Efficient Set Membership Proofs using MPC-in-the-Head

Aarushi Goel[1], Matthew Green[1], Mathias Hall-Andersen[2], and Gabriel Kaptchuk[3]

[1]Johns Hopkins University, {aarushig,mgreen}@cs.jhu.edu
[2]Aarhus University, ma@cs.au.edu
[3]Boston University, kaptchuk@bu.edu

Abstract

Set membership proofs are an invaluable part of privacy preserving systems. These proofs allow a prover to demonstrate knowledge of a witness w corresponding to a secret element x of a public set, such that they jointly satisfy a given NP relation, *i.e.* $\mathcal{R}(w, x) = 1$ and x is a member of a public set $\{x_1, \ldots, x_\ell\}$. This allows the identity of the prover to remain hidden, eg. ring signatures and confidential transactions in cryptocurrencies.

In this work, we develop a new technique for efficiently adding logarithmic-sized set membership proofs to any MPC-in-the-head based zero-knowledge protocol (Ishai et al. [STOC'07]). We integrate our technique into an open source implementation of the state-of-the-art, post quantum secure zero-knowledge protocol of Katz et al. [CCS'18]. We find that using our techniques to construct ring signatures results in signatures (based only on symmetric key primitives) that are between 5 and 10 times smaller than state-of-the-art techniques based on the same assumptions. We also show that our techniques can be used to efficiently construct post-quantum secure RingCT from only symmetric key primitives.

1 Introduction

Zero-knowledge proofs and arguments of knowledge[1][GMW91] allow a prover to convince a verifier that they possess a witness for an NP statement, without revealing anything about the witness itself. The flexibility and power of zero-knowledge protocols have made these primitives a key building block in larger privacy-preserving protocols, supporting a wide array of practical applications. This has spurred interest in the development of more concretely efficient protocols that can improve the performance characteristics of deployed applications.

An important sub-component of many practically useful NP statements is *set membership*. Proving set membership consists of demonstrating that a secret element x is a member of a publicly-known set $\{x_1, x_2, \ldots, x_\ell\}$, *i.e.* there exists a $1 \leq j \leq \ell$ such that $x = x_j$. This sub-component allows a prover to then demonstrate some property of the value x without revealing *which* element x is. The prover will then prove that it has some witness w such that $\mathcal{R}(x, w) = 1$, where $\mathcal{R}$ is a public NP relation. For example, ring signatures [RST01] can be formulated as a proof in which the signer's task is to convince the verifier that it knows a signature that verifies against some public key pk that is part of a "ring" of public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell\}$. In this case, the proof is composed of demonstrating membership of the chosen public key in the correct set and demonstrating that the prover knows a valid signature under this public key.

Membership proofs are an invaluable part of privacy preserving systems. Standard zero-knowledge proofs hide the witness from the verifier, but this may not be sufficient to hide the identity of the prover. Consider the case where it is well-known that only the prover holds a witness to a certain statement. In this case, any verifying proof can be directly linked back to the prover. A privacy-conscious prover can use membership

[1]In this work we will use the terms proofs and arguments interchangeably, as has become commonplace for practically oriented work in this area.

proofs to "hide in the crowd," proving that it knows a witness from a large set of relevant statements, each of which relates to someone else. Ring signatures are the quintessential example of this paradigm, as the verifier only learns that the signer is a member of a set. This anonymity property has recently been leveraged by privacy-focused cryptocurrencies, like Monero [Noe15] and ZCash [MGGR13], to protect the identities of payers, even when all transactions are recorded in a public ledger. Evaluating a function represented as a lookup table can also be represented as a set membership problem: selecting the correct output value o given an input value i from a set of mappings $\{(i_1, o_1), \ldots, (i_\ell, o_\ell)\}$.

Given the importance of set membership proofs, finding efficient set membership proving techniques is a critical research direction. While there are generic techniques to prove such statements, generic approaches often result in concretely inefficient constructions. To be of practical interest, membership proofs should satisfy the following properties: (1) The set membership must compose well with a larger zero-knowledge proof. This is because simply proving membership alone is not interesting; the prover will likely want to use the values in the set as input to some other arbitrary relation $\mathcal{R}$. (2) The communication complexity of set membership proofs must not grow quickly with the size of the set, as large sets are what provide provers meaningful anonymity. Ideally, the proof size should be logarithmic in the set size with small concrete constants. (3) Because of the use in blockchain applications, set membership proofs should be amenable to non-interactivity transformations like Fiat-Shamir [FS87]. And finally, (4) It would be desirable for any techniques to be plausibly post-quantum secure so that privacy cannot be violated by future adversaries with access to quantum computation. The recent NIST PQC competition has demonstrated the importance that new cryptographic techniques adapt to the post-quantum setting.

In this work we design a novel technique for seamlessly integrating membership proofs into generic MPC-in-the-head [IKOS07] zero-knowledge proofs. While memberships have a long research history, they have largely been overlooked in the context of efficient and versatile MPC-in-the-head protocols.[2]

The most significant benefit of focusing on MPC-in-the-head protocols is the ease with which the set membership proof can be integrated with another NP relation. Because the MPC protocol being used can easily operate over any domain, there is no need to "match" the set membership approach and the NP relation, which could result in either a less efficient membership proof or less efficient representation of the relation. Additionally, our membership proofs introduce only a logarithmic (in the size of the set) communication overhead to the cost of proving the relation circuit $\mathcal{R}$, they are Fiat-Shamir friendly, and post-quantum secure. Applying our techniques is highly efficient, producing the smallest post-quantum ring signatures from symmetric key primitives when used in conjunction with the signature scheme initially proposed by Chase et al. [CDG$^+$17].

Applications. We integrate our membership proof technique into an open source implementation of the MPC-in-the-Head protocol proposed by Katz *et al.* [KKW18]. We use our implementation to implement the smallest post-quantum ring signatures from symmetric key assumptions. The signature sizes are approximately 42KB + 1.5KB $\times \log(\ell)$, where ℓ is the ring size. This is dramatically smaller than previous ring-signature constructions from similar assumptions, and is competitive with lattice-based ring signature constructions (see Table 1 for a comparison of ring signature sizes).

Additionally, there has recently been a sequence of work showing how to leverage lattice-based cryptography to create post-quantum secure RingCT, the technique used to create privacy preserving cryptocurrency transactions for Monero [TSS$^+$18, TKS$^+$19, TSSK20, EZS$^+$19]. In our work, we show how to efficiently instantiate RingCT using our set membership proofs. Our construction is elegant, simple, and efficient, with the same asymptotic size as the lattice-based constructions, but different constants. Our construction also illustrates the benefits of an efficient set membership proof that naturally integrates with a generic zero-knowledge proof system: no technical difficulties arise when getting different parts of the statement to fit together. For instance, our construction requires no range proofs, which are necessary when subcomponents are arithmetizatized for compatibility.

[2]As we discuss later, Katz et al. [KKW18] is one of the few works to address this shortfall; they construct ring signatures using MPC-in-the-head by adding a Merkle tree membership check to the circuit, a sub-component that dominates the overall circuit size.

Ring size:	2^7	2^{10}	2^{13}	Assumption
Derler et al. [DRS18]	982 KB	1352 KB	1722 KB	Symmetric Key
Katz et al. [KKW18]	285 KB	388 KB	492 KB	Symmetric Key
This Work	**52 KB**	**56 KB**	**60 KB**	Symmetric Key
Ring size:	2^3	2^6	2^{12}	Assumption
Libert et al. [LLNW16]	52 MB	94 MB	179 MB	SIS
Torres et al. [TSS+18]	> 124 KB	> 900 KB	61 MB	Ring-SIS
Esgin et al. [ESLL19]	41 KB	58 KB	256 KB	M-LWE & M-SIS
Esgin et al. [EZS+19]	29 KB	34 KB	148 KB	M-LWE & M-SIS
Lyubashevsky et al. [LNS21]	< 16 KB	< 18 KB	< 19 KB	Ex-M-LWE & M-SIS
This Work	**46 KB**	**50 KB**	**59 KB**	Symmetric Key

Table 1: Performance of our post-quantum ring signature scheme compared to prior works, all for 128 bits of post-quantum security. The performance of the lattice-based signatures come from [ESLL19]; these bodies of literature use different benchmarks, so we include both. Our ring signatures outperform the best known work relying on symmetric key assumptions by a factor of 5 to 8 and is competitive with the best known lattice-based approaches.

1.1 Our Contributions

We now give an overview of the contributions we present in this work.

Efficient Set Membership Proofs using MPC-in-the-head. We present an efficient, novel set membership proof that can be used with any MPC-in-the-head [IKOS07] protocol. Our protocol introduces a input-independent pre-processing phase to the simulated MPC protocol (similar to the work of Katz *et al.* [KKW18][3]) in which the simulated parties prepare secret shares of the elements in the set. Composing a membership proof for a set of size ℓ with an MPC-in-the-head proof increases the communication complexity by only $O(\log(\ell))$ and increases computation complexity by $O(\ell)$.

Implementation. We integrate our techniques into Reverie [git20], an open source implementation of the Katz et al. [KKW18] MPC-in-the-head protocol. In our implementation, membership proofs only incur an additional communication overhead of around 1.5KB $\times \log(\ell) + |x|$ at 128 bits of post-quantum security, where $|x|$ is the size of the elements of the set.

Post-Quantum Ring Signatures. We use our modified version of Reverie to implement post-quantum ring signatures. Our ring signatures, based on the post-quantum signatures presented in [CDG+17, KKW18, KZ20], are the smallest known post-quantum ring signatures from symmetric key assumptions, and can be nearly an order of magnitude smaller than those based on the same assumptions generated in [KKW18].

Post-Quantum RingCT. We give a simple and elegant construction of post-quantum RingCT based on symmetric key primitives and MPC-in-the-head. Due to our efficient set membership proofs, we are able to make use of generic zero-knowledge and still arrive at an efficient construction. Our work diversifies the assumptions from which we now know efficient post-quantum RingCT, minimizing the reliance on lattice-based primitives. Our construction is more efficient than prior work for the most important kind of transaction: small numbers of inputs and outputs, but a large sender anonymity set. This offers a different tradeoff from prior work, which scales poorly with the size of the anonymity, but better with the numbers of inputs and outputs.

1.2 Technical Overview

As discussed earlier, we make use of the MPC-in-the-head paradigm introduced by Ishai et al. [IKOS07]. Since its inception, a sequence of papers including (but not limited to) ZKBoo [GMO16], ZKB++ [CDG+17], and the work of Katz *et al.* [KKW18] have developed new techniques in this regime to construct concretely

[3]See Section 1.3 for a full comparison of our work at that of Katz *et al.*

efficient zero-knowledge proofs. In this section, we begin by briefly recalling some of these techniques, and then discuss our main ideas.

MPC-in-the-Head. MPC-in-the-head [IKOS07] is a technique used for designing three-round, public-coin, zero-knowledge proofs of knowledge using MPC protocols (also called Σ-protocols). At a high level, the prover simulates an n-party MPC protocol Π virtually on the relation circuit $\mathcal{R}(x, \cdot)$, that has the statement hard-wired in it. The input to this MPC is the witness w (which is shared among the virtual parties) corresponding to the statement x, such that $\mathcal{R}(x, w) = 1$. In the first round of the protocol, the prover commits to the views of all the parties in this MPC execution. In the second round, an honest verifier then selects a random subset of the views to be opened. In the third round, the prover opens to the views of the selected players. The verifier then verifies that those views are consistent with each other and with an honest execution, where the output of Π is 1.

The computation and communication complexity of such a protocol depends on the complexity of the underlying MPC protocol which depends on the size of the relation circuit $|\mathcal{R}(x, \cdot)|$. Since this technique yields a public-coin honest verifier zero-knowledge (HVZK) protocol, it can be made non-interactive, in the Random Oracle Model via the Fiat-Shamir transformation [FS87]. This approach was first made practical using MPC protocols with very few parties — *e.g.* in ZKBoo [GMO16] and ZKB++ [CDG$^+$17] — which require many parallel iterations to achieve negligible soundness error.

Set Membership using MPC-in-the-head. A naïve use of this approach for set membership will result in a protocol where the prover simulates an MPC execution for the following circuit, that takes witness w as input:

$$(\mathcal{R}(x_1, w) = 1) \vee \ldots \vee (\mathcal{R}(x_\ell, w) = 1)$$

Here $x_1, \ldots, x_\ell$ are elements of the publicly known set and $\mathcal{R}$ is the relation circuit. Complexity of the resulting Σ-protocol in this case is dependent on $\ell \times |\mathcal{R}(\cdot, \cdot)|$, which is highly inefficient.

It would be far better to treat both w and x as part of the witness, so $\mathcal{R}$ need only be executed once. However, in this case, the chosen value of x remains hidden. To guarantee that x used in the proof is indeed valid, we can explicitly check its membership using the underlying MPC protocol. In other words, consider the following circuit that takes an element x and a witness w as input:

$$(\mathcal{R}(x, w) = 1) \wedge (x = x_1 \vee \ldots \vee x = x_\ell)$$

It is easy to see that the complexity of an MPC-in-the-head protocol for this relation is dependent on $|\mathcal{R}(\cdot, \cdot)| + \ell |x|$. While this is a significant asymptotic improvement, it still has a linear dependence on the size of the set. As discussed in the introduction, it would be ideal to design a protocol where the proof size is logarithmic in the size of the set.

Logarithmic Dependence on Size of the Set. Linear dependence on the size of the set in the previous approach is a result of the explicit membership check done inside the MPC protocol. A simple idea to optimize this check is to use known techniques such as using a Merkle hash to succinctly accumulate all the members of the set and using a logarithmic sized Merkle proof to prove membership. While the size of such a membership proof is asymptotically logarithmic in the size of the set, the computation and communication is concretely very high. This is because verifying a Merkle proof inside the MPC requires expressing the hash function as a circuit, adding a *significant* number of gates to the simulated MPC. This approach can easily dominate the communication cost of the overall protocol. For example, Katz et al. [KKW18] use this approach for constructing ring signatures; the circuitry to verify the Merkle proof in their construction is bigger than the actual relation circuit $\mathcal{R}(\cdot, \cdot)$ by a *multiplicative* factor of $\log(\ell)$.

