supports online/offline signing (resp. verification), where the output of the offline signing (resp. verification) can be re-used across signatures of the same ring. For a ring size n, our framework requires an SoK of the NP statement with size $\log n$.

To instantiate our framework, we adapt the well-known post-quantum secure non-interactive argument of knowledge (NIAoK), ethSTARK, into an SoK. This SoK is inherently post-quantum secure and has a signature size poly-logarithmic in the size of the NP statement. Thus, our resulting LRS has a signature size of $O(\text{polylog}(\log n))$. By comparison, existing post-quantum ring signatures, regardless of linkability considerations, have signature sizes of $O(\log n)$ at best. Furthermore, leveraging online/offline verification, part of the verification of signatures on the same ring can be shared, resulting in a state-of-the-art amortized verification cost of $O(\text{polylog}(\log n))$.

Our LRS also performs favourably against existing schemes in practical scenarios. Concretely, our scheme has the smallest signature size among all post-quantum linkable ring signatures with non-slanderability for ring size larger than 32. In our experiment, at 128-bit security and ring size of 1024, our LRS has a size of 29KB, and an amortized verification cost of 0.3 ms, surpassing the state-of-the-art by a significant margin. Even without considering amortization, the verification time for a single signature is 128 ms, comparable to those featuring linear signature size. A similar performance advantage can also be seen at signing. Furthermore, our LRS has extremely short public keys (32 bytes), while public keys of existing constructions are in the order of kilobytes.

Keywords: linkable ring signature · post-quantum cryptography · signature of knowledge

1 Introduction

Ring signatures [39] allow a user to sign messages anonymously on behalf of a group without revealing the signer's identity. Initially introduced by Rivest et al. [39], the primary motivation behind ring signatures is to allow whistleblowers to disclose information while keeping their identity confidential and proving the reliability of the information. Unlike group signatures [14], which require a central manager to handle tasks such as generating users' public keys, managing group membership, and deanonymizing the signer, ring signatures achieve anonymity without relying on a central manager, and each member can spontaneously form ad-hoc groups.

Linkable ring signatures [30] (LRS) are ring signatures with reduced anonymity to safeguard against potential abuses of complete anonymity. Specifically, LRS are ring signatures with linkability, meaning that multiple signatures from the same signer can be detected (i.e., linked).

In the literature, various notions of linkability have been considered. The origin linkable ring signatures [30,31] allow linking of signatures generated using the same key on the same ring (referred to as ring-based linkability hereafter). In other words, in ring-based LRS, signatures on different rings from the same signer will not be linked. In [22], signatures generated using the same key on the same message can be linked, and we used the term message-based linkability to describe this kind of linking. A variant called event-oriented linkability (aka prefix linkability in [10]) is considered in [44,4,11,10]. In an event-oriented LRS, signatures consist of an additional component called event-id, and signatures generated from the same key with the same event-id can be linked. Another common type of LRS offers one-time linkability [2,5,33,26]. In these schemes, signatures generated using the same key can be linked, and typically, the signer will use their key only once. [45] presents a transformation that turns any ring signatures into a one-time LRS.

It is important to note that event-oriented linkability is the most general form of linkability among the aforementioned notions. By setting the event-id to be the ring or the message, the resulting event-oriented LRS becomes ring-based or message-based linkability, respectively. Similarly, if we set the event-id to be a fixed string, we have one-time linkability.

Linkable ring signatures are employed in various applications such as e-voting [16] and privacy-oriented cryptocurrencies [43,38]. Anonymity decouples voters from their ballots and prevents transactions from being linked to specific accounts, while linkability prevents double-voting and double-spending.

The security of many existing linkable ring signature schemes [39,30,29] rely on the hardness of integer factorization or discrete logarithms problem, making them vulnerable to quantum computers. To defend against quantum attacks that might emerge in the coming decades, post-quantum secure solutions are of paramount importance. Presently, post-quantum cryptography research mainly falls within five categories [15]: lattice-based, hash-based, code-based, isogeny-based, and Multivariate polynomial cryptography.

Among the alternatives, existing post-quantum linkable ring signature schemes primarily concentrate on lattice-based [5,8,2,48,32] and isogeny-based approaches [8]. However, these schemes encounter practical limitations due to either their substantial signature sizes [5,2,32,48], or comparatively slow runtime [8], especially in scenarios involving large rings. Specifically, the verification time complexity of all existing post-quantum solutions is $O(n)$ for ring size n, and the current smallest signature size with 128-bit security and a ring size of 1024 is 55 KB from [8]. To address this limitation, we construct a hash-based linkable ring signature scheme with $O(\text{polylog}(\log n))$ amortized verifier time and signature size. At the same security level and ring size, our LRS has a signature size of only 29 KB. More importantly, our scheme is an order of magnitude faster than [8] in both signing and verification even without consideration of amortization. Indeed, the time complexity of our scheme is comparable to the Raptor [34], the fastest LRS in the literature featuring linear signature size. A comparison of existing post-quantum ring signature schemes is presented in Table 1.

Table 1. Comparison of post-quantum linkable ring signatures. OTL MBL and RBL respectively denote one-time, message-based and ring-based linkability.

	OTL	MBL	RBL	Signature size	Verifier time		Hardness assumptions	Random Oracle
					offline	online		
[47]	✗	✗	✗	$O(n)$	-	$O(n)$	M-LWE, M-SIS	Yes
[37]	✗	✗	✗	$O(\log n)$	-	$O(n)$	M-LWE, M-SIS	Yes
[21]	✓	✗	✗	$O(\log n)$	-	$O(n)$	M-LWE, M-SIS	Yes
[2]	✓	✗	✗	$O(n)$	-	$O(n)$	Ring-SIS	Yes
[5]	✓	✗	✗	$O(n)$	-	$O(n)$	M-LWE, M-SIS	Yes
[34]	✓	✗	✗	$O(n)$	-	$O(n)$	NTRU	Yes
[8]	✓	✗	✗	$O(\log n)$	-	$O(n)$	M-LWE, M-SIS / CSIDH-512	Yes
Ours	✓	✓	✓	$O(\text{polylog}(\log n))$	$O(n)$	$O(\text{polylog}(\log n))$	CRHF	Yes

1.1 Our Contribution

We summarize our contribution as follows.

- First, we introduce a new framework for constructing event-oriented linkable ring signatures based on hash functions and signature of knowledge (SoK) [13]. We provide rigorous security proof for this generic construction, demonstrating that its security hinges on the security of the underlying hash functions and SoK.
- Second, we instantiate our framework by adapting the hash-based post-quantum non-interactive argument of knowledge (NIAoK) ethSTARK [42] into an SoK. Our adaption involves adding zero-knowledge to ethSTARK, which is crucial in transforming it into an SoK (through the Fiat-Shamir heuristic). Also, we crafted the program representation of the execution trace to enhance the efficiency of the signing process. The results in the first post-quantum event-oriented linkable ring signature scheme.

- Third, we evaluate the performance of our LRS and show that it is highly efficient. Asymptotically, our LRS achieves $O(\text{polylog}(\log n))$ signature size and amortized verification time for ring size n. When targeting 128-bit security, our instantiation features a notably compact public key size of just 32 bytes, which is significantly smaller compared to existing lattice-based schemes, where public key sizes can reach several kilobytes. Moreover, when ring size is $n \geq 32$ and security level is 128-bit, our LRS features the smallest signature size among all post-quantum LRS with non-slanderability. The online verification time is about 0.3 ms for ring size $n = 8192$. The overall verification time is always an order of magnitude faster than those with sublinear signature size and is comparable with the state-of-the-art linear-size LRS. This high efficiency, even when dealing with large ring sizes, makes our scheme ideal for applications involving a significant number of users.

1.2 Overview of Our Construction

Our Framework. We first describe our framework for constructing LRS from hash functions and SoK. In our framework, each user's public key pk is derived from their private key sk via hashing operation: $pk = \mathcal{H}ash(sk)$. A collection of user public keys, $pk_1, pk_2, ..., pk_n$, forms the ring R.

To sign message m on behalf of ring R with respect to event-id e, the signer first constructs a Merkle tree using all public keys in R as its leaf nodes and obtains the Merkle root rt. The signer then calculates the Merkle path[3] $\boldsymbol{P}$ from the hash leaf of the signer's public key pk_l to the root rt. An example of the Merkle path is illustrated in Fig. 1. The signer further computes tag $T = \mathcal{H}ash(sk_l, e)$ from its private key sk_l and event-id e.

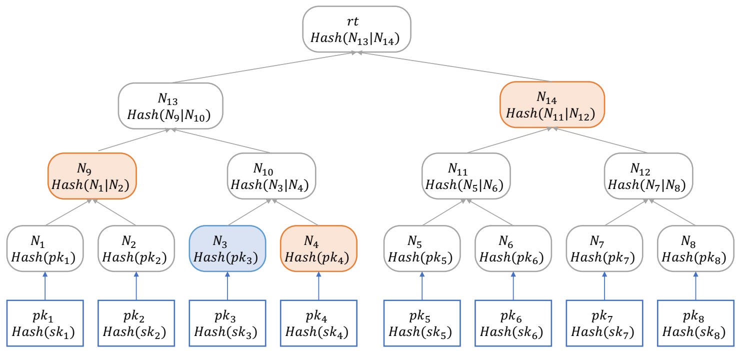
Finally, the signer constructs a signature of knowledge SoK_m on message m for the NP-statement (e, rt, T): 1. $\mathcal{H}ash(sk_l)$ is a leaf of rt, 2. $T = \mathcal{H}ash(sk_l, e)$, using witness $(l, sk_l, \boldsymbol{P})$. The output of SoK_m is a signature σ_s that demonstrates the possession of witness with respect to the instance and the correct signing of message m. For concreteness, one can think of an SoK as the proof-of-knowledge turned into a signature using the Fiat-Shamir heuristic. Finally, the signer outputs the LRS (σ_s, T).

On receiving the linkable ring signature (σ_s, T) on message m, event e and ring R, the verifier will compute the Merkle root rt from R and form the instance (e, rt, T). Then, the verifier will utilize the verification algorithm of SoK_m to check whether σ_s is a valid proof for instance (e, rt, T) on message m. Linkability is achieved by checking whether the receiving signatures share the same tag T as previous ones. If there is a match, the two signatures share the same signer.

