Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains

Dan Boneh, Benedikt Bünz, Ben Fisch
Stanford University

Abstract

We present batching techniques for cryptographic accumulators and vector commitments in groups of unknown order. Our techniques are tailored for distributed settings where no trusted accumulator manager exists and updates to the accumulator are processed in batches. We develop techniques for non-interactively aggregating membership proofs that can be verified with a constant number of group operations. We also provide a constant sized batch non-membership proof for a large number of elements. These proofs can be used to build the first positional vector commitment (VC) with constant sized openings and constant sized public parameters. As a core building block for our batching techniques we develop several succinct proof systems in groups of unknown order. These extend a recent construction of a succinct proof of correct exponentiation, and include a succinct proof of knowledge of an integer discrete logarithm between two group elements. We circumvent an impossibility result for Sigma-protocols in these groups by using a short trapdoor-free CRS. We use these new accumulator and vector commitment constructions to design a stateless blockchain, where nodes only need a constant amount of storage in order to participate in consensus. Further, we show how to use these techniques to reduce the size of IOP instantiations, such as STARKs.

1 Introduction

A cryptographic accumulator [Bd94] is a primitive that produces a short binding commitment to a set of elements together with short membership and/or non-membership proofs for any element in the set. These proofs can be publicly verified against the commitment. The simplest accumulator is the Merkle tree [Mer88], but several other accumulators are known, as discussed below. An accumulator is said to be *dynamic* if the commitment and membership proofs can be updated efficiently as elements are added or removed from the set, at unit cost independent of the number of accumulated elements. Otherwise we say that the accumulator is *static*. A *universal* accumulator is dynamic and supports both membership and non-membership proofs.

A *vector commitment* (VC) is a closely related primitive [CF13]. It provides the same functionality as an accumulator, but for an ordered list of elements. A VC is a *position binding* commitment and can be opened at any position to a unique value with a short proof (sublinear in the length of the vector). The Merkle tree is a VC with logarithmic size openings. Subvector commitments [LM18] are VCs where a subset of the vector positions can be opened in a single short proof (sublinear in the size of the subset).

The typical way in which an accumulator or VC is used is as a communication-efficient authenticated data structure (ADS) for a remotely stored database where users can retrieve individual items along with their membership proofs in the data structure. Accumulators have been used for many applications within this realm, including accountable certificate management [BLL00, NN98], timestamping [Bd94], group signatures and anonymous credentials [CL02], computations on authenticated data [ABC+12], anonymous e-cash [STS99b, MGGR13a], privacy-preserving data outsourcing [Sla12], updatable signatures [PS14, CJ10], and decentralized bulletin boards [FVY14, GGM14].

Our present work is motivated by two particular applications of accumulators and vector commitments: stateless transaction validation in blockchains, or "stateless blockchains" and short interactive oracle proofs (IOPs) [BCS16].

"Stateless" blockchains. A *blockchain* has become the popular term for a ledger-based payment system, in which peer-to-peer payment transactions are asynchronously broadcasted and recorded in an ordered ledger that is replicated across nodes in the network. Bitcoin and Ethereum are two famous examples. Verifying the validity of a transaction requires querying the ledger *state*. The state can be computed uniquely from the ordered log of transactions, but provides a more compact index to the information required for transaction validation.

For example, in Ethereum the state is a key/value store of account balances where account keys are the public key addresses of users. In Bitcoin, the state is the set of *unspent transaction outputs* (UTXOs). In Bitcoin, every transaction completely transfers all the funds associated with a set of source addresses to a set of target addresses. It is only valid if every source address is the output of a previous transaction that has not yet been consumed (i.e. "spent"). It is important that all nodes agree on the ledger state.

Currently, in Bitcoin, every node in the system stores the entire UTXO set in order to verify incoming transactions. This has become cumbersome as the size of UTXO set has grown to gigabytes. An accumulator commitment to the UTXO set would alleviate this need. Transactions would include membership proofs for all its inputs. A node would only need to store the current state of the accumulator and verify transactions by checking membership proofs against the UTXO accumulator state. In fact, with dynamic accumulators, no single node in the network would be required to maintain the entire UTXO set. Only the individual nodes who are interested in a set of UTXOs (e.g. the users who can spend these outputs) would need to store them along with their membership proofs. Every node can efficiently update the UTXO set commitment and membership proofs for individual UTXOs with every new batch of transactions. The same idea can be applied to the Ethereum key-value store using a VC instead of an accumulator.

This design concept is referred to as a "stateless blockchain" [Tod16] because nodes may participate in transaction validation without storing the entire state of the ledger, but rather only a short commitment to the state. The idea of committing to a ledgers state was introduced long before Bitcoin by Sanders and Ta-Shma for E-Cash[STS99a]. While the stateless blockchain design reduces the storage burden of node performing transaction validation, it increases the network communication due to the addition of membership proofs to each transaction payload. A design goal is to minimize the communication impact. Therefore, stateless blockchains would benefit from an accumulator with smaller membership proofs, or the ability to aggregate many membership proofs for a batch of transactions into a single constant-size proof.

Interactive oracle proofs (IOPs). Micali [Mic94] showed how *probabilistically checkable proofs* (PCPs) can be used to construct succinct non-interactive arguments. In this construction the prover commits to a long PCP using a Merkle tree and then uses a random oracle to generate a few random query positions. The prover then verifiably opens the proof at the queried positions by providing Merkle inclusion paths.

This technique has been generalized to the broader class of *interactive oracle proofs* (IOPs)[BCS16]. In an IOP the prover sends multiple proof oracles to a verifier. The verifier uses these oracles to query a small subsets of the proof, and afterwards accepts or rejects the proof. If the proof oracle is instantiated with a Merkle tree commitment and the verifier is public coin, then an IOP can be compiled into a non-interactive proof secure in the random oracle model [BCS16]. In particular, this compiler is used to build short non-interactive (zero-knowledge) proof of knowledge with a quasilinear prover and polylogarithmic verifier. Recent practical instantiations of proof systems from IOPs include Ligero [AHIV17], STARKs [BBHR18], and Aurora [BSCR$^+$18].

IOPs use Merkle trees as a vector commitment. Merkle trees have two significant drawbacks for this application: first, position openings are not constant size, and second, the openings of several positions cannot be compressed into a single constant size proof (i.e. it is not a subvector commitment). A vector commitment with these properties would have dramatic benefits for reducing the communication of an IOP (or size of the non-interactive proof compiled from an IOP).

1.1 Summary of contributions

Our technical contributions consist of a set of batching and aggregation techniques for accumulators. The results of these techniques have a wide range of implications, from concrete practical improvements in the proof-size of IOP-based succinct arguments (e.g. STARKS) and minimizing the network communication blowup of stateless blockchains to theoretical achievements in VCs and IOPs.

To summarize the theoretical achievements first, we show that it is possible to construct a VC with constant size subvector openings and constant size public parameters. Previously, it was only known how to construct a VC with constant size subvector openings and public parameters *linear* in the length of the vector. This has immediate implications for IOP compilers. The Merkle-tree IOP compiler outputs a non-interactive proof that is $O(\lambda q \log n)$ larger (additive blowup) than the original IOP communication, where q is the number of oracle queries, n is the maximum length[1] of the IOP proof oracles, and λ is the Merkle tree security parameter. When replacing the Merkle-tree in the IOP compiler with our new VC, we achieve only $O(r\lambda)$ blowup in proof size, independent of q and n, but dependent on the number of IOP rounds r. In the special case of a PCP there is a single round (i.e. $r = 1$). A similar result was recently demonstrated [LM18] using the vector commitments of Catalano and Fiore (CF) [CF13], but the construction requires the verifier to access public parameters linear in n. It was not previously known how to achieve this with constant size public parameters.

Lai and Malavolta apply the CF vector commitments to "CS-proofs", a special case of a compiled IOP where the IOP is a single round PCP. Instantiated with theoretical PCPs [Kil92, Mic94], this results in the shortest known setup-free non-interactive arguments (for NP) with random oracles consisting of just 2 elements in a hidden order group and 240 additional bits of the PCP proof for 80-bit statistical security. Instantiating the group with class groups and targeting 100-bit security yields a proof of ≈ 540 bytes. However, the verifier must either use linear storage or perform linear work for each proof verification to generate the public proof parameters. In similar vein, we can use our new VCs to build the same non-interactive argument system, but with sublinear size parameters (in fact constant size). Under the same parameters our proofs are slightly larger, consisting of 5 group elements, a 128-bit integer, and the 240 bits of the PCP proof ($\approx 1.3KB$).

Our VCs also make concrete improvements to practical IOPs. Targeting 100-bit security with class groups, replacing Merkle trees with our VCs would incur only 1 KB per round of the IOP. In Aurora [BSCR+18], it was reported that Merkle proofs take up 154 KB of the 222 KB proof for a circuit of size 2^{20}. Our VCs would reduce the size of the proof to less than 100 KB, a 54% reduction. For STARKs, a recent benchmark indicates that the Merkle paths make up over 400 KB of the 600 KB proof for a circuit of 2^{52} gates [BBHR18]. With our VCs, under the same parameters the membership proofs would take up roughly 22 KB, reducing the overall proof size to approximately 222 KB, nearly a 63% reduction.

Furthermore, replacing Merkle trees with our new VCs maintains good performance for proof verification. Roughly, each Merkle path verification of a k-bit block is substituted with k modular multiplications of λ-bit integers. The performance comparison is thus $\log n$ hashes vs k multiplications, which is even an improvement for $k < \log n$. In the benchmarked STARK example, Merkle path verification comprises roughly 80% of the verification time.

[1] In each round of an IOP, the prover prepares a message and sends the verifier a "proof oracle", which gives the verifier random read access to the prover's message. The "length" of the proof oracle is the length of this message.

1.2 Overview of techniques

Batching and aggregation. We use the term *batching* to describe a single action applied to n items instead of one action per item. For example a verifier can batch verify n proofs faster than n times verifying a single membership proof. *Aggregation* is a batching technique that is used when non-interactively combining n items to a single item. For example, a prover can aggregate n membership proofs to a single constant size proof.

Succinct proofs for hidden order groups. Wesolowski [Wes18] recently introduced a constant sized and efficient to verify proof that a triple (u, w, t) satisfies $w = u^{2^t}$, where u and w are elements in a group $\mathbb{G}$ of unknown order. The proof extends to exponents that are not a power of two and still provides significant efficiency gains over direct verification by computation.

We expand on this technique to provide a new proof of knowledge of an exponent, which we call a *PoKE* proof. It is a proof that a computationally bounded prover knows the discrete logarithm between two elements in a group of unknown order. The proof is succinct in that the proof size and verification time is independent of the size of the discrete-log and has good soundness. We also generalize the technique to pre-images of homomorphisms from $\mathbb{Z}^q$ to $\mathbb{G}$ of unknown order. We prove security in the generic group model, where an adversarial prover operates over a generic group. Nevertheless, our extractor is classical and does not get to see the adversary's queries to the generic group oracles. We also rely on a short unstructured common reference string (CRS). Using the generic group model for extraction and relying on a CRS is necessary to bypass certain impossibility results for proofs of knowledge in groups of unknown order [BCK10, TW12].

We also extend the protocol to obtain a (honest verifier zero-knowledge) Σ-Protocol of DLOG in $\mathbb{G}$. This protocol is the first succinct Σ-protocol of this kind.

Distributed accumulator with batching. Next, we extend current RSA-based accumulators [CL02, LLX07] to create a universal accumulator for a distributed/decentralized setting where no single trusted accumulator manager exists and where updates are processed in batches. Despite this we show how membership and non-membership proofs can be efficiently aggregated. Moreover, items can efficiently be removed from the accumulator without a trapdoor or even knowledge of the accumulated set. Since the trapdoor is not required for our construction we can extend Lipmaa's [Lip12] work on accumulators in groups of unknown order without a trusted setup by adding dynamic additions and deletions to the accumulator's functionality. Class groups of imaginary quadratic order are a candidate group of unknown order without a trusted setup[BH01].

Batching non-membership proofs. We next show how our techniques can be amplified to create a succinct and efficiently verifiable batch membership and batch non-membership proofs. We then use these batch proofs to create the first vector commitment construction with constant sized batch openings (recently called subvector commitments [LM18]) and $O(1)$ setup. This improves on previous work [CF13, LRY16] which required superlinear setup time and linear public parameter size. It also improves on Merkle tree constructions which have logarithmic sized non-batchable openings. The efficient setup also allows us to create sparse vector commitments which can be used as a key-value map commitment.

Soundness lower bounds in hidden order groups. Certain families of sigma protocols for a relation in a generic group of unknown order can achieve at most soundness $1/2$ per challenge [BCK10, TW12]. Yet, our work gives sigma protocols in a generic group of unknown order that have negligible soundness error. This does not contradict the known impossibility result because our protocols involve a CRS, whereas the family of sigma protocols to which the $1/2$ soundness lower bound applies do not have a CRS. Our results are significant as we show that it suffices

to have a CRS containing two fresh random generic group generators to circumvent the soundness lower bound.

Note that we only prove how to extract a witness from a successful prover that is restricted to the generic group model. Proving extraction from an arbitrary prover under a falsifiable assumption is preferable and remains an open problem.

1.3 Additional related work

Dynamic accumulators can be constructed from the strong RSA assumption in groups of unknown order (such as an RSA group or the class group) [BP97, CL02, LLX07, Lip12], from bilinear maps [DT08, CKS09, Ngu05], and from Merkle hash trees [Mer88, CHKO08]. These accumulators (with the exception of Merkle trees) naturally support batching of membership proofs, but not batching of non-membership proofs. Vector commitments based on similar techniques [LY10, CF13, LRY16] have constant size openings, but large setup parameters.

Accumulators traditionally utilize a trusted *accumulator manager* which possesses a trapdoor to efficiently delete elements from the accumulator. This trapdoor also allows the manager to create membership witnesses for arbitrary elements. Lipmaa [Lip12] was the first to construct a static accumulator without a trusted setup from hidden order groups.

In concurrent work, Chepurnoy et. al. [CPZ18] also note that accumulators and vector commitments can be used to build stateless blockchains. The work proposes a new homomorphic vector commitment based on bilinear maps and multivariate polynomials. This is applied to a blockchain design where each account stores a balance, and balances can be updated homomorphically knowing only the vector commitment to the current blockchain balances. However, the construction requires linear public parameters, does not have a trustless setup, and does not support batching of inclusion proofs. The linear public parameter imply that a bound on the total number of accounts needs to be known at setup time.

2 Preliminaries

Notation.

- $a \parallel b$ is the concatenation of two lists a, b
- $\boldsymbol{a}$ is a vector of elements and a_i is the ith component
- $[\ell]$ denotes the set of integers $\{0, 1, \ldots, \ell - 1\}$.
- $\mathsf{negl}(\lambda)$ is a negligible function of the security parameter λ
- $\mathsf{Primes}(\lambda)$ is the set of integer primes less than 2^λ
- $x \xleftarrow{\$} S$ denotes sampling a uniformly random element $x \in S$.
  $x \xleftarrow{\$} \mathcal{A}(\cdot)$ denotes the random variable that is the output of a randomized algorithm $\mathcal{A}$.
- $GGen(\lambda)$ is a randomized algorithm that generates a group of unknown order in a range $[a, b]$ such that a, b, and $a - b$ are all integers exponential in λ.

2.1 Assumptions

The adaptive root assumption, introduced in [Wes18], is as follows.