Our Idea. To reduce the practical cost of this Merkle proof, we aim to move the Merkle proof verification *outside* the MPC protocol. In other words, use MPC-in-the-head approach to only prove that $\exists x, w$, such that $\mathcal{R}(x, w) = 1$ and give a separate (cleartext) Merkle proof to prove that $x \in \{x_1, \ldots, x_\ell\}$. This would accomplish two goals: First, the *communication complexity* of the resulting proof would no longer rely on the circuit representation of the hash function. Second, verifying the Merkle proof could be done directly on hardware, making both prover time and verification time faster.

However, in order to securely implement this idea, we need to overcome the following obstacles:

- **Soundness:** We need to tie the two proofs together to make sure that the same x is used in the Merkle proof as well as in the MPC-in-the-head protocol.
- **Zero-Knowledge:** Moreover, we need to make sure that the Merkle Proof does not leak x.

To overcome the first issue, recall that the inputs to the MPC in an MPC-in-the-head protocol are (secret) shared amongst the parties. The prover must convince the verifier that the shares of input x correspond to a valid sharing of one of the elements in the set $\{x_1, \dots, x_\ell\}$. Therefore, instead of simply accumulating the set $\{x_1, \dots, x_\ell\}$ using a Merkle tree (or any accumulator), the prover first computes shares of these set elements, accumulates these shares and commits to the accumulated value. It can then use the Merkle proof to show that the shares of x assigned to the parties whose views were opened in the MPC-in-the-head protocol are members of the accumulated values. While this idea helps create a dependency between the shares used inside the MPC and the Merkle proof, it does not guarantee that the shares committed inside the Merkle tree were indeed honestly computed shares of the set $\{x_1, \dots, x_\ell\}$.

We observe that the *privacy-free* nature of $\{x_1, \dots, x_\ell\}$ can be leveraged to use the well-known *cut-and-choose* technique to ensure that the prover computes the shares honestly. In more detail, the prover computes multiple copies of the Merkle tree. The verifier then chooses to open a subset of these. Because the verifier already knows $x_1, \dots, x_\ell$, they can check whether or not the x_i's were honestly secret shared and committed. The remaining task is to selectively assign shares of the correct x to the parties without revealing its index.

This can be solved by allowing the prover to accumulate the shares in a random order. In particular, the prover selects an independent random permutation for the secret sharings of $x_1, \dots, x_\ell$ for each iteration. This permutation is revealed during cut-and-choose, so the verifier can check that the shares are still honest. When the simulated parties now use shares from the actual x, the random permutation prevents the verifier from detecting the value to which the shares correspond.

Adapting To Concrete MPC-in-the-head Protocols. For the sake of simplicity, we kept the above discussion oblivious to the exact specifications of the MPC-in-the-head protocol and its underlying MPC protocol (*e.g.* privacy threshold, number of "input-bearing" parties, etc...). However, the particularities of concrete MPC-in-the-head protocols may impact the efficiency of the above techniques. As such, we design ways to ensure that our techniques are flexible and widely applicable.

In a generic MPC-in-the-head protocol, the prover shares the witness among the input-bearing parties using an *additive sharing.* The soundness argument of the resulting protocol relies on the verifier's ability to identify a cheating computational party, but is agnostic to the behavior of the input-bearing parties. Therefore, the number of input-bearing parties relative to the protocols privacy threshold is irrelevant. In our case, however, the verifier must check that the input parties act honestly with respect to their inputs. Thus, the soundness error also depends on the probability with which the verifier catches a misbehaving *input-bearing party.* Using additive secret sharing in this case results in very poor soundness, as a single malicious input party can effectively change the MPC inputs. To improve soundness, we secret share the statement x amongst the input-bearing parties using a *threshold secret sharing*, where the threshold is the same as the privacy threshold of the underlying MPC.[4] This allows the soundness error of our techniques to only depend on the soundness error of the underlying MPC-in-the-head protocol and on the failure probability of the cut-and-choose step.

Concrete MPC-in-the-head protocols often require different numbers of parallel repetition to amplify soundness. The number of repetitions, which we denote τ, can vary from 1 to a linear function of the security parameter. The most direct—if wasteful—way to extend the above approach for repetitions would be to run the above set membership idea separately for each of the τ repetitions. We instead propose a more flexible approach: the prover can independently decide the value of τ and the number of Merkle trees (say M) that it must generated for set membership, based on the soundness required in each phase. In order to minimize the failure probability of cut and choose, the verifier then randomly asks to open $M/2$ Merkle trees (to check for correctness). The prover can then run a separate online phase using each of the remaining $M/2$

[4]We however, note that this optimization can only be applied when the number of input-bearing parties is higher than the privacy threshold of the underlying MPC. While this assumption is not universal, it is extremely common among proposed MPC-in-the-head protocols.

unopened Merkle Trees. However, if $M/2 > \tau$ this approach leads to unnecessary repititions in the online phase. To avoid this, our idea is to use *multiple* Merkle trees for a single online phase execution and check that all selected values are the same in each Merkle Tree, within the MPC circuit of the underlying protocol. On the other extreme, if the number of unopened preprocessings remaining after the cut-and-choose phase is τ, a single preprocessing can be used for each MPC emulation.

Communication Complexity. Using some simple additional optimizations such as deriving the randomness used for secret sharing and for accumulation from a small seed using PRGs and using compressive commitments, the above techniques will yield a proof size[5] of $O(n|x| + \log(\ell)) + \mathtt{CC}_\Sigma(\mathcal{R})$, where n are the number of parties used in the emulation of the MPC protocol and $\mathtt{CC}_\Sigma(\mathcal{R})$ is the communication complexity of the underlying MPC-in-the-head protocol when used to prove the relation circuit $\mathcal{R}$. We discuss this in detail in Section 3. Finally, we note that while we described the high level idea using Merkle trees, we note that any accumulator can be used.

Round Complexity. The above approach results in a 5 round protocol, where the first two-rounds are used for cut-and-choose (henceforth, we will refer to the generation each set of shares for the cut-and-choose as a pre-processing) and the next three rounds are for executing the MPC-in-the-head protocol. However, note that this is still a public-coin proof and can be made non-interactive in the random oracle model using the Fiat-Shamir transform. We also note that in many circumstances it would be possible to use our approach to construct a three-round protocol in the plain model with slightly higher communication costs. We elaborate more on this in the technical sections.

Implementation and Applications. We integrate our membership proof techniques into Reverie, an open source, Rust implementation of Katz et al. [KKW18]. Our implementation supports both boolean and arithmetic circuits. We use this implementation to evaluate the concrete efficiency of our techniques in two contexts: post-quantum ring signatures and post-quantum RingCT.

We implement post-quantum ring-signatures based on the post-quantum signatures of [CDG$^+$17]. These signatures use only symmetric key primitives to achieve post-quantum security. In short, the signer's public key is $\mathsf{PRF}(K, 0)$ where K is the signer's private key. Signatures are non-interactive proofs that the signer knows K that are tightly bound to the signed message m. The size of our signatures can be seen in Table 1. For a signer set of 2^{13}, our signatures are only 60 KB, 8 times smaller than [KKW18] and 5 times smaller than [ESLL19].

Additionally, we present a simple construction of post-quantum RingCT [Noe15, NMRL16, SALY17, YSL$^+$20, TSS$^+$18, TKS$^+$19, EZS$^+$19, TSSK20] based on symmetric key primitives. Our technique replaces the ring signatures, balance proof, and range proofs common in prior approaches with a single zero-knowledge proof that provides assurances of all the necessary properties at once. We note that this approach is only efficient when the prover has access to an efficient set membership proof that can easily integrate into a larger proof. The resulting transaction sizes are competitive with prior work, while removing the need for lattice-based assumptions. Specifically, our construction is more efficient when there are few inputs and outputs, but the spender wants a large anonymity set.

1.3 Comparison to Prior Work

Comparison to Katz et al. [KKW18]. Katz et al. [KKW18] design an efficient MPC-in-the-head NIZK protocol and use it to build ring signatures. While [KKW18] made significant contributions to the state of MPC-in-the-head NIZKs, their work does not substantially contribute to the state-of-the-art in set membership (as this was not their goal). Their signatures are direct descendants of the NIZK-based signatures that Chase et al. [CDG$^+$17] constructed from symmetric-key primitives. To avoid linear dependence on the ring size, [KKW18] embed a Merkle tree inside the relation circuit, a technique established in prior work (*eg.* [ST99]). This non-black box use of the hash function produces a massive subcircuit that dwarfs the size of the base signature. Our approach is black box in the hash function, improving inefficiency. Another

[5]Here we are ignoring linear multiplicative factor in the security parameter for simplicity. A more detailed calculation of our complexity appears in Section 3

benefit of our work is that we are able to us standard hash functions with large multiplicative complexity (eg. SHA256) instead of a poorly analyzed hash function with low multiplicative complexity (Davies-Meyer on LowMC).

Set Membership Proofs. Camenisch et al. [CCs08] achieve set membership by having the verifier sign the set such that the prover need only show that it knows a signature over the element. More recently Benarroch et al. [BCFK19] build SNARKs for set membership and set non-membership proofs. To our knowledge, there has been no work looking at optimizing set membership proofs in the context of MPC-in-the-head techniques. Set membership is also a subset of disjunctive statements, for which there has recently been significant work on zero-knowledge [HK20, BMRS20, GGHAK21]. The key improvement of these works is that the communication complexity of the resulting proof system is proportional to the size of (the relation circuit corresponding to) the largest clause in the disjunction, but the proof computation and verification times in these systems still depend on the size of all the clauses.

Ring Signatures and RingCT. We focus on applying our set membership techniques to post-quantum ring signatures and post-quantum RingCT. Ring Signatures were initially proposed by Rivest, Shamir and Tauman [RST01] and have been the subject of a significant body of research [CWLY06, BK10, BCC$^+$15]. More recently, post-quantum ring signatures have been proposed from plausibly post-quantum assumptions like symmetric key primitives [DRS18, KKW18] and lattice-based assumptions [LLNW16, TSS$^+$18, ESLL19, EZS$^+$19, BCOS20, LNS21]. Post-quantum RingCT has also been the focus of recent work [TSS$^+$18, TKS$^+$19, EZS$^+$19, TSSK20] as enthusiasm for post-quantum cryptography grows. We give a more in-depth comparison of our techniques in Section 6.

Generic Zero Knowledge. There has been a tremendous amount of recent work on concretely efficient zero-knowledge proof systems, *e.g.* [JKO13, GGPR13, Gro16, GMO16, CDG$^+$17, KKW18, BBB$^+$18, AHIV17, BCR$^+$19, MBKM19, COS19, XZZ$^+$19, Set20]. Broadly, these constructions can be classified either as (1) systems where the proof is succinct i.e., sublinear in the length of the witness (e.g. ZK-SNARKs [PHGR13, GGPR13, Gro16]) and (2) systems where the proof size is a function of the size of the relation circuit (e.g. [JKO13, GMO16, CDG$^+$17, KKW18, AHIV17]). Succinct proof systems generally suffer from prohibitively large computation times. On the other hand, concretely efficient non-succinct proof systems have communication and computation complexity that are both dependent on the size of the relation circuit. This is prohibitive for set membership proofs, as membership is a heavy-weight primitive when explicitly represented as a circuit.

2 MPC-in-the-Head Based Σ-Protocols

In this section, we recall the template of an MPC-in-the-Head Σ-protocol [IKOS07] and establish some notation that will be useful in future sections.

Let Π be an n-party MPC protocol. Let $\mathcal{P}$ be the set of parties. Let $\mathcal{V}_i$ be the view of a party $P_i \in \mathcal{P}$ in Π. Let $\mathcal{P}_{\mathsf{inp}} \subseteq \mathcal{P}$ be the set of input-bearing parties in this MPC. Let $\mathcal{C}$ be the set of all admissible corrupt party sets. Let $\mathcal{L}$ be an NP-language and $\mathcal{R}$ be the corresponding NP-relation. An MPC-in-the-head style zero-knowledge protocol, where both parties get a statement $x \in \mathcal{L}$ and the prover gets a witness w, s.t. $\mathcal{R}(x, w) = 1$ proceeds as follows:

- **Round 1:** The prover additively secret shares w amongst the virtual parties in $\mathcal{P}_{\mathsf{inp}}$. Let $\{w_i\}_{i\in\mathcal{P}_{\mathsf{inp}}}$ be this set of shares. It then emulates an execution of Π in its head for the following function: $\mathcal{F}(\{w_i\}_{i\in\mathcal{P}_{\mathsf{inp}}}) = \mathcal{R}(x, \bigoplus_{i\in\mathcal{P}_{\mathsf{inp}}} w_i)$. Let $\{\mathcal{V}_i\}_{i\in[n]}$ be the views of the respective parties in this execution. The prover computes $\mathsf{Com} = \mathsf{Commit}(\{\mathcal{V}_i\}_{i\in[n]})$ and sends Com to the verifier. Depending on the efficiency requirements, Commit here could be any generic or a specific tailor-made commitment scheme.

- **Round 2:** The verifier randomly selects a set of parties from the set of all admissible corrupt party sets, *i.e.*, $\mathcal{I} \in \mathcal{C}$ and sends it to the prover.

- **Round 3:** The prover sends an opening Open to the verifier, that would enable the verifier to *only* obtain the $\{\mathcal{V}_i\}_{i\in\mathcal{I}}$.

- **Verify:** The verifier computes $\{\mathcal{V}_i\}_{i\in\mathcal{I}}$ using Com and Open. It then checks if these views are consistent amongst each other. If the checks succeed, the verifier outputs 1, else it outputs 0.

Zero-Knowledge. In order to argue the zero-knowledge property, we must show existence of a PPT simulator that outputs a transcript (without knowledge of the witness w) that is indistinguishable from the transcript obtained in a real protocol. The simulator works by sampling a random $\mathcal{I} \in \mathcal{C}$. Assuming this set to be the set of corrupt parties, It then uses the simulator of the underlying MPC protocol to simulate views $\{\mathcal{V}_i\}_{i\in\mathcal{I}}$ for this set of parties using random inputs $\{w_i\}_{i\in\mathcal{I}\cap\mathcal{P}_{\mathsf{inp}}}$. It sets $\mathcal{V}_i = 0$ for the remaining parties $i \notin \mathcal{I}$. It then proceeds to commit to all these views and compute an opening that only enables recovery of $\{\mathcal{V}_i\}_{i\in\mathcal{I}}$. Indistinguishability of this simulated transcript from the real transcript follows from privacy of the underlying MPC protocol and from the hiding property of the commitment scheme, that ensures that the views of the parties $i \notin \mathcal{I}$ are not leaked.