The above signing and verification processes both require the construction of a Merkle tree, which can be done offline after the ring R is known, while before knowing the signing message. As a result, our framework naturally divides into online/offline signing and verification phases. The online phase involves the signing and verification of a SoK_m.

[3] A Merkle path of a leaf node consists of all sibling nodes along the path from the root to the leaf node.

Fig. 1. An Example Merkle path for pk_3



For public key pk_3 in list $\{pk_i\}_{i\in[8]}$, the path $\boldsymbol{P}$ is (N_4, N_9, N_{14}).

Our Instantiation. In our instantiation, we adopt the non-interactive ethSTARK [42] as the underlying argument system to build SoK_m. We choose ethSTARK for several reasons. Firstly, ethSTARK is transparent, eliminating the need for a trusted setup. Secondly, ethSTARK is a hash-based NIAoK resistant to attacks from quantum computers. Lastly, for the proof of computation with purely hash operations, the verification time and proof size in ethSTARK are poly-logarithmic to the number of hash operations.

However, the plain ethSTARK lacks zero-knowledge property and thus can not be directly utilized or transformed into an SoK. To accommodate this, we augment ethSTARK with the zero-knowledge property, which can be considered an independent interest.

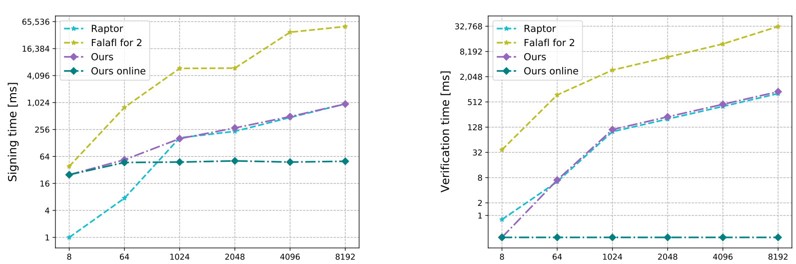
Furthermore, we optimize the NP statement of our SoK_m from ethSTARK to improve efficiency. As mentioned in [27], while general-purpose virtual machines for the STARK program are available, e.g., Cairo [23], hand-optimized representations are often needed for better efficiency. We construct an execution trace consisting solely of individual traces of hash operations. We also optimized the representation of the trace table so that the number of hash operations scales logarithmically with the ring size. Specifically, our execution trace consists only of 8 registers, and the total number of states is linear in $\log n$, where n is the ring size. Recall that in our case, the verification time and proof size in ethSTARK scale poly-logarithmically in the number of hash operations. Thus, by leveraging our hand-optimized representation, our scheme achieves a further improvement in efficiency, resulting in a verification time and proof size of $O(\text{polylog}(\log n))$. This improvement in efficiency is a crucial aspect of our instantiation.

Table 2 provides a signature size comparison between our instantiation and existing post-quantum linkable ring signature schemes. In ethSTARK, the trace table, which represents the execution of the computation, is required to have a length that is a power of 2. Our construction has an actual trace length of $8\log(n) + 24$, where n is the ring size. We pad the trace table to the next power of 2 for compatibility. As a result, the trace length remains constant for a range of ring sizes, e.g., 128 for ring sizes between 2^5 and 2^{13}. Since the efficiency of ethSTARK depends on the trace length, this consistency in trace length under the same constraints leads to a consistent performance across the corresponding range of ring sizes. The predictable signature size allows for easier integration and evaluation of our instantiation in various scenarios.

In Figure 2a, 2b, we present the comparisons of the signing time and verification time, all with a security level of 128-bit. In comparison to Raptor [34], which is based on NTRU, our method achieves a higher security level and offers a smaller signature size when the ring size exceeds 2^5. Furthermore, our construction demonstrates a smaller signature size and faster runtime when compared to Falafl for 2 [8], which relies on module short integer solution (M-SIS) problem and module learning with error (M-LWE) problem. While the isogeny-based scheme Calamari [8] has the smallest signature size, its runtime is significantly slower, and it only provides 128 bits of classical security and 60 bits of quantum security. Due to its slow performance, we exclude it from the comparison in Figure 2a, 2b.

Table 2. Signature size comparison

		Number of users						
	Security bits	2^3	2^6	2^8	2^{10}	2^{12}	2^{13}	2^{14}
Raptor [34]	100 bits	11KB	83KB	327KB	1302KB	5203KB	10327KB	20644KB
Falafl for 2 [8]	$\geq$ 128 bits	50KB	52KB	53KB	54KB	55KB	55KB	56KB
Calamari [8]	60 bits	5KB	8KB	10KB	12KB	14KB	15KB	16KB
This work	99 bits	17KB	20KB	20KB	20KB	20KB	20KB	26KB
This work	128 bits	25KB	29KB	29KB	29KB	29KB	29KB	38KB



(a) Signing time comparison (b) Verification time comparison

Fig. 2. Performance comparison with 128-bit security

In our performance comparison, we focus on post-quantum linkable ring signature schemes that offer non-slanderability and have provided concrete performance results. We note that some existing post-quantum linkable ring signature schemes, like SMILE [37] and MatRiCT+ [21], can be adapted to achieve one-time linkability with non-slanderability by incorporating a post-quantum one-time signature. For instance, by incorporating Dilithium 3 [18], which has a public key size 1.9KB and signature size 3.2KB, we estimate that linkable SMILE and MatRiCT+ with non-slanderability will feature signature size of 24KB and 31KB, respectively, for ring size of 2^{15}.[4] Asymptotically, both SMILE and MatRiCT+ have signature size logarithmic in the ring size N, while for our scheme, we have $O(\text{polylog}(\log N))$ signature size.

Moreover, our scheme maintains a significantly smaller public key size of only 32 bytes, which offers a notable advantage in terms of communication costs, including the size of the public key. The public key size for Raptor is 0.9KB, while MatRiCT+ has a public key size over 1KB for very small rings and reaches 3KB when the ring size is around 2^{10}. SMILE and Falafl do not provide concrete public key sizes in the paper. According to our estimation, the public key size for both schemes will be around the kilobyte level.

1.3 Related Work

Post-quantum Ring Signatures Brakerski and Kalai [12] introduced a generic ring signature scheme based on the short integer solution (SIS) assumption in 2010. However, it is weakly secure and requires extra effort to transform into a fully secure scheme. Building upon Lyubashevsky's [35] lattice-based signature scheme, Aguilar-Melchor et al. [1] further extended it to construct a ring signature scheme with a linear size. To shorten the signature size, Libert et al. [28] proposed the first logarithmic-sized post-quantum ring signature scheme. This scheme utilizes accumulators to prove membership by demonstrating the possession of a hash chain. Thereafter, subsequent works based on zero-knowledge proofs were introduced. Esgin et al. [20,19,21] presented a lattice-based one-out-of-many proof based on the proposals [24,9]. Lyubashevsky et al. [37] proposed a set membership proof from ideal lattices and transformed it into a logarithmic size ring signature.

Different from the constructions based on accumulators and zero-knowledge proofs, Yuen et al. [47] introduced a novel ring signature scheme consisting of two rings, a commitment ring and a challenge ring. Their scheme can be instantiated from both DL-based and lattice-based cryptography. Apart from the previous lattice-based solutions, Scafuro and Zhang [40] proposed one-time traceable ring signatures constructed purely from hash functions. Their scheme requires no hardness assumptions and uses hash functions in a black-box way.

Post-quantum Linkable Ring Signatures Yang et al. [46] proposed the first post-quantum linkable ring signature scheme of logarithmic size. Their scheme was

[4] We remark that no concrete signature sizes in this setting were reported, and the adaption only provides one-time linkability.

built on the lattice-based weak pseudo-random function. Torres et al. [2] constructed a one-time linkable ring signature with unconditional anonymity based on the lattice-based signature scheme BLISS [17]. In concurrent work, Baum et al. [5] presented a one-time linkable ring signature scheme constructed from a collision-resistant lattice-based hash function. The paper achieved linkability without heavy zero-knowledge proof.

In the line of general lattice-based linkable ring signatures, Zhang et al. [48] proposed a logarithmic size construction based on ideal lattices using lattice signatures [36]. To be more applicable in cryptocurrencies, Liu et al. [32] presented a lattice-based linkable ring signature scheme with stealth addresses to capture practical situations under adversarially chosen-key attacks. Lu et al. [34] presented a practical lattice-based linkable ring signature scheme based on the generic ring signature framework from Rivest et al. [39] adapted towards the lattice setting. Specifically, while Revist et al.'s framework employs the one-way trapdoor permutation, Lu et al.'s framework was built on a new primitive called Chameleon Hash Plus, and they presented an instantiation from the NTRU lattice. Beullens et al. [8] proposed logarithmic-size linkable ring signatures based on a group action. Their scheme can be instantiated using isogeny or lattice assumption, and is the first construction of linkable ring signatures from isogeny assumption.

2 Preliminaries

2.1 Notations

We consider the field $\mathbb{F}_p$ to be a prime field that contains a sufficiently large multiplicative sub-group. We use the notation $[d]$ to denote the set $\{1, 2, \ldots, d\}$, and $l[i]$ to denote the ith value in the vector $l = (l_1, \ldots, l_n) \in \mathbb{F}_p^n$. We use $(D^{(0)}, D^{(1)})$ to represent the two elements in $D \in \mathbb{F}_p^2$ and use $t \leftarrow\$ T$ to represent that we randomly select an element t from the set T.

2.2 Signatures of knowledge

We follow the notion of signature of knowledge as described in [10]. In a signature of knowledge scheme, a signature is issued on behalf of any NP statement, that can be interpreted as "One who has signed message m holds a valid witness $\mathbb{w}$ to the NP statement $\mathbb{x}$".