Definition 1. *We say that the* ***adaptive root assumption*** *holds for GGen if there is no efficient adversary* $(\mathcal{A}_0, \mathcal{A}_1)$ *that succeeds in the following task. First,* $\mathcal{A}_0$

outputs an element $w \in \mathbb{G}$ and some state*. Then, a random prime ℓ in* $\mathsf{Primes}(\lambda)$ *is chosen and $\mathcal{A}_1(\ell, \mathsf{state})$ outputs $w^{1/\ell} \in \mathbb{G}$. More precisely, for all efficient $(\mathcal{A}_0, \mathcal{A}_1)$:*

$$\mathsf{Adv}^{\mathrm{AR}}_{(\mathcal{A}_0,\mathcal{A}_1)}(\lambda) := \Pr\left[u^\ell = w \neq 1 \ : \ \begin{array}{l} \mathbb{G} \xleftarrow{\$} GGen(\lambda) \\ (w, \mathsf{state}) \xleftarrow{\$} \mathcal{A}_0(\mathbb{G}) \\ \ell \xleftarrow{\$} \mathsf{Primes}(\lambda) \\ u \xleftarrow{\$} \mathcal{A}_1(\ell, \mathsf{state}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

The adaptive root assumption implies that the adversary can't compute the order of any non trivial element. For any element with known order the adversary can compute arbitrary roots that are co-prime to the order. This immediately allows the adversary to win the adaptive root game. For the group Z_N this means that we need to exclude $\{-1, 1\}$

We will also need the strong RSA assumption for general groups of unknown order. The adaptive root and strong RSA assumptions are incomparable. The former states that it is hard to take a random root of a chosen group element, while the latter says that it is hard to take a chosen root of a random group element. In groups of unknown order that do not require a trusted setup the adversary A additionally gets access to $GGen$'s random coins.

Definition 2 (Strong RSA assumption)**.** *GGen satisfies the strong RSA assumption if for all efficient $\mathcal{A}$:*

$$\Pr\left[u^\ell = g \text{ and } \ell \text{ is an odd prime} : \begin{array}{l} \mathbb{G} \xleftarrow{\$} GGen(\lambda),\ g \xleftarrow{\$} \mathbb{G}, \\ (u, \ell) \in \mathbb{G} \times \mathbb{Z} \xleftarrow{\$} \mathcal{A}(\mathbb{G}, g) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

2.2 Generic group model for groups of unknown order

We will use the generic group model for groups of unknown order as defined by Damgard and Koprowski [DK02]. The group is parameterized by two integer public parameters A, B. The order of the group is sampled uniformly from $[A, B]$. The group $\mathbb{G}$ is defined by a random injective function $\sigma : \mathbb{Z}_{|\mathbb{G}|} \to \{0,1\}^\ell$, for some ℓ where $2^\ell \gg |\mathbb{G}|$. The group elements are $\sigma(0), \sigma(1), \ldots, \sigma(|\mathbb{G}| - 1)$. A *generic group algorithm* $\mathcal{A}$ is a probabilistic algorithm. Let $\mathcal{L}$ be a list that is initialized with the encodings given to $\mathcal{A}$ as input. The algorithm can query two generic group oracles:

- $\mathcal{O}_1$ samples a random $r \in \mathbb{Z}_{|\mathbb{G}|}$ and returns $\sigma(r)$, which is appended to the list of encodings $\mathcal{L}$.
- When $\mathcal{L}$ has size q, the second oracle $\mathcal{O}_2(i, j, \pm)$ takes two indices $i, j \in \{1, \ldots, q\}$ and a sign bit, and returns $\sigma(x_i \pm x_j)$, which is appended to $\mathcal{L}$.

Note that unlike Shoup's generic group model [Sho97], the algorithm is not given $|\mathbb{G}|$, the order of the group $\mathbb{G}$.

2.3 Argument systems

An argument system for a relation $\mathcal{R} \subset \mathcal{X} \times \mathcal{W}$ is a triple of randomized polynomial time algorithms $(\mathsf{Pgen}, \mathsf{P}, \mathsf{V})$, where Pgen takes an (implicit) security parameter λ and outputs a common reference string (crs) pp. If the setup algorithm uses only public randomness we say that the setup is transparent and that the crs is unstructured. The prover P takes as input a statement $x \in \mathcal{X}$, a witness $w \in \mathcal{W}$, and the crs pp. The verifier V takes as input pp and x and after interaction with P outputs 0 or 1. We denote the transcript between the prover and verifier by $\langle \mathsf{V}(\mathsf{pp}, x), \mathsf{P}(\mathsf{pp}, x, w) \rangle$ and write $\langle \mathsf{V}(\mathsf{pp}, x), \mathsf{P}(\mathsf{pp}, x, w) \rangle = 1$ to indicate that the verifier accepted the transcript. If V uses only public randomness we say that the protocol is public coin.

Definition 3 (Completeness)**.** *We say that an argument system* $(\mathsf{Pgen}, \mathsf{P}, \mathsf{V})$ *for a relation $\mathcal{R}$ is* ***complete*** *if for all $(x, w) \in \mathcal{R}$:*

$$\Pr\big[\ \langle \mathsf{V}(pp, x), \mathsf{P}(pp, x, w) \rangle = 1 : pp \xleftarrow{\$} \mathsf{Pgen}(\lambda)\big] = 1.$$

We now define soundness and knowledge extraction for our protocols. The adversary is modeled as two algorithms $\mathcal{A}_0$ and $\mathcal{A}_1$, where $\mathcal{A}_0$ outputs the instance $x \in \mathcal{X}$ *after* Pgen is run, and $\mathcal{A}_1$ runs the interactive protocol with the verifier using a state output by A_0. In slight deviation from the soundness definition used in statistically sound proof systems, we do not universally quantify over the instance x (i.e. we do not require security to hold for all input instances x). This is due to the fact that in the computationally-sound setting the instance itself may encode a trapdoor of the crs pp (e.g. the order of a group of unknown order), which can enable the adversary to fool a verifier. Requiring that an efficient adversary outputs the instance x prevents this. In our soundness definition the adversary $\mathcal{A}_1$ succeeds if he can make the verifier accept when no witness for x exists. For the stronger *argument of knowledge* definition we require that an extractor with access to $\mathcal{A}_1$'s internal state can extract a valid witness whenever $\mathcal{A}_1$ is convincing. We model this by enabling the extractor to rewind $\mathcal{A}_1$ and reinitialize the verifier's randomness.

Definition 4 (Arguments (of Knowledge))**.** *We say that an argument system* $(\mathsf{Pgen}, \mathsf{P}, \mathsf{V})$ *is sound if for all poly-time adversaries* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*:*

$$\Pr\left[\begin{array}{ll} \langle \mathsf{V}(\mathit{pp}, x), \mathcal{A}_1(\mathit{pp}, x, \mathit{state})\rangle = 1 & \mathit{pp} \xleftarrow{\$} \mathsf{Pgen}(1^\lambda) \\ \textit{and } \nexists w \quad (x, w) \in \mathcal{R} : & (x, \mathit{state}) \leftarrow \mathcal{A}_0(\mathit{pp}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Additionally, the argument system is an argument of knowledge if for all poly-time adversaries $\mathcal{A}_1$ *there exists a poly-time extractor* Ext *such that for all poly-time adversaries* $\mathcal{A}_0$*:*

$$\Pr\left[\begin{array}{ll} \langle \mathsf{V}(\mathit{pp}, x), \mathcal{A}_1(\mathit{pp}, x, \mathit{state})\rangle = 1 & \mathit{pp} \xleftarrow{\$} \mathsf{Pgen}(1^\lambda) \\ \textit{and } (x, w') \notin \mathcal{R} : & (x, \mathit{state}) \leftarrow \mathcal{A}_0(\mathit{pp}) \\ & w' \xleftarrow{\$} \mathsf{Ext}(\mathit{pp}, x, \mathit{state}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Any argument of knowledge is also sound. In some cases we may further restrict $\mathcal{A}$ in the security analysis, in which case we would say the system is an argument of knowledge for a restricted class of adversaries. For example, in this work we construct argument systems for relations that depend on a group $\mathbb{G}$ of unknown order. In the analysis we replace $\mathbb{G}$ with a generic group and restrict $\mathcal{A}$ to a generic group algorithm that interacts with the oracles for this group. For simplicity, although slightly imprecise, we say the protocol is an *argument of knowledge in the generic group model.* Groth [Gro16] recently proposed a SNARK system for arbitrary relations that is an argument of knowledge in the generic group model in a slightly different sense, where the generic group is used as part of the construction rather than the relation and the adversary is a generic group algorithm with respect to this group generated by the setup.

Definition 5 (Non interactive arguments)**.** *A* ***non-interactive argument system*** *is an argument system where the interaction between* P *and* V *consists of only a single round. We then write the prover* P *as* $\pi \xleftarrow{\$} \mathsf{Prove}(\mathit{pp}, x, w)$ *and the verifier as* $\{0, 1\} \leftarrow \mathsf{Vf}(\mathit{pp}, x, \pi)$*.*

The Fiat-Shamir heuristic [FS87] and its generalization to multi-round protocols [BCS16] can be used to transform public coin argument systems to non-interactive systems.

3 Succinct proofs for hidden order groups

In this section we present several new succinct proofs in groups of unknown order. The proofs build on a proof of exponentiation recently proposed by Wesolowski [Wes18] in the context of verifiable delay functions [BBBF18]. We show that the Wesolowski proof is a *succinct* proof of knowledge of a discrete-log in a group of unknown order. We then derive a *succinct* zero-knowledge argument of

knowledge for a discrete-log relation, and more generally for knowledge of the inverse of a homomorphism $h : \mathbb{Z}^n \to \mathbb{G}$, where $\mathbb{G}$ is a group of unknown order. Using the Fiat-Shamir heuristic, the non-interactive version of this protocol is a special purpose SNARK for the pre-image of a homomorphism.

3.1 A succinct proof of exponentiation

Let $\mathbb{G}$ be a group of unknown order. Let $[\ell] := \{0, 1, \ldots, \ell - 1\}$ and let $\mathsf{Primes}(\lambda)$ denote the set of odd prime numbers in $[0, 2^\lambda]$. We begin by reviewing Wesolowski's (non-ZK) proof of exponentiation [Wes18] in the group $\mathbb{G}$. Here both the prover and verifier are given (u, w, x) and the prover wants to convince the verifier that $w = u^x$ holds in $\mathbb{G}$. That is, the protocol is an argument system for the relation

$$\mathcal{R}_{\mathsf{PoE}} = \big\{\big((u, w \in \mathbb{G}, x \in \mathbb{Z});\ \perp\big)\ :\ w = u^x \in \mathbb{G}\big\}.$$

The verifier's work should be much less than computing u^x by itself. Note that $x \in \mathbb{Z}$ can be much larger than $|\mathbb{G}|$, which is where the protocol is most useful. The protocol works as follows:

<u>Protocol PoE (Proof of exponentiation) for $\mathcal{R}_{\mathsf{PoE}}$ [Wes18]</u>

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$; Inputs: $u, w \in \mathbb{G}$, $x \in \mathbb{Z}$; Claim: $u^x = w$

1. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ to prover.
2. Prover computes the quotient $q = \lfloor x/\ell \rfloor \in \mathbb{Z}$ and residue $r \in [\ell]$ such that $x = q\ell + r$.
   Prover sends $Q \leftarrow u^q \in \mathbb{G}$ to the Verifier.
3. Verifier computes $r \leftarrow (x \bmod \ell) \in [\ell]$ and accepts if $Q^\ell u^r = w$ holds in $\mathbb{G}$.

The protocol above is a minor generalization of the protocol from [Wes18] in that we allow an arbitrary exponent $x \in \mathbb{Z}$, where as in [Wes18] the exponent was restricted to be a power of two. This does not change the soundness property captured in the following theorem, whose proof is given in [Wes18, Prop. 2] (see also [BBF18, Thm. 2]) and relies on the adaptive root assumption for *GGen*.

Theorem 1 (Soundness PoE [Wes18])**.** *Protocol PoE is an argument system for Relation $\mathcal{R}_{\mathsf{PoE}}$ with negligible soundness error, assuming the adaptive root assumption holds for GGen.*

For the protocol to be useful the verifier must be able to compute $r = x \bmod \ell$ faster than computing $u^x \in \mathbb{G}$. The original protocol presented by Wesolowski assumed that $x = 2^T$ is a power of two, so that computing $x \bmod \ell$ requires only $\log(T)$ multiplications in $\mathbb{Z}_\ell$ whereas computing u^x requires T group operations.

For a general exponent $x \in \mathbb{Z}$, computing $x \bmod \ell$ takes $O((\log x)/\lambda)$ multiplications in $\mathbb{Z}_\ell$. In contrast, computing $u^x \in \mathbb{G}$ takes $O(\log x)$ group operations in $\mathbb{G}$. Hence, for the current groups of unknown order, computing u^x takes λ^3 times as long as computing $x \bmod \ell$. Concretely, when ℓ is a 128 bit integer, a multiplication in $\mathbb{Z}_\ell$ is approximately 5000 time faster than a group operation in a 2048-bit RSA group. Hence, the verifier's work is much less than computing $w = u^x$ in $\mathbb{G}$ on its own.

Note that the adaptive root assumption is not only a sufficient security requirement but also necessary. In particular it is important that no known order elements are in the group $\mathbb{G}$. Assume for example that $-1 \in \mathbb{G}$ such that $(-1)^2 = 1 \in \mathbb{G}$. If $g^x = y$ then an adversary can succeed in $\mathsf{PoE}(g, -y, x)$ by setting $Q' \leftarrow -1 \cdot g^{\lfloor x/\ell \rfloor}$. It is, therefore, important to not directly use the multiplicative RSA group $\mathbb{G} := (\mathbb{Z}/N)^*$ but rather $\mathbb{G}^+ := \mathbb{G}/\{-1, 1\}$ as described in [BBF18].

The PoE protocol can be generalized to a relation involving any homomorphism $\phi : \mathbb{Z}^n \to \mathbb{G}$ for which the adaptive root assumption holds in $\mathbb{G}$. The details of this generalization are discussed in Appendix A.1.

3.2 A succinct proof of knowledge of a discrete-log

We next show how the protocol PoE can be adapted to provide an argument of knowledge of discrete-log, namely an argument of knowledge for the relation:

$$\mathcal{R}_{\mathsf{PoKE}} = \big\{\big((u, w \in \mathbb{G});\ x \in \mathbb{Z}\big)\ :\ w = u^x \in \mathbb{G}\big\}.$$

The goal is to construct a protocol that has communication complexity that is much lower than simply sending x to the verifier. As a stepping stone we first provide an argument of knowledge for a modified PoKE relation, where the base $u \in \mathbb{G}$ is fixed and encoded in a CRS. Concretely let CRS consist of the unknown-order group $\mathbb{G}$ and the generator g. We construct an argument of knowledge for the following relation:

$$\mathcal{R}_{\mathsf{PoKE}^*} = \big\{\big(w \in \mathbb{G};\ x \in \mathbb{Z}\big)\ :\ w = g^x \in \mathbb{G}\big\}.$$

The argument modifies the PoE Protocol in that x is not given to the verifier, and the remainder $r \in [\ell]$ is sent from the prover to the verifier:

Protocol PoKE* (Proof of knowledge of exponent) for Relation $\mathcal{R}_{\mathsf{PoKE}^*}$

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$, $g \in \mathbb{G}$; Inputs: $w \in \mathbb{G}$; Witness: $x \in \mathbb{Z}$; Claim: $g^x = w$

1. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
2. Prover computes the quotient $q \in \mathbb{Z}$ and residue $r \in [\ell]$ such that $x = q\ell + r$. Prover sends the pair $(Q \leftarrow g^q,\ r)$ to the Verifier.
3. Verifier accepts if $r \in [\ell]$ and $Q^\ell g^r = w$ holds in $\mathbb{G}$.

Here the verifier does not have the witness x, but the prover additionally sends $r := (x \bmod \ell)$ along with Q in its response to the verifier's challenge. Note that the verifier no longer computes r on its own, but instead relies on the value from the prover. We will demonstrate an extractor that extracts the witness $x \in \mathbb{Z}$ from a successful prover, and prove that this extractor succeeds with overwhelming probability against a generic group prover. In fact, in the next section we will present a generalization of Protocol PoKE* to group representations in terms of bases $\{g_i\}_{i=1}^n$ included in the CRS, i.e. a proof of knowledge of an integer vector $\mathbf{x} \in \mathbb{Z}^n$ such that $\prod_i g_i^{x_i} = w$. We will prove that this protocol is an argument of knowledge against a generic group adversary. The security of Protocol PoKE* above follows as a special case. Hence, the following theorem is a special case of Theorem 7 below.