Soundness. Soundness relies heavily on the corruption threshold and the robustness guarantees of the underlying MPC protocol. Informally speaking, robustness puts an upper-bound on the number of parties that are allowed to act maliciously, such that they are unable to impact the output of the protocol. For instance, a semi-honest protocol may or may not be robust against any number of players, but a maliciously secure protocol is bound to be robust against at least t-parties, where t is the number of corrupt parties against which it also guarantees privacy. Therefore, depending on the security of the underlying MPC protocol Π, a single iteration of the above protocol may or may not yield negligible soundness error. In those cases, the above protocol may need to be repeated to amplify soundness. The verifier in that case accepts only if the outcome of all repetitions is 1. We refer the reader to [IKOS07] for a more detailed proof and discussion.

3 Set Membership using MPC-in-the-head

In this section, we describe our generic compiler that transforms any MPC-in-the-head style Σ-protocol into a five round public coin protocol for set membership with only an additive overhead in the communication complexity that is logarithmic in the size of the set. As discussed in the introduction, we first present an optimized version for the case where the number of input parties in the underlying MPC protocol is more than its privacy threshold and later discuss how it can be extended for any MPC-in-the-head protocol.

Notation and Building Blocks. Let κ be a tunable security parameter. We use $\|$ to denote concatenation. We will use ϕ to denote a permutation. We will denote the number of parallel repetitions of the preprocessing (*i.e.*, generation of shares of the set) by M. Let $\{x_1, \ldots, x_\ell\}$ be the publically known set and $\mathcal{R}$ be an NP-relation. The prover wants to convince the verifier that there exists an element $x \in \{x_1, \ldots, x_\ell\}$ and a witness w such that $\mathcal{R}(x, w) = 1$. Let x_α denote the active element in the set, *i.e.*, the element for which the prover has a corresponding witness.

Our protocol uses the following primitives: (1) Pseudorandom generator $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^{\mathsf{poly}(\kappa)}$ (used implicitly to sample random values), (2) An Accumulator ($\mathsf{Acc.Gen}, \mathsf{Acc.Eval}, \mathsf{Acc.Proof}, \mathsf{Acc.Verify}$) (see Appendix A.3), (3) A hiding and binding non-interactive commitment scheme $\mathsf{Com}(m; r)$, (4) a threshold secret sharing scheme ($\mathsf{Share}(m, n, t; r), \mathsf{Recon}(\{s_i\}_{i\in[n]}, n, t)$) [Sha79], and (5) An MPC-in-the-head based Σ-protocol (see Section 2)

For simplicity, we assume that all parties in the MPC protocol used in Σ are input-bearing parties. Looking ahead, it will be obvious from context, that this protocol can be trivially extended to accommodate scenarios where only a subset of the parties might be input-bearing.

Overview. As discussed in the technical overview, the proof proceeds in two phases: (1) demonstrating correctness of the witness independent preprocessing of the public set $\{x_1, \ldots, x_\ell\}$ using cut-and-choose, and (2) demonstrating that the MPC-in-the-head protocol has been executed honestly with respect to the preprocessing. For succinctness, as much information as possible is generated by expanding small seeds using a PRG. The protocol is split over two figures; first, the prover and verifier participate in an interactive protocol described in Figure 1, and then the verifier runs the algorithm from Figure 2 to verify validity of the transcript obtained from Figure 1.

Set Membership Proof using MPC-in-the-Head

Public Inputs: Statements $x_1, \dots, x_\ell$, Accumulator Parameters pp
Prover Inputs: Active Statement x_α, Witness w
Protocol Parameters: Number of Preprocessings M, Number of MPC Parties n, MPC Corruption Threshold t, Number of Online Phases τ, Numer of Preprocessings Consumed by Each Online Phase η

- **Round 1:** For each $j \in [M]$, the prover computes the following:
  - **Share a Random Mask Among the Players:**
    1. Sample a random mask $\mathsf{mask}_j \xleftarrow{\$} \mathbb{F}$.
    2. For each $k \in [\ell]$, compute $\Delta_{j,k} = x_k - \mathsf{mask}_j \in \mathbb{F}$.
    3. Share the mask: $r_j^{(\mathsf{share})} \xleftarrow{\$} \{0,1\}^\kappa; [\mathsf{mask}_j]_1, \dots, [\mathsf{mask}_j]_n \leftarrow \mathsf{Share}(\mathsf{mask}_j, n, t; r_j^{(\mathsf{share})})$.
  - **Committing to Shares and Deltas:**
    1. For each $i \in [n]$, commit to the share: $r_{j,i}^{(\mathsf{mask\text{-}com})} \xleftarrow{\$} \{0,1\}^\kappa; \mathsf{com}_{j,i}^{(\mathsf{mask})} = \mathsf{Com}([\mathsf{mask}_j]_i; r_{j,i}^{(\mathsf{mask\text{-}com})})$.
    2. Sample $r_j^{(\Delta\text{-}\mathsf{com})} \xleftarrow{\$} \{0,1\}^\kappa$ and compute $\{r_{j,k}^{(\Delta\text{-}\mathsf{com})}\}_{k\in[\ell]} \leftarrow \mathsf{PRG}(r_j^{(\Delta\text{-}\mathsf{com})})$.
    3. For each $k \in [\ell]$ commit to the delta: $\mathsf{com}_{j,k}^{(\Delta)} = \mathsf{Com}(\Delta_{j,k}; r_{j,k}^{(\Delta\text{-}\mathsf{com})})$.
  - **Permuting and Accumulating Delta Commitments:**
    1. Randomly sample a permutation ϕ_j from the space of all permutations of size ℓ.
    2. Accumulate permuted commitments: $(\mathsf{aux}_j, \mathsf{acc}_j) = \mathsf{Acc.Eval}(\mathsf{pp}, \mathsf{com}_{j,\phi_j(1)}^{(\Delta)}, \dots, \mathsf{com}_{j,\phi_j(\ell)}^{(\Delta)})$.
    3. Let $\mathsf{R}_j = (\mathsf{com}_{j,1}^{(\mathsf{mask})}, \dots, \mathsf{com}_{j,n}^{(\mathsf{mask})}, \mathsf{acc}_j)$

  Finally, send $(\mathsf{R}_1, \dots, \mathsf{R}_M)$ to the verifier.
- **Round 2:** The verifier randomly partitions $[M]$ into $(\tau + 1)$ subsets: $C_1, \dots, C_\tau$ and S such that $|C_i| = \eta$ for $i \in [\tau]$ and $|S| = M - \tau\eta$.
- **Round 3:** The prover (1) opens all repetitions in S, and (2) runs the MPC for the remaining repetitions:
  1. For each $j \in S$, the prover sends $(\{r_{j,i}^{(\mathsf{mask\text{-}com})}\}_{i\in[n]}, r_j^{(\Delta\text{-}\mathsf{com})}, r_j^{(\mathsf{share})}, \mathsf{mask}_j, \phi_j)$ to the verifier.
  2. For each $i \in [\tau]$:
     (a) Additively secret share w: pick random $[w]_1, \dots, [w]_n$ st. $w = \sum_{m\in[n]} [w]_m$
     (b) Additively secret share x_α: pick random $[x_\alpha]_1, \dots, [x_\alpha]_n$ st. $x_\alpha = \sum_{m\in[n]} [x_\alpha]_m$
     (c) Let $\{\Delta_{j,\alpha}\}_{j\in C_i}$ be (public) constants in the MPC protocol, each party P_m is assigned input $(\{[\mathsf{mask}_j]_m\}_{j\in C_i}, [x_\alpha]_m, [w]_m)$. The prover computes the 'MPC-in-the-head' for the following relation:
     $$\mathcal{F}\left(\{\{[\mathsf{mask}_j]_m\}_{j\in C_i}, [x_\alpha]_m, [w]_m\}_{m\in[n]}\right) :=$$
     $$\mathcal{R}\left(\textstyle\sum_{m\in[n]} [x_\alpha]_m, \sum_{m\in[n]} [w]_m\right) \wedge \forall j \in C_i : \mathsf{Recon}\left(\{[\mathsf{mask}_j]_m\}_{m\in[n]}, n, t\right) + \Delta_{j,\alpha} = \textstyle\sum_{m\in[n]} [x_\alpha]_m$$
     Let $\mathcal{V}_{i,m}$ be the resulting view of party P_m.
     (d) For each $m \in [n]$, commit to each view: $r_{i,m}^{(\mathsf{view\text{-}com})} \xleftarrow{\$} \{0,1\}^\kappa; \mathsf{com}_{i,m}^{(\mathsf{view})} = \mathsf{Com}(\mathcal{V}_{i,m}; r_{i,m}^{(\mathsf{view\text{-}com})})$
     (e) For $j \in C_i$ compute the membership proof for $\Delta_{j,\alpha}$: $\pi_j \leftarrow \mathsf{Acc.Proof}(\mathsf{pp}, \mathsf{acc}_j, \mathsf{com}_{j,\phi_j(\alpha)}^{(\Delta)}, \mathsf{aux}_j)$.
     (f) Send $\{(\pi_j, \Delta_{j,\alpha}, r_{j,\alpha}^{(\Delta\text{-}\mathsf{com})})\}_{j\in C_i}$, and $\{\mathsf{com}_{i,m}^{(\mathsf{view})}\}_{m\in[n]}$ to the verifier.
- **Round 4:** For each $i \in [\tau]$, the verifier picks a random t-sized subset of the parties $\mathcal{I}_i \subseteq_t [n]$.
- **Round 5:** The prover opens the parties specified in $\mathcal{I}_1, \dots, \mathcal{I}_\tau$. For each $i \in [\tau]$:
  1. Send $\{\mathcal{V}_{i,m}, r_{i,m}^{(\mathsf{view\text{-}com})}, \{r_{j,m}^{(\mathsf{mask\text{-}com})}\}_{j\in C_i}\}_{m\in\mathcal{I}_i}$ to the verifier.

Figure 1: A general compiler to obtain an efficient set membership proof from any MPC-in-the-head based Σ-protocol.

Verification for the Set Membership Proof

The verifier proceeds as follows:

- **Verifying Correctness of Opened Preprocessing:** For each $j \in S$, compute the following:
  1. Receive $(\{r_{j,i}^{(\textsf{mask-com})}\}_{i\in[n]}, r_j^{(\Delta\textsf{-com})}, r_j^{(\textsf{share})}, \textsf{mask}_j, \phi_j)$ from the prover.
  2. Compute $\{r_{j,k}^{(\Delta\textsf{-com})}\}_{k\in[\ell]} \leftarrow \textsf{PRG}(r_j^{(\Delta\textsf{-com})})$.
  3. Compute $[\textsf{mask}_j]_1, \ldots, [\textsf{mask}_j]_n \leftarrow \textsf{Share}(\textsf{mask}_j, n, t; r_j^{(\textsf{share})})$.
  4. For each $i \in [n]$, recompute the commitments to the shares $\textsf{com}_{j,i}^{(\textsf{mask})\prime} = \textsf{Com}([\textsf{mask}_j]_i; r_{j,i}^{(\textsf{mask-com})})$.
  5. For each $k \in [\ell]$, recompute the deltas $\Delta_{j,k} = x_k - \textsf{mask}_j$; $\textsf{com}_{j,k}^{(\Delta)\prime} = \textsf{Com}(\Delta_{j,k}; r_{j,k}^{(\Delta\textsf{-com})})$
  6. Recompute the accumulator $(\_, \textsf{acc}'_j) = \textsf{Acc.Eval}(\textsf{pp}, \textsf{com}_{j,\phi_j(1)}^{(\Delta)\prime}, \ldots, \textsf{com}_{j,\phi_j(\ell)}^{(\Delta)\prime})$.
  7. Perform the following checks: $\forall i \in [n], \textsf{com}_{j,i}^{(\textsf{mask})} \overset{?}{=} \textsf{com}_{j,i}^{(\textsf{mask})}$; $\forall k \in [\ell], \textsf{com}_{j,k}^{(\Delta)} \overset{?}{=} \textsf{com}_{j,k}^{(\Delta)\prime}$; $\textsf{acc}_j \overset{?}{=} \textsf{acc}'_j$.
- **Verify Consistency of MPC-in-the-head Execution and Delta Memberships:** For each $i \in [\tau]$:
  1. Receive $(\{(\pi_j, \Delta_j, r_j^{(\Delta\textsf{-com})})\}_{j\in C_i}, \{\textsf{com}_{i,m}^{(\textsf{view})}\}_{m\in[n]}, \{\mathcal{V}_{i,m}, r_{i,m}^{(\textsf{view-com})}, \{r_{j,m}^{(\textsf{mask-com})}\}_{j\in C_i}\}_{m\in\mathcal{I}_i})$ from the prover.
  2. Recompute commitment to provided delta: $\textsf{com}_j^{(\Delta)} = \textsf{Com}(\Delta_j; r_j^{(\Delta\textsf{-com})})$
  3. Recompute commitments to the opened views, for $m \in \mathcal{I}_i$: $\textsf{com}_{i,m}^{(\textsf{view})\prime} = \textsf{Com}(\mathcal{V}_{i,m}; r_{i,m}^{(\textsf{view-com})})$
  4. Verify commitments to player views: Check, for $m \in \mathcal{I}_i$: $\textsf{com}_{i,m}^{(\textsf{view})\prime} \overset{?}{=} \textsf{com}_{i,m}^{(\textsf{view})}$
  5. Verify the accumulator inclusion proofs: Check $\forall j \in C_i : \textsf{Acc.Verify}(\textsf{pp}, \textsf{acc}_j, \textsf{com}_j^{(\Delta)}, \pi_j) \overset{?}{=} 1$.
  6. Verify commitments to opened player's masks: Check, for $m \in \mathcal{I}_i$, $j \in C_i$: $\textsf{com}_{j,m}^{(\textsf{mask})} = \textsf{Com}([\textsf{mask}_j]_m; r_{j,m}^{(\textsf{mask-com})})$, where $[\textsf{mask}_j]_m$ is part of $\mathcal{V}_{i,m}$.
  7. Check consistency of the opened MPC views against the function:

$$\mathcal{F}\left(\{\{[\textsf{mask}_j]_m\}_{j\in C_i}, [x_\alpha]_m, [w]_m\}_{m\in[n]}\right) :=$$
$$\mathcal{R}\left(\textstyle\sum_{m\in[n]}[x_\alpha]_m, \sum_{m\in[n]}[w]_m\right) \wedge \forall j \in C_i : \textsf{Recon}\left(\{[\textsf{mask}_j]_m\}_{m\in[n]}, n, t\right) + \Delta_j = \textstyle\sum_{m\in[n]}[x_\alpha]_m$$

Parameterized by the constants $\{\Delta_j\}_{j\in C_i}$.