A signature of knowledge is a set of probabilistic polynomial time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$: takes the security parameter λ as input and outputs public parameters pp.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathbb{x}, \mathbb{w}, m)$: takes the statement $\mathbb{x}$, witness $\mathbb{w}$ and a message m as inputs and outputs signature σ.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathbb{x}, \sigma, m)$: takes the statement $\mathbb{x}$, a signature σ, and a message m as inputs and outputs a bit representing accept(1) or reject(0).

The triple of efficient algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is called a signature of knowledge for a relation $\mathcal{R}$ if the following properties hold:

– **Correctness.** For all $\lambda \in \mathbb{N}, m \in \{0,1\}^*, (\mathbb{x}, \mathbb{w}) \in \mathcal{R}$,

$$\Pr\left[\ \mathsf{Verify}(\mathsf{pp}, \mathbb{x}, \sigma, m) = 1 \,\middle|\, \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathbb{x}, \mathbb{w}, m) \end{array} \right] = 1.$$

– **Simulatability.** There exists a polynomial time simulator Sim consisting of algorithms SimG and SimS,
  - $(\mathsf{pp}, \tau) \leftarrow \mathsf{SimG}(1^\lambda)$: takes the security parameter λ as input and outputs public parameters pp and trapdoor τ.
  - $\sigma \leftarrow \mathsf{SimS}(\mathsf{pp}, \tau, \mathbb{x}, m)$: takes the public parameters pp, trapdoor τ, statement $\mathbb{x}$, and a message m as input and produces a simulated signature σ.

  The oracle Sim receives the input values $(\mathbb{x}, \mathbb{w}, m)$, checks whether $\mathbb{w}$ is valid and returns $\sigma \leftarrow \mathsf{SimS}(\mathsf{pp}, \tau, \mathbb{x}, m)$. For any non-uniform polynomial time adversary $\mathcal{A}$ with oracle access to Sim and signer S,

$$\Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathsf{Sim}}(pp) \,\middle|\, (\mathsf{pp}, \tau) \leftarrow \mathsf{SimG}(1^\lambda)\ \right] \approx \Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathsf{S}}(pp) \,\middle|\, \mathsf{pp} \leftarrow \mathsf{G}(1^\lambda)\ \right].$$

– **Simulation Extractability.** In addition to oracle Sim, there exists a polynomial time extractor Ex such that for any non-uniform polynomial time adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} (\mathsf{pp}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} \vee \\ (\mathbb{x}, \mathbb{w}, m) \in \mathsf{Q} \vee \\ \mathsf{Verify}(\mathbb{x}, \sigma, m) = 0 \end{array} \,\middle|\, \begin{array}{l} (\mathsf{pp}, \tau) \leftarrow \mathsf{SimGen}(1^\lambda), \\ (\mathbb{x}, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sim}}(pp), \\ \mathbb{w} \leftarrow \mathsf{Ex}(\mathsf{pp}, \tau, \mathbb{x}, m, \sigma) \end{array} \right] \approx 1.$$

  where Q denotes all successful queries $(\mathbb{x}, \mathbb{w}, m)$ that $\mathcal{A}$ has sent to Sim.

2.3 ethSTARK Protocol

STARK [7] (Scalable Transparent ARgument of Knowledge) is a class of proof system addressing the computational integrity (CI) statements, where the system translates CI statements such as "u is the result of executing hash function f for T steps on input v" into formal algebraic language. ethSTARK [42] is a member of the STARK family.

An execution trace of a program running for T steps is a $w \times T$ table, in which w is the number of registers. Each column in the execution trace table corresponds to a specific register, tracking its contents and changes over time as the program executes. Each row in the table represents the state of the computation at a particular moment during the execution. In ethSTARK, the verification of a CI statement is initially reduced to the task of checking whether the Domain Extension for Eliminating Pretenders (DEEP) composition polynomial has a low degree. The passing of the low-degree test indicates that the execution trace satisfies the given constraints. This low-degree test is achieved by leveraging the Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI) [6] protocol. Furthermore, the protocol can be converted to be non-interactive via the Fiat-Shamir heuristic. For more scheme details, please refer to the literature [42].

2.4 Merkle Tree

In the following, we describe a set of algorithms for the implementation of the Merkle tree. First, we conclude a hash function $\mathcal{H} : \mathcal{X}_t \to \mathcal{Y}_t$ using two algorithms $(\mathsf{HGen}, \mathcal{H})$.

- $\mathsf{pp}_{\mathcal{H}} \leftarrow \mathsf{HGen}(1^\lambda)$: takes the security parameter as input and outputs public parameters $\mathsf{pp}_{\mathcal{H}}$.
- $D \leftarrow \mathcal{H}(m)$: takes the message m as input and outputs the hash output D.

Next, we define the algorithms for the Merkle tree as follows,

Definition 1 (Merkle Tree). *Given a hash function $\mathcal{H}$ and a list of elements $\boldsymbol{s} = (s_1, \ldots, s_n)$, the Merkle tree consists of three algorithms as* $(\mathsf{MTree}, \mathsf{GPath}, \mathsf{MPath})$ *where:*

- $(rt, mtree) \leftarrow \mathsf{MTree}(\boldsymbol{s})$*: on input a list of elements $\boldsymbol{s}$, it uses $\{\mathcal{H}(s_i)\}_{i\in[n]}$ as leaves to construct a Merkle tree. The algorithm outputs a description of the tree $mtree$ and the root rt.*
- $\boldsymbol{P} \leftarrow \mathsf{GPath}(l, mtree)$*: on input an index l, a description of a Merkle tree $mtree$, it outputs the Merkle path $\boldsymbol{P}$ to the leaf node $\mathcal{H}(s_l)$, which contains the siblings of s_l and its ancestors'.*
- $rt' \leftarrow \mathsf{MPath}(s_l, \boldsymbol{P}, l)$*: on input an element s_l in the list $\boldsymbol{s}$, a Merkle path $\boldsymbol{P}$ and an index l, it outputs the reconstructed root rt'.*

3 Linkable Ring Signature Schemes

We now review the definition of linkable ring signatures in [29,34].

Definition 2 (Linkable ring signature scheme). *A linkable ring signature scheme consists of five PPT algorithms as* $\mathsf{LRS} = (\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link})$ *where:*

- $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$*: takes the security parameter λ as input and outputs public parameters* pp.
- $(pk_i, sk_i) \leftarrow \mathsf{LRS.KeyGen}(\mathsf{pp})$*: takes the public parameters* pp *as input and outputs a pair of public and private keys.*
- $\sigma \leftarrow \mathsf{LRS.Sign}(e, sk_l, m, \mathsf{R})$*: on input an event-id e, a private key sk_l, a message m, a list of public keys* R *that includes the public key corresponding to the private key sk_l, it outputs a signature σ.*
- $0/1 \leftarrow \mathsf{LRS.Verify}(e, \sigma, m, \mathsf{R})$*: takes an event-id e, a signature σ, a message m, a list of public keys* R *as input and outputs a bit representing accept(1) or reject(0).*
- $0/1 \leftarrow \mathsf{LRS.Link}(e, \sigma, \sigma', m, m', \mathsf{R}, \mathsf{R}')$*: takes an event-id e, signatures σ, σ', messages m, m', lists of public keys R, R' as input and outputs a bit representing linked(1) or unlinked(0).*

Security notions we introduce the following oracles which can be accessed by adversaries during the game.

– Joining oracle $pk_i \leftarrow \mathcal{JO}(\perp)$: the joining oracle $\mathcal{JO}$ adds a new member to the system and returns a public key for the new member.
– Corruption oracle $sk_i \leftarrow \mathcal{CO}(pk_i)$: given a public key pk_i produced by $\mathcal{JO}$, the corruption oracle $\mathcal{CO}$ returns the associated private key sk_i.
– Signing oracle $\sigma \leftarrow \mathcal{SO}(e, \mathsf{R}, pk_i, m)$: given en event-id e, a list of public keys R, a public key $pk_i \in \mathsf{R}$ and a message m, the signing oracle $\mathcal{SO}$ returns a valid signature σ.

Unforgeability requires that an adversary $\mathcal{A}$ cannot create a valid signature without having any secret key in that ring. We define the unforgeability game $Game^{forge}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as:

- $\mathcal{C}$ runs $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$ and sends pp to $\mathcal{A}$.
- $(e, \sigma, m, \mathsf{R}) \leftarrow \mathcal{A}^{\mathcal{SO},\mathcal{CO},\mathcal{JO}}(\mathsf{pp})$.
- $\mathcal{A}$ wins $Game^{forge}$ if (i) $\mathsf{LRS.Verify}(e, \sigma, m, \mathsf{R})= 1$. (ii) all public keys in R are produced by $\mathcal{JO}$. (iii) no public key in R has been input to $\mathcal{CO}$. (iv) σ is not generated by $\mathcal{SO}$.

The advantage of $\mathcal{A}$ in $Game^{forge}$ is defined as $adv_A^{forge} = \Pr[\mathcal{A} \text{ wins } Game^{forge}]$.

Definition 3 (Unforgeability). *A linkable ring signature scheme is unforgeable if for any polynomial-time adversary $\mathcal{A}$, the advantage adv_A^{forge} for $\mathcal{A}$ to win the unforgeability game $Game^{forge}$ is negligible.*

Anonymity requires that an adversary $\mathcal{A}$ cannot identify which is the signer who produced the signature. We define the anonymity game $Game^{anon}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as:

- $\mathcal{C}$ runs $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$ and sends pp to $\mathcal{A}$.
- $\mathcal{A}$ picks an event-id e, a message m and a set of public keys $\mathsf{R} = \{pk_i\}_{i\in[n]}$ where $\mathsf{R} \leftarrow \mathcal{A}^{\mathcal{JO}}(pp)$, and sends (e, m, R) to $\mathcal{C}$.
- $\mathcal{C}$ picks $b \leftarrow\!\$\ [n]$ and runs $\sigma \leftarrow \mathsf{LRS.Sign}(e, sk_b, m, \mathsf{R})$ and sends σ to $\mathcal{A}$.
- $\mathcal{A}$ outputs b' and wins the game $Game^{anon}$ if $b' = b$.

The advantage of $\mathcal{A}$ in $Game^{anon}$ is defined as $adv_A^{anon} = \Pr[\mathcal{A} \text{ wins } Game^{anon}]$.