Theorem 2. Protocol PoKE* *is an argument of knowledge for relation* $\mathcal{R}_{\mathsf{PoKE}^*}$ *in the generic group model.*

An attack. Protocol PoKE* requires the discrete logarithm base g to be encoded in the CRS. When this protocol is applied to a base freely chosen by the adversary it becomes insecure. In other words, Protocol PoKE* is not a secure protocol for the relation $\mathcal{R}_{\mathsf{PoKE}}$.

To describe the attack, let g be a generator of $\mathbb{G}$ and let $u = g^x$ and $w = g^y$ where $y \neq 1$ and x does not divide y. Suppose that the adversary knows both x and y but not the discrete log of w base u. Computing an integer discrete logarithm of w base u is still difficult in a generic group (as follows from Lemma 3), however an efficient adversary can nonetheless succeed in fooling the verifier as follows. Since the challenge ℓ is co-prime with x with overwhelming probability, the adversary can compute $q, r \in \mathbb{Z}$ such that $q\ell + rx = y$. The adversary sends $(Q = g^q,\ r)$ to the verifier, and the verifier checks that indeed $Q^\ell u^r = w$. Hence, the verifier accepts despite the adversary not knowing the discrete log of w base u.

This does not qualify as an "attack" when $x = 1$, or more generally when x divides y, since then the adversary does know the discrete logarithm y/x such that $u^{y/x} = w$.

Extending PoKE for general bases. To obtain a protocol for the relation $\mathcal{R}_{\mathsf{PoKE}}$ we start by modifying protocol PoKE^* so that the prover first sends $z = g^x$, for a fixed base g, and then executes two PoKE^* style protocols, one base g and one base u, in parallel, showing that the discrete logarithm of w base u equals the one of z base g. We show that the resulting protocol is a secure argument of knowledge (in the generic group model) for the relation $\mathcal{R}_{\mathsf{PoKE}}$. The transcript of this modified protocol now consists of two group elements instead of one.

Protocol PoKE (Proof of knowledge of exponent)

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$, $g \in \mathbb{G}$; Inputs: $u, w \in \mathbb{G}$; Witness: $x \in \mathbb{Z}$;
Claim: $u^x = w$

1. Prover sends $z = g^x \in \mathbb{G}$ to the verifier.
2. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
3. Prover finds the quotient $q \in \mathbb{Z}$ and residue $r \in [\ell]$ such that $x = q\ell + r$. Prover sends $Q = u^q$ and $Q' = g^q$ and r to the Verifier.
4. Verifier accepts if $r \in [\ell]$, $Q^\ell u^r = w$, and $Q'^\ell g^r = z$.

The intuition for the security proof is as follows. The extractor first uses the same extractor for Protocol PoKE^* (specified in Theorem 7) to extract the discrete logarithm x of z base g. It then suffices to argue that this extracted discrete logarithm x is a *correct* discrete logarithm of w base u. We use the adaptive root assumption to argue that the extracted x is a correct discrete logarithm of w base u.

We can optimize the protocol to bring down the proof size back to a single group element. We do so in the protocol PoKE2 below by adding one round of interaction. The additional round has no effect on proof size after making the protocol non-interactive using Fiat-Shamir.

Protocol PoKE2 (Proof of knowledge of exponent)

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$; Inputs: $u, w \in \mathbb{G}$; Witness: $x \in \mathbb{Z}$; Claim: $u^x = w$

1. Verifier sends $g \xleftarrow{\$} \mathbb{G}$ to the Prover.
2. Prover sends $z \leftarrow g^x \in \mathbb{G}$ to the verifier.
3. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ and $\alpha \xleftarrow{\$} [0, 2^\lambda)$.
4. Prover finds the quotient $q \in \mathbb{Z}$ and residue $r \in [\ell]$ such that $x = q\ell + r$. Prover sends $Q = u^q g^{\alpha q}$ and r to the Verifier.
5. Verifier accepts if $r \in [\ell]$ and $Q^\ell u^r g^{\alpha r} = wz^\alpha$.

The intuition for the security proof is the same as for Protocol PoKE, but we additionally show that (in the generic group model) a similar extraction argument holds when the prover instead sends $Q \leftarrow u^q g^q$ and r such that $Q^\ell u^r g^r = wz$. The extraction argument uses the fact that with overwhelming probability the generic adversary did not obtain g from any of its group oracle queries prior to forming w and therefore the adversary's representation of w does not contain g as a base with a non-zero exponent. The extractor is able to obtain an exponent x such that $(gu)^x = wz$. This alone does not yet imply that $u^x = w$, however if the prover sends Q, r such that $Q^\ell u^r g^{\alpha r} = wz^\alpha$, then the extractor obtains a fixed x such that $(g^\alpha u)^x = wz^\alpha$ with high probability over the random choice of α. This implies that either $u^x = w$ or w/u^x is an element of low order, which breaks the adaptive root assumption. We summarize this in the following theorem. See Appendix C for the proof.

Theorem 3 (PoKE Argument of Knowledge)**.** *Protocol PoKE and Protocol PoKE2 are arguments of knowledge for relation $\mathcal{R}_{\mathsf{PoKE}}$ in the generic group model.*

The PoKE argument of knowledge can be extended to an argument of knowledge for the pre-image of a homomorphism $\phi : \mathbb{Z}^n \to \mathbb{G}$. This is included in Appendix A.2.

We can also construct a (honest-verifier) zero-knowledge version of the PoKE argument of knowledge protocol using a method similar to the classic Schnorr Σ-protocol for hidden order groups. This is covered in Appendix A.4.

3.3 Aggregating Knowledge of Co-prime Roots

Unlike exponents, providing a root of an element in a hidden order group is already succinct (it is simply a group element). There is a simple aggregation technique for providing a succinct proof of knowledge for multiple *co-prime* roots $x_1, ..., x_n$ simultaneously. This is useful for aggregating PoKE proofs.

When the roots are all for the same element α then the witness is trivially a root α^{1/x^*} where $x^* = x_1 \cdots x_n$. From this witness one can publicly extract the x_ith root of α for each i. We show a method where the elements need not be the same, i.e. the witness is a list of elements $w_1, ..., w_n$ for public elements $\alpha_1, ..., \alpha_n$ and public integers $x_1, ..., x_n$ such that $w_i^{x_i} = \alpha_i$ for each i and $gcd(x_i, x_j) = 1 \forall i, j \in [1, n], i \neq j$. The size of the proof is still a single element.

Concretely the PoKCR protocol is a proof for the relation:

$$\mathcal{R}_{\mathsf{PoKCR}} = \left\{ \left(\alpha \in \mathbb{G}^n;\ \mathbf{x} \in \mathbb{Z}^n \right)\ :\ w = \phi(\mathbf{x}) \in \mathbb{G} \right\}.$$

The proof is the product of witnesses, $w \leftarrow w_1 \cdots w_n$. From this product and the public x_i's and α_i's it is possible to extract an x_ith root of each α_i. (This is not necessarily the same as w_i as roots are not unique). Moreover, the verification algorithm does not need to run this extraction procedure in full, it only needs to check that $w^{x^*} = \prod_i \alpha_i^{x^*/x_i}$. This equation can be verifier with $O(n \log n)$ group exponentiations with exponents of size at most $\max_i |x_i|$ using the optimized recursive **MultiExp** algorithm shown below.

Protocol PoKCR for Relation $\mathcal{R}_{\mathsf{PoKCR}}$
Input: $\mathbb{G}$, $\alpha_1, ..., \alpha_n \in \mathbb{G}$, $x_1, ..., x_n \in \mathbb{Z}$ s.t. $gcd(x_1, ..., x_n) = 1$;
Witness: $\mathbf{w} \in \mathbb{G}^n$ s.t. $w_i^{x_i} = \alpha_i$

1. Prover sends $w \leftarrow \prod_{i=1}^n w_i$ to the Verifier.
2. Verifier computes $x^* \leftarrow \prod_{i=1}^n x_i$, and $y \leftarrow \prod_{i=1}^n \alpha_i^{x^*/x_i}$ using $\mathbf{MultiExp}(n, \boldsymbol{\alpha}, \boldsymbol{x})$. Verifier accepts if $w^{x^*} = y$.

$\mathbf{MultiExp}(n, \boldsymbol{\alpha}, \boldsymbol{x})$:

1. **if** $n = 1$ **return** α
2. $\boldsymbol{\alpha_L} \leftarrow (\alpha_1, ..., \alpha_{n/2})$; $\boldsymbol{\alpha_R} \leftarrow (\alpha_{n/2+1}, ..., \alpha_n)$
3. $\boldsymbol{x_L} \leftarrow (x_1, ..., x_{n/2})$; $\boldsymbol{x_R} \leftarrow (x_{n/2+1}, ..., x_n)$
4. $x_L^* \leftarrow x_1 \cdots x_{n/2}$; $x_R^* \leftarrow x_{n/2+1} \cdots x_n$
5. $L \leftarrow \mathbf{MultiExp}(n/2, \boldsymbol{\alpha_L}, \boldsymbol{x_L})$; $R \leftarrow \mathbf{MultiExp}(n/2, \boldsymbol{\alpha_R}, \boldsymbol{x_R})$
6. **return** $L^{x_R^*} \cdot R^{x_L^*}$

Lemma 1. *Protocol PoKCR is an argument of knowledge for Relation $\mathcal{R}_{\mathsf{PoKCR}}$.*

Proof. We show that given any w such that $w^{x^*} = y = \prod_{i=1}^n \alpha_i^{x^*/x_i}$ it is possible to compute directly an x_ith root of α_i for all i. For each i and $j \neq i$ let $z_{ij} = x^*/(x_i x_j)$. For each i, let $A_j = \prod_{i \neq j} \alpha_i^{z_{ij}}$, then we can express $y = A_j^{x_i} \alpha_i^{x^*/x_i}$. This shows that the element $u = w^{(x^*/x_i)} A_j^{-1}$ is an x_ith root of $\alpha_i^{x^*/x_i}$. Since $gcd(x^*/x_i, x_i) = 1$, there exist Bezout coefficients a, b such that $a(x^*/x_i) + bx_i = 1$. Finally, $u^a \alpha_i^b$ is an x_ith root of α_i as $(u^a \alpha_i^b)^{x_i} = \alpha_i^{(ax^*/x_i) + bx_i} = \alpha_i$. □

Non-interactive proofs All of the protocols can be made non-interactive using the standard Fiat-Shamir transform. In the Fiat-Shamir transform, the prover non-interactively builds a simulated transcript of the protocol by replacing each of the verifier's challenges with a hash of the protocol transcript preceding the challenge

λ: Security Parameter
t: A discrete time counter
A_t: Accumulator value at time t
S_t: The set of elements currently accumulated
w_x^t, u_x^t: Membership and non-membership proofs
pp: Public parameters implicitly available to all methods
upmsg: Information used to update proofs
$\mathbf{Setup}(\lambda, z) \to \mathsf{pp}, A_0$ Generate the public parameters
$\mathbf{Add}(A_t, x) \to \{A_{t+1}, \mathbf{upmsg}\}$ Update the accumulator
$\mathbf{Del}(A_t, x) \to \{A_{t+1}, \mathbf{upmsg}\}$ Delete a value from the accumulator
$\mathbf{MemWitCreate}(A_t, S, x) \to w_x^t$ Create an membership proof
$\mathbf{NonMemWitCreate}(A_t, S, x) \to u_x^t$ Create a non-membership proof
$\mathbf{MemWitUp}(A_t, w_x^t, x, \mathbf{upmsg}) \to w_x^{t+1}$ Update an membership proof
$\mathbf{NonMemWitUp}(A_t, w_x^t, x, \mathbf{upmsg}) \to u_x^{t+1}$ Update a non-membership proof
$\mathbf{VerMem}(A_t, x, w_x^t) \to \{0, 1\}$ Verify membership proof
$\mathbf{VerNonMem}(A_t, x, u_x^t) \to \{0, 1\}$ Verify non-membership proof

Figure 1: A trapdoorless universal accumulator.

using a collision-resistant hash function H as a heuristic substitute for a random oracle. In our protocols, the verifier's challenges are sampled from $\mathsf{Primes}(\lambda)$ and $\mathbb{G}$. Therefore, the non-interactive version must involve a canonical mapping of the output seed σ of the random oracle to a random prime or element of $\mathbb{G}$. Furthermore, it is important that hashing to an element $g \in \mathbb{G}$ does not reveal the discrete log relation between g and any another element (i.e. $g \leftarrow u^\sigma$ is not secure). The simplest way to map σ to a prime in $\mathsf{Primes}(\lambda)$ is find the smallest integer i such that the first λ bits of $H(\sigma, i)$ is prime. More efficient methods are described in Section 7. It is these non-interactive, succinct, and efficiently verifiable proofs that are most useful for the applications discussed later in this paper. Appendix D summarizes the non-interactive proofs that will be used later.

Aggregating PoKE proofs Several non-interactive PoE/PoKE/PoKE2 proofs can be aggregated using the PoKCR protocol. The value Q sent to the verifier in this proof is the ℓth root of yg^{-r}. As long as the primes sampled in each proof instance are distinct then these proofs (specifically the values Q_i) are a witness for an instance of PoKCR. Since the primes are generated by hashing the inputs to the proof they need not be included in the proof.

4 Trapdoorless Universal Accumulator

In this section we describe a number of new techniques for manipulating accumulators built from the strong RSA assumption in a group of unknown order. We show how to efficiently remove elements from the accumulator, how to use the proof techniques from Section 3 to give short membership proofs for multiple elements, and how to non-interactively aggregate inclusion and exclusion proofs. All our techniques are geared towards the setting where there is no trusted setup. We begin by defining what an accumulator is and what it means for an accumulator to be secure.

Our presentation of a trapdoorless universal accumulator mostly follows the definitions and naming conventions of [BCD+17]. Figure 1 summarizes the accumulator syntax and list of associated operations. One notable difference in our syntax is the presence of a common reference string pp generated by the **Setup** algorithm in place of private/public keys.

The security definition we follow [Lip12] formulates an *undeniability* property for accumulators. For background on how this definition relates to others that have been proposed see [BCD+17], which gives generic transformations between different accumulators with different properties and at different security levels.

The following definition states that an accumulator is secure if an adversary cannot construct an accumulator, an element x and a valid membership witness w_x^t

and a non-membership witness u_x^t where w_x^t shows that x is in the accumulator and u_x^t shows that it is not. Lipmaa [Lip12] also defines undeniability without a trusted setup. In that definition the adversary has access to the random coins used by **Setup**.

Definition 6 (Accumulator Security (Undeniability))**.**

$$\Pr\left[\begin{array}{l} pp, A_0 \in \mathbb{G} \xleftarrow{\$} \textbf{\textit{Setup}}(\lambda) \\ (A, x, w_x, u_x) \xleftarrow{\$} \mathcal{A}(pp, A_0) \\ \textbf{\textit{VerMem}}(A, x, w_x^t) \wedge \textbf{\textit{VerNonMem}}(A, x, u_x^t) \end{array}\right] = \mathsf{negl}(\lambda)$$

4.1 Accumulator construction

Several sub-procedures that are used heavily in the construction are summarized below. **Bezout**(x,y) refers to a sub-procedure that outputs Bezout coefficients $a, b \in \mathbb{Z}$ for a pair of co-prime integers x, y (i.e. satisfying the relation $ax + by = 1$). **ShamirTrick** uses Bezout coefficient's to compute an (xy)-th root of a group element g from an x-th root of g and a yth root of g. **RootFactor** is a procedure that given an element $y = g^x$ and the factorization of the exponent $x = x_1 \cdots x_n$ computes an x_i-th root of y for all $i = 1, \ldots, n$ in total time $O(n \log(n))$. Naively this procedure would take time $O(n^2)$. It is related to the **MultiExp** algorithm described earlier and was originally described by [STSY01].