If all the above checks succeed, the verifier outputs 1, else it outputs 0.

Figure 2: Verification algorithm for our Set Membership Proof from Figure 1.

For each of the M parallel iterations of the preprocessing step, the prover begins by sampling a random mask. It then uses a threshold secret sharing scheme to compute shares of this mask using the privacy threshold t of the underlying MPC protocol. It also computes the Δs between the random mask and every element in the set, where we denote the Δ corresponding to the k^{th} element in the j^{th} iteration by $\Delta_{j,k}$. The prover then computes commitments to these n shares of the random mask (one for each party) and the ℓ Δ values (one for each element in the set). It then randomly permutes the commitments to the ℓ Δ values and accumulates these permuted commitments using an accumulator. Finally, it computes a hash of the accumulated value and the n commitments to the respective seeds assigned to each party. This hash value is sent over to the verifier.

The verifier then partitions the M preprocessings into two types: (1) a subset S that will be opened and checked for correctness, and (2) preprocessings that will be used to execute the MPC-in-the-head protocol.

The verifier further subdivides this second type of preprocessing into τ equally sized subsets $C_1, \ldots, C_\tau$, each of which will be used to execute a single MPC-in-the-head instance. Note that value of τ comes directly from the number of repetitions required by the underlying MPC-in-the-head protocol, and is not a newly variable introduced by our techniques. For example, if the MPC-in-the-head protocol does not require repetition, then the prover can simply set $\tau = 1$.

In the third round, the prover responds with all the randomness used to compute the preprocessings in S. Then the prover runs τ parallel copies of the the underlying MPC-in-the-head protocol using the remaining "unopened" preprocessing. In particular, for each copy, the prover computes random additive shares for the witness w and a random additive secret sharing for the statements x_α. the witness shares and random mask shares (computed in the preprocessing phase) form the inputs of the parties. The prover computes the views of all the virtual parties in the underlying MPC for the following functionality:

$$\mathcal{F}\left(\{\{[\mathsf{mask}_j]_m\}_{j \in C_i}, [x_\alpha]_m, [w]_m\}_{m \in [n]}\right) \coloneqq$$
$$\mathcal{R}\left(\sum_{m \in [n]} [x_\alpha]_m, \sum_{m \in [n]} [w]_m\right) \wedge \forall j \in C_i : \mathsf{Recon}\left(\{[\mathsf{mask}_j]_m\}_{m \in [n]}, n, t\right) + \Delta_{j,\alpha} = \sum_{m \in [n]} [x_\alpha]_m$$

This functionality contains the following parts:

(1) reconstructs the additive secret sharings of the active statement x_α and witness w supplied as input;

(2) executes the relation circuit $\mathcal{R}(x_\alpha, w)$; and

(3) ensures that statement encoded in each preprocessing is the same and matches x_α.

We assume w.l.o.g., that the reconstruction algorithm Recon outputs $\perp$ if all the n input shares are not consistent. The prover commits to these resulting views exactly as it would in the underlying protocol.

Finally, the verifier chooses a subset of the parties for each execution of the MPC. The prover responds by sending the views of those parties (which include the inputs assigned to those parties). The prover additionally sends the Δ value corresponding to the active element in the unopened preprocessing and the randomness used to commit to it. It then gives a proof of inclusion (using the accumulator) to prove that this commitment was part of the accumulated values.

Completeness of the protocol follows trivially from the correctness of the underlying primitives used in this protocol. We argue zero-knowledge and soundness of this protocol in Appendix B.

Remark. Since the above approach requires us to secret share the mask amongst all the input-bearing parties using the privacy threshold of the underlying MPC protocol, this approach only works if the number of input-bearing parties is more than the privacy threshold in the underlying MPC protocol. For all other protocols, we can simply use an additive secret sharing scheme to secret share the masks. However, as discussed in the introduction, the soundness error of the resulting protocol in this case will depend both on the soundness error of the underlying MPC-in-the-head protocol and on the number of parties.

Complexity Analysis. Let $\mathtt{CC}_\Sigma(\mathcal{F})$ be the communication complexity of a single run underlying MPC-in-the-head protocol when used to give a proof for the relation circuit $\mathcal{F}$. Then the communication complexity of a single run of the protocol described in Figure 1 is: $M(n+1)\kappa + (n\kappa + 3\kappa + |x|)(M - \tau\eta) + \tau\eta\kappa(\log \ell + 2) + \mathtt{CC}_\Sigma(\mathcal{F}) = O(Mn(\kappa + |x|) + M\kappa \log \ell) + \mathtt{CC}_\Sigma(\mathcal{F})$ The computation complexity of this protocol is $O(Mn(\kappa + |x|) + (\kappa + |x|)M\ell) + \overline{\mathtt{CC}}_\Sigma(\mathcal{F})$, where $\overline{\mathtt{CC}}_\Sigma(\mathcal{F})$ is the computation complexity of a single run of the underlying MPC-in-the-head protocol when used to give a proof for the relation circuit $\mathcal{F}$.

Set Membership for Multiple Values. It is easy to see that the above ideas can be trivially extended to obtain a set membership proof for multiple values (say k values), *i.e.*, where the prover would want to prove that $\exists x'_1, \ldots, x'_k, w$ such that $\{x'_1, \ldots, x'_k\} \subset \{x_1, \ldots, x_n\}$ and $\mathcal{R}(x'_1, \ldots, x'_k, w) = 1$. This can be done using the above protocol with the only modification that the underlying MPC-in-the-head protocol will be used to prove the relation

$$\mathcal{F}\left(\{\{[\mathsf{mask}_j]_m\}_{j\in C_i}, [x_1]_m, \ldots, [x_k]_m, [w]_m\}_{m\in[n]}\right) :=$$

$$\mathcal{R}\left(\sum_{m\in[n]}[x_1]_m, \ldots, \sum_{m\in[n]}[x_k]_m, \sum_{m\in[n]}[w]_m\right) \wedge \forall j \in C_i \text{ and } \beta \in [k] : \mathsf{Recon}\left(\{[\mathsf{mask}_j]_m\}_{m\in[n]}, n, t\right) + \Delta_{j,\beta} = \sum_{m\in[n]}[x_\beta]_m$$

where $\Delta_{j,\beta}$ are public constants for $\beta \in [k]$ and the prover will be required to give a proof of inclusion in the accumulator for each $\Delta_{j,\beta}$ value. Looking ahead, this will be useful in our construction of RingCT.

3.1 Optimizations.

We now discuss some optimizations to improve the communication of the above protocol.

Secret Sharing Based Protocols: In the above protocol we use the underlying MPC protocol to additionally run the reconstruction algorithm of the threshold secret sharing scheme in order to reconstruct shares of the masks. However, if the underlying MPC protocol itself is based on a threshold secret sharing scheme (e.g. [BH05, GMW87]), then this explicit reconstruction can be avoided. This is because most such protocols maintain an invariant that all the parties hold secret shares of all the intermediate wire values in the circuit that they are evaluating. As a result, we can simply have the parties compute the following function inside the MPC, assuming that they already have a secret sharing of the mask.

$$\mathcal{F}\left(\{\mathsf{mask}_j\}_{j\in C_i}, [x_\alpha]_m, [w]_m\}_{m\in[n]}\right) := \mathcal{R}\left(\sum_{m\in[n]}[x_\alpha]_m, \sum_{m\in[n]}[w]_m\right) \wedge \forall j \in C_i : \mathsf{mask}_j + \Delta_{j,\alpha} = \sum_{m\in[n]}[x_\alpha]_m$$

Round Complexity: Recall that our five-round protocol is a public-coin protocol, since the verifier is only required to send random values. Such a protocol can be easily made non-interactive in the random oracle model using the Fiat-Shamir transform in both the classical setting [FS87] and the quantum setting [LZ19, DFMS19].

Additionally, notice that in the case where $\eta = 1$ (*i.e.* $|C_i| = 1$),[6] the above protocol can also be slightly modified to obtain a three-round protocol in the plain model. This can be done by requiring the prover to compute the first round messages in the underlying MPC-in-the-head protocol for all M repetitions of the pre-processing phase and committing to them in the first round itself. In the second round, the verifier chooses a random $S \subset [M]$. In the third round, the prover only send the third round messages of the MPC-in-the-head protocol for the executions not in S. Put another way, the prover pessimistically runs the full MPC-in-the-head protocol for *each* preprocessing, and reveals only the preprocessing for S and only the MPC-in-the-head protocol for the remainder. If we use a compressing commitment to commit to all the first round messages, this modification will not increase the communication complexity by a lot, but it does significantly increase the computation time, since the prover needs to execute the first round of the underlying MPC-in-the-head protocol M times.

Using Tree based PRGs: Instead of sampling an independent random seed for each repetition of the pre-processing phase, we could simply sample a "master seed" and derive all the other seeds using a Tree-based PRG. When the verifier asks to open all but one pre-processings, the prover can simply send a punctured master seed (which is only logarithmic in the number of the leaves in the tree) that allows the verifier to derive all but one seed. However, if one of the above optimizations are used, where only a small subset of the pre-processings are opened, we will have to puncture the seed on multiple indices (which increases the size of the punctured seed). In that case, one might have to carefully decide if it's still worth using a Tree-based PRG or if sampling an independent seed for each iteration is better.

[6] As we will see in Section 4, this case matches the KKW MPC-in-the-head protocol, and is thus of practical importance.

Similarly, if the underlying MPC protocol is in the dishonest majority setting, we can again use a Tree-based PRG to derive the seeds used to commit to the shares of parties in each pre-proprocessing. When the verifier asks to open the views of all but one party, the prover can simply send a punctured seed.

Finally, we note that all of the above optimization suggestions only try to cover some broad categories of MPC-in-the-head protocols. Given the wide variety of MPC-in-the-head protocols, it is impossible to suggest a general technique that gives the best efficiency with all protocols. Indeed, if one is willing to use the underlying MPC-in-the-head protocol in a non-black box way, it might be possible to further customize some of these optimizations. In the next section, we explore some of these choices in more detail when our compiler is used in conjunction with the protocol of Katz et al. [KKW18].

4 Compiling Non-Standard MPC-in-the-head Protocols

The compiler described in the previous section works seamlessly with any MPC-in-the-head based Σ protocol that follows the template presented in Section 2. However, some recent protocols in this regime slightly deviate from this template in order to get better efficiency. In this section, we demonstrate the versatility of our main ideas by showing how they can extend to two such protocols. Additionally, we implement our compiler as applied to the second example.

4.1 Integrating Into Ligero [AHIV17]

Ligero is a MPC-in-the-head protocol with sub-linear communication complexity that leverages a highly non-traditional MPC protocol. In the first phase, a special "sender" party evaluates the relation circuit using the witness in the clear, and computes $\sqrt{|C|}$ packed secret sharings of the values induced on all the intermediate wires of this relation circuit, where $|C|$ is the size of the circuit and each sharing holds a block of $\sqrt{|C|}$ values. These sharings are distributed to n virtual servers. In the second phase, the servers obtain a public random string r sampled via a coin flipping oracle, and perform only local computation. Finally, each server sends a single message to a "receiver."

In the 5 round MPC-in-the-head protocol based on this MPC protocol, the verifier supplies the value r and chooses to see a subset of the views of the servers and the verifier, but is not allowed to see the sender's view, as this would trivially violate zero-knowledge. At a high level, the first phase of the underlying MPC protocol transforms the circuit into a set of polynomial encodings. The verifier then uses the second phase of the MPC to check that these encodings satisfy low degree constraints derived from the structure of the relation circuit.

The Ligero protocol does not require repetition, as a single iteration of the protocol already has negligible soundness error. Thus, when instantiating our set membership techniques with Ligero, we set τ=1 and $\eta = M/2$. Because the sender party's view cannot be opened, the inputs for x_α must be provided to the *servers* instead and be appropriately encoded into packed secret shares. The remaining protocol can be run without further modification.

4.2 Integrating Into Katz et al. [KKW18]

Katz et al. [KKW18] presented an efficient zero-knowledge protocol using MPC-in-the-head technique. The goal of their approach is to increase the number of virtual players participating in the MPC protocol (and thereby reduce the number of parallel repetitions required for soundness) without increasing the size of resulting proofs. To accomplish this, they design a special-purpose MPC protocol consisting of two phases: (1) a circuit dependent *pre-processing phase* that is independent of the parties' inputs to the MPC and (2) an *online phase* that depends on the output of the pre-processing phase and the parties' inputs. Using several clever optimizations, the size of the resulting proof is (mostly) independent of the number of virtual players.

A straightforward way to integrate our set membership proof techniques into a Katz et al. proof would be to use their protocol in a black-box way inside our compiler (Figure 1), computing η based on the value of τ used in Katz et al. However, this ignores the shared structure of our compiler and Katz et al.'s protocol: both have an input-independent pre-processing phase, followed by an evaluation phase over secret inputs. A more natural approach would be to run our set membership in parallel with Katz et al.'s protocol. Specifically, during the pre-processing phase, the prover generates both correlated randomness and secret shares of the set (as in the compiler) and sets $\eta = 1$. The correctness of both can be verified using a cut-and-choose approach. During the online phase, the virtual parties run the MPC protocol in Katz et al. leveraging the correlated randomness, using the secret shares of the secret element directly.