Definition 4 (Anonymity). *A linkable ring signature scheme is anonymous if for any polynomial-time adversary $\mathcal{A}$, with the ring size n, the advantage adv_A^{anon} for $\mathcal{A}$ to win the anonymity game $Game^{anon}$ is negligible close to $1/n$.*

Linkability requires that an adversary $\mathcal{A}$ cannot produce two unlinked signatures using the same private key. We define the linkability game $Game^{link}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as:

- $\mathcal{C}$ runs $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$ and sends pp to $\mathcal{A}$.
- $(e, \sigma_i, m_i, \mathsf{R}_i) \leftarrow \mathcal{A}^{\mathcal{SO},\mathcal{CO},\mathcal{JO}}(\mathsf{pp})$ for $i \in [n]$.

- $\mathcal{A}$ wins the game $Game_A^{link}$ if (i) $\mathsf{LRS.Link}(e, \sigma_i, \sigma_j, m_i, m_j, \mathsf{R}_i, \mathsf{R}_j) = 0$ for $i, j \in [n]$ and $i \neq j$. (ii) $\mathsf{LRS.Verify}(e, \sigma_i, m_i, \mathsf{R}_i) = 1$. (iii) no σ_i is generated by $\mathcal{SO}$. (iv) all public keys in R_i are produced by $\mathcal{JO}$. (v) $\mathcal{A}$ queried $\mathcal{CO}$ less than n times.

The advantage of $\mathcal{A}$ in $Game^{link}$ is defined as $adv_A^{link} = \Pr[\mathcal{A} \text{ wins } Game^{link}]$.

Definition 5 (Linkability). *A linkable ring signature scheme is linkable if for any polynomial-time adversary $\mathcal{A}$, the advantage adv_A^{link} for $\mathcal{A}$ to win the linkability game $Game^{link}$ is negligible.*

Non-slanderability requires that an adversary $\mathcal{A}$ cannot produce a valid signature that links to a signature generated by an honest signer. We define the non-slanderability game $Game^{slan}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as:

- $\mathcal{C}$ runs $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$ and sends pp to $\mathcal{A}$.
- $\mathcal{A}$ sends $\mathcal{C}$ an event-id e, a message m, a list of public key R and a public key pk, where $pk \in \mathsf{R}$.
- $\mathcal{C}$ runs $\sigma \leftarrow \mathsf{LRS.Sign}(e, sk, m, \mathsf{R})$ and sends the signature σ to $\mathcal{A}$, where sk is the associated private key of pk.
- $(\sigma', m', \mathsf{R}') \leftarrow \mathcal{A}^{\mathcal{SO},\mathcal{CO},\mathcal{JO}}(\mathsf{pp}, \sigma)$.
- $\mathcal{A}$ wins the game $Game^{slan}$ if (i) pk has not been input to $\mathcal{CO}$ and $\mathcal{SO}$. (ii) σ' is not generated by $\mathcal{SO}$. (iii) $\mathsf{LRS.Verify}(e, \sigma', m', \mathsf{R}') = 1$. (iv) $\mathsf{LRS.Link}(e, \sigma, \sigma', m, m', \mathsf{R}, \mathsf{R}') = 1$.

The advantage of $\mathcal{A}$ in $Game^{slan}$ is defined as $adv_A^{slan} = \Pr[\mathcal{A} \text{ wins } Game^{slan}]$.

Definition 6 (Non-slanderability). *A linkable ring signature scheme is non-slanderable if for any polynomial-time adversary $\mathcal{A}$, the advantage adv_A^{slan} for $\mathcal{A}$ to win the non-slanderability game $Game^{slan}$ is negligible.*

4 Our Construction

We present our event-oriented linkable ring signature scheme. We begin by introducing the framework and then describe the instantiation of our framework.

4.1 Framework

Assuming the number of members in the ring R is n, and R is a vector of public keys $(pk_1, \ldots, pk_n)$. We start by showing how to prove the signer's private key sk_l was used to generate the tag T, while also confirming that its associated public key pk_l exists within the ring R.

Let e be the event-id, l be the signer's index in binary form, rt be the Merkle root of tree $mtree$ generated using the ring R, and $\boldsymbol{P}$ be the Merkle path for pk_l in $mtree$. We define two one-way and collision resistant hash functions $\mathcal{H}_k$:

$\mathcal{X}_k \to \mathcal{Y}_k$ and $\mathcal{H}_t : \mathcal{X}_t \to \mathcal{Y}_t$. On input of a security parameter λ, let $k = \lceil \log n \rceil$, the relation $\mathcal{R}_s$ is defined as :

$$\mathcal{R}_s = \{((e, rt, T), (\boldsymbol{P}, l, sk_l)) : e, sk_l \in \mathcal{X}_k \wedge rt, T \in \mathcal{Y}_t \wedge l \in \{0,1\}^k \wedge \\ \boldsymbol{P} = \{P_i\}_{i\in[k]} \wedge P_i \in \mathcal{Y}_t \wedge T = \mathcal{H}_t(sk_l, e) \wedge rt = \mathsf{MPath}(\mathcal{H}_k(sk_l), \boldsymbol{P}, l)\}$$

where the algorithms ($\mathsf{MTree}, \mathsf{GPath}, \mathsf{MPath}$) defined in section 2.4 are used to prove the membership of signer's public key pk_l in the ring.

Let m be the message to be signed. We incorporate a signature of knowledge SoK_m for the relation $\mathcal{R}_s$ on m and output a signature σ_s. A signature of knowledge issues the public key signatures on behalf of NP statements. That is, if σ_s is a valid signature, it indicates that the signer possesses the witness $(\boldsymbol{P}, l, sk_l)$, and the relation $\mathcal{R}_s$ holds.

The public parameters in the framework include the public parameters for $\mathcal{H}_k, \mathcal{H}_t$ in the setup phase HGen, and the public parameters for $\mathsf{SoK_m}$. In the offline phase, the prover and the verifier both construct a Merkle tree using all the public keys in the ring R to obtain the tree root rt. In the online phase, the signer and the verifier engage in the $\mathsf{SoK_m}$ protocol on message m. We describe the framework of our linkable ring signature scheme in Figure 3.

Setup: $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$

Define functions $\mathcal{H}_k : \mathcal{X}_k \to \mathcal{Y}_k, \mathcal{H}_t : \mathcal{X}_t \to \mathcal{Y}_t$.
$\mathsf{pp}_{\mathcal{H}_k} \leftarrow \mathsf{HGen}(1^\lambda), \mathsf{pp}_{\mathcal{H}_t} \leftarrow \mathsf{HGen}(1^\lambda)$,
$\mathsf{pp}_s \leftarrow \mathsf{SoK_m.Gen}(1^\lambda)$.
Return $\mathsf{pp} = (\mathcal{H}_k, \mathcal{H}_t, \mathsf{pp}_{\mathcal{H}_k}, \mathsf{pp}_{\mathcal{H}_t}, \mathsf{pp}_s)$.

Key Generation: $(pk, sk) \leftarrow \mathsf{LRS.KeyGen}(\mathsf{pp})$

$sk \leftarrow\!\$\; \mathcal{X}_k, pk = \mathcal{H}_k(sk) \in \mathcal{Y}_k$.
Return (pk, sk).

Signing: $\sigma \leftarrow \mathsf{LRS.Sign}(e, sk_l, m, \mathsf{R})$

- One-time offline signing per ring :
  $rt, mtree \leftarrow \mathsf{MTree}(\mathsf{R})$.
- Online signing :
  $P = \mathsf{GPath}(l, mtree), T = \mathcal{H}_t(sk_l, e)$,
  $\sigma_s \leftarrow \mathsf{SoK_m.Sign}(\mathsf{pp}, (e, rt, T), (\boldsymbol{P}, l, sk_l), m)$.
  Return $\sigma = (\sigma_s, T)$.

Verification $0/1 \leftarrow \mathsf{LRS.Verify}(e, \sigma, m, \mathsf{R})$

- One-time offline verification per ring :
  $rt, mtree \leftarrow \mathsf{MTree}(\mathsf{R})$.
- Online verification :
  Parse $\sigma = (\sigma_s, T)$.
  Return $0/1 \leftarrow \mathsf{SoK_m.Verify}(\mathsf{pp}, (e, rt, T), \sigma_s, m)$.

Linking: $0/1 \leftarrow \mathsf{LRS.Link}(e, \sigma, \sigma', m, m', R, R')$

Parse $\sigma = (\sigma_s, T), \sigma' = (\sigma'_s, T')$. If $T' = T$, return 1, otherwise 0.

Fig. 3. Linkable Ring Signature Scheme Framework

4.2 Security Proof

Theorem 1. *Our linkable ring signature is linkable in the random oracle model if the underlying SoK is correct, simulatable and simulation extractable, and the underlying hash function is one-way and collision-resistant.*

If $\mathcal{A}$ is able to win the linkability game defined in Definition 5 with a non-negligible probability, we can construct $\mathcal{S}$ to break either the one-wayness or the collision-resistance of the hash function $\mathcal{H}_k$. For breaking one-wayness, on given a hash output h_o, one is required to output x such that $h_o = \mathcal{H}_k(x)$. For breaking collision-resistance, on given x, one is required to output x' such that $\mathcal{H}_k(x) = h_c = \mathcal{H}_k(x')$.