ShamirTrick(w_1, w_2, x, y): [Sha83]
1. **if** $w_1^x \neq w_2^y$ **return** $\perp$
2. $a, b \leftarrow$ **Bezout**(x, y)
3. **return** $w_1^b w_2^a$

$\mathsf{H}_{\mathsf{prime}}(x)$:
1. $y \leftarrow \mathsf{H}(x)$
2. **while** y is not odd prime:
3. $\quad y \leftarrow \mathsf{H}(y)$
4. **return** y

RootFactor$(g, x_1, \ldots, x_n)$:
1. **if** $n = 1$ **return** g
2. $n' \leftarrow \lfloor \frac{n}{2} \rfloor$
3. $g_L \leftarrow g^{\prod_{j=1}^{n'} x_j}$
4. $g_R \leftarrow g^{\prod_{j=n'+1}^{n} x_j}$
5. $L \leftarrow$**RootFactor**$(g_R, x_1, \ldots, x_{n'})$
6. $R \leftarrow$**RootFactor**$(g_L, x_{n'+1}, \ldots, x_n)$
7. **return** $L \parallel R$

Groups of unknown order The accumulator requires a procedure $GGen(\lambda)$ which samples a group of unknown order in which the strong root assumption (Definition 2) holds. One can use the quotient group $(\mathbb{Z}/N)^*/\{-1, 1\}$, where N is an RSA modulus, which may require a trusted setup to generate the modulus N. Alternatively, one can use a class group which eliminates the trusted setup. Note that the adaptive root assumption requires that these groups have no known elements of low order, and hence the group $(\mathbb{Z}/N)^*$ is not suitable because $(-1) \in (\mathbb{Z}/N)^*$ has order two [BBF18]. Given an element of order two it is possible to convince a PoE-verifier that $g^x = -y$ when in fact $g^x = y$.

The basic RSA accumulator. We review he classic RSA accumulator [CL02, Lip12] below, omitting all the procedures that require trapdoor information. All accumulated values are odd primes. If the strong RSA assumption (Definition 2) holds in $\mathbb{G}$, then the accumulator satisfies the undeniability definition [Lip12].

The core procedures for the basic dynamic accumulator are the following:

- **Setup** generates a group of unknown order and initializes the group with a generator of that group.
- **Add** takes the current accumulator A_t, an element from the odd primes domain, and computes $A_{t+1} = A_t$.
- **Del** does not have such a trapdoor and therefore needs to reconstruct the set from scratch. The **RootFactor** algorithm can be used for pre-computation. Storing 2^k elements and doing $n \cdot k$ work, the online removal will only take $(1 - \frac{1}{2}^k) \cdot n$ steps.

- A membership witness is simply the accumulator without the aggregated item.
- A membership non-witness, proposed by [LLX07], uses the fact that for any $x \notin S$, $\gcd(x, \prod_{s \in S} s) = 1$. The Bezout coefficients $(a, b) \leftarrow \mathbf{Bezout}(x, \prod_{s \in S} s)$ are therefore a valid membership witness. The actual witness is the pair (a, g^b) which is short because $|a| \approx |x|$.
- Membership and non-membership witnesses can be efficiently updated as in [LLX07]

Setup(λ):
1. $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$
2. $g \xleftarrow{\$} \mathbb{G}$
3. **return** $\mathbb{G}, g$

Add(A_t, S, x):
1. **if** $x \in S$: **return** A_t
2. **else** :
3. $\quad S \leftarrow S \cup \{x\}$
4. $\quad$ **upmsg** $\leftarrow x$
5. $\quad$ **return** A_t^x, **upmsg**

Del(A_t, S, x):
1. **if** : $x \notin S$: **return** A_t
2. **else** :
3. $\quad S \leftarrow S \setminus \{x\}$
4. $\quad A_{t+1} \leftarrow g^{\prod_{s \in S} s}$
5. $\quad$ **upmsg** $\leftarrow \{x, A_t, A_{t+1}\}$
6. $\quad$ **return** A_{t+1}, **upmsg**

MemWitCreate(A, S, x) :
1. $w_x^t \leftarrow g^{\prod_{s \in S, s \neq x} s}$
2. **return** w_x^t

NonMemWitCreate(A, S, x) :
1. $s^* \leftarrow \prod_{s \in S} s$
2. $a, b \leftarrow$ **Bezout**(s^*, x)
3. $B \leftarrow g^b$
4. **return** $u_x^t \leftarrow \{a, B\}$

VerMem(A, w_x, x) :
1. **return** 1 **if** $(w_x)^x = A$

VerNonMem(A, u_x, x) :
1. $\{a, B\} \leftarrow u_x$
2. **return** 1 **if** $A^a B^x = g$

Theorem 4 (Security accumulator [Lip12])**.** *Assume that the strong RSA assumption (Definition 2) holds in $\mathbb{G}$. Then the accumulator satisfies undeniability (Definition 6) and is therefore secure.*

Proof. We construct an $\mathcal{A}_{RSA}$ that given an $\mathcal{A}_{Acc}$ for the accumulator breaks the strong RSA assumption. $\mathcal{A}_{RSA}$ receives a group $\mathbb{G} \leftarrow GGen(\lambda)$ and a challenge $g \xleftarrow{\$} \mathbb{G}$. We now run $\mathcal{A}_{Acc}$ on input $\mathbb{G}$ and $A_0 = g$. $\mathcal{A}_{Acc}$ returns a tuple (A, x, w_x, u_x) such that $\mathbf{VerMem}(A, x, w_x) = 1$ and $\mathbf{VerNonMem}(A, x, u_x) = 1$. $\mathcal{A}_{RSA}$ parses $(a, B) = u_x$ and computes $B \cdot (w_x)^a$ as the xth root of g. x is an odd prime by definition and $(B \cdot w_x^a)^x = B^x \cdot A^b = g$. This contradicts the strong RSA assumption and thus shows that the accumulator construction satisfies undeniability. □

Updating membership witnesses [CL02, LLX07] Updating membership witnesses when an item is added simply consists of adding the item to the witness which itself is an accumulator. The membership witness is an xth root of the accumulator A_t. After removal of $\hat{x}$, A_{t+1} is an $\hat{x}$th root of A_t. We can use the **ShamirTrick** to compute an $x \cdot \hat{x}$th root of A_t which corresponds to the updated witness. Updating the non-membership witnesses is done by computing the Bezout coefficients between x and the newly added/deleted item $\hat{x}$ and then updating non-membership witness such that it represents the Bezout coefficient's between x and the product of the accumulated elements. For a complete description and correctness proof see [LLX07].

4.2 Batching and aggregation of accumulator witnesses

Aggregating membership witnesses Aggregating membership witnesses for many elements into a single membership witness for the set is straightforward using **ShamirTrick**. However, verification of this membership witness is linear in the number of group operations. Note that the individual membership witnesses can still be extracted from the aggregated witness as $w_x = w_{xy}^y$. Security, therefore, still

holds for an accumulator construction with aggregated membership witnesses. The succinct proof of exponentiation (NI-PoE) enables us to produce a single membership witness that can be verified in constant time. The verification **VerAggMemWit** simply checks the proof of exponentiation.

Aggregating existing membership witnesses for elements in several distinct accumulators (that use the same setup parameters) can be done as well. The algorithm **MemWitX** simply multiplies together the witnesses w_x for an element $x \in A_1$ and w_y for $y \in A_2$ to create an inclusion proof w_{xy} for x *and* y. The verification checks $w_{xy}^{x \cdot y} = A_1^y A_2^x$. If x and y are co-prime then we can directly recover w_x and w_y from the proof w_{xy}. In particular $w_x = \textbf{ShamirTrick}(A_1^y, A_1, w_{xy}^y A_2^{-1}, y, x)$ and $w_y = \textbf{ShamirTrick}(A_2^x, A_2, w_{xy}^x A_1^{-1}, x, y)$.

$\textbf{AggMemWit}(A, w_x, w_y, x, y)$:
1. $w_{x \cdot y} \leftarrow \textbf{ShamirTrick}(A, w_x, w_y, x, y)$
2. **return** $w_{x \cdot y}, \mathsf{NI\text{-}PoE}(w_{x \cdot y}, x \cdot y, A)$

$\textbf{MemWitCreate*}(A, \{x_1, \ldots, x_n\})$:
1. $x^* = \prod_{i=1}^n x_i$
2. $w_{x^*} \leftarrow \textbf{MemWitCreate}(A, x^*)$
3. **return** $w_{x^*}, \mathsf{NI\text{-}PoE}(x, w_{x^*}, A)$

$\textbf{VerMem*}(A, \{x_1, \ldots, x_n\}, w = \{w_x, \pi\})$:
1. **return** $\mathsf{NI\text{-}PoE.verify}(\prod_{i=1}^n x_i, w, A, \pi)$

$\textbf{MemWitX}(A_1, A_2, w_x, w_y, x, y)$:
1. **return** $w_{xy} \leftarrow w_x \cdot w_y$

$\textbf{VerMemWitX}(A_1, A_2, w_{xy}, x, y)$:
1. **if** $gcd(x, y) \neq 1$
2. **return** $\perp$
3. **else**
4. **return** $w_{xy}^{x \cdot y} \leftarrow A_1^y A_2^x$

Distributed accumulator updates In the decentralized/distributed setting, the accumulator is managed by a distributed network of participants who only store the accumulator state and a subset of the accumulator elements along with their membership witnesses. These participants broadcast their own updates and listen for updates from other participants, updating their local state and membership witnesses appropriately when needed.

We observe that the basic accumulator functions do not require a trapdoor or knowledge of the entire state, summarized in Figure 2. In particular, deleting an item requires knowledge of the item's current membership witness (the accumulator state after deletion is this witness). Moreover, operations can be performed in batches as follows:

The techniques are summarized as follows:

- **BatchAdd** An NI-PoE proof can be used to improve the amortized verification efficiency of a batch of updates that add elements $x_1, ..., x_m$ at once and update the accumulator to $A_{t+1} \leftarrow A_t^{x^*}$. A network participant would check that $x^* = \prod_i x_i$ and verify the proof rather than compute the m exponentiations.

- **BatchDel** Deleting elements in a batch uses the **AggMemWit** function to a compute the aggregate membership witness from the individual membership witnesses of each element. This is the new state of the accumulator. A NI-PoE proof improves the verification efficiency of this batch update.

- **CreateAllMemWit** It is possible for users to update membership and non-membership witnesses [LLX07]. The updates do not require knowledge of the accumulated set S but do require that every accumulator update is processed. Since this is cumbersome some users may rely on service providers for maintaining the witness. The service provider may store the entire state or just the users witnesses. Creating all users witnesses naively requires $O(n^2)$ operations. Using the **RootFactor** algorithm this time can be reduced to $O(n \log(n))$ operations or amortized $O(\log(n))$ operations per witness.

[1]The condition that $gcd(x, y) = 1$ is minor as we can simply use a different set of primes as the domains for each accumulator. Equivalently we can utilize different collision resistant hash functions with prime domain for each accumulator. The concrete security assumption would be that it is difficult to find two values a, b such that both hash functions map to the same prime. We utilize this aggregation technique in our IOP application (Section 6.2).

- **CreateManyNonMemWit** Similarly to **CreateAllMemWit** it is possible to create m non-membership witness using $O(\max(n,m) + m\log(m))$ operations. This stands in contrast to the naive algorithm that would take $O(m \cdot n)$ operations. The algorithm is in Figure 4.2.

Add(A_t, x):
1. **return** A_t^x

BatchAdd$(A_t, \{x_1, \ldots, x_m\})$:
1. $x^* \leftarrow \prod_{i=1}^m x_i$
2. $A_{t+1} \leftarrow A_t^{x^*}$
3. **return** $A_{t+1}, \mathsf{NI\text{-}PoE}(x^*, A_t, A_{t+1})$

DelWMem(A_t, w_x^t, x):
1. **if VerMem**$(A_t, w_x^t, x) = 1$
2. **return** w_x^t

BatchDel$(A_t, (x_1, w_{x_1}^t) \ldots, (x_m, w_{x_m}^t))$:
1. $A_{t+1} \leftarrow w_{x_1}^t$
2. $x^* \leftarrow x_1$
3. **for** $i \leftarrow 2, i \leq m$
4. $\quad A_{t+1} \leftarrow$ **ShamirTrick**$(A_{t+1}, w_{x_i}^t, x, x_i)$
5. $\quad x^* \leftarrow x^* \cdot x_i$
6. **return** $A_{t+1}, \mathsf{NI\text{-}PoE}(x^*, A_{t+1}, A_t)$

CreateAllMemWit(S) :
1. **return RootFactor**(g, S)

Figure 2: Distributed and stateless accumulator functions.

Batching non-membership witnesses A non-membership witness u_x for x in an accumulator with state A for a set S is $u_x = \{a, g^b\}$ such that $as^* + bx = 1$ for $s^* \leftarrow \prod_{s \in S} s$. The verification checks $A^a g^{bx} = g$. Since $gcd(s^*, x) = 1$ and $gcd(s^*, y) = 1$ if and only if $gcd(s^*, xy) = 1$, to batch non-membership witnesses we could simply construct a non-membership witness for $x \cdot y$. A prover computes $a', b' \leftarrow$ **Bezout**(s^*, xy) and sets $u_{xy} \leftarrow a', g^{b'}$. This is still secure as a non-membership witness for both x and y because we can easily extract a non-membership witness for x as well as for y from the combined witness (a', B') by setting $u_x = (a', (B')^y)$ and $u_y = (a', (B')^x)$.

Unfortunately, $|a'| \approx |xy|$ so the size of this batched non-membership witness is linear in the number of elements included in the batch. A natural idea is to set $u_{xy} = (V, B) \leftarrow (A^{a'}, g^{b'}) \in \mathbb{G}^2$ instead of $(a', B) \in \mathbb{Z} \times \mathbb{G}$ as the former has constant size. The verification would check that $VB^{xy} = g$. This idea doesn't quite work as an adversary can simply set $V = gB^{-xy}$ without knowing a discrete logarithm between A and V. Our solution is to use the $\mathsf{NI\text{-}PoKE2}$ protocol to ensure that V was created honestly. Intuitively, soundness is achieved because the knowledge extractor for the $\mathsf{NI\text{-}PoKE2}$ can extract a' such that (a', B) is a standard non-membership witness for xy.

The new membership witness is $V, B, \pi \leftarrow \mathsf{NI\text{-}PoKE(A,v;b)}$. The size of this witness is independent of the size of the statement. We can further improve the verification by adding a proof of exponentiation that the verification equation holds: $\mathsf{NI\text{-}PoE}(x \cdot y, B, g \cdot V^{-1})$. Lastly, recall from Section 3 that the two independent

CreateManyNonMemWit$(A, S, \{x_1, \ldots, x_m\})$:
1. $x^* = \prod_{i=1}^m x_i$
2. $\{a, B\} =$ **NonMemWitCreate**(A, S, x^*)
3. **return BreakUpNonMemWit**$(A, \{a, B\}, \{x_1, \ldots, x_m\})$

BreakUpNonMemWit$(A, \{a, B\}, \{x_1, \ldots, x_m\})$:
1. **if** $m = 1$ **return** $\{a, B\}$
2. $x_L = \prod_{i=1}^{m/2} x_i$
3. $x_R = \prod_{i=\lfloor m/2 \rfloor + 1}^m x_i$
4. $B_L = B^{x_R} A^{\left\lfloor \frac{a}{x_L} \right\rfloor}, a_L = a \bmod x_L$
5. $B_R = B^{x_L} A^{\left\lfloor \frac{a}{x_R} \right\rfloor}, a_R = a \bmod x_R$
6. $u_L =$ **BreakUpNonMemWit**$(A, \{a_L, B_L\}, \{x_1, \ldots, x_{\lfloor m/2 \rfloor}\})$
7. $u_R =$ **BreakUpNonMemWit**$(A, \{a_R, B_R\}, \{x_{\lfloor m/2 \rfloor + 1}, \ldots, x_m\})$
8. **return** $u_L || u_R$

Figure 3: Algorithm for creating multiple non membership witnesses

NI-PoKE2 and NI-PoE proofs can be aggregated into a single group element.
We present the non-membership protocol bellow as **NonMemWitCreate***. The verification algorithm **VerNonMem*** simply verifies the NI-PoKE2 and NI-PoE.