The benefits of this approach is that there is no need to add extra rounds to the protocol.[7] Moreover, since their MPC protocol works in the dishonest majority, as discussed in the previous section, instead of sampling a different seed for every party, we can use a tree-based PRG to derive randomness for the parties to further reduce the communication complexity. This protocol is only based on symmetric key primitives and is therefore post-quantum secure. The total communication complexity of this protocol is $O(\kappa \log n + |\mathcal{R}| + |x| + |w| + \kappa \log \ell)$.

Implementing Set Membership Proofs for Katz et al. We implemented this optimized integration of set membership techniques on top of Reverie, the only public general purpose implementation of the Katz et al. protocol. The entire code base is written in Rust and supports parallel computation, proof streaming, and bit slicing, and is freely available under the GPLv3.[8] Throughout our evaluation, we use this implementation.

Our implementation automatically applies Fiat-Shamir to the three-round variant of [KKW18]. Because the input from the set membership proof needs to be treated differently than standard input, we separate the input into two tapes. Circuits are specified in the Bristol format; whenever the prover encounters an `INPUT` gate, it reads from the first (witness) tape and whenever it encounters a `BRANCH` gate it reads from the (second) membership proof tape.

The code base instantiates the cryptographic primitives with ChaCha12[Ber08] and Blake3[OANWO20] as follows: The PRG is implemented with ChaCha12, including as the length-doubling PRG inside the TreePRG. We use Blake3 as the collision resistant hash function and to instantiate the random oracle (in KDF mode)[9]. We instantiate our commitments with hash-based commitments, also using Blake3 in keyed mode. We use Blake3 without keying when committing to high entropy values, as in Katz et. al [KKW18], but note that this particular optimization cannot be applied when committing to the shares in the set membership proof, as they are correlated. Our implementation is generic: allowing instantiation of the NIZKPoK with any ring and taking any algebraic circuit over said ring. We use this to optimize the ring signature circuit for the particular PRF instantiation as described below.

5 Post-Quantum Ring Signatures

Ring signatures are a privacy preserving version of signatures introduced by Rivest, Shamir, and Tauman [RST01]. Ring signatures give the signer $k-$anonymity, in that a signature can only be linked back to a *set* (or ring) or public keys. We now show how to leverage our efficient set membership techniques for MPC-in-the-head to get the smallest constructions of post-quantum ring signatures from symmetric key assumptions.

5.1 Definition of Ring Signatures

We provide formal definitions for ring signatures here.

Definition 1 *A ring signature scheme consists of a tuple of three probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *defined as follows:*

[7]Indeed, this five-round protocol for set-membership can then be compressed into a three-round protocol (in the plain model) using the ideas from [KKW18]

[8]Code is available at `https://github.com/trailofbits/reverie/tree/stacked-ikos`.

[9]To provide domain separation from the CRH.

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$*: The key-generation algorithm* Gen *takes as input the security parameter* λ *and generates public key* pk *and the associated signing key* sk.

- $\mathsf{Sign}(\{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}, M) \to \sigma$*: The signing algorithm* Sign *takes as input a set (or "ring")* $R = \{\mathsf{pk}_i\}_{i\in[\ell]}$ *of distinct public keys, a secret key sk corresponding to one of the public keys in* R*, and a message* M*. It outputs a signature* σ*.*

- $\mathsf{Verify}(\{\mathsf{pk}_i\}_{i\in[\ell]}, M, \sigma) \to b$ *The verification algorithm* Verify *takes as input a ring* R *of distinct public keys, a message* M*, and a signature* σ*. It outputs a single bit* b *indicating acceptance or rejection.*

We proceed to define three properties of ring signatures: *correctness, unforgeability* and *anonymity*. These definitions (adapted from [BKM09]) are taken verbatim from [KKW18].

Correctness. Correctness requires that for any collection of keys $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[\ell]}$ output by Gen, any message M, and any $j \in [\ell]$, we have

$$\mathsf{Verify}(\{\mathsf{pk}_i\}_{i\in[\ell]}, M, \mathsf{Sign}(\{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}_j, M)) = 1.$$

Unforgeability. Intuitively, unforgeability means that an adversary not in R should not be able to generate a valid signature σ on a message M relative to a ring R unless some honest user in R had previously signed M (relative to the same ring).

Definition 2 *Ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is unforgeable if, for any PPT adversary* $\mathcal{A}$ *and any polynomial* ℓ*, the probability that* $\mathcal{A}$ *succeeds in the following experiment is negligible in* λ*:*

1. *Keys* $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[\ell]}$ *are generated by* $\mathsf{Gen}(1^\lambda)$*. The public keys* $S = \{\mathsf{pk}_i\}_{i\in[\ell]}$ *are given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *may query an oracle* $\mathsf{Sign}'(\cdot,\cdot,\cdot)$*, where* $\mathsf{Sign}'(R, i, M)$ *(with* $\mathsf{pk}_i \in R$*) outputs* $\mathsf{Sign}(R, \mathsf{sk}_i, M)$*. (Note that we do not require* $R \subseteq S$*.)*

3. $\mathcal{A}$ *may also query a corruption oracle* $\mathsf{corrupt}$ *that on input* i *returns* sk_i*. If* $\mathcal{A}$ *queries* $\mathsf{corrupt}(i)$ *then we say that* pk_i *is corrupted. We let* C *be the set of corrupted public keys at the end of the experiment.*

4. $\mathcal{A}$ *outputs* M^*, R^*, σ^**. It succeeds if (1)* $\mathsf{Verify}(R^*, M^*, \sigma^*) = 1$*, (2)* $R^* \subseteq S \setminus C$ *and (3)* $\mathcal{A}$ *never queried* $\mathsf{Sign}'(R^*, \cdot, M^*)$*.*

Security. We now prove security of this construction. We show that if this construction is not unforgeable, then either there is a hash collision or the our NIZK PoK is not sound or it does not satisfy zero-knowledge. Let us assume for contradiction that there exists an adversary who can break the unforgeability of this construction.

- If $\sigma^* = \sigma$: If $\sigma^* = \sigma$, then the challenge on which the prover computes must also be identical in the two proofs. Since the challenge computed by H^{RO} depends on the message and the ring. However, we know that either $M^* \neq M$ or $R^* \neq R$. From collision resistance of hash function, it follows that the two challenges cannot be identical in this case, except with some negligible probability.

- If $\sigma^* \neq \sigma$: From soundness of our NIZKPoK, we know that we can extract the witness corresponding to an accepting proof, which in this case is sk_i. Since $\mathsf{sk}_i \notin C$, the only way an adversary could have learned sk_i is if it breaks the zero-knowledge property of the NIZKPoK construction, which only happens with at most negligible probability.

Anonymity of the scheme trivially follows from the zero-knowledge property of our NIZKPoK.

Anonymity. Intuitively, anonymity ensures that a valid signature with respect to a ring R does not reveal which secret key (corresponding to some public key in R) was used to generate the signature.

Post-Quantum Secure Ring Signatures

Let $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a pseudorandom function.

- $\mathsf{Gen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$: The key-generation algorithm Gen chooses a uniform signing key $\mathsf{sk} \in \{0,1\}^\kappa$ and the corresponding public key is defined as $\mathsf{pk} = \mathsf{PRF}_{\mathsf{sk}}(0^\kappa)$. It outputs $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{Sign}(\{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}, M) \coloneqq \sigma$: The signing algorithm Sign computes and outputs a NIZKPoK for:

$$\text{NIZKPoK}\left\{(\mathsf{sk}, \mathsf{pk}) : (\mathsf{PRF}_{\mathsf{sk}}(0^\kappa) = \mathsf{pk}) \wedge \mathsf{pk} \in \{\mathsf{pk}_1, \dots \mathsf{pk}_\ell\}\right\},$$

  where the message M being signed is included as input to the hash function H^{RO} used to compute the challenge.

- $\mathsf{Verify}(\{\mathsf{pk}_i\}_{i\in[\ell]}, M, \sigma) \coloneqq b$: The verification algorithm verifies the NIZK proof σ and outputs 1 if the proof accepts and 0 otherwise.

Figure 3: The signature construction that we implement, based on [CDG+17, KKW18]. When proving the security of the scheme, PRF is modeled as a one-way function.

Definition 3 *Ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *satisfies anonymity if, for any PPT adversary $\mathcal{A}$ and any polynomial ℓ, the probability that $\mathcal{A}$ succeeds in the following experiment is at most* $1/2 + \mathsf{negl}(\lambda)$*:*

1. *Keys* $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[\ell]}$ *are generated by* $\mathsf{Gen}(1^\lambda)$ *and all keys (both public and private) are given to $\mathcal{A}$.*
2. *$\mathcal{A}$ outputs a message M, a ring R, and $i_0, i_1 \in [\ell]$. A uniform $b \in \{0,1\}$ is chosen, and $\mathcal{A}$ is given* $\mathsf{Sign}'(R', \mathsf{sk}_{i_b}, M)$ *where* $R' = R \cup \{\mathsf{pk}_{i_0}, \mathsf{pk}_{i_1}\}$*.*
3. *$\mathcal{A}$ outputs a bit b' and succeeds if $b' = b$.*

5.2 Construction and Implementation

The work of Chase *et. al.* [CDG+17], and later Katz *et. al.* [KKW18], construct post-quantum ring signatures in the random oracle model that only depend on symmetric key primitives. These signatures use random PRF keys as the secret keys sk and the corresponding public key is defined as $\mathsf{PRF}_{\mathsf{sk}}(0^\kappa)$. The regular signatures constitute a NIZKPoK$\{(\mathsf{sk}) : \mathsf{PRF}_{\mathsf{sk}}(0^\kappa) = \mathsf{pk}\}$ in which the message is fed into the random oracle during flattening. This means that the simulated players execute the PRF and that the public key is fed explicitly into the circuit at the end to perform the equality check. Clearly, this reveals the identity of the signer.

Katz *et. al.* [KKW18] showed how to extend this template to create both ring and group signatures by adding a Merkle membership proof inside the circuit. More formally, the signature would be of the form NIZKPoK$\left\{(\mathsf{sk}, \mathsf{pk}) : (\mathsf{PRF}_{\mathsf{sk}}(0^\kappa) = \mathsf{pk}) \wedge \mathsf{pk} \in \{\mathsf{pk}_1, \dots \mathsf{pk}_\ell\}\right\}$, where the membership proof is implemented using a Merkle tree. This introduced a *multiplicative* overhead to the signature size proportional to the the logarithm of the ring size, as each hash in the Merkle tree is the same size as the relation itself. Taking a similar approach, but we use the membership proof construction presented in Section 4.2.

Implementation and Evaluation We implemented our ring signatures using Reverie. We instantiate the PRF using LowMC with the `picnic-L5-full` parameters[10] from the Picnic specification [Zav20] offering an estimated 128 bits of post-quantum security. The underlying MPC protocol is instantiated with $n = 64$, $M = 1662$, $\tau = 44$ to optimize for signature size at 256 bits of statistical security (yielding a NIZK with 128 bits of security against Grover's algorithm [Gro96]). We bitslice the LowMC circuit inside the MPC

[10]LowMC with 255-bit state, 255-bit key, 4 rounds and full S-Box layers.

Table 2: Performance of the post-quantum ring signature scheme compared to prior works. $|\sigma|$ is the size of the resulting signature. t is the time for signing/verifying the signature. The average is taken over 1000 executions. The server consists of two Intel(R) Xeon(R) CPU E5-2695 @ 2.10GHz. The laptop consists of a single Intel(R) i7-4510U CPU @ 2.00GHz. Derler et al. do not report runtime. The experiments of Katz. el al. were run on a Intel Xeon E5-2666v3

Ring size:	2^7		2^{10}		2^{13}	
	$\|\sigma\|$	t	$\|\sigma\|$	t	$\|\sigma\|$	t
Derler et al.	982 KB	—	1352 KB	—	1722 KB	—
Katz et al. (Server w/ 10 cores)	285 KB	2000 ms	388 KB	2800 ms	492 KB	3600 ms
This Work (Server w/ 36 cores)	52 KB	126 ms	56 KB	210 ms	60 KB	1980 ms
This Work (Laptop w/ 2 cores)	52 KB	2163 ms	56 KB	3437 ms	60 KB	16080 ms

protocol[11], by letting the MPC protocol operate over vectors from $(\mathbb{F}_2)^{85}$ (rather $\mathbb{F}_2$) to improve concrete prover/verifier time.

We find that our ring signatures are the smallest post-quantum ring signatures currently known from symmetric key assumptions. A single iteration of LowMC requires $\approx$ 42KB of communication, which is linear in the number of non-linear gates in the the LowMC circuit times the number of online repetitions. The membership proof requires $\approx 1.5\text{KB} \times \log(\ell)$, where ℓ is the size of the ring. We present concrete sizes for our signatures in Table 1. Additionally we report signature generation/verification time in Table 2. We note that signature generation/verification is an embarrassingly parallel problem, so computing on multiple cores significantly increases the performance of the system.

6 Post-Quantum RingCT

RingCT [Noe15, NMRL16] is a protocol used by the cryptocurrency Monero to provide additional privacy to transactions posted on the blockchain. RingCT provides two primary benefits over more standard cryptocurrencies: the identity of a transaction's sender is obscured using an anonymity set of the sender's choice, and the amounts sent in each transaction are kept private to the sender and receiver. These properties make it difficult to track the flow of currency through the network, providing significantly improved privacy over blockchains like Bitcoin [Nak08] and Ethereum [B$^+$].

In order to provide these privacy properties, RingCT augments standard blockchain transactions with the following: (1) protect the identity of the sender with a ring signature, rather than a traditional signature, (2) encapsulate the input and output balances of transactions inside commitments, (3) prove that the input balances sum to the output balances, and (4) prove that all balances are well formed. Monero instantiates these augmentations using ring signatures, homomorphic Pederson Commitments, and range proofs based on Bulletproofs [BBB$^+$18].

There has been recent interest in instantiating RingCT using plausibly post-quantum secure primitives [TSS$^+$18, TKS$^+$19, EZS$^+$19, TSSK20]. These works follow the classical constructions of RingCT closely, leveraging lattice-based cryptography to create the required primitives. We improve upon this work by reducing the assumptions to only symmetric-key primitives.