At the beginning of the game, simulator $\mathcal{S}$ receives the one-wayness instance h_o and collision-resistance instance x_c of $\mathcal{H}_k$. $\mathcal{S}$ will sample other public parameters by running $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$. $\mathcal{H}_t$ will be programmed as random oracle. For the Oracle simulation,

- $\mathcal{JO}(\perp)$: Assume that $\mathcal{A}$ makes total q_j join queries. $\mathcal{S}$ first samples $q_j^{(o)}, q_j^{(c)} \leftarrow\!\!\$ \; [1, \cdots, q_j]$. For the ith query, if $i \neq q_j^{(o)}$ or $q_j^{(c)}$, $\mathcal{S}$ runs $\mathsf{LRS.KeyGen}(\mathsf{pp})$ to generate pk_i. If $i = q_j^{(o)}$, $\mathcal{S}$ returns $pk_i = pk_o = h_o$. If $i = q_j^{(c)}$, $\mathcal{S}$ returns $pk_i = pk_c = \mathcal{H}_k(x_c)$ and sets $sk_i = sk_c = x_c$. From the adversary's view, the join oracle will be identical to the original one.
- $\mathcal{CO}(pk_i)$: Consider $\mathcal{A}$ makes q_r queries to $\mathcal{CO}$, where $q_r \leq n-1$. For $pk_i = h_o$, $\mathcal{S}$ aborts the game. For $pk_i = pk_c$, $\mathcal{S}$ returns the private key $sk_i = x_c$. Otherwise, $\mathcal{S}$ returns the corresponding private key sk_i.
- $\mathcal{SO}(\mathsf{R}, e, pk_i, m)$: When $\mathcal{A}$ queries $\mathcal{SO}$ on message m, event-id e, a list of public keys R and the public key for the signer pk_i, where $\mathsf{pk}_i \in \mathsf{R}$. If $pk_i \neq pk_o$, $\mathcal{S}$ runs $\sigma \leftarrow \mathsf{LRS.Sign}(e, sk_i, m, \mathsf{R})$ and sends the signature σ to $\mathcal{A}$. If $pk_i = pk_o$, $\mathcal{S}$ samples $T \leftarrow\!\!\$ \; \mathcal{Y}_t$ and sets $\mathbb{x} = \{rt, T, e, m\}$ where rt is the Merkle root generated from R. $\mathcal{S}$ then employs the simulator Sim in SoK to simulate $\mathsf{SimG}(1^\lambda) \to (pp, \tau)$, $\mathsf{SimS}(\mathsf{pp}, \tau, \mathbb{x}) \to tr$, and returns the signature as (T, tr). $\mathcal{S}$ will record $\{(\cdot, e), T, \mathsf{pk}_o\}$ to the hash table.
- When $\mathcal{A}$ queries random oracle $\mathcal{H}_t$ on an input $x \in \mathcal{X}_k$ and $e \in \mathcal{X}_k$, $\mathcal{S}$ will check whether (x, e) is already in the hash table. If so, $\mathcal{S}$ responds to $\mathcal{A}$ according to this entry. $\mathcal{S}$ will also check whether $\mathcal{H}_k(x) = pk_o$ holds, and is there an entry $\{(\cdot, e), T, \mathsf{pk}_o\}$ in the hash table. If so, x will be returned by $\mathcal{S}$ as the one-wayness instance response and $\mathcal{S}$ will send T to $\mathcal{A}$. Otherwise, $\mathcal{S}$ samples $y \in \mathcal{Y}_k$ uniformly at random and sends to $\mathcal{A}$. $\mathcal{S}$ than adds $\{(x, e), y, \cdot\}$ to the hash table.

In the challenge phase, $\mathcal{A}$ outputs a set of unlinkable tuples $(e, \sigma_i, m_i, \mathsf{R}_i)$, where $\sigma_i = (\sigma_{s,i}, T_i)$ for $i \in [n]$. However, $\mathcal{A}$ can only make at most $n - 1$ queries, meaning that at least one of the secret keys used to generate the n linkable ring signatures is not the query output of $\mathcal{CO}$. There are two cases, either 1. $\mathcal{A}$ obtains a sk^* that corresponds to a pk^* never queried to $\mathcal{CO}$, or 2. $\mathcal{A}$ obtains a sk^* that the corresponding pk^* has been queried to $\mathcal{CO}$ with an output $sk' \neq sk^*$. Assume the advantage for $\mathcal{A}$ wining this game is adv_A, and $\mathcal{A}$ wins by case 1 with probability pr_A^1, $\mathcal{A}$ wins by case 2 with probability pr_A^2, such that $pr_A^1 + pr_A^2 = adv_A$.

Since $\mathsf{LRS.Verify}(e, \sigma_i, m_i, \mathsf{R}) = 1$ for $i \in [n]$, given the simulation extractability property of the SoK, we can use the extractor E to extract witnesses sk_i for $\{e, \sigma_i, m_i, \mathsf{R}\}$, $i \in [n]$. We use $sk^* \in \{sk_i\}_{i\in[n]}$ to represent the secret key that is not a query output of $\mathcal{CO}$. The probability for $\mathcal{A}$ wining in case 1, and $pk^* = pk_o$ is $\frac{q_j - q_r}{q_j} \cdot pr_A^1 \cdot \frac{1}{q_j - d + 1}$ which is non-negligible. In this case, $\mathcal{S}$ returns (sk^*, C) to the $\mathcal{H}_k$ one-wayness challenger. The probability for $\mathcal{A}$ wining in case 2, and $pk^* = pk_c$ is $\frac{q_j - q_r}{q_j} \cdot pr_A^2 \cdot \frac{1}{q_j}$ which is non-negligible. In this case, $\mathcal{S}$ returns (sk^*, C) to the $\mathcal{H}_k$ collision resistance challenger.

Theorem 2. *Our linkable ring signature is anonymous in the random oracle model if the underlying SoK is correct, simulatable and simulation extractable, and the underlying hash function is one-way and collision-resistant.*

Suppose there exists a Simulator $\mathcal{S}$ that plays the anonymity game with adversary $\mathcal{A}$ in Definition 4.

$\mathcal{S}$ generates public parameters $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$ and sends pp to $\mathcal{A}$. The hash functions $\mathcal{H}_k$ and $\mathcal{H}_t$ are modeled as random oracles.

For the oracle simulation, when $\mathcal{A}$ queries joining oracle $\mathcal{JO}$, $\mathcal{S}$ samples pk uniformly at random and returns it to $\mathcal{A}$. When $\mathcal{A}$ queries random oracle $\mathcal{H}_k$ and $\mathcal{H}_t$ on an input $x \in \mathcal{X}_k$, $\mathcal{S}$ will check whether x already in the hash table. If so, $\mathcal{S}$ responds to $\mathcal{A}$ according to this entry. Otherwise, $\mathcal{S}$ samples $y \in \mathcal{Y}_k$ uniformly at random and sends to $\mathcal{A}$. $\mathcal{S}$ than adds (x, y) to the hash table.

In the challenge phase, $\mathcal{A}$ chooses a set of public keys $\mathsf{R} = \{pk_i\}_{i\in[n]}$, an event-id e and a message m, then sends (R, e, m) to $\mathcal{S}$.

$\mathcal{S}$ constructs a Merkle root rt using the set of public keys R and samples tag $T \in \mathcal{Y}_t$ uniformly at random. $\mathcal{S}$ also picks $b \leftarrow\$ [1, \cdots, n]$.

Given $\mathbb{x} = \{rt, T, e, m\}$, $\mathcal{S}$ employs the simulator Sim in SoK to run $\mathsf{SimG}(1^\lambda) \rightarrow (pp, \tau)$, and $\mathsf{SimS}(\mathsf{pp}, \tau, \mathbb{x}) \rightarrow tr$, and sends the signature $\sigma = (tr, T)$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs b'.

Since the underlying SoK is simulatable, the simulated signature tr is computationally indistinguishable from the one in the original scheme. Moreover, tr is generated without witness and tag T is sampled uniformly random. The probability for $b' = b$ is $\frac{1}{n}$.

Theorem 3. *Our linkable ring signature is non-slanderable in the random oracle model if the underlying SoK is correct, simulatable and simulation extractable, and the underlying hash function is one-way and collision-resistant.*

If $\mathcal{A}$ is able to win the non-slanderability game defined in Definition 6 with a non-negligible probability, we can construct $\mathcal{S}$ to break the one-wayness of the hash function $\mathcal{H}_t$. For breaking one-wayness, on given a hash digest h_o, one is required to output x such that $h_o = \mathcal{H}_t(x)$.

$\mathcal{S}$ samples public parameters by running $\mathsf{pp} \leftarrow \mathsf{LRS.Gen}(1^\lambda)$ and sends pp to $\mathcal{A}$, where $\mathcal{H}_k$ is programmed as random oracle. For the Oracle simulation,

- $\mathcal{JO}(\perp)$: Whenever $\mathcal{A}$ queries to $\mathcal{JO}$, $\mathcal{S}$ samples $pk_i \leftarrow\$ \mathcal{Y}_k$ and returns it to adversary.
- $\mathcal{CO}(pk_i)$: $\mathcal{S}$ samples $sk_i \leftarrow\$ \mathcal{X}_k$ and programs random oracle $\mathcal{H}_k$ such that $pk_i = \mathcal{H}_k(sk_i)$. $\mathcal{S}$ returns sk_i and records $\{sk_i, pk_i, \cdot\}$ to the hash table.
- $\mathcal{SO}(\mathsf{R}, e, pk_i, m)$: when $\mathcal{A}$ queries $\mathcal{SO}$ on message m, event-id e, a list of public keys R and the public key for the signer pk, where $pk \in \mathsf{R}$. If pk has been queried to $\mathcal{CO}$, sign the message using $\mathsf{LRS.Sign}$. If pk has not been queried to $\mathcal{CO}$, $\mathcal{S}$ samples $sk_i \leftarrow\$ \mathcal{X}_k$ and programs random oracle $\mathcal{H}_k$ such that $pk_i = \mathcal{H}_k(sk_i)$. $\mathcal{S}$ then signs the message using $\mathsf{LRS.Sign}$. $\mathcal{S}$ returns the signature and records $\{sk_i, pk_i, \cdot\}$ to the hash table.
- When $\mathcal{A}$ queries random oracle $\mathcal{H}_k$ on an input $x \in \mathcal{X}_k$, $\mathcal{S}$ will check whether $\{x, \cdot, \cdot\}$ is already in the hash table. If so, $\mathcal{S}$ responds to $\mathcal{A}$ according to this entry. $\mathcal{S}$ will also check whether for the entry $\{\cdot, pk, (e, T)\}$ in the hash table, it has $\mathcal{H}_t(x, e) = T$. If so, (x, e) will be returned by $\mathcal{S}$ as the response to the one-wayness game and $\mathcal{S}$ will use pk to answer the query. Otherwise, $\mathcal{S}$ samples $y \in \mathcal{Y}_k$ uniformly at random and sends to $\mathcal{A}$. $\mathcal{S}$ than adds $\{x, y, \cdot\}$ to the hash table.