NonMemWitCreate*(A, s^*, x^*) : // $A = g^{s^*}$, $s^* = \prod_{s \in S} s$, $x = \prod x_i, x_i \in \mathsf{Primes}(\lambda)$
1. $a, b \leftarrow$ **Bezout**(s^*, x^*)
2. $V \leftarrow A^a, B \leftarrow g^b$
3. $\pi_V \leftarrow \mathsf{NI\text{-}PoKE2}(A, V; a)$ // $V = A^a$
4. $\pi_g \leftarrow \mathsf{NI\text{-}PoE}(x^*, B, g \cdot V^{-1})$ // $B^x = g \cdot V^{-1}$
5. **return** $\{V, B, \pi_V, \pi_g\}$

VerNonMem*$(A, u = \{V, B, \pi_V, \pi_g\}, \{x_1, \ldots, x_n\})$:
1. **return** $\mathsf{NI\text{-}PoKE2.verify}(A, V, \pi_V) \wedge \mathsf{NI\text{-}PoE.verify}(\prod_{i=1}^{n} x_i, B, g \cdot V^{-1}, \pi_g)$

Soundness of batch non-membership witnesses Using the knowledge extractor for NI-PoKE2 and relying on the soundness of NI-PoE, and given an adversary who outputs a valid batch non-membership witness (V, B, π_V, π_g) for a set of odd prime elements $x_1, ..., x_k$ with respect to an accumulator state A, we can extract individual non-membership witnesses for each x_i. The knowledge extractor for NI-PoKE2 (Theorem 3) obtains a such that $V = A^a$ and the soundness of NI-PoE (Theorem 1) guarantees that $B^{x^*} \cdot V = g$ where $x^* = \prod_i x_i$. Given a and B we can compute a non-membership witness for $x_i | x^*$ as $B^{\frac{x^*}{x_i}}, a$ because $(B^{\frac{x^*}{x_i}})^{x_i} A^a = B^{x^*} V = g$ Recall that we proved the existence of a knowledge extractor only for the interactive form of PoKE2 and soundness for the interactive form of PoE, relying on the generic group model. The existence of a knowledge extractor for NI-PoKE2 and soundness of NI-PoE are derived from the Fiat-Shamir heuristic.

Batch accumulator security We now formally define security for an accumulator with batch membership and non-membership witnesses. The definition naturally generalizes Definition 6. We omit a correctness definition as it follows directly from the definition of the batch witnesses. We assume that correctness holds perfectly.

Definition 7 (Batch Accumulator Security (Undeniability))**.**

$$\Pr\left[\begin{array}{l} pp, A_0 \in \mathbb{G} \xleftarrow{\$} \boldsymbol{Setup}(\lambda) \\ (A, I, E, w_I, u_E) \xleftarrow{\$} \mathcal{A}(pp, A_0) : \\ \boldsymbol{VerMem^*}(A, I, w_I) \wedge \boldsymbol{VerNonMem^*}(A, S, u_S) \wedge I \cap S \neq \emptyset \end{array}\right] = \mathsf{negl}(\lambda)$$

From the batch witnesses w_I and u_S we can extract individual accumulator witnesses for each element in I and S. Since the intersection of the two sets is not empty we have an element x and extracted witnesses w_x and u_x for that element. As in the proof of Theorem 4 this lets us compute and xth root of g which directly contradicts the strong RSA assumption. Our security proof will be in the generic group model as it implies the strong RSA assumption, the adaptive root assumption and can be used to formulate extraction for the PoKE2 protocol. Our security proof uses the interactive versions of PoKE2 and PoE protocols but extraction/soundness holds for their non-interactive variants as well.

Theorem 5. *The batch accumulator construction presented in Section 4.2 is secure (Definition 7) in the generic group model.*

Proof. We will prove security by showing that given an adversary that can break the accumulator security we can construct an efficient extractor that will break the strong RSA assumption (Definition 2). This, however, contradicts the generic group model in which strong RSA holds [DK02]. Given a strong RSA challenge $g \in \mathbb{G}$ we set A_0 the accumulator base value to g. Now assume there exists such an adversary $\mathcal{A}$ that on input $(\mathbb{G}, g)$ with non-negligible probability outputs (A, I, E, w_I, u_E) such that w_I and u_E are valid witnesses for the accumulator A and the inclusion proof elements I intersect with the exclusion proof elements E. Let $x \in I \cap S$ be in the intersection. The batch membership witness w_I is such that $w_I^{\prod_{x_i \in I} x_i} = A$

with overwhelming probability. This follows directly from the soundness of the accompanying PoE proof (Theorem 1). We can directly compute $w_x = w_I^{\prod x_i \in I, x_i \neq x}$, i.e. a membership witness for x.

The batch non-membership witness u_E consists of $B, V \in G$ as well as a PoKE2 and a PoE We now use the PoKE2 extractor to compute $a \in \mathbb{Z}, B \in \mathbb{G}$. Given that the extractor succeeds with overwhelming probability (Theorem 3) and the overwhelming soundness of PoE(Theorem 1), a, B satisfy $A^a B^{\prod_{x_i \in E} x_i} = g$. From this we can compute $B' = B^{\prod_{x_i \in E, x_i \neq x} x_i}$ such that $A^a B'^x = g$. As in the proof of Theorem 4 we can now compute $B' w_x^a$ as an xth root of g. This is because $B'^x (w_x^a)^x = B'^x A^a = g$. This, however, contradicts the strong RSA assumption. □

Aggregating non-membership witnesses We have shown how to create a constant sized batch non-membership witness for arbitrary many witnesses. However this process required knowledge of the accumulated set S. Is it possible to aggregate multiple independent non-membership witnesses into a single constant size witness? We show that we can aggregate unbatched witnesses and the apply the same batching technique to the aggregated witness. This is useful in the stateless blockchain setting where a node may want to aggregate non-membership witnesses created by independent actors. Additionally it will allow us to aggregate vector commitment openings for our novel construction presented in Section 5.

Given two non-membership witness $u_x = \{a_x, B_x\}$ and $u_y = \{a_y, B_y\}$ for two distinct elements x and y and accumulator A we want to create a witness for $x \cdot y$. As shown for batch non-membership witnesses a non-membership witness for $x \cdot y$ is equivalent to a witness for x and y if x and y are co-prime.

Concretely we will compute $a_{xy} \in \mathbb{Z}$ and $B_{xy} \in \mathbb{G}$ such that $A^{a_{xy}} B_{xy}^{x \cdot y} = g$. First we compute $\alpha, \beta \leftarrow \textbf{Bezout}(x, y)$. Then we set $B' \leftarrow B_x^\beta B_y^\alpha$ and set $a' \leftarrow \beta a_x y + \alpha a_y x$. Note that $B' \in \mathbb{G}$, $a' \in \mathbb{Z}$ already satisfy $A^{a'} B'^{xy} = (A^{a_x} B_x^x)^{\beta y} (A^{a_y} B_y^y)^{\alpha x} = g^{\beta y + \alpha x} = g$ but that $|a'|$ is not necessarily less than xy. To enforce this we simply reduce a' mod xy, setting $a_{xy} \leftarrow a' \bmod xy$ and $B_{xy} \leftarrow B' A^{\lfloor \frac{a'}{xy} \rfloor}$. The verification equation $A^{a_{xy}} B_{xy}^{xy} = A^{a'} B'^{xy} = g$ is still satisfied.

The full protocol is presented below:

AggNonMemWit$(A, x, y, u_x = (a_x \in [x], B_x \in \mathbb{G}), u_y = (a_y, B_y))$:

1. $\alpha, \beta \leftarrow \textbf{Bezout}(x, y)$
2. $B' \leftarrow B_x^\beta B_y^\alpha$
3. $a' \leftarrow \beta a_x y + \alpha a_y x$
4. $a_{xy} \leftarrow a' \bmod xy$
5. $B_{xy} \leftarrow B' A^{\lfloor \frac{a'}{xy} \rfloor}$
6. **return** $\{a_{xy}, B_{xy}\}$

As in Theorem 5, non-membership witnesses for x and y individually can be computed from an aggregated non-membership witness for x and y. Note that we can also use a PoKE proof and apply the non-membership batching technique presented above to make the proof constant size. The final witness can be verified using the **VerNonMem*** algorithm.

Unions and Multiset accumulators Our succinct proofs can be used to prove that an accumulator is the union of two other accumulators. This uses a succinct proof of a DDH tuple, another special case of a homomorphism preimage. Further details are given in Appendix B.1. In the distributed accumulator setting, it is necessary to assume that no item is added twice to the accumulator. Otherwise, the distributed delete operation will fail. Alternatively, the construction can be viewed as a multi-set accumulator, where every element has a counter. Generating a valid membership witness for an element requires knowing the count of that element in the accumulator multi-set. Further details on this construction are given in Appendix B.2.

5 Batchable Vector Commitments with Small Parameters

5.1 VC Definitions

We review briefly the formal definition of a vector commitment. We only consider static commitments that do not allow updates, but our scheme can naturally be modified to be dynamic.

Vector commitment syntax A VC is a tuple of four algorithms: $\mathsf{VC.Setup}$, $\mathsf{VC.Com}$, $\mathsf{VC.Open}$, $\mathsf{VC.Verify}$.

1. $\mathsf{VC.Setup}(\lambda, n, \mathcal{M}) \to \mathsf{pp}$ Given security parameter λ, length n of the vector, and message space of vector components $\mathcal{M}$, output public parameters pp, which are implicit inputs to all the following algorithms.

2. $\mathsf{VC.Com}(\boldsymbol{m}) \to \tau, com$ Given an input $\boldsymbol{m} = (m_1, ..., m_n)$ output a commitment com and advice τ.

3. $\mathsf{VC.Update}(com, m, i, \tau) \to \tau, com$ Given an input message m and position i output a commitment com and advice τ.

4. $\mathsf{VC.Open}(com, m, i, \tau) \to \pi$ On input $m \in \mathcal{M}$ and $i \in [1, n]$, the commitment com, and advice τ output an opening π that proves m is the ith committed element of com.

5. $\mathsf{VC.Verify}(com, m, i, \pi) \to 0/1$ On input commitment com, an index $i \in [n]$, and an opening proof π output 1 (accept) or 0 (reject).

If the vector commitment does not have an $\mathsf{VC.Update}$ functionality we call it a *static* vector commitment.

Definition 8 (Static Correctness)**.** *A static vector commitment scheme VC is correct if for all $\boldsymbol{m} \in \mathcal{M}^n$ and $i \in [1, n]$:*

$$\Pr\left[\mathsf{VC.Verify}(com, m_i, i, \pi) = 1 : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{VC.Setup}(\lambda, n, \mathcal{M}) \\ \tau, com \leftarrow \mathsf{VC.Com}(\boldsymbol{m}) \\ \pi \leftarrow VC.Open(com, m_i, i, \tau) \end{array}\right] = 1$$

The correctness definition for dynamic vector commitments also incorporates updates. Concretely whenever $\mathsf{VC.Update}$ is invoked the underlying committed vector $\boldsymbol{m}$ is updated correctly.

Binding commitments The main security property of vector commitments (of interest in the present work) is position binding. The security game augments the standard binding commitment game

Definition 9 (Binding)**.** *A vector commitment scheme VC is position binding if for all $O(\mathsf{poly}(\lambda))$-time adversaries $\mathcal{A}$ the probability over $\mathsf{pp} \leftarrow \mathsf{VC.Setup}(\lambda, n, \mathcal{M})$ and $(com, i, m, m', \pi, \pi') \leftarrow \mathcal{A}(\mathsf{pp})$ the probability that $\mathsf{VC.Verify}(com, m, i, \pi) = \mathsf{VC.Verify}(com, m', i, \pi') = 1$ and $m \neq m'$ is negligible in λ.*

5.2 VC construction

We first present a VC construction for bit vectors, i.e. using the message space $\mathcal{M} = \{0, 1\}$. We then explain how this can be easily adapted for a message space of arbitrary bit length.

Our VC construction associates a unique prime[2] integer p_i with each ith index of the bitvector $\boldsymbol{m}$ and uses an accumulator to commit to the set of all primes

[2]Examples include $\mathsf{H_{prime}}$ (described earlier), or alternatively the function that maps i to the next prime after $f(i) = 2(i+2) \cdot \log_2(i+2)^2$, which maps the integers $[0, N)$ to smaller primes than $\mathsf{H_{prime}}$ (in expectation).

corresponding to indices where $m_i = 1$. The opening of the ith index to $m_i = 1$ is an inclusion proof of p_i and the opening to $m_i = 0$ is an exclusion proof of p_i. By using our accumulator from Section 4, the opening of each index is constant-size. Moreover, the opening of several indices can be batched into a constant-size proof by aggregating all the membership witnesses for primes on the indices opened to 1 and batching all the non-membership witnesses for primes on the indices opened to 0.

The VC for vectors on a message space of arbitrary bit length is exactly the same, interpreting the input vector as a bit vector. Opening a λ-bit component is then just a special case of batch opening several indices of a VC to a bit vector. The full details are in Figure 4.

$\mathsf{VC.Setup}(\lambda)$:
- $A \leftarrow \mathit{Accumulator}.\mathbf{Setup}(\lambda)$
- **return** $pp \leftarrow (A, n)$

$\mathsf{VC.Com}(\boldsymbol{m}, pp)$:
- $\mathcal{P} \leftarrow \{p_i | i \in [1, n] \wedge m_i = 1\}$
- $A.\mathbf{BatchAdd}(\mathcal{P})$
- **return** A

$\mathsf{VC.Update}(b, b' \in \{0, 1\}, i \in [1, n])$:
- **if** $b = b'$ **return** A
- **elseif** $b = 1$
- **return** $A.\mathbf{Add}(p_i)$
- **else**
- **return** $A.\mathbf{Del}(p_i)$

$\mathsf{VC.Open}(b \in \{0, 1\}, i \in [1, n])$:
- **if** $b = 1$
- **return** $A.\mathbf{MemWitCreate}(p_i)$
- **else**
- **return** $A.\mathbf{NonMemWitCreate}(p_i)$

$\mathsf{VC.Verify}(A, b \in \{0, 1\}, i, \pi)$:
- **if** $b = 1$:
- **return** $A.\mathbf{VerMem}(\pi, p_i)$
- **else** :
- **return** $A.\mathbf{VerNonMem}(\pi, p_i)$

$\mathsf{VC.BatchOpen}(\boldsymbol{b} \in \{0, 1\}^m, \boldsymbol{i} \in [1, n]^m)$:
- $\mathsf{Ones} \leftarrow \{j \in [1, m] : b_j = 1\}$
- $\mathsf{Zeros} \leftarrow \{j \in [1, m] : b_j = 0\}$
- $p^+ \leftarrow \prod_{j \in \mathsf{Ones}} p_{\boldsymbol{i}[j]}$; $p^- \leftarrow \prod_{j \in \mathsf{Zeros}} p_{\boldsymbol{i}[j]}$
- $\pi_I \leftarrow A.\mathbf{MemWitCreate*}(p^+)$
- $\pi_E \leftarrow A.\mathbf{NonMemWitCreate*}(p^-)$
- **return** $\{\pi_I, \pi_E\}$

$\mathsf{VC.BatchVerify}(A, \boldsymbol{b}, \boldsymbol{i}, \pi_I, \pi_E)$:
- $\mathsf{Ones} \leftarrow \{j \in [1, m] : b_j = 1\}$
- $\mathsf{Zeros} \leftarrow \{j \in [1, m] : b_j = 0\}$
- $p^+ \leftarrow \prod_{j \in \mathsf{Ones}} p_{\boldsymbol{i}[j]}$; $p^- \leftarrow \prod_{j \in \mathsf{Zeros}} p_{\boldsymbol{i}[j]}$
- **return** $A.\mathbf{VerMem}(p^+, \pi_I) \wedge A.\mathbf{VerNonMem*}(p^-, \pi_E)$

$\mathsf{VC.BatchOpen}^*(\boldsymbol{b} \in \{0, 1\}^m, \boldsymbol{i} \in [1, n]^m)$:
- $\mathsf{Ones} \leftarrow \{j \in [1, m] : b_j = 1\}$
- $\mathsf{Zeros} \leftarrow \{j \in [1, m] : b_j = 0\}$
- $p^+ \leftarrow \prod_{j \in \mathsf{Ones}} p_{\boldsymbol{i}[j]}$; $p^- \leftarrow \prod_{j \in \mathsf{Zeros}} p_{\boldsymbol{i}[j]}$
- $\pi_I \leftarrow A.\mathbf{MemWitCreate*}(p^+)$
- $\pi_E \leftarrow A.\mathbf{NonMemWitCreate}(p^-)$
- **return** $\{\pi_I, \pi_E\}$

$\mathsf{VC.BatchVerify}^*(A, \boldsymbol{b}, \boldsymbol{i}, \pi_I, \pi_E)$:
- $\mathsf{Ones} \leftarrow \{j \in [1, m] : b_j = 1\}$
- $\mathsf{Zeros} \leftarrow \{j \in [1, m] : b_j = 0\}$
- $p^+ \leftarrow \prod_{j \in \mathsf{Ones}} p_{\boldsymbol{i}[j]}$; $p^- \leftarrow \prod_{j \in \mathsf{Zeros}} p_{\boldsymbol{i}[j]}$
- **return** $A.\mathbf{VerMem}(p^+, \pi_I) \wedge A.\mathbf{VerNonMem}(p^-, \pi_E)$

Figure 4: Vector commitment scheme from accumulator with batchable membership and non-membership witnesses.