6.1 Definition of RingCT

In this work, we adopt the definitions of RingCT 2.0 [SALY17], as they are cleaner that subsequent formalizations [LRR$^+$19, EZS$^+$19, YSL$^+$20] and are sufficient to capture the application in which we are interested. A RingCT scheme consists of the following algorithms:

[11]As done in the optimized Picnic implementation.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$: On input the security parameter, generate any necessary public parameters. We let these public parameters be an implicit input to the algorithms below.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$: On input the security parameter, generate a random public-private keypair $(\mathsf{pk}, \mathsf{sk})$.
- $(\mathtt{coin}, ck) \leftarrow \mathsf{Mint}(\mathsf{pk}, \epsilon)$: On input a public key pk and an amount ϵ, create a coin $\mathtt{coin}$ and a coin key ck. In general, the coin can be thought of as a commitment to the value ϵ under the randomness ck.
- $(\mathtt{tx}, \pi, S) \leftarrow \mathsf{Spend}(K_s, A_s, A, R)$: On input a set K_s of private information tuples $(\mathsf{pk}, (\epsilon, ck))$ (where sk is a secret key, ϵ is an amount, and ck is a coin key), each of which is associated with an input account $(\mathsf{pk}, \mathtt{coin})$ in the set A_s, a larger set of anonymizing accounts A, and a set of output accounts R, generate a transaction $\mathtt{tx}$, a proof π that the transaction is well formed, and a set of serial numbers S.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathtt{tx}, \pi, S)$: On inputs a transaction $\mathtt{tx}$, a proof π and a set of serial numbers S, determine if the bundle is valid.

We now give informal descriptions of the security properties that these algorithms must satisfy. We refer the reader to [SALY17] for a more formal definition of these properties:

- **Correctness.** A user can run the Spend algorithm to produce a valid tuple $(\mathtt{tx}, \pi, S)$ spend from accounts that they control to any set of (adversarial chosen) receiver accounts and any (adversarial chosen) anonymity set.
- **Balance.** A user should (1) only be able to spend from accounts that they control, and (2) it should be impossible to use transactions to fabricate more currency (*i.e.* the sum of the input amounts to a transaction should be at least as large as the sum of the outputs of that transaction). Relatedly, it should only be able to spend from accounts once.
- **Anonymity.** An observer should not be able to determine which of the accounts in A are the input accounts.
- **Non-Slanderability.** The non-slanderabiliy property requires that a malicious user cannot slander any honest user after observing an honestly generated spending. It is infeasible for any malicious user to produce a valid spending that shares at least one serial number with a previously generated honest spending

6.2 Constructing RingCT From Symmetric Key Primitives

Recall that the goal is to construct RingCT from only symmetric key primitives. Adapting the intuition of prior implementations, we choose to instantiate coins as a commitment to the value using the coin key as randomness. Specifically, we will use hash based commitments, as they rely only on symmetric key primitives, instantiated using LowMC and the Davies-Meyer transformation[12]. Thus, the Mint algorithm is simply generating a random string ck and then committing to the input value. Additionally, we will make use of the signatures discussed in the previous section and compute the serial numbers for each coin as a deterministic, random-looking function of ck. Because we use a pseudorandom function both for the generation of serial numbers and signatures, we separate the two pseudorandom functions as PRF_1 and PRF_2 respectively.

Warmup. As a warmup, we start by showing the algorithm Spend when $|A_s| = |R| = 1$. In this case, $|A_s|$ consists of a single account $(\mathsf{pk}_{\text{in}}, \mathtt{coin}_{\text{in}})$, and K_s is the private information necessary to spend from this account; namely, K_s contains $\mathsf{sk}_{\text{in}}, (\epsilon_{\text{in}}, ck_{\text{in}})$ such that $\mathtt{coin}_{\text{in}} = \mathsf{Com}(\epsilon_{\text{in}}, ck_{\text{in}})$ and $\mathsf{pk}_{\text{in}} = \mathsf{PRF}_2(\mathsf{sk}_{\text{in}}, 0)$. With only a single output, R consists of just $(\mathsf{pk}_{\text{out}}, \epsilon_{\text{out}})$. Finally, A is an arbitrary set of accounts such that $A_s \subseteq A$.

[12]The Davies-Meyer transformation constructs a collision resistant hash function from a block cipher $E(k, m)$ as $H(m_1 \| m_2) = E(m_1, m_2) \oplus m_2$.

In our construction, the proof π is a single zero-knowledge proof that simultaneously provides most of the required security properties. Namely, it plays both the role of the signature and the balance checks. Concretely, for our warmup case, Spend generates a new output coin $\mathtt{coin}_{\text{out}}$ containing value ϵ_{out} and returns $(\mathtt{tx}, \pi_{\text{single}}, S)$, where $\mathtt{tx}$ is $(A, (\mathsf{pk}_{\text{out}}, \mathtt{coin}_{\text{out}}))$, S is a set containing the serial number sn, and π_{single} is computed as

$$\begin{aligned}\pi_{\text{single}} = \text{NIZKPoK}\{&(K_s, A_s, (\epsilon_{\text{out}}, ck_{\text{out}})) : \\ &(\mathsf{pk}_{\text{in}} = \mathsf{PRF}_2(\mathsf{sk}_{\text{in}}, 0) \wedge (\mathsf{pk}_{\text{in}}, \mathtt{coin}_{\text{in}}) \in A) \wedge \\ &(\mathtt{coin}_{\text{in}} = \mathsf{Com}(\epsilon_{\text{in}}; ck_{\text{in}}) \wedge sn = \mathsf{PRF}_1(\mathsf{sk}_{\text{in}}, ck_{\text{in}})) \wedge \\ &(\epsilon_{\text{in}} = \epsilon_{\text{out}} \wedge \mathtt{coin}_{\text{out}} = \mathsf{Com}(\epsilon_{\text{out}}; ck_{\text{out}}))\}.\end{aligned}$$

Intuitively, there are 3 parts of this proof, that together provide the necessary security properties for RingCT:

- **Showing Authorization to Spend Input Coin.** First the prover uses a membership proof to show that $(\mathsf{pk}_{\text{in}}, \mathtt{coin}_{\text{in}})$ is in A and a signature to show that the prover has authorization to spend from that account.
- **Proving Knowledge About Input Coin.** Next, the prover demonstrates that it knows an opening to the coin and that the public serial number corresponds to the coin.
- **Proving Output Coin is Well Formed.** Finally, the prover shows that the output coin holds the correct balance and is well formed.

Once these elements have been constructed, the spender sends them to the blockchain. There, the miners verify the correctness of the proof and check that sn has not been seen before. If these checks pass, the transaction will be included in the blockchain. Finally, the spender will send ϵ_{out} and ck_{out} to the receiver so that they can spend in the future. We note that this approach is significantly simpler than prior constructions. We emphasize that using a generic zero knowledge proof in this way is only possible because of an efficient construction of set-membership proofs.

Generalizing to Multi-input and Multi-output. Using the blueprint from the warmup above, we now present the construction for multi-input and multi-output RingCT. The input sets K_s, A_s and R now contain many elements each. The checks used in the warmup are applied to each input account and output coin, as appropriate. The only significant difference is that the proof checks if the sums of the input accounts equal the sums of the output accounts instead of doing a direct equality check. This task is incredibly straightforward in the MPC-in-the-head setting. Specifically, if the statement is represented as a boolean circuit, then summing the amounts just requires a cascade of full adders of the appropriate width (3 `AND` gates per bit). Another advantage of this representation is that *no range proof is necessary*, as a boolean circuit can natively support unsigned arithmetic. This is a big departure from prior work, where the amounts had to be represented as arithmetic values, which opens the possibility of "overflowing" the ring in which they were being represented. The full construction can be found in Figure 4.

As is expected from such a simple construction that makes such heavy use of generic zero knowledge, security of this construction trivially follows from the security of the underlying set membership proof system. In particular, correctness follows from the correctness of the PRF, commitment scheme and completeness of the set membership proof system. Balance follows from the soundness of the set membership proof system. Anonymity follows from the zero-knowledge property of the set membership proof system. It is well-known [CDG+17] that the Fiat-Shamir transform yields non-malleable NIZKPoKs, both in the classical setting [FKMV12] and in the quantum setting [Unr17]. Non-slanderability of this RingCT construction follows from non-malleability of our NIZKPoK for set membership.

Efficiency Analysis. We now compute the size of the proof π in Figure 4 in terms of $|A|$, $|A_s|$ and $|R|$. For each input in A_s, the prover must prove the following information: (1) that it knows a valid opening for the input coin (1 commitment evaluation), (2) the serial number is correctly computed as the output of

Post-Quantum RingCT from Symmetric Key Assumptions

Let $\mathsf{PRF}_1, \mathsf{PRF}_2 : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a pseudorandom functions. Let $\mathsf{Com} : \{0,1\}^* \to \{0,1\}^\kappa$ be a hash-based commitment scheme.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$: Sample $\mathsf{sk} \xleftarrow{\$} \{0,1\}^\kappa$ and compute $\mathsf{pk} \leftarrow \mathsf{PRF}_2(\mathsf{sk}, 0)$
- $(\mathtt{coin}, ck) \leftarrow \mathsf{Mint}(\mathsf{pk}, \epsilon)$: Sample $ck \leftarrow \{0,1\}^\kappa$ and compute $\mathtt{coin} = \mathsf{Com}(\epsilon; ck)$
- $(\mathtt{tx}, \pi, S) \leftarrow \mathsf{Spend}(K_s, A_s, A, R)$:

  // $K_s = \{(\mathsf{sk}_{\text{in},i}, (\epsilon_{\text{in},i}, ck_{\text{in},i}))\}_{i\in[|K_s|]}$ are secret key material to spend from A_s

  // $A_s = \{(\mathsf{pk}_{\text{in},i}, \mathtt{coin}_{\text{in},i})\}_{i\in[|A_s|]}$ are accounts from which the spender spends

  // $A = \{(\mathsf{pk}_{\text{in},i}, \mathtt{coin}_{\text{in},i})\}_{i\in[|A|]}$ is the anonymity set (ring) hiding input accounts A_S

  // Note that A may be represented succinctly using a random low degree polynomial

  // $R = \{(\mathsf{pk}_{\text{out},j}, \epsilon_{\text{out},j})\}_{j\in[|R|]}$ are receiver's public keys and amounts

  - For $(\mathsf{pk}_{\text{out},j}, \epsilon_{\text{out},j}) \in R$, compute $(\mathtt{coin}_{\text{out},j}, ck_{\text{out},j}) \leftarrow \mathsf{Mint}(\mathsf{pk}_{\text{out},j}, \epsilon_{\text{out},j})$
  - For $(\mathsf{sk}_{\text{in},i}, (\epsilon_{\text{in},i}, ck_{\text{in},i})) \in A_s$, compute $sn_i \leftarrow \mathsf{PRF}_1(\mathsf{sk}_{\text{in},i}, ck_{\text{in},i})$
  - Let $\mathtt{tx} = (A, \{(\mathsf{pk}_{\text{out},j}, \mathtt{coin}_{\text{out},j})\}_{j\in[|R|]})$
  - Let $S = \{sn_i\}_{i\in[|A_s|]}$
  - Compute π as

$$\begin{aligned}\pi = \text{NIZKPoK}\{(K_s, A_s, \{\epsilon_{\text{out},j}, ck_{\text{out},j}\}_{j\in|R|}) : \big(\{\mathsf{pk}_{\text{in},i} = \mathsf{PRF}_2(\mathsf{sk}_{\text{in},i}, 0)\}_{i\in[|A_s|]}\big) \ \wedge \ (A_s \subseteq A) \\ \wedge \ \big(\{\mathtt{coin}_{\text{in},i} = \mathsf{Com}(\epsilon_{\text{in},i}; ck_{\text{in},i})\}_{i\in[|A_s|]}\big) \ \wedge \ \big(\{sn_i = \mathsf{PRF}_1(\mathsf{sk}_{\text{in},i}, ck_{\text{in},i})\}_{i\in[|A_s|]}\big) \\ \wedge \ \left(\sum_{i\in[|A_s|]} \epsilon_{\text{in},i} = \sum_{j\in[|R|]} \epsilon_{\text{out},j}\right) \ \wedge \ \big(\{\mathtt{coin}_{\text{out},j} = \mathsf{Com}(\epsilon_{\text{out},j}; ck_{\text{out},j})\}_{j\in[|R|]}\big)\}\end{aligned}$$

- $\{0/1\} \leftarrow \mathsf{Verify}(\mathtt{tx}, \pi, S)$: Return 1 if and only if:
  - π verifies
  - A is a subset of existing accounts
  - No element of S appears previously

Figure 4: A post-quantum RingCT construction based on symmetric key primitives.

the PRF (1 PRF evaluation), (3) the prover knows the secret key corresponding to the public key associated with the input coin (1 PRF evaluation), and (4) the public key and input coin are indeed part of the ring (1 set membership proof). Thus, the total cost for each input is $3 \times |\text{LowMC}| + |\text{Membership Proof}|$. For each output in R, the prover need only demonstrate that the coin is well-formed with respect to the output value ϵ_{out}. The sum check requires $3 \times b$ AND gates per addition, where b is the bit width of the amount representation. With $b = 64$ and $\tau = 44$, this means each addition costs only 2.1KB of communication. Thus, the total cost for each output is $|\text{LowMC}|$. Finally, the prover demonstrates that the inputs and outputs sum correctly. In total, for a transaction with $|A_s|$ inputs, $|R|$ outputs, and a ring size of $|A_s|$, π is

$$(3|A_s| + |R|)|\text{LowMC}| + |A_s||\text{Membership Proof}| + |\text{Sum Check}|.$$

Each iteration of LowMC, using the same parameter choices from Section 5 costs 11.2KB[13] with a baseline

[13]Elements are in $\mathbb{F}_{85}$. There are 4 rounds, in which each bit of the element in $\mathbb{F}_{85}$ is multiplied 3 times. Each MPC requires

overhead of 31KB. Thus, the full cost is

$$31\text{KB} + (3|A_s| + |R|)(11.2\text{KB}) + |A_s|\log(|A|)(1.5\text{KB}) + (|A_s| + |R| - 2)(2.1\text{KB})$$

Comparing our work to the most efficient recent work on post-quantum RingCT [EZS$^+$19], we find that our techniques yield proof sizes that significantly more efficient when the anonymity set is large.[14] For instance, when $|A| = 2^{15}, |A_s| = 1$, and $|R| = 2$, our proofs are of size 111KB, which is *half* the size of [EZS$^+$19] transactions, which are $\approx$ 250KB. The relative performance of our techniques improves as $|A|$ grows. Once $|A| = 2^{21}$, the proof size of [EZS$^+$19] is already 400KB, whereas ours would only be 120.6KB. Both of our techniques scale linearly with $|A_s|$, but ours offers slightly worse constants. Our proofs grow by $\approx$ 35KB per input. However, as noted by [EZS$^+$19], the most common kinds of transactions have $|A_s|, |R| \leq 2$. Finally, we note that the public keys in our construction are only 256 bits, whereas the public keys in [EZS$^+$19] are 4.36KB.