In the challenge phase, $\mathcal{A}$ sends a set of public keys $\mathsf{R} = \{pk_i\}_{i\in[n]}$, a public key pk, a message m and an event-id e to $\mathcal{S}$, where $pk \in \mathsf{R}$. $\mathcal{S}$ sets $T = h_o$ and sets $\mathbb{x} = \{rt, T, e, m\}$ where rt is the Merkle root generated from R. $\mathcal{S}$ then employs the simulator Sim in SoK to simulate $\mathsf{SimG}(1^\lambda) \rightarrow (pp, \tau)$, $\mathsf{SimS}(\mathsf{pp}, \tau, \mathbb{x}) \rightarrow tr$ and returns the signature as (T, tr). $\mathcal{S}$ will record $\{\cdot, pk, (e, T)\}$ to the hash table.

$\mathcal{A}$ outputs a list of public keys R', messsage m' and a signature $\sigma' = (\sigma'_s, T')$ where $\mathsf{LRS.Verify}(e, \sigma', m', \mathsf{R}') = 1$. In addition, pk should not be an input to $\mathcal{CO}$ and $\mathcal{SO}$.

Given the simulation extractability property of the SoK, we can extract witnesses sk' from σ' using the extractor E such that $T' = \mathcal{H}_t(sk', e)$. Since we have $\mathsf{LRS.Link}(e, \sigma, \sigma', m, m', \mathsf{R}, \mathsf{R}') = 1$ and $T' = T$, $\mathcal{S}$ then can return (sk', e) to the one-wayness game challenger.

Theorem 4. *Our linkable ring signature is unforgeable in the random oracle model if the underlying SoK is correct, simulatable and simulation extractable, and the underlying hash function is one-way and collision-resistant.*

Unforgeability is implied by linkability and non-slanderability.

4.3 Instantiation

In this section, we instantiate our framework using the Rescue-Prime hash function [41] and an SoK based on ethSTARK [42].

We choose a prime field $\mathbb{F}_p$ with $p = 2^{128} - 45 \cdot 2^{40} + 1$. In our construction, we use the Rescue-Prime hash function, referred to as $\mathcal{H}$, for both hash functions $\mathcal{H}_k$ and $\mathcal{H}_t$. This particular hash function is chosen because it is arithmetization-friendly, meaning it requires fewer operations in the underlying finite field than more complex hash functions like SHA3, making them more efficient within arithmetic circuits and is therefore well-suited for use in zero-knowledge proof systems. For clarity of reference throughout the rest of the paper, we will use the notation $\mathcal{H}$ to refer to both $\mathcal{H}_k$ and $\mathcal{H}_t$.

To achieve 128-bit security, we configure the Rescue-Prime hash function $\mathcal{H}$ to have a state width of $w_h = 6$. We also set the rate of the sponge construction of $\mathcal{H}$ to 2, meaning that $\mathcal{H} : \mathbb{F}_p^* \to \mathbb{F}_p^2$ outputs 2 field elements. Furthermore, we set the number of rounds in $\mathcal{H}$ to 7. Considering the key generation process within our system, given a signer's private key $sk_l \in \mathbb{F}_p$, we calculate the corresponding public key pk_l as $pk_l = \mathcal{H}(sk_l) \in \mathbb{F}_p^2$.

Construct SoK from ethSTARK. The current version of ethSTARK does not consider the zero-knowledge property. To adapt ethSTARK into an SoK, we first incorporate zero-knowledge properties into non-interactive ethSTARK to build a non-interactive zero-knowledge argument of knowledge. The details of this process are provided in the full version [**?**]. In non-interactive ethSTARK, the non-interaction property is achieved through the use of the Fiat-Shamir transformation. We denote the hash function utilized in this transformation as $\mathcal{H}_f$, which hashes a description of the statement and the public input. We argue that if $\mathcal{H}_f$ also takes the message $m \in \{0,1\}^*$ as input to generate the challenge, the resulting zero-knowledge non-interactive ethSTARK will result in a signature of knowledge $\mathsf{SoK_m}$ on m.

Note that STARK [7] is utilized to verify the computational integrity (CI) of the computation. To prove the following relation R_s, we transform the original statement into a CI statement that concerns the correctness of the computation of the procedure I and some additional constraints.

$$\mathcal{R}_s = \{((e, rt, T), (\boldsymbol{P}, l, sk_l)) : e, sk_l \in \mathbb{F}_p \wedge rt, T \in \mathbb{F}_p^2 \wedge l \in \{0,1\}^k \wedge$$
$$\boldsymbol{P} = \{P_i\}_{i \in [k]} \wedge P_i \in \mathbb{F}_p^2 \wedge T = \mathcal{H}_t(sk_l, e) \wedge rt = \mathsf{MPath}(\mathcal{H}_k(sk_l), \boldsymbol{P}, l)\}$$

We define the procedure I as in Algorithm 1.

Algorithm 1 Procedure $I((e, rt, T), (\boldsymbol{P}, l, sk_l))$

$T = \mathcal{H}_t(sk_l, e)$
$pk_l = \mathcal{H}_k(sk_l)$
$rt' = \mathcal{H}_k(pk_l)$
for $i \leftarrow 1, log(n)$ **do**
  if $l[i] == 1$ **then**
    $rt' \leftarrow \mathcal{H}_k(P_i, rt')$
  else
    $rt' \leftarrow \mathcal{H}_k(rt', P_i)$
  end if
end for

Construct execution trace. Having transformed the CI statement, we proceed to reduce it to an execution trace and a set of polynomial constraints, which involves constructing our hand-optimized representations of the program. Recall that an execution trace is a sequence of machine states with w registers that lasts for T states. The width of our execution trace table is $w = 8$, and we denote these registers as $r_1, \ldots, r_8$. To be more specific, each hash operation takes $N = 8$ rows and $w_h = 6$ columns in the trace table. We concatenate the traces of individual hashes, resulting in the total number of states $C = N \cdot log(n) + 3N$. However, the actual trace length must be a power of 2, we pad the trace to length T, which is the smallest power of 2 that is greater than C.

ethSTARK utilizes *periodic columns* to specify the periodic list of constants, which includes the round constants for the hash function. These periodic columns are available to the verifier and are not included in the execution trace as part of the witness. In our implementation, we construct periodic columns r_t, r_p. The cell values of r_t are set to 0 in every state except for S_1 and S_N which are 1. The cell values of r_p are 0 in state S_{bN} and 1 in other states, where $b \in [\frac{T}{N}]$. We present the execution trace table and the periodic columns in Table 3.

Table 3. Execution trace for our scheme. Set $l[1] = 0, l[2] = 1, l[i] = 1$.

	r_t	r_p	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
S_1	1	1	0	sk_l	sk_l	e	0	0	0	0
	0	1	0	sk_l	Hash					
S_8	1	0	0	sk_l	$T^{(0)}$	$T^{(1)}$	-	-	-	-
S_9	0	1	0	0	sk_l	1	0	0	0	0
	0	1	0	0	Hash					
S_{16}	0	0	0	0	$pk^{(0)}$	$pk^{(1)}$	-	-	-	-
S_{17}	0	1	0	0	$pk^{(0)}$	$pk^{(1)}$	0	0	0	0
	0	1	0	0	Hash					
S_{24}	0	0	0	0	$rt^{(0)}$	$rt^{(1)}$	-	-	-	-
S_{25}	0	1	$l[1]$	0	$rt^{(0)}$	$rt^{(1)}$	$P^{(0)}[1]$	$P^{(1)}[1]$	0	0
	0	1	0	0	Hash					
S_{32}	0	0	0	0	$rt^{(0)}$	$rt^{(1)}$	-	-	-	-
S_{33}	0	1	$l[2]$	0	$P^{(0)}[2]$	$P^{(1)}[2]$	$rt^{(0)}$	$rt^{(1)}$	0	0
	0	1	0	0	Hash					
S_{40}	0	0	0	0	$rt^{(0)}$	$rt^{(1)}$	-	-	-	-
	...	...	...	...	...					
S_{25+8i}	0	1	$l[i]$	0	$P^{(0)}[i]$	$P^{(1)}[i]$	$rt^{(0)}$	$rt^{(1)}$	0	0
	0	1	0	0	Hash					
S_{32+8i}	0	0	0	0	$rt^{(0)}$	$rt^{(1)}$	-	-	-	-
	...	...	...	...	...					
S_C	0	0	0	0	$rt^{(0)}$	$rt^{(1)}$	-	-	-	-
	...	...	...	...	...					
S_T	0	0	...	...	...					

Represent constraints in polynomial form. As defined in [7], we have two types of constraints, where the transition constraints guarantee that every pair of successive states in the trace table meets the constraints specified by the computation, and the boundary constraints ensure that the values of particular cells in the trace table are equal to the given values.

We denote r_i as the current state and r'_i as the next state of the register. Let $f_{R^{XLIX}}(\cdot,\cdot)$ be a function that captures a single round of rescue permutation. Denote the trace cell value in register i, state j as $r_{i,j}$. For the transition constraints, we require that all hash operations are executed correctly, both the tag and public key calculations be performed using the same private key, and that the Merkle tree reconstruction be performed using the computed public key from the previous step. We enforce the transition constraints on every row as:

(i) Vector l is a bit string: $(r_p - 1) \cdot r'_1 \cdot (r'_1 - 1) = 0$.
(ii) Value sk_l in cell $r_{2,1}$ is the same in $r_{3,1}$: $r_p \cdot r_t \cdot (r_2 - r_3) = 0$.
(iii) Value sk_l in register r_2 is the same from S_1 to S_8: $r_p \cdot (r_2 - r'_2) = 0$.
(iv) Value sk_l in cell $r_{2,8}$ is the same in $r_{3,9}$: $(r_p - 1) \cdot r_t \cdot (r_2 - r'_3) = 0$.
(v) Merkle root reconstruction is computed correctly:
$(r_p - 1) \cdot (r_t - 1) \cdot r'_1 \cdot (r_3 - r'_5) \cdot (r_4 - r'_6) = 0$,
$(r_p - 1) \cdot (r_t - 1) \cdot (r'_1 - 1) \cdot (r_3 - r'_3) \cdot (r_4 - r'_4) = 0$.
(vi) All Rescue-XLIX permutations are computed correctly:
$r_p \cdot f_{R^{XLIX}}(r'_i, r_i) = 0$ for $i \in [3, 8]$.