Both the accumulator's CRS as well as PrimeGen can be represented in constant space independent of n. This means that the public parameters for the vector commitment are also constant-size and independent of n, unlike all previous vector commitments with $O(1)$ size openings [CF13, LRY16, LM18]. The batch opening of several (mixed value) indices consists of 2 elements in $\mathbb{G}$ for the aggregate membership-witness and an additional 5 elements in $\mathbb{G}$ for the batch non-membership witness, plus one λ-bit integer.

Aggregating Openings Just as for our accumulator construction we can aggregate vector commitment openings. The aggregation does not require knowledge of the vector contents and the running time of the aggregation is independent of the length of the vector. The opening of a bit in the vector commitment consists of an accumulator inclusion proof and an exclusion proof, both of which we can aggregate as shown in Section 4.2.

This aggregation protocol works for outputs of VC.Open, but unfortunately it does not extend to outputs of VC.BatchOpen. The latter contain PoKE proofs created by **VerNonMem***, which would somehow need to be aggregated as well along with their inputs. When opening only a small number of bit indices, say in a 256-

bit component of the vector, $\mathsf{VC.BatchOpen}^*$ could be used instead so that these openings can be later aggregated. While the output size of $\mathsf{VC.BatchOpen}^*$ grows linearly in m, the number of batched indices, it still contains only three group elements and an integer whose size is proportional to the product of at most m λ-bit primes. These are the unique primes associated with indices of the vector and heuristically can be chosen to be much smaller than λ bits, i.e. closer to $\log n$ bits. After the aggregation step is completed, the aggregate non-membership witness can be further compressed with a PoKE proof.

This aggregation protocol is important for the account-based stateless blockchain application, described in more detail in Section 6. In this application, there is a distributed network of participants who who each hold openings for only a partial number of the vector components (e.g. every user knows the value corresponding to their own account data). A batch of transactions will typically contain many openings (of small values) produced by many different participants in the network. In this case, it makes sense for the participants to each produce an opening of the form $\mathsf{VC.BatchOpen}^*$ so that after the individual participants post all the openings they can be aggregated into a single constant size value that is persisted in the transaction log.

Optimization The number of group elements can be reduced by utilizing a PoKCR for all of the PoE and PoKE roots involved in the membership/non-membership witness generation. It is important that all PoE and PoKE protocols use different challenges. These challenges are then guaranteed to be co-prime. This reduces the number of opening proof elements to $4 \in \mathbb{G}$ and 1 λ-bit integer.

5.3 Comparison

Table 5.3 compares the performance of our new VC scheme, the Catalano-Fiore (CF)[CF13] RSA-based VC scheme, and Merkle trees. The table assumes the VC input is a length n vector of k bit elements with security parameter λ. The performance for the CF scheme include batch openings which were introduced by Lai and Malatova[LM18]. We note that the **MultiExp** algorithm from Section 3.3 also applies to the CF scheme. In particular it can improve the Setup and Open time. The comparison reflects these improvements.

Metric	This Work	Catalono-Fiore [CF13, LM18]	Merkle Tree
	Setup		
Setup	O(1)	$O(n \cdot \log(n) \cdot \lambda)\ \mathbb{G}$	O(1)
$\lvert\mathsf{pp}\rvert$	O(1)	O(n) $\mathbb{G}$	O(1)
$\mathsf{Com}(\mathbf{m}) \to c$	$O(n \cdot \log(n) \cdot k)\ \mathbb{G}$	$O(n \cdot k)\ \mathbb{G}$	$O(n)\ \mathsf{H}$
$\lvert\mathbf{c}\rvert$	$1\ \mathbb{G}$	$1\ \mathbb{G}$	$1\ \lvert\mathsf{H}\rvert$
	Proofs		
$\mathsf{Open}(m_i, i) \to \pi$	$O(k \cdot n \log n)\ \mathbb{G}$	$O(n \cdot (k+\lambda))\ \mathbb{G}$.	$O(\log n)\ \mathsf{H}$
$\mathsf{Verify}(m_i, i, \pi)$	$O(\lambda)\ \mathbb{G} + k \cdot \log n\ \mathbb{F}$	$O(k+\lambda)\ \mathbb{G}$	$O(\log n)\ \mathsf{H}$
$\lvert\pi\rvert$	$O(1)\ \lvert\mathbb{G}\rvert$	$1\ \lvert\mathbb{G}\rvert$	$O(\log n)\ \lvert\mathsf{H}\rvert$
$\mathsf{Open}(\mathbf{m}, \boldsymbol{i}) \to \pi_{\boldsymbol{i}}$	$O(k \cdot n \log n)\ \mathbb{G}$	$O(n \log n \cdot (k+\lambda))\ \mathbb{G}$	$O(n \log n)\ \mathsf{H}$
$\mathsf{Verify}(\mathbf{m}, \boldsymbol{i}, \pi_{\mathbf{i}})$	$O(\lambda)\ \mathbb{G} + O(k \cdot n \log n)\ \mathbb{F}$	$O(nk)\ \mathbb{G}$	$O(n \log n)\ \mathsf{H}$
$\lvert\pi_{\mathbf{i}}\rvert$	$4\ \lvert\mathbb{G}\rvert + \lambda$	$1\ \lvert\mathbb{G}\rvert$	$O(n \log n)\ \lvert\mathsf{H}\rvert$
Aggregatable	Yes	No	No

Table 1: Comparison between Merkle trees, Catalano-Fiore RSA VCs and the new VCs presented in this work. The input $\boldsymbol{m}$ is a vector of n. $\mathbb{G}$ refers to a group operation in $\mathbb{G}$, $\mathbb{F}$ to a multiplication in a field of size roughly 2^λ, and H to a hash operation. Group operations are generally far more expensive than hashes which are more expensive than multiplication in $\mathbb{F}$. $\lvert\mathbb{G}\rvert$ is the size of a hidden order group element and $\lvert\mathsf{H}\rvert$ is the size of a hash output.

5.4 Key-Value Map Commitment

Our vector-commitment can be used to build a commitment to a key-value map. A key-value map can be built from a sparse vector. The key-space is represented by positions in the vector and the associated value is the data at the keys position. The vector length is exponential in the key length and most positions are zero (null). Our VC commitment naturally supports sparse vectors because the complexity of the commitment is proportional to the number of bit indices that are set to 1, and otherwise independent of the vector length.

Optimization with honest updates In order to commit to arbitrary length values we can hash the value and then commit to the resulting hash. Unfortunately this still requires setting λ bits in the vector commitment which corresponds to adding λ, λ-bit primes to the underlying accumulator. This can make updating the commitment computationally quite expensive. In some settings we can do better than this. Note that the VC and the accumulator definitions (Definition 6) assume that the adversary outputs the commitment. This requirement is too strong for settings where every update follows the rules of the system, i.e. is performed by the challenger. In this case we can implement a key-value map commitment by storing in the VC which keys exist and storing in the accumulator a key, value tuple. If the key already exists then an update will update the entry in the accumulator. Otherwise it will add an entry to the accumulator and set the corresponding VC bit to 1. The construction requires only 1 bit to be stored per key in the VC and 1 entry in the accumulator. The construction also is not secure if the adversary can output an accumulator value as it could contain multiple entries for the same key. We omit a formal security definition and proof but note that security follows directly from the binding guarantees of the underlying accumulator and vector commitment constructions. The VC ensures that each key appears at most ones in the accumulator and the accumulator ensures the integrity of the committed data.

6 Applications

6.1 Stateless Blockchains

UTXO commitment We first consider a simplified blockchain design which closely corresponds to Bitcoin's UTXO design where users own coins and issue transaction by spending old coins and creating new coins. We call the set of unspent coins the UTXO set. Updates to the blockchain can be viewed as asynchronous updates to the UTXO set. In most current blockchain designs (with some exceptions[MGGR13a, BCG+14]) nodes participating in transaction validation store the whole UTXO set and use it to verify whether a coin was unspent. Instead, we consider a blockchain design where the network maintains the UTXO set in a dynamic accumulator [STS99a, TMA13, Tod16, Dra]. We instantiate this accumulator with our new construction from Section 4.1, taking advantage of our distributed batch updates and aggregate membership proofs.

Each transaction block will contain an accumulator state, which is a commitment to the current UTXO set. To spend a coin, a user provides a membership witness for the coin (UTXO) that is being spent inside a transaction. Any validator (aka miner) may verify the transactions against the latest accumulator state and also uses **BatchDel** to delete all spent coins from the accumulator, derive its new state, and output a proof of correctness for the deletions. The proof is propagated to other validators in the network. For the newly minted coins, the validator uses **BatchAdd** to add them to the accumulator and produce a second proof of correctness to propagate. Other validators are able to verify that the accumulator was updated correctly using only a constant number of group operations and highly efficient arithmetic over λ-bit integers.

In this design, users store the membership witnesses for their own coins and are required to update their witnesses with every block of transactions. It is plausible that users use third-party services to help with this maintenance. These services

are not trusted for integrity, but only for availability. Note that a may produce many (e.g. n) membership witnesses at once in $O(n \log(n))$ time using the **CreateAllMemWit** algorithm

Accounts commitment Some currencies such as Ethereum [Woo14] or Stellar [SYB14] use an account-based system where the state is a key-value map. A transaction updates the balances of the sending and the receiving accounts. To enable stateless validation in this setting, a user can provide proofs of the balances of the sending and receiving accounts in the current ledger state. Instead of using an accumulator to commit to this state, we use the new key-value map commitment from Section 5.4. This commitment supports batch distributed updates, similar to our new accumulator. Using the aggregation of vector commitment openings a miner or validator can perform the aggregation and batching operations without storing the state providing efficient proofs that the openings are correct. Other nodes can verify these opening proofs efficiently requiring only a constant number of group operations.

6.2 Short IOPs

Merkle tree paths contribute significant overhead to both the proof size of a compiled IOP proof and its verification time. Vector commitments with smaller openings than Merkle trees, or batchable openings (i.e. subvector commitments), can help reduce this overhead [LM18]. Using our new VCs, the opening proof for each round of the compiled IOP is just 4 group elements in $\mathbb{G}$ and a λ-bit integer (plus one additional element for the VC commitment itself). Instantiating $\mathbb{G}$ with a class group of quadratic imaginary order and tuning security to 128-bits requires elements of size approximately 2048-bits [HM00]. Thus, the VC openings contribute 8320 bits to the proof size per IOP round. When applied to the "CS-proof" SNARK considered by Lai and Malavolta, which is based on a theoretical PCP that checks 3 bits per query and has 80 queries, the proof size is $5 \cdot 2048 + 128 + 3 \cdot 80 = 10608$ bits, or 1.3 KB. This is the shortest (theoretical) setup-free SNARK with sublinear public parameters to date.

Our VCs also achieve concrete improvements to practical IOPs. Targeting 100-bit security in the VC component and otherwise apples-to-apples comparisons with benchmarks for Aurora [BSCR$^+$18] and STARKS [BBHR18], we can conservatively use 2048-bit class group elements. With these parameters, our VCs reduce the size of the Aurora proofs on a 2^{20} size circuit from 222 KB to less than 100 KB, a 54% reduction, and the size of STARK proofs for a circuit of 2^{52} gates from 600 KB to approximately 222 KB, a 63% reduction. This rough estimate is based on the Merkle path length 42 and round number 21 extrapolated from the most recent STARK benchmarks for this size circuit [BBHR18].

Replacing Merkle trees with our VCs does not significantly impact the verification cost, and in some cases it may even improve verification time. Recall that verifying a batch VC proof costs approximately one *lamdba*-bit integer multiplication and a primality check per bit. Furthermore, using the optimization described in Section 7 eliminates the primality checks for the verifier (at a slight cost to the prover). Computing a SHA256 hash function (whether SHA256 or AES with Davies-Meyer) is comparable to the cost of a λ-bit integer multiplication. Thus, as a loose estimate, replacing each Merkle path per query with a single λ-bit multiplication would achieve a factor $\log n = 36$ reduction. In STARKS, Merkle paths are constructed over 256-bit blocks of the proof rather than bits, thus the comparison is 36 hashes vs 256 modular multiplications. The Merkle path validation accounts for 80% of the verification time.

While using our vector commitment has many benefits for IOPs, there are several sever downsides. Our vector commitment is not quantum secure as a quantum computer can find the order of the group and break the Strong-RSA assumption. Merkle trees are more plausibly quantum secure. Additionally, the prover for an IOP instantiated with our vector commitment would be significantly slower than

one with a Merkle tree.

7 Hashing To Primes

Our constructions use a hash-function with prime domains in several places: Elements in the accumulator are mapped to primes, using a collision resistant hash function with prime domain. The vector commitment associates a unique prime with each index. All of the proofs presented in Section 3 use a random prime as a challenge. When the proofs are made non-interactive, using the Fiat-Shamir heuristic the challenge is generated by hashing the previous transcript.

In Section 4.1 we present a simple algorithm for a collision-resistant hash function $\mathsf{H}_{\mathsf{prime}}$ with prime-domain built from a collision resistant hash function H with domain $\mathbb{Z}_{2^\lambda}$. The hash function iteratively hashes a message and a counter, increasing the counter until the output is prime. If we model H as a random function with then the expected running time of $\mathsf{H}_{\mathsf{prime}}$ is $O(\lambda)$. This is because there are $O(\frac{n}{\log(n)})$ primes below n.

The problem of hashing to primes has been studied in several contexts: Cramer and Shoup [CS99] provide a way to generate primes with efficiently checkable certificates. Fouque and Tibouchi[FT14] showed how to quickly generate random primes. Seeding the random generation with a collision resistant hash function can be used to generate an efficient hash function with prime domain. Despite these improvements, the hash function actually introduces a significant overhead for verification and in this section we present several techniques how the hashing can be further sped up.