7 Conclusion

In this work we presented an efficient set membership proof method that can be used with any MPC-in-the-head zero-knowledge proof system. We demonstrated that this technique has meaningful applications to privacy preserving systems, including the smallest post-quantum ring signatures from symmetric key primitives. Additionally, we presented a simple construction of post-quantum RingCT that showed the benefits of set membership proofs that naturally integrate with generic zero-knowledge.

Acknowledgments. The first and second authors are supported in part by NSF under awards CNS-1653110, and CNS-1801479, and the Office of Naval Research under contract N00014-19-1-2292. The first author is also supported in part by NSF CNS grant 1814919, NSF CAREER award 1942789 and the Johns Hopkins University Catalyst award. The second author is also funded by DARPA under Contract No. HR001120C0084, as well as a Security and Privacy research award from Google. The third author is funded by Concordium Blockhain Research Center, Aarhus University, Denmark. The forth author is supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project and is supported by DARPA under Agreements No. HR00112020021 and Agreements No. HR001120C0084. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

[B$^+$] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BCC$^+$15] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on DDH. In Günther Pernul, Peter Y. A. Ryan,

2 bits per AND gate. Finally, there are 44 parallel iterations of the online phase. Thus, in total, each instance of LowMC requires 89,760 bits = 11.2KB.

[14] We note that extracting concrete proof sizes for [EZS$^+$19] is difficult, as their work only includes the results in graph form. We refer the reader to Figure 1 in their work.

and Edgar R. Weippl, editors, *ESORICS 2015, Part I*, volume 9326 of *LNCS*, pages 243–265. Springer, Heidelberg, September 2015.

[BCFK19] Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255, 2019. `https://eprint.iacr.org/2019/1255`.

[BCOS20] Cecilia Boschini, Jan Camenisch, Max Ovsiankin, and Nicholas Spooner. Efficient post-quantum SNARKs for RSIS and RLWE and their applications to privacy. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 247–267. Springer, Heidelberg, 2020.

[BCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

[Ber08] Daniel Bernstein. Chacha, a variant of salsa20. 01 2008.

[BH05] Michael Ben-Or and Avinatan Hassidim. Fast quantum byzantine agreement. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 481–485. ACM Press, May 2005.

[BK10] Zvika Brakerski and Yael Tauman 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`.

[BKM09] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. *Journal of Cryptology*, 22(1):114–138, January 2009.

[BMRS20] Carsten Baum, Alex J. Malozemoff, Marc Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for arithmetic circuits with nested disjunctions. Cryptology ePrint Archive, Report 2020/1410, 2020. `https://eprint.iacr.org/2020/1410`.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252. Springer, Heidelberg, December 2008.

[CDG$^+$17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1825–1842. ACM Press, October / November 2017.

[COS19] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. Cryptology ePrint Archive, Report 2019/1076, 2019. `https://eprint.iacr.org/2019/1076`.

[CWLY06] Sherman S. M. Chow, Victor K.-W. Wei, Joseph K. Liu, and Tsz Hon Yuen. Ring signatures without random oracles. In Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Lin, Shiuhpyng Shieh, and Sushil Jajodia, editors, *ASIACCS 06*, pages 297–302. ACM Press, March 2006.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 356–383. Springer, Heidelberg, August 2019.

[DRS18] David Derler, Sebastian Ramacher, and Daniel Slamanig. Post-quantum zero-knowledge proofs for accumulators with applications to ring signatures from symmetric-key primitives. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 419–440. Springer, Heidelberg, 2018.

[ESLL19] Muhammed F. Esgin, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. Lattice-based zero-knowledge proofs: New techniques for shorter and faster constructions and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 115–146. Springer, Heidelberg, August 2019.

[EZS$^+$19] Muhammed F. Esgin, Raymond K. Zhao, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. MatRiCT: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 567–584. ACM Press, November 2019.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GGHAK21] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose σ-protocols for disjunctions. Cryptology ePrint Archive, Report 2021/422, 2021. `https://eprint.iacr.org/2021/422`.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[git20] Reverie (github project), 2020. `https://github.com/trailofbits/reverie`.

[GK03] S. Goldwasser and Y. T. Kalai. On the (in)security of the fiat-shamir paradigm. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 102–113, 2003.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 569–598. Springer, Heidelberg, May 2020.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

[KZ20] Daniel Kales and Greg Zaverucha. Improving the performance of the picnic signature scheme. Cryptology ePrint Archive, Report 2020/427, 2020. `https://eprint.iacr.org/2020/427`.

[LLNW16] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 1–31. Springer, Heidelberg, May 2016.

[LNS21] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. SMILE: Set membership from ideal lattices with applications to ring signatures and confidential transactions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 611–640, Virtual Event, August 2021. Springer, Heidelberg.

[LRR$^+$19] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, November 2019.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 326–355. Springer, Heidelberg, August 2019.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press, May 2013.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, 2008.

[NMRL16] Shen Noether, Adam Mackenzie, and the Monero Research Lab. Ring confidential transactions. *Ledger*, 1:1–18, Dec. 2016.

[Noe15] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. `https://eprint.iacr.org/2015/1098`.

[OANWO20] Jack O'Connor, Jean-Phillip Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. Blake3: One function, fast everywhere. 01 2020.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Heidelberg, December 2001.

[SALY17] Shi-Feng Sun, Man Ho Au, Joseph K. Liu, and Tsz Hon Yuen. RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 456–474. Springer, Heidelberg, September 2017.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[ST99] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 555–572. Springer, Heidelberg, August 1999.

[TKS$^+$19] Wilson Abel Alberto Torres, Veronika Kuchta, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, and Jacob Cheng. Lattice RingCT V2.0 with multiple input and multiple output wallets. In Julian Jang-Jaccard and Fuchun Guo, editors, *ACISP 19*, volume 11547 of *LNCS*, pages 156–175. Springer, Heidelberg, July 2019.

[TSS$^+$18] Wilson Abel Alberto Torres, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, Veronika Kuchta, Nandita Bhattacharjee, Man Ho Au, and Jacob Cheng. Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice RingCT v1.0). In Willy Susilo and Guomin Yang, editors, *ACISP 18*, volume 10946 of *LNCS*, pages 558–576. Springer, Heidelberg, July 2018.

[TSSK20] Wilson Alberto Torres, Ron Steinfeld, Amin Sakzad, and Veronika Kuchta. Post-quantum linkable ring signature enabling distributed authorised ring confidential transactions in blockchain. Cryptology ePrint Archive, Report 2020/1121, 2020. `https://eprint.iacr.org/2020/1121`.

[Unr17] Dominique Unruh. Post-quantum security of Fiat-Shamir. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 65–95. Springer, Heidelberg, December 2017.

[XZZ$^+$19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Heidelberg, August 2019.

[YSL+20] Tsz Hon Yuen, Shifeng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu. RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 464–483. Springer, Heidelberg, February 2020.

[Zav20] Greg Zaverucha. The picnic signature algorithm. Technical report, 2020. `https://github.com/microsoft/Picnic/raw/master/spec/spec-v3.0.pdf`.

A Preliminaries

In this section, we give formal definitions of the primitives used in our constructions.

A.1 Sigma Protocols

Σ-protocols are a large class of particularly 'simple' zero-knowledge proofs of knowledge. They share three defining qualities:

- *3-round Arthur-Merlin:* the protocol consists of 3 moves: (1) a message 'a' from the prover to the verifier. (2) a random message 'c' from the verifier to the prover, sampled independently of a. (3) a message 'z' from the prover to the verifier.
- *Special Honest-Verifier Zero-Knowledge:* there exists a polynomial-time simulator which given the statement 'x' and challenge 'e' produces the same distribution over transcripts (a, e, z) as the honest interaction between the prover and verifier with the challenge 'e'.
- *s-Special Soundness:* from s distinct transcripts (a, e_1, z_1), ..., (a, e_s, z_s), sharing the first message 'a', but with distinct challenges 'e', there exists a polynomial-time extractor which recovers a witness 'w' for 'x'.

A.2 Fiat-Shamir Transform

Any Σ-protocol can be compiled into a NIZKPoK by generating $e = \mathsf{H}(a)$ where H is modelled as a random oracle, rather than having the verifier send e. To check the validity of the resulting proof $\pi = (a, e, z)$, the NIZKPoK verifier emulates the verifier from the Σ-protocol on the transcript π and checks that $e = \mathsf{H}(a)$. When H is instantiated with any concrete family of hash functions there are (contrived) examples where the Fiat-Shamir transform is unsound [GK03], however in practice the transform appears to yield secure and highly efficient NIZKPoKs from natrual interactive proofs. The CRS for the resulting NIZKPoK is a description of a programmable hash function H, the extractor/simulator is obtained from the Σ-protocol extractor/simulator in the natural way.

A.3 Accumulators

Let κ be the security parameter. A secure accumulator for inputs in $\mathcal{Y}$ is tuple of the 4 PPT algorithms ($\mathsf{Acc.Gen}$, $\mathsf{Acc.Eval}$, $\mathsf{Acc.Proof}$, $\mathsf{Acc.Verify}$) defined as follows:

- $\mathsf{Acc.Gen}(1^\kappa, n)$: On input κ and the number of values that can be securely accumulated n, this algorithm returns a key key.
- $\mathsf{Acc.Eval}(\mathsf{key}, Y)$: On input the key and accumulation set $Y = \{y_1, \ldots, y_{n'}\} \in \mathcal{Y}^{n'}$, where $n' \leq n$, this algorithm returns an accumulated value z and auxiliary information aux.
- $\mathsf{Acc.Proof}(\mathsf{key}, y, z, \mathsf{aux})$: On input key, a value y, an accumulated value z of some set Y, and some auxiliary information aux, this algorithm returns a proof π if $y \in Y$, else it returns the special symbol $\perp$.

- $\mathsf{Acc.Verify}(\mathsf{key}, y, z, \pi)$: This is a deterministic algorithm that takes in a key key, a value y, a proof π, and an accumulated value z, and returns a bit $b \in \{0, 1\}$.

These algorithms satisfy the following properties:

- **Completeness:** For any input set $Y = \{y_1, \ldots, y_{n'}\} \in \mathcal{Y}^{n'}$, and and $i \in [n]$, it holds that:

$$\begin{aligned}\Pr[&\mathsf{Acc.Verify}(\mathsf{key}, y, z, w) = 1 \mid \\ &\qquad \mathsf{key} \leftarrow \mathsf{Acc.Gen}(1^\kappa) z, \mathsf{aux} \leftarrow \mathsf{Acc.Eval}(\mathsf{key}, Y), \\ &\qquad\qquad w \leftarrow \mathsf{Acc.Proof}(\mathsf{key}, y, z, \mathsf{aux})] = 1\end{aligned}$$

- **Soundness:** No PPT adversary Adv, can win the following game with more than negligible probability (in κ):

  1. The challenger samples $\mathsf{key} \leftarrow \mathsf{Acc.Gen}(1^\kappa)$ and sends to Adv.
  2. Adv responds with $(\{y_j\}_{j\in[n']})$.
  3. The challenger computes $z, \mathsf{aux} \leftarrow \mathsf{Acc.Eval}(\mathsf{key}, Y = \{y_1, \ldots, y_n\}$ and sends (z, aux) to Adv.
  4. Adv responds with a pair (y', w), and wins if $\mathsf{Acc.Verify}(\mathsf{key}, y', z, w) = 1$ and $y' \neq y_i$ for every $i \in [n]$.

A.4 Merkle Hash Proof System

In this section, we give a definition of the Merkle Hash Proof System [Mer88]. For simplicity, we give a definition based on a key-less hash function. If any leaves does not have enough entropy, it must be committed to using a hiding commitment scheme (*e.g.* adding random salt).

A Merkle hash proof system corresponding to a hash function $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is defined by a tuple of algorithms $(\mathsf{Merkle.Hash}, \mathsf{Merkle.Proof}, \mathsf{Merkle.Verify})$ as follows:

- $\mathsf{Merkle.Hash}(x_1, \ldots, x_n)$: On input a vector $x_1, \ldots, x_n$, the Merkle hash algorithm computes and returns a hash y using a Merkle tree as follows:
  - For each $i \in [n]$, compute $y_i^0 = \mathsf{H}(x_i)$.
  - For each $\ell \in [\log(n)]$ and $i \in [n/2^\ell]$,[15] compute $y_i^\ell = \mathsf{H}(y_{2i-1}^{\ell-1} || y_{2i}^{\ell-1})$. Set $y = y_1^{\log(n)}$.
- $\mathsf{Merkle.Proof}(x_1, \ldots, x_n, x_i)$: On input a vector $x_1, \ldots, x_n$, and an element x_i, the Merkle proof algorithm computes and outputs a proof p as follows:
  - For each $k \in [n]$, compute $y_k^0 = \mathsf{H}(x_k)$.
  - For each $\ell \in [\log(n)]$ and $k \in [n/2^\ell]$, compute $y_k^\ell = \mathsf{H}(y_{2k-1}^{\ell-1} || y_{2k}^{\ell-1})$.
  - Initialize $p = \{(i, \mathsf{sibling}(y_i^0))\}$ and for each $\ell \in [\log(n)]$, set $p = p \cup \{(\lceil i/2^\ell \rceil, \mathsf{sibling}(y_{\lceil i/2^\ell \rceil}^\ell))\}$.
- $\mathsf{Merkle.Verify}(x_i, y, p)$: On input an element x_i, Merkle hash y, and a Merkle proof p, the Merkle verification algorithm parses $p = ((i_0, x^0), \ldots, (i_{\log(n)}, x^{\log(n)}))$ and returns $\{0,1\}$, proceeding as follows:
  - If i_0 is an even number, compute $y^1 = \mathsf{H}(\mathsf{H}(x_i) || x^0)$, else compute $y^1 = \mathsf{H}(x^0 || \mathsf{H}(x_i))$.
  - For each $\ell \in [\log(n)]$, if i_ℓ is an even number, compute $y^\ell = \mathsf{H}(\mathsf{H}(y^{\ell-1}) || x^\ell)$, else compute $y^\ell = \mathsf{H}(x^\ell || \mathsf{H}(y^{\ell-1}))$.
  - If $y^{\log(n)} = y$, output 1; else, output 0.