For the boundary constraints, we require the message m, tag T, e and the reconstructed root rt' to be the claimed value, where the verifier checks whether $rt' = rt^{(0)}, rt^{(1)}$ is the same as the root rt computed in the preprocessing phase. We place the boundary constraints: (i) $r_{4,1} = e$. (ii) $r_{3,8}, r_{4,8} = T^{(0)}, T^{(1)}$. (iii) $r_{3,C}, r_{4,C} = rt^{(0)}, rt^{(1)}$. After interpreting each column in the trace table as a polynomial over the trace evaluation domain, we have the witness and constraints in polynomial form. If the computation is honest, the execution trace will satisfy all constraints.

4.4 Efficiency Analysis

We evaluate the performance on an i9-12900k CPU with 64GB RAM. For parameter settings in ethSTARK with 128-bit security, we set the number of queries to be 32, the blowup factor to be 16, the folding factor to be 16, and the grinding factor to be 20. The parameters of 99-bit security are the same as those of 128-bit, except that the number of queries is 20. Specifically, the number of queries indicates the number of queries to the FRI protocol in ethSTARK. The higher the number of queries, the higher the security level, but at the cost of larger proofs and greater complexity for both the prover and verifier. For a fixed security level, increasing the blowup factor would increase the prover time while reducing the proof size and verification time. A higher grinding factor increases the computational expense for a malicious prover to generate a false proof, thereby contributing to greater security but also demanding more time from the prover.

Table 4. Performance measurements of our scheme with 128-bit security bits.

Number of users	2^3	2^6	2^{10}	2^{11}	2^{12}	2^{13}
Online signing time	25 ms	47 ms	48 ms	51 ms	48 ms	50 ms
Online verifying time	0.3 ms	0.3 ms	0.3 ms	0.3 ms	0.3 ms	0.3 ms
Total signing time	25 ms	54 ms	160 ms	277 ms	497 ms	949 ms
Total verifying time	0.3 ms	7 ms	112 ms	226 ms	449 ms	899 ms

Computational Efficiency. Our scheme has a linear offline time and $O(\text{polylog}(\log n))$ online verifier time in the ring size n. As shown in Figure 2b, our verification time is primarily influenced by the linear offline preprocessing phase, which accounts for 99% of the total time for a ring size of 2^8, and the portion of preprocessing time increases as the ring size grows. In particular, for a ring size of 2^{13}, our online verification only takes 0.3 ms, whereas the offline preprocessing takes 899 ms. Thus, we recommend applying our signature to applications with static rings so that only one offline processing is required. We present our evaluation in table 4. Similarly to the signature size, the efficiency of the online signing and verification processes is influenced by the length of the execution trace. Therefore, when the ring size lies in a certain range, the online performance remains consistent.

Signature Size. Our scheme exhibits a signature size that scales $O(\text{polylog}(\log n))$ with the ring size n. We present a size comparison of our scheme with other schemes in Table 2. In practical scenarios, our performance is significantly influenced by two factors: the choice of the SoK and the choice of the hash function $\mathcal{H}$. For instance, if we adopt GMiMC [3], the offline preprocessing time is faster, but the proof size becomes larger compared to adopting the Rescue Prime hash [41]. This discrepancy arises because GMiMC itself is faster, but it requires more permutation rounds for implementation.

Acknowledgements. This work is partially supported by the General Research Fund of the Research Grant Council of Hong Kong (Project No.: 17201421, 15211120) and The Hong Kong Polytechnic University (Project No.: A0048350, P0046340, A0044374).

References

1. 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: 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings 6*, pages 1–25. Springer, 2013.
2. 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: 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings 23*, pages 558–576. Springer, 2018.

3. M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger. Feistel structures for mpc, and more. In *ESORICS 2019*, pages 151–171. Springer, 2019.
4. M. H. Au, J. K. Liu, W. Susilo, and T. H. Yuen. Secure id-based linkable and revocable-iff-linked ring signature with constant-size construction. *Theoretical Computer Science*, 469:1–14, 2013.
5. C. Baum, H. Lin, and S. Oechsner. Towards practical lattice-based one-time linkable ring signatures. In *Information and Communications Security: 20th International Conference, ICICS 2018, Lille, France, October 29-31, 2018, Proceedings*, pages 303–322. Springer, 2018.
6. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *ICALP 2018*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
7. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, 2018.
8. W. Beullens, S. Katsumata, and F. Pintore. Calamari and falafl: logarithmic (linkable) ring signatures from isogenies and lattices. In *ASIACRYPT 2020*, pages 464–492. Springer, 2020.
9. J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, and C. Petit. Short accountable ring signatures based on ddh. In *ESORICS 2015*, pages 243–265. Springer, 2016.
10. J. Bootle, K. Elkhiyaoui, J. Hesse, and Y. Manevich. Dualdory: Logarithmic-verifier linkable ring signatures through preprocessing. In *ESORICS 2020*, pages 427–446. Springer, 2022.
11. X. Boyen and T. Haines. Forward-secure linkable ring signatures. In *ACISP 2018*, pages 245–264. Springer, 2018.
12. Z. Brakerski and Y. T. Kalai. A framework for efficient signatures, ring signatures and identity based encryption in the standard model. Cryptology ePrint Archive, Paper 2010/086, 2010. https://eprint.iacr.org/2010/086.
13. M. Chase and A. Lysyanskaya. On signatures of knowledge. In *Advances in Cryptology-CRYPTO 2006: 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006. Proceedings 26*, pages 78–96. Springer, 2006.
14. D. Chaum and E. Van Heyst. Group signatures. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pages 257–265. Springer, 1991.
15. L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. A. Perlner, and D. Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016.
16. S. S. Chow, J. K. Liu, and D. S. Wong. Robust receipt-free election system with ballot secrecy and verifiability. In *NDSS*, volume 8, pages 81–94, 2008.
17. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal gaussians. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 40–56. Springer, 2013.

18. L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.
19. M. F. Esgin, R. Steinfeld, J. K. Liu, and D. Liu. Lattice-based zero-knowledge proofs: new techniques for shorter and faster constructions and applications. In *Annual International Cryptology Conference*, pages 115–146. Springer, 2019.
20. M. F. Esgin, R. Steinfeld, A. Sakzad, J. K. Liu, and D. Liu. Short lattice-based one-out-of-many proofs and applications to ring signatures. In *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*, pages 67–88. Springer, 2019.
21. M. F. Esgin, R. Steinfeld, and R. K. Zhao. Matrict+: More efficient post-quantum private blockchain payments. In *IEEE S&P 2022*, pages 1281–1298. IEEE, 2022.
22. E. Fujisaki and K. Suzuki. Traceable ring signature. In *PKC 2007*, pages 181–200. Springer, 2007.
23. L. Goldberg, S. Papini, and M. Riabzev. Cairo–a turing-complete stark-friendly cpu architecture. *Cryptology ePrint Archive*, 2021.
24. J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *EUROCRYPT 2015*, pages 253–280. Springer, 2015.
25. V. Guruswami and M. Sudan. Improved decoding of reed-solomon and algebraic-geometric codes. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 28–37. IEEE, 1998.
26. M. Hu and Z. Liu. Lattice-based linkable ring signature in the standard model. *Cryptology ePrint Archive*, 2022.
27. I. Khaburzaniya, K. Chalkias, K. Lewi, and H. Malvai. Aggregating and thresholdizing hash-based signatures using starks. In *ACM CCS 2022*, pages 393–407, 2022.
28. 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 *EUROCRYPT 2016*, pages 1–31. Springer, 2016.
29. J. K. Liu, M. H. Au, W. Susilo, and J. Zhou. Linkable ring signature with unconditional anonymity. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):157–165, 2013.
30. J. K. Liu, V. K. Wei, and D. S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *ACISP*, volume 4, pages 325–335. Springer, 2004.
31. J. K. Liu and D. S. Wong. Linkable ring signatures: Security models and new schemes. In *ICCSA 2005*, pages 614–623. Springer, 2005.
32. Z. Liu, K. Nguyen, G. Yang, H. Wang, and D. S. Wong. A lattice-based linkable ring signature supporting stealth addresses. In *ESORICS 2019*, pages 726–746. Springer, 2019.
33. X. Lu, M. H. Au, and Z. Zhang. (linkable) ring signature from hash-then-one-way signature. In *IEEE TrustCom 2019*, pages 578–585, 2019.
34. X. Lu, M. H. Au, and Z. Zhang. Raptor: a practical lattice-based (linkable) ring signature. In *ACNS 2019*, pages 110–130. Springer, 2019.
35. V. Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT 2009*, pages 598–616. Springer, 2009.
36. V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT 2012*, pages 738–755. Springer, 2012.
37. V. Lyubashevsky, N. K. Nguyen, and G. Seiler. Smile: set membership from ideal lattices with applications to ring signatures and confidential transactions. In *Annual International Cryptology Conference*, pages 611–640. Springer, 2021.