PoE,PoKE proofs We first investigate the PoE,PoKE family of protocols. In the non-interactive variant the challenge ℓ is generated by hashing the previous transcript to a prime. The protocol can be modified by having the prover provide a short nonce such that $\ell \leftarrow H(transcript||nonce)$ with $\ell \in \mathsf{Primes}(\lambda)$. In expectation the nonce is just $\log(\lambda)$ bits and with overwhelming probability it is less than $2\log(\lambda)$ bits. This modification allows the adversary to produce different challenges for the same transcript. However it does not increase an adversary's advantage. The prover can always alter the input to generate new challenges. By changing the nonce the prover can grind a polynomial number of challenges but the soundness error in all of our protocols is negligible. The change improves the verification as the verifier only needs to do a single primality check instead of λ. The change is particularly interesting if proof verification is done in a circuit model of computation, where variable time operations are difficult and costly to handle. Circuit computations have become increasingly popular for general purpose zero-knowledge proofs[GGPR13, BBB$^+$18, BSCR$^+$18]. Using the adapted protocol verification becomes a constant time operation which uses only a single primality check.

Accumulator A similar improvement can be applied to accumulators. The users can provide a nonce such that $element||nonce$ is accumulated instead of just the element. This of course allows an adversary to accumulate the same element twice. In some applications this is acceptable. In other applications such as stateless blockchains it is guaranteed that no element is accumulated twice(see Section 6). One way to guarantee uniqueness is to commit to the current state of the accumulator for every added element. In an inclusion proof, the prover would provide the nonce as part of the proof. The verifier now only does a single primality check to ensure that $\mathsf{H}(element||nonce)$ is indeed prime. This stands in contrast to $O(\lambda)$ primality checks if $\mathsf{H}_{\mathsf{prime}}$ is used. The nonce construction prohibits efficient exclusion proofs but these are not required in some applications, such as the blockchain application.

Vector Commitments The vector commitment construction uses one prime per index to indicate whether the vector is 1 at that index or 0. The security definition for a vector commitment states that a secure vector commitment cannot be opened

to two different openings at the same index. In our construction this would involve giving both an inclusion as well as an exclusion proof for a prime in an accumulator, which is impossible if the accumulator itself is secure. Using a prime for each index again requires using a collision resistant hash function with prime domain which uses $O(\lambda)$ primality checks or an injective function which runs in time $O(\log(n)^2)$, where n is the length of the vector. What if instead of accumulating a prime for each index we accumulate a random λ bit number at each index? The random number could simply be the hash of the index. Is this construction still secure? First consider the case where each index's number has a unique prime factor. This adapted construction is trivially still secure. What, however, if x_k, associated with index k, is the product of x_i and x_j. Then accumulating x_i and x_j lets an adversary also give an inclusion proof for x_k. Surprisingly, this does still not break security. While it is possible to give an inclusion proof for x_k, i.e. open the vector at index k to 1 it is suddenly impossible to give an exclusion proof for x_k, i.e. open the vector at index k to 0. The scenario only breaks the correctness property of the scheme, in that it is impossible to commit to a vector that is 1 at i and j but 0 at k. In a setting, where the vector commitment is used as a static commitment to a vector, correctness only needs to hold for the particular vector that is being committed to. In the IOP application, described in Section 6.2, the prover commits to a long proof using a vector commitment. If these correctness failures only happen for few vectors, it may still be possible to use the scheme. This is especially true because in the IOP application the proof and also the proof elements can be modified by hashing the proof elements along with a nonce. A prover would modify the nonces until he finds a proof, i.e. a vector that he can commit to. To analyze the number of correctness failures we can compute the probability that a k-bit element divides the product of n k-bit random elements. Fortunately, this question has been analyzed by Coron and Naccache[CN00] with respect to the Gennaro-Halevi-Rabin Signature Scheme[GHR99]. They find that for 50 Million integers and 256-bit numbers the probability that even just a single correctness failure occurs is 1%. Furthermore we find experimentally that for 2^{20} integers and 80-bit numbers only about $8,000$ integers do not have a unique prime factor. Thus, any vector that is 1 at these $8,000$ positions can be committed to using just 80-bit integers. Our results suggest that using random integer indices instead of prime indices can be useful, if a) perfect completeness is not required b) primality checks are a major cost to the verifier.

8 Conclusion

We expect that our techniques and constructions will have more applications beyond what was discussed. Several interesting open questions remain: What practical limitations occur when deploying the scheme? Is it possible to efficiently compute unions of accumulators? This is certainly true for Merkle trees but these do not have the batching properties and constant size of RSA accumulators. Similarly can one build an accumulator with constant sized witnesses from a quantum resistant assumption? Additionally, we hope that this research motivates further study of class groups as a group of unknown order.

Acknowledgments

This work was partially supported by NSF, ONR, the Simons Foundation and the ZCash foundation. We thank Dario Fiore and Oliver Tran for pointing out typos and small errors.

References

[ABC+12] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on authenticated data. In

Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 1–20. Springer, Heidelberg, March 2012.

[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 17*, pages 2087–2104. ACM Press, October / November 2017.

[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.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. `https://eprint.iacr.org/2018/712`.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[BCD+17] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. Cryptology ePrint Archive, Report 2017/043, 2017. `http://eprint.iacr.org/2017/043`.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCK10] Endre Bangerter, Jan Camenisch, and Stephan Krenn. Efficiency limitations for S-protocols for group homomorphisms. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 553–571. Springer, Heidelberg, February 2010.

[BCM05] Endre Bangerter, Jan Camenisch, and Ueli Maurer. Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 154–171. Springer, Heidelberg, January 2005.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

[Bd94] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

[BH01] Johannes Buchmann and Safuat Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2001.

[BLL00] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In S. Jajodia and P. Samarati, editors, *ACM CCS 00*, pages 9–17. ACM Press, November 2000.

[BP97] Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, Heidelberg, May 1997.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *IEEE Symposium on Security and Privacy*, 2014.

[BSCR+18] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for r1cs. Cryptology ePrint Archive, Report 2018/828, 2018. `https://eprint.iacr.org/2018/828`.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, February / March 2013.

[CHKO08] Philippe Camacho, Alejandro Hevia, Marcos A. Kiwi, and Roberto Opazo. Strong accumulators from collision-resistant hashing. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC 2008*, volume 5222 of *LNCS*, pages 471–486. Springer, Heidelberg, September 2008.

[CJ10] Sébastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In Josef Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 179–194. Springer, Heidelberg, March 2010.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 481–500. Springer, Heidelberg, March 2009.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002.

[CN00] Jean-Sébastien Coron and David Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 91–101. Springer, Heidelberg, May 2000.

[CPZ18] Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang. Edrax: A cryptocurrency with stateless transaction validation. Cryptology ePrint Archive, Report 2018/968, 2018. `https://eprint.iacr.org/2018/968`.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. Cryptology ePrint Archive, Report 1999/001, 1999. `http://eprint.iacr.org/1999/001`.

[DK02] Ivan Damgård and Maciej Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 256–271. Springer, Heidelberg, April / May 2002.

[Dra] Justin Drake. Accumulators, scalability of utxo blockchains, and data availability. https://ethresear.ch/t/accumulators-scalability-of-utxo-blockchains-and-data-availability/176.

[DT08] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. http://eprint.iacr.org/2008/538.

[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.

[FT14] Pierre-Alain Fouque and Mehdi Tibouchi. Close to uniform prime number generation with fewer random bits. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 991–1002. Springer, Heidelberg, July 2014.

[FVY14] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A decentralized public key infrastructure with identity retention. Cryptology ePrint Archive, Report 2014/803, 2014. http://eprint.iacr.org/2014/803.

[GGM14] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *NDSS 2014*. The Internet Society, February 2014.

[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.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 123–139. Springer, Heidelberg, May 1999.

[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.

[HM00] Safuat Hamdy and Bodo Möller. Security of cryptosystems based on class groups of imaginary quadratic orders. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 234–247. Springer, Heidelberg, December 2000.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[Lip12] Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 12*, volume 7341 of *LNCS*, pages 224–240. Springer, Heidelberg, June 2012.

[LLX07] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 253–269. Springer, Heidelberg, June 2007.

[LM18] Russell W.F. Lai and Giulio Malavolta. Optimal succinct arguments via hidden order groups. Cryptology ePrint Archive, Report 2018/705, 2018. `https://eprint.iacr.org/2018/705`.

[LRY16] Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl, July 2016.

[LY10] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517. Springer, Heidelberg, February 2010.

[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.

[MGGR13a] 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.

[MGGR13b] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *IEEE Symposium on Security and Privacy*, 2013.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

[Ngu05] L. Nguyen. Accumulators from bilinear maps and applications. *CT-RSA*, 3376:275–292, 2005.

[NN98] Kobbi Nissim and Moni Naor. Certificate revocation and certificate update. In *Usenix*, 1998.

[PS14] Henrich Christopher Pöhls and Kai Samelin. On updatable redactable signatures. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS 14*, volume 8479 of *LNCS*, pages 457–475. Springer, Heidelberg, June 2014.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems (TOCS)*, 1(1):38–44, 1983.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[Sla12] Daniel Slamanig. Dynamic accumulator based discretionary access control for outsourced storage with unlinkable access - (short paper). In Angelos D. Keromytis, editor, *FC 2012*, volume 7397 of *LNCS*, pages 215–222. Springer, Heidelberg, February / March 2012.

[STS99a] 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.

[STS99b] Tomas Sander and Amnon Ta-Shma. Flow control: A new approach for anonymity control in electronic cash systems. In Matthew Franklin, editor, *FC'99*, volume 1648 of *LNCS*, pages 46–61. Springer, Heidelberg, February 1999.

[STSY01] Tomas Sander, Amnon Ta-Shma, and Moti Yung. Blind, auditable membership proofs. In Yair Frankel, editor, *FC 2000*, volume 1962 of *LNCS*, pages 53–71. Springer, Heidelberg, February 2001.

[SYB14] David Schwartz, Noah Youngs, and Arthur Britto. The Ripple Protocol Consensus Algorithm, September 2014.

[TMA13] Peter Todd, Gregory Maxwell, and Oleg Andreev. Reducing UTXO: users send parent transactions with their merkle branches. `bitcointalk.org`, October 2013.

[Tod16] Peter Todd. Making UTXO Set Growth Irrelevant With Low-Latency Delayed TXO Commitments . `https://petertodd.org/2016/delayed-txo-commitments`, May 2016.

[TW12] Björn Terelius and Douglas Wikström. Efficiency limitations of S-protocols for group homomorphisms revisited. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 461–476. Springer, Heidelberg, September 2012.

[Wes18] Benjamin Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. `https://eprint.iacr.org/2018/623`.

[Woo14] Gavin Wood. Ethereum: A secure decentralized transaction ledger. `http://gavwood.com/paper.pdf`, 2014.

A PoE/PoKE Generalizations and Zero Knowledge

A.1 A succinct proof of homomorphism preimage

We observe that the protocol PoE can be generalized to a relation for any homomorphism $\phi : \mathbb{Z}^n \to \mathbb{G}$ for which the adaptive root assumption holds in $\mathbb{G}$. Specifically, Protocol PoHP below is a protocol for the relation:

$$\mathcal{R}_{\phi,\mathsf{PoHP}} = \big\{\big((w \in \mathbb{G},\ \mathbf{x} \in \mathbb{Z}^n);\ \perp\big)\ :\ w = \phi(\mathbf{x}) \in \mathbb{G}\big\}.$$

This generalization will be useful in our applications.

Protocol PoHP (Proof of homomorphism preimage) for $\mathcal{R}_{\phi,\mathsf{PoHP}}$

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$, $\phi : \mathbb{Z}^n \to \mathbb{G}$; Inputs: $\mathbf{x} \in \mathbb{Z}^n$, $w \in \mathbb{G}$; Claim: $\phi(\mathbf{x}) = w$

1. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
2. For $i = 1, \ldots, n$: Prover finds integers q_i and $r_i \in [\ell]$ s.t. $x_i = q_i\ell + r_i$.
   Let $\mathbf{q} \leftarrow (q_1, ..., q_n) \in \mathbb{Z}^n$ and $\mathbf{r} \leftarrow (r_1, ..., r_n) \in [\ell]^n$.
   Prover sends $Q \leftarrow \phi(\mathbf{q}) \in \mathbb{G}$ to Verifier.
3. Verifier computes $r_i = (x_i \bmod \ell) \in [\ell]$ for all $i = 1, \ldots, n$, sets $\mathbf{r} = (r_1, \ldots, r_n)$, and accepts if $Q^\ell \phi(\mathbf{r}) = w$ holds in $\mathbb{G}$.

Theorem 6 (Soundness PoHP)**.** *Protocol PoHP is an argument system for Relation $\mathcal{R}_{\phi,\mathsf{PoHP}}$ with negligible soundness error, assuming the adaptive root assumption holds for GGen.*

Proof. Suppose that $\phi(\mathbf{x}) \neq w$, but the adversary succeeds in making the verifier accept with non-negligible probability. Let $\mathbf{q}$ and $\mathbf{r}$ be as defined in step (2) of the protocol and let Q be the prover's message to the verifier. Then $[Q/\phi(\mathbf{q})]^\ell = [w/\phi(\mathbf{r})]/[\phi(\mathbf{x})/\phi(\mathbf{r})] = w/\phi(\mathbf{x}) \neq 1$. We thus obtain an algorithm to break the adaptive root assumption for the instance $\hat{w} := w/\phi(\mathbf{x})$ by interacting with the adversary, giving it the adaptive root challenge ℓ, and outputting $\hat{u} := Q/\phi(\mathbf{q}) \in \mathbb{G}$, where Q is the value output by the adversary. □

A.2 A succinct proof of knowledge of a homomorphism preimage

The PoKE argument of knowledge can be extended to an argument of knowledge for the pre-image of a homomorphism $\phi : \mathbb{Z}^n \to \mathbb{G}$.

$$\mathcal{R}_\phi = \big\{\big(w \in \mathbb{G};\ \mathbf{x} \in \mathbb{Z}^n\big)\ :\ w = \phi(\mathbf{x}) \in \mathbb{G}\big\}.$$

For a general homomorphism ϕ we run into the same extraction challenge that we encountered in extending Protocol PoKE* to work for general bases. The solution for Protocol PoKE was to additionally send g^x where g is either a base in the CRS or chosen randomly by the verifier and execute a parallel PoKE for $g \mapsto g^x$. We can apply exactly the same technique here on each component x_i of the witness, i.e. send g^{x_i} to the verifier and execute a parallel PoKE that $g \mapsto g^{x_i}$. This allows the extractor to obtain the witness x, and the soundness of the protocol then follows from the soundness of Protocol PoHP. However, as an optimization to reduce the communication we can instead use the group representation homomorphism $Rep : \mathbb{Z}^n \to \mathbb{G}$ defined as

$$Rep(\mathbf{x}) = \prod_{i=1}^{n} g_i^{x_i}$$

for base elements g_i defined in the CRS. The prover sends $Rep(\mathbf{x})$ in its first message, which is a single group element independent of n.

Protocol PoKHP (Proof of knowledge of homomorphism preimage)

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$, $(g_1, ..., g_n) \in \mathbb{G}^n$, $\phi : \mathbb{Z}^n \to \mathbb{G}$; Inputs: $w \in \mathbb{G}$; Witness: $\mathbf{x} \in \mathbb{Z}$; Claim: $\phi(\mathbf{x}) = w$

1. Prover sends $z = Rep(\mathbf{x}) = \prod_i g_i^{x_i} \in \mathbb{G}$ to the verifier.
2. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
3. For each x_i, Prover computes q_i, r_i s.t. $x_i = q_i\ell + r_i$, sets $\boldsymbol{q} \leftarrow (q_1, ..., q_n) \in \mathbb{Z}^n$ and $\boldsymbol{r} \leftarrow (r_1, ..., r_n) \in [\ell]^n$. Prover sends $Q_\phi \leftarrow \phi(\boldsymbol{q}) \in \mathbb{G}$, $Q_{Rep} \leftarrow Rep(\boldsymbol{q}) \in \mathbb{G}$, and $\boldsymbol{r}$ to Verifier.
4. Verifier accepts if $\boldsymbol{r} \in [\ell]^n$, $Q_\phi^\ell \phi(\mathbf{r}) = w$, and $Q_{Rep}^\ell Rep(\mathbf{r}) = z$.