[15] For simplicity, we assume that n is a power of 2. The general case follows either by including additional elements 0^λ or by just computing a canonical balanced tree.

A Merkle Hash Proof System has the following properties.

Theorem 1 (Merkle Hash Proof System) *Assuming existence of a length-halving, seeded, collision resistant hash function* $\mathsf{H} : \{0,1\}^*\lambda \to \{0,1\}^\lambda$, *the Merkle hash proof system* ($\mathsf{Merkle.Hash}, \mathsf{Merkle.Proof}, \mathsf{Merkle.Verify}$) *satisfies the following properties:*

- ***Completeness:*** *For any input string* $x_1, \ldots, x_n \in \{0,1\}^{n\lambda}$ *and* $i \in [n]$, *it holds that:*

$$\Pr\left[\mathsf{Merkle.Verify}(x_i, y, p) = 1 \,\middle|\, \begin{array}{c} y = \mathsf{Merkle.Hash}(x_1, \ldots, x_n) \\ p = \mathsf{Merkle.Proof}(x_1, \ldots, x_n, x_i) \end{array}\right] = 1$$

- ***Soundness:*** *No PPT adversary* Adv, *can win the following game with more than negligible probability (in* κ*):*

  1. Adv *responds with* $(i, \{x_j\}_{j\in[n]\setminus\{i\}})$.
  2. *The challenger samples* $x_i \xleftarrow{\$} \{0,1\}^\lambda$, *computes* $\mathsf{Merkle.Hash}(x_1, \ldots, x_n) = y$ *and sends* (x_i, y) *to* Adv.
  3. Adv *responds with a pair* (x', p), *and wins if* $\mathsf{Merkle.Verify}(x', y, p) = 1$ *and* $x' \neq x_i$ *for every* $i \in [n]$.

A.5 Tree-Based PRG

In order to generate the preprocessing for many parties efficiently (and to send the states of many parties to the verifier succinctly), we requires a puncturable, tree-based pseudo-random generator. This construction uses a single seed as the root of a tree and outputs the leaves. To puncture a leaf value, preventing it from being recovered from the seed, the seed is replaced by all the siblings of nodes along the path from the leaf to the root seed.

Definition 4 *A tree based pseudo-random generator corresponding to a pseudorandom generator* $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ *is defined by a tuple of algorithms* ($\mathsf{TreePRG.Gen}, \mathsf{TreePRG.Punc}, \mathsf{TreePRG.GenPunc}$) *as follows:*

- $\mathsf{TreePRG.Gen}(\mathit{seed}, \ell)$: *On input the seed and number of leaves* ℓ, *this algorithm outputs* ℓ *seeds, computed as follows:*
  - *Set* $\mathit{seed}_1^0 = \mathit{seed}$.
  - *For each* $k \in [\log(\ell)]$ *and* $i \in [2^{k-1}]$, *compute* $\mathit{seed}_{2i-1}^k \| \mathit{seed}_{2i}^k = \mathsf{PRG}(\mathit{seed}_i^{k-1})$
  - *Output* $\mathit{seed}_1^{\log(\ell)}, \ldots, \mathit{seed}_\ell^{\log(\ell)}$
- $\mathsf{TreePRG.Punc}(\mathit{seed}, \ell, q)$: *On input the seed, number of leaves* ℓ *and an index* q, *this algorithm computes a punctured seed* S, *that is computed as follows:*
  - *Set* $\mathit{seed}_1^0 = \mathit{seed}$.
  - *For each* $k \in [\log(\ell)]$ *and* $i \in [2^{k-1}]$, *compute* $\mathit{seed}_{2i-1}^k \| \mathit{seed}_{2i}^k = \mathsf{PRG}(\mathit{seed}_i^{k-1})$
  - *Inititalize* $S = \{\}$. *For each* $k \in [\log(\ell)]$, *set* $S = S \cup \{\mathsf{sibling}(\mathit{seed}^k_{\lfloor \frac{q}{2^{\log(\ell)-k}} \rfloor})\}$.
- $\mathsf{TreePRG.GenPunc}(S, \ell, q)$: *On input the punctured seed* S, *number of leaves* ℓ *and the punctured index* q, *parse* $S = ((1, \mathit{seed}^1), \ldots, (\log(\ell), \mathit{seed}^{\log(\ell)}))$, *this algorithm recovers all the leaf seeds except for the punctured leaf seed as follows:*

– *For each* $k \in [\log(\ell)-1]$, *for each* $i \in [2^k] \backslash \{\lfloor \frac{q}{2^{\log(\ell)-k}} \rfloor\}$, *compute* $\textit{seed}_{2i-1}^{k+1} \| \textit{seed}_{2i}^{k+1} = \mathsf{PRG}(\textit{seed}_i^k)$.

– *Output* $\{\textit{seed}_i^{\log(\ell)}\}_{i \in [\ell] \backslash \{q\}}$.

We require standard notions of pseudorandomness from the PRG, which implies pseudorandomness for the leaves and nodes of the tree.

B Security of our Set Membership Proof System

In this section, we prove zero-knowledge and soundness of the protocol presented in Section 3.

Zero-Knowledge. We can use the simulator of the underlying MPC-in-the-head protocol to design a simulator for the new protocol. Let Sim_Σ be the simulator that exists from the zero-knoweldge property of the underlying Σ protocol. The simulator proceeds as follows:

1. Sample a random $\alpha \xleftarrow{\$} [\ell]$, and a random $(M - \tau\eta)$-subset $S \subset [M]$ and η-subsets $C_1, \ldots, C_\tau$.
2. Use Sim_Σ to sample subsets (of appropriate size) $\mathcal{I}_i \subset [n]$ ($\forall i \in [\tau]$) of the parties.
3. For each $j \in [M] \setminus S$, compute the preprocessing exactly as described in the real protocol.
4. For $j \in [M] \setminus S$, compute the preprocessings exactly as described in the real protocol, except for each $k \neq \alpha$, compute $\mathtt{com}_{j,k}^{(\Delta)} = \mathsf{Com}(0; r_{j,k}^{(\Delta\text{-com})})$ and for each $m \notin \mathcal{I}_i$ compute $\mathtt{com}_{j,m}^{(\mathsf{mask})} = \mathsf{Com}(0; r_{j,m}^{(\mathsf{mask\text{-}com})})$, where i is such that $j \in C_i$.
5. Sample random values $\{w_i\}_{i \in \mathcal{I}}$.
6. Shares of the active element for parties $i \in \mathcal{I}$ are chosen exactly as described in the real protocol.
7. Use these shares and simulator Sim_Σ, to simulate the remaining third round messages (which correspond to the first round messages in the underlying MPC-in-the-head protocol).
8. Last round messages of the underlying MPC-in-the-head protocol are also computed using Sim_Σ.

We now proceed to show that the transcripts output by the simulator is computationally indistinguishable from transcripts of real executions of the protocol with an honest verifier. We do this by constructing a sequence of hybrids as follows:

- $\mathcal{H}_0$: Real Transcript
- $\mathcal{H}_1$: The simulator samples a random $S \subset [M]$, and a random subsets $\mathcal{I}_1, \ldots, \mathcal{I}_\tau$. It behaves exactly like an honest prover, except for each $j \in [M] \setminus S$ and $k \neq \alpha$, it computes $\mathtt{com}_{j,k}^{(\Delta)} = \mathsf{Com}(0; r_{j,k}^{(\Delta\text{-com})})$ and for each $m \notin \mathcal{I}_i$ compute $\mathtt{com}_{j,m}^{(\mathsf{mask})} = \mathsf{Com}(0; r_{j,m}^{(\mathsf{mask\text{-}com})})$,where i is such that $j \in C_i$.

  $\underline{\mathcal{H}_0 \approx_c \mathcal{H}_1 :}$ Since these $\mathtt{com}_{j,k}^{(\Delta)}$ and $\mathtt{com}_{j,m}^{(\mathsf{mask})}$ are never opened, indistinguishability of hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ follows from the hiding property of the commitment scheme.

- $\mathcal{H}_2$: The simulator proceeds exactly as in hybrid $\mathcal{H}_1$, except that it samples uniform $\{w_i\}_{i \in \mathcal{I}}$ and runs Sim_Σ along with preprocessing information to simulate the first and third round messages in the underlying MPC-in-the-head protocol (which correspond to the third and fifth round messages (resp.,) in our compiled protocol).

  $\underline{\mathcal{H}_1 \approx_c \mathcal{H}_2 :}$ Indistinguishability of hybrids $\mathcal{H}_1$ and $\mathcal{H}_2$ follows from the zero-knowledge property of the underlying Σ-protocol.

- $\mathcal{H}_3$: The simulator behaves exactly as in hybrid $\mathcal{H}_2$, except that instead of using the actual α, it samples a random $\alpha \in [\ell]$.

  $\underline{\mathcal{H}_2 \approx_c \mathcal{H}_3 :}$ Since we permute the set before accumulating, $\mathcal{H}_2$ and $\mathcal{H}_3$ are identically distributed.

Soundness. Let the underlying MPC-in-the-head protocol have s-special soundness, require τ repetitions and the soundness error in each reptition be $1/\epsilon$. Intuitively a malicious prover P^* can violate soundness of our compiled protocol, if each opened pre-processing commits to the set $\{x_1, \ldots, x_\ell\}$ (otherwise the verifier will reject), yet in each of the τ online executions either: (1) the η consumed preprocessings C_i does not commit to $x_1, \ldots, x_\ell$ (e.g. by replacing some x_i by x'_i). (2) P^* successfully cheated in an instance of the underlying MPC-in-the-head protocol.

We start by calculating the probability when the above conditions hold true and the malicious prover P^* is able to violate soundness. Later, we will show that if the none of the above happens, then it is possible to extract the *valid* witness from prover's messages corresponding to $s+1$ different verifier challenge messages.

Soundness Error: For notational convenience let $\mathbb{S}(N, S) \coloneqq \prod_{i=0}^{N} \binom{S-i\eta}{\eta}$ be the number of ways to sample N disjoint subsets, each of size η from a set with S elements. Let A be the number of preprocessings in which P^* cheats (e.g. by committing to the wrong set), to bound the probability of successfully cheating in the online execution we consider the probability of at least k online executions being executed with η malicious preprocessing (meaning P^* does not need to cheat in the underlying MPC-in-the-head protocol) and P^* successfully cheating in the underlying MPC-in-the-head protocol for the remaining $\tau - k$ repetitions. The latter is bounded by $1/\epsilon^{(\tau-k)}$ while the former is bounded $\mathbb{S}(k,A)\cdot\mathbb{S}(\tau-k,\tau\eta-k\eta)/\mathbb{S}(\tau,\tau\eta)$. The probability of passing the opening check is $\binom{M-A}{M-\tau\eta}/\binom{M}{M-\tau\eta}$. Using a union bound, the soundness/knowledge error is therefore upper bounded by the maximum over A as:

$$p_{\text{err}} \leq \max_{A} \left\{ \frac{\binom{M-A}{M-\tau\eta}}{\binom{M}{M-\tau\eta}} \cdot \sum_{k=0}^{\lfloor A/\eta \rfloor} \left(\frac{\mathbb{S}(k, A) \cdot \mathbb{S}(\tau - k, \tau\eta - k\eta)}{\mathbb{S}(\tau, \tau\eta)} \cdot 1/\epsilon^{(\tau-k)} \right) \right\}$$

Extracting the Witness: We know show that using $s+1$-transcripts of our compiled protocol, except with probability p_{err}, it is possible to extract the witness from this protocol. In our case, for each $i \in [\tau]$, this witness is $(\{\mathsf{mask}_j\}_{j \in C_i}, x_\alpha, w)$. From soundness of the underlying protocol, it holds that $\mathcal{F}\left(\{\{[\mathsf{mask}_j]_m\}_{j \in C_i}, [x_\alpha]_m, [w]_m\}_{m \in [n]}\right) = 1$. In order to argue soundness of our compiled protocol, all that is left to show is that x_α is indeed a member of the set $\{x_1, \ldots, x_\ell\}$.

Let $(S, \{C_1, \ldots, C_\tau\}, \mathcal{I}_1), \ldots, (S, \{C_1, \ldots, C_\tau\}, \mathcal{I}_s)$ be the S challenges whose corresponding transcripts were used for extracting $(\{\mathsf{mask}_j\}_{j \in C_i}, x_\alpha, w)$. We rely on an accepting transcript for one more challenge of the form $(S', *, *)$, where $S \neq S'$. Given these transcripts, unless the prover is able to guess c, the following holds:

1. On challenge $(S', *, *)$: The verifier gets the randomness used for computing the pre-processings for each $j \in S'$. It uses this to reconstruct the preprocessings for each $j \in S'$.
2. On challenge $(S, \{C_1, \ldots, C_\tau\}, \mathcal{I})$: The verifier gets the randomness used for computing the pre-processings for each $j \in S$. It uses this to reconstruct the preprocessings for each $j \in S$. It also gets some partial randomness to compute some commitments in the remaining $[M] \setminus S$ preprocessings.

From binding property of the commitment scheme, we know that the transcripts obtained across the above two challenges are consistent (in particular, the set $\{x_1, \ldots, x_\ell\}$ was honestly used in the preprocessings and the $\Delta_{\cdot,\alpha}$ values computed using the pre-processing randomness in one transcript is the same as the one obtained from the prover in the third round in the other transcript). Moreover, from soundness of the accumulator, we know that the $\Delta_{\cdot,\alpha}$ values were indeed part of the accumulated sets. It now trivially follows that $x_\alpha = \mathsf{mask}_j + \Delta_{j,\alpha}$ $(\forall j \in C_i)$ is indeed a member of the set $\{x_1, \ldots, x_\ell\}$.