38. S. Noether, A. Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.
39. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT 2001*, pages 552–565. Springer, 2001.
40. A. Scafuro and B. Zhang. One-time traceable ring signatures. In *ESORICS 2021*, pages 481–500. Springer, 2021.
41. A. Szepieniec, T. Ashur, and S. Dhooghe. Rescue-prime: a standard specification (sok). Cryptology ePrint Archive, Paper 2020/1143, 2020. https://eprint.iacr.org/2020/1143.
42. S. Team. ethstark documentation. *IACR Cryptol. ePrint Arch.*, 2021:582, 2021.
43. P. P. Tsang and V. K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In *ISC 2005*, pages 48–60. Springer, 2005.
44. P. P. Tsang, V. K. Wei, T. K. Chan, M. H. Au, J. K. Liu, and D. S. Wong. Separable linkable threshold ring signatures. In *INDOCRYPT 2004*, pages 384–398. Springer, 2005.
45. X. Wang, Y. Chen, and X. Ma. Adding linkability to ring signatures with one-time signatures. In *ISC 2019*, pages 445–464. Springer, 2019.
46. R. Yang, M. H. Au, J. Lai, Q. Xu, and Z. Yu. Lattice-based techniques for accountable anonymity: composition of abstract stern's protocols and weak prf with efficient protocols from lwr. *Cryptology ePrint Archive*, 2017.
47. T. H. Yuen, M. F. Esgin, J. K. Liu, M. H. Au, and Z. Ding. Dualring: generic construction of ring signatures with efficient instantiations. In *CRYPTO 2021*, pages 251–281. Springer, 2021.
48. H. Zhang, F. Zhang, H. Tian, and M. H. Au. Anonymous post-quantum cryptocash. In *FC 2018*, pages 461–479. Springer, 2018.

A Add zero-knowledge to ethSTARK

We informally describe the process of incorporating zero-knowledge into ethSTARK using the same approach as in ZK-STARK [7]. The intuition behind this is to randomize trace polynomials and add random mask polynomials. As introduced in Section 2.3, let w denote the trace width, T denote the trace length and z denote the ZK parameter. Let $\mathbb{F}_q$ be the finite field, s be the number of constraints, and d be the maximal degree of the constraints, we have:

- $\mathbb{K}$: finite extension of $\mathbb{F}_q$, with size q^e and $e \geq 1$.
- H_0: trace evaluation domain, which is a multiplicative subgroup of $\mathbb{F}_p^\times$ of size T, generated by generator g.
- D: evaluation domain, which is a nontrivial coset of a multiplicative group $D_0 \subset \mathbb{K}^*$, where $H_0 \subset D_0$, and $D \subset \mathbb{K}^*$ is disjoint from H_0.
- $I \subset \{1, \ldots, w\} \times \{0, \ldots, T-1\}$: a set of mask indices, where $Y = \{Y_{i,j} : (i,j) \in I\}$ is a set of mask variables that is indexed by elements of I.
- $C_1, \ldots, C_s$: a set of constraints, each constraint is an ordered pair $C_i = (Q_i, H_i)$, where
  - ⋄ $Q_i \in \mathbb{F}^{\leq d}[Y]$: i-th constraint polynomial, which is a multivariate polynomial over the mask variables.
  - ⋄ $H_i \subseteq H_0$: i-th constraint enforcement domain, which is a subset of the trace domain.

For prover P and verifier V, the protocol provides them with an instance $(\mathbb{F}_q, w, d, s, g, I, \{C_i\}_{i\in[s]})$ and auxiliary interactive oracle proofs (IOP) parameters $(\mathbb{K}, e, D, \mathsf{aux_{FRI}})$, where $\mathsf{aux_{FRI}}$ is auxiliary information required by the FRI protocol.

For the definition of completeness, soundness, knowledge soundness, and zero knowledge, we refer to the original papers [42,7].

Description of the zero-knowledge protocol:

1. **Prover sends execution trace oracle**:
   - With a $w \times T$ execution trace table, for $i \in [w]$, P interprets each trace column as a trace polynomial $P_i : H_0 \to \mathbb{F}_q$ of degree smaller than T.
   - For $i \in [w]$, P draw a uniformly random polynomial $P'_j(X)$ for degree less than $z+T$ such that for every $y \in H_0$ it satisfies $P'_j(y) = P_j(y)$. P evaluates each P'_i on D to generate oracle functions $f_1, \dots, f_w : D \to \mathbb{K}$, and sends them to V.
2. **Prover sends constraint oracles**:
   - V samples and sends randomness $R = (\alpha_1, \alpha'_1, \dots, \alpha_s, \alpha'_s) \leftarrow\!\$\ \mathbb{K}^{2s}$ to P.
   - Given constraints $C_1, \dots, C_s$, P replaces variables Y in multivariate constraint polynomial Q_j with trace polynomial values that satisfy the assignment to get a univariate polynomial $(Q_j \circ \overrightarrow{P})(X)$, where $\overrightarrow{P} = (P_1, \dots, P_w)$ such that $\forall i \in [s] : x \in H_i \Rightarrow (Q_j \circ \overrightarrow{P})(x) = 0$. P additionally samples a random polynomial $R_0(X) \in \mathbb{F}_q[X]$, and then calculates the random linear combination of the constraint polynomials as

     $$C'(X) = \sum_{j=1}^{s} (\alpha_j + \alpha'_j \cdot X^{e_j}) \cdot \frac{(Q_j \circ \overrightarrow{P})(X)}{Z_{H_j}(X)} + R_0(X), \tag{1}$$

     let d_j be the degree of polynomial $(Q_j \circ \overrightarrow{P})(X)/Z_{H_j}(X)$, and d_{max} be the smallest integral power of 2 that is strictly greater than $max_{j\in[s]}\mathsf{d}_j$, we have the degree correction parameter $e_j = \mathsf{d}_{max} - \mathsf{d}_j - 1$.
   - Instead of representing the constraint composition polynomial $C'(X) : H_0 \to \mathbb{F}_q$ as polynomial with degree smaller than d_{max}, P represents it as m polynomials $C'_1(X), \dots C'_m(X)$ of degree smaller than $z+T$, such that

     $$C'(X) = \sum_{k=1}^{m} X^{k-1} \cdot C'_k(X^m). \tag{2}$$

   - P evaluates $R_0(X)$ and each $C'_k(X)$ on D to generate oracle functions $r_0, c_1, \dots, c_d : D \to \mathbb{K}$, and sends them to V.
3. **Verification**:
   - V samples and queries $z \leftarrow\!\$\ \mathbb{K}^* \backslash (H_0 \cup \bar{D})$, where $\bar{D} = \{y \in \mathbb{K}^* : y^m \in D\}$.
   - P responds with $f_1(z), f_1(gz), \dots, f_w(z), f_w(gz), c_1(z), \dots, c_m(z), r_0(z)$.

– V calculates $C'(z)$ using $f_1(z), \ldots, f_w(gz), r_0(z)$ and constraints $\{C_j\}_{j\in[s]}$. V then checks $\sum_{k=1}^{m} z^{k-1} \cdot c_k(z^m) \stackrel{?}{=} C'(z)$.
– V samples and sends randomness $\{\gamma_i\}_{i\in[1,2w+m]} \leftarrow\$ \mathbb{K}$.
– P additionally samples random polynomial $R_1(X) \in \mathbb{F}_p[X]$ with $\deg(R_1) < z + T$, and sends oracle function $r_1 : D \to \mathbb{K}$ to V.
– P computes the oracle function $g_\gamma : D \to \mathbb{K}$ for DEEP composition polynomial as

$$\begin{aligned} g_\gamma(x) = &\sum_{j=1}^{w} \left(\gamma_j \frac{f_j(x) - f_j(z)}{x - z} + \gamma_{j+w} \frac{f_j(x) - f_j(gz)}{x - gz} \right) \\ &+ \sum_{l=1}^{m} \gamma_{l+2w} \frac{c_l(x) - c_l(z)}{x - z^m} + r_1(x). \end{aligned} \tag{3}$$

– P and V run FRI for g_γ over domain D.

Intuitive explanation for the proof of zero-knowledge. We denote the simulator as Sim. Given a verifier V', the simulator Sim operates as follows:

1. Sim invokes V' and records the first message, which is the randomness R provided by V'. Sim then instantiates a sub-prover P', such that all further messages and queries from V' to FRI protocol are handled by Sim through the invocation of P'.
2. Next, Sim samples uniformly random functions $f_1, \ldots, f_w, c_1, \ldots, c_m, g \in \mathbb{F}[X]$ of degree smaller than $z + T$. It continues to run V' and responds to queries in the following manner:
   – For queries on $f_1, \ldots, f_w$ at a point $x_0 \in D$, Sim returns $\{f_i(x_0)\}_{i\in[w]}$ and $\{f_i(x_0 g)\}_{i\in[w]}$.
   – For queries on $c_1, \ldots, c_m$ at a point $x_0 \in D$, Sim computes $\{c_i(x_0)\}_{i\in[m]}$ and thus determines the value of $C'(x_0)$ through Equation 2. Observe that $f_1(x_0), f_1(x_0 g), \ldots, f_w(x_0), f_w(x_0 g)$ are fixed, implying that the right-hand side of Equation 1, except for the term $R_0(x_0)$, is also fixed. Given these values, Sim determines $R_0(x_0)$ as the unique field element that satisfies the linear constraint.
   – Similarly, for queries on g at a point $y_0 \in D$, Sim computes $g(y_0)$ such that both sides of Equation 3, except for the term $r_1(y_0)$, are fixed. Consequently, Sim determines $r_1(y_0)$ as the unique field element that satisfies the linear constraint.

In the honest prover's execution, the functions $(f_1, \ldots, f_w, r_0, r_1)$ are sampled uniformly and independently. The functions $(c_1, \ldots, c_m, g)$ are then computed based on the sampled functions $\{f_i\}_{i\in[w]}$ and r_0, r_1 according to Equations 1, 2 and 3. The distribution of messages exchanged between the verifier V' and the sub-prover P', which is part of the FRI protocols, is designed to be identical to the distribution provided by the simulator Sim. This is because both the honest prover and Sim provide P' with the same uniformly random input polynomial g. As a result, the distribution of transcripts generated by the simulator Sim interacting with the verifier V' is indistinguishable from the distribution of transcripts generated by an honest prover interacting with V'.

Reference for the proof of Soundness and Knowledge Soundness. To prove the Knowledge Soundness, an extractor is introduced to extract a valid witness. The extractor in [7,42] operates by running the Guruswami–Sudan list decoding algorithm [25] to obtain candidate codewords, and checking each candidate to find a satisfying witness. Our protocol adds zero-knowledge using the same techniques as those in ZK-STARK [7]. For more details on the soundness and knowledge soundness proofs, please refer to the ZK-STARK and ethSTARK paper [42]. These works provide a comprehensive analysis of the soundness and knowledge soundness of the protocol, which can be adapted to our setting.