In order to analyze the security of this protocol, it is helpful to first consider a special case of Protocol PoKHP protocol for the homomorphism $Rep : \mathbb{Z}^n \to \mathbb{G}$, which is a generalization of Protocol PoKE*. In this case the prover of course does not need to separately send $Rep(\mathbf{x})$ in the first message. The protocol is as follows:

Protocol PoKRep (Proof of knowledge of representation)

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda), (g_1, ..., g_n) \in \mathbb{G}^n$; Inputs: $w \in \mathbb{G}$; Witness: $\boldsymbol{x} \in \mathbb{Z}$; Claim: $Rep(\boldsymbol{x}) = \prod_{i=1}^n g_i^{x_i} = w$

1. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
2. For each x_i, Prover finds q_i, r_i s.t. $x_i = q_i\ell + r_i$, sets $\boldsymbol{q} \leftarrow (q_1, ..., q_n) \in \mathbb{Z}^n$ and $\boldsymbol{r} \leftarrow (r_1, ..., r_n) \in [\ell]^n$. Prover sends $Q \leftarrow Rep(\boldsymbol{q}) = \prod_i g_i^{q_i} \in \mathbb{G}$ and $\boldsymbol{r}$ to Verifier.
3. Verifier accepts if $\boldsymbol{r} \in [\ell]^n$, $Q^\ell Rep(\boldsymbol{r}) = w$.

The following theorems prove security of the two protocols above.

Theorem 7 (PoKRep Argument of Knowledge)**.** Protocol PoKRep *is an argument of knowledge for relation $\mathcal{R}_{\mathsf{Rep}}$ in the generic group model.*

Proof. See Appendix C. □

Theorem 8 (PoKHP Argument of Knowledge)**.** *Protocol PoKHP is an argument of knowledge for the relation $\mathcal{R}_\phi$ in the generic group model.*

Proof. See Appendix C. □

A.3 A succinct proof of integer exponent mod n

There are several applications of accumulators that require proving complex statements about integer values committed in an accumulator (e.g. [BSCG+14, MGGR13b]). Practical succinct argument systems (SNARGs/SNARKs/STARKs) operate on statements defined as an arithmetic circuit, and the prover efficiency scales with the multiplication complexity of the statement. Since RSA accumulators are an algebraic accumulator, in constrast to Merkle trees, one would hope that the arithmetic complexity of statements involving RSA accumulator elements would be much lower than those involving Merkle tree elements. Unfortunately, this is not the case because RSA accumulator operations are in $\mathbb{Z}_N$ for composite N with unknown factorization, whereas arithmetic circuits for SNARGs are always defined over finite fields $\mathbb{Z}_p$. Finding ways to combine RSA accumulators with SNARKs more efficiently is an interesting research direction.

We present a variant of PoKE* for a group of unknown order $\mathbb{G}$, which is an argument of knowledge that the integer discrete log x of an element $y \in \mathbb{G}$ is equivalent to $\hat{x}$ modulo a public odd prime integer n. Concretely, the new protocol PoKEMon is for the following relation:

$$\mathcal{R}_{\mathsf{PoKEMon}} = \big\{\big(w \in \mathbb{G}, \hat{x} \in [n]; x \in \mathbb{Z}\big) \ : \ w = g^x \in \mathbb{G}, x \bmod n = \hat{x}\big\}.$$

As with PoKE*, the base element g and the unknown order group $\mathbb{G}$ are fixed public parameters. PoKEMon modifies PoKE* by setting the challenge to be $\ell \cdot n$ where $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.

Protocol PoKEMon (Proof of equality mod n) for Relation $\mathcal{R}_{\mathsf{PoKEMon}}$

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$, $g \in \mathbb{G}$; Inputs: Odd prime $n, w \in \mathbb{G}, \hat{x} \in [n]$; Witness: $x \in \mathbb{Z}$; Claim: $g^x = w$ and $x \bmod n = \hat{x}$

1. Verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
2. Prover computes the quotient $q \in \mathbb{Z}$ and residue $r \in [\ell \cdot n]$ such that $x = q(\ell \cdot n) + r$. Prover sends the pair $(Q \leftarrow g^q,\ r)$ to the Verifier.
3. Verifier accepts if $r \in [\ell \cdot n]$ and $Q^{\ell \cdot n} g^r = w$ holds in $\mathbb{G}$ and $r \bmod n = \hat{x}$.

The same technique can be applied to PoKE, where the base element can be freely chosen by the prover.

We can prove security by directly reducing it to the security of the PoKE* protocol and additionally the strong RSA assumption, which assumes it is hard to compute an ℓth root of a random group element g for odd prime ℓ.

Theorem 9 (PoKEMon Argument of Knowledge)**.** *Protocol PoKEMon is an argument of knowledge for the relation $\mathcal{R}_{\mathsf{PoKEMon}}$ if Protocol PoKE* is an argument of knowledge for the relation $\mathcal{R}_{\mathsf{PoKE}^*}$ and the strong RSA assumption holds for GGen.*

Proof. We use the extractor Ext* of the PoKE* protocol to build an extractor Ext for PoKEMon, which succeeds with overwhelming probability in extracting x such that $g^x = w$ and $x = \hat{x} \bmod n$ from any PoKEMon adversary that has a non-negligible success rate.

Ext runs a copy of Ext* and simulates both the PoKE* challenges and a PoKE* adversary's response. When Ext receives the challenge ℓ and the PoKEMon adversary's response (Q, r), it computes $q' = \lceil r/\ell \rceil$ and $r' = r \bmod \ell$ so that $r = q'\ell + r'$ and sets $Q' \leftarrow Q^n g^{q'}$. It forwards (ℓ, Q', r') to Ext*. If the PoKEMon adversary's response is valid then $Q^{\ell n} g^r = w$, implying that $Q'^{\ell} g^{r'} = w$. Thus, Ext simulates for Ext* a transcript of the PoKE* protocol for a PoKE* adversary that succeeds with the same rate as the PoKEMon adversary. By hypothesis, Ext* succeeds with overwhelming probability to output x such that $g^x = w$.

Consider any iteration in which Ext had received an accepting PoKEMon transcript (ℓ, Q, r). We claim that $x = r \bmod \ell \cdot n$ with overwhelming probability, by the strong RSA assumption.

Suppose that $x - r \neq 0 \bmod \ell \cdot n$ then given that ℓ and n are prime we have that either $\gcd(\ell, x - r) = 1$ or $\gcd(n, x - r) = 1$. Without loss of generality we assume the latter. Let $Q' = Q^\ell$ then $Q'^n = g^{x-r}$ and $\gcd(n, x - r) = 1$. Now using Shamir's trick we can compute an nth root of g. Let $a, b = \mathbf{Bezout}(n, x - r)$ such that $an + b(x - r) = 1$. Then $u = Q'^b g^a$ is an nth root of g as $u^n = Q'^{bn} g^{an} = g^{b \cdot (x-r)} g^{an} = g$. This shows that (u, n) breaks the strong RSA assumption for $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$.

This contradicts the hypothesis and we, therefore, have that $r = x \bmod \ell \cdot n$ which implies that $r = x \bmod n$ for an overwhelming number of accepting transcripts. And since in any accepting transcript $\hat{x} = r \bmod n$ we have that $x = \hat{x} \bmod n$ with overwhelming probability.

□

A.4 A succinct zero-knowledge proof of discrete-log

The PoKE protocol for succinctly proving knowledge of an exponent can further be made zero-knowledge using a method similar to the classic Schnorr Σ-protocol for hidden order groups. The Schnorr protocol for hidden order groups has the same structure as the standard Schnorr protocol for proving knowledge of a discrete logarithm x such that $u^x = w$ in a group of known order. Here, the prover first samples a blinding factor $k \in [-B, B]$ and sends $A = u^k$, obtains a challenge c, and returns $s = k + cx$. The verifier checks that $u^z = aw^c$. In hidden order groups, k must be sampled from a range of integers $[-B, B]$ such that $|\mathbb{G}|/B$ is negligible.

The classical Schnorr protocol for hidden order groups is an honest verifier statistical zero-knowledge (HVSZK) protocol and has soundness error of only $1/2$ against a classical adversary [BCK10]. Only for a small subclass of homomorphisms better

soundness can be proven [BCM05]. Unfortunately, [BCK10] proved that the soundness limitation is fundamental and cannot be improved against a classical adversary, and therefore requires many rounds of repetition. However, we are able to show that we can prove much tighter soundness if the adversary is restricted to operating in a generic group.

Definition 10 (Zero Knowledge)**.** *We say an argument system* $(\mathsf{Pgen}, \mathsf{P}, \mathsf{V})$ *for* $\mathcal{R}$ *has* ***statistical zero-knowledge*** *if there exists a poly-time simulator* Sim *such that for* $(x, w) \in \mathcal{R}$ *the following two distribution are statistically indistinguishable:*

$$D_1 = \Big\{ \langle \mathsf{P}(pp, x, w), \mathsf{V}(pp, x) \rangle,\ pp \xleftarrow{\$} \mathsf{Pgen}(\lambda) \Big\}$$

$$D_2 = \Big\{ \mathsf{Sim}(pp, x, \mathsf{V}(pp, x)),\ pp \xleftarrow{\$} \mathsf{Pgen}(\lambda) \Big\}$$

The protocol. Our ZK protocol applies Protocol PoKE to the last step of the Schnorr protocol, which greatly improves the communication efficiency of the classical protocol when the witness is large. In fact, we can interleave the first step of Protocol PoKE where the verifier sends a random prime ℓ with the second step of the Schnorr protocol where the verifier sends a challenge c. This works for the case when u is a base specified in the CRS, i.e. it is the output of a query to the generic group oracle $\mathcal{O}_1$, however a subtlety arises when u is selected by the prover. In fact, we cannot even prove that the Schnorr protocol itself is secure (with negligible soundness error) when u is selected by the prover. The method we used for PoKE on general bases involved sending g^x for g specified in the CRS. This would immediately break ZK since the simulator cannot simulate g^x without knowing the witness x. Instead, in the first step the prover will send a Pedersen commitment $g^x h^\rho$ where ρ is sampled randomly in some interval and h is another base specified in the CRS.

We will first present a ZK proof of knowledge of a representation in terms of bases specified in the CRS and show that there is an extractor that can extract the witness. We then use this as a building block for constructing a ZK protocol for the relation $\mathcal{R}_{\mathsf{PoKE}}$.

Protocol ZKPoKRep for Relation $\mathcal{R}_\phi$ where $\phi := Rep$

Params: $(g_1, \dots g_n) \in \mathbb{G}$, $\mathbb{G} \xleftarrow{\$} GGen(\lambda), B > 2^{2\lambda}|\mathbb{G}|$; Inputs: $w \in \mathbb{G}$;
Witness: $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{Z}^n$; Claim: $Rep(\boldsymbol{x}) = \prod_{i=1}^n g_i^{x_i} = w$

1. Prover chooses random $k_1, \dots, k_n \xleftarrow{\$} [-B, B]$, sends $A = \prod_{i=1}^n g_i^{k_i}$ to Verifier.
2. Verifier sends $c \xleftarrow{\$} [0, 2^\lambda], \ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
3. Prover computes $s_i = k_i + c \cdot x_i \forall i \in [1, n]$ and then derives quotients $\boldsymbol{q} \in \mathbb{Z}^n$ and residues $\boldsymbol{r} \in [\ell]^n$ such that $q_i \cdot \ell + r_i = s_i$ for all $1 \le i \le n$. Prover sends $Q = \prod_{i=1}^n g_i^{q_i}$ and $\boldsymbol{r}$ to the Verifier.
4. Verifier accepts if $r_i \in [\ell]$ for all $1 \le i \le n$ and that $Q^\ell \prod_{i=1}^n g_i^{r_i} = Aw^c$.

Theorem 10 (Protocol ZKPoKRep)**.** *Protocol ZKPoKRep is an honest-verifier statistically zero-knowledge argument of knowledge for relation* $\mathcal{R}_{\mathsf{Rep}}$ *in the generic group model.*

Proof. See Appendix C. □

Finally, we use the protocol above to obtain a ZK protocol for the relation $\mathcal{R}_{\mathsf{PoKE}}$. The protocol applies (in parallel) the Σ-protocol for PoKRep to a Pedersen commitment $g^x h^\rho$ for g and h specified in the CRS. In order to achieve statistical zero-knowledge we require that g and h generate the same subgroup of $\mathbb{G}$. This requirement can be lifted when computation zero-knowledge suffices. The extractor for this protocol will invoke the PoKRep extractor to open the commitment. The protocol works as follows:

Protocol ZKPoKE for $\mathcal{R}_{\mathsf{PoKE}}$

Params: $(g, h) \in \mathbb{G}$ s.t. $\langle g \rangle = \langle h \rangle$, $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$; Inputs: $u, w \in \mathbb{G}$, $B > 2^{2\lambda}|\mathbb{G}|$;
Witness: $x \in \mathbb{Z}$; Claim: $u^x = w$
Let $Com(x; r) := g^x h^r$.

1. Prover chooses random $k, \rho_x, \rho_k \xleftarrow{\$} [-B, B]$ and sends (z, A_g, A_u) to the verifier where $z = Com(x; \rho_x)$, $A_g = Com(k; \rho_k)$, $A_u = u^k$.
2. Verifier sends $c \xleftarrow{\$} [0, 2^\lambda], \ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
3. Prover computes $s_x = k + c \cdot x$ and $s_\rho = \rho_k + c \cdot \rho_x$ and then derives quotients $q_1, q_2 \in \mathbb{Z}$ and residues $r_x, r_\rho \in [\ell]$ such that $q_x \cdot \ell + r_x = s_x$ and $q_\rho \cdot \ell + r_\rho = s_\rho$.
   Prover sends $Q_g = Com(q_x; q_\rho)$, $Q_u = u^{q_x}$ and r_x, r_ρ to the Verifier.
4. Verifier accepts if $r_x, r_\rho \in [\ell]$ and

$$Q_g^\ell \cdot Com(r_x; r_\rho) = A_g z^c \qquad \text{and} \qquad Q_u^\ell \cdot u^{r_x} = A_u w^c.$$

Theorem 11 (Protocol ZKPoKE)**.** *Protocol ZKPoKE is an honest verifier statistically zero-knowledge argument of knowledge for relation $\mathcal{R}_{\mathsf{PoKE}}$ in the generic group model.*

Proof. See Appendix C. □

B More Accumulator techniques

B.1 Accumulator unions

Yet another application of our succinct proofs to accumulators is the ability to prove that an accumulator is the union of two other accumulators. Given three accumulators $A_1 = g_1^{\prod_{s \in S_1} s}$, $A_2 = g_2^{\prod_{s \in S_2} s}$ and $A_3 = A_2^{\prod_{s \in S_1} s}$ a prover can use the NI-PoDDH protocol to convince a verifier that (A_1, A_2, A_3) forms a valid DDH tuple. If S_1 and S_2 are guaranteed to be disjoint, then A_3 will be an accumulator of $S_1 \cup S_2$. If they are not disjoint, then resulting accumulator will be an accumulator for a multi-set as described in the next paragraph. The NI-PoDDH is independent of the size of S_1 and S_2 in both the proof size and the verification time. This union proof can be used to batch exclusion proofs over multiple accumulators. The prover verifiably combines the accumulators and then creates a single aggregate non-membership proof in the union of the accumulators. This is sound but only works if the domains of the accumulators are separate.

B.2 Multiset accumulator

A dynamic multiset accumulator is an accumulator where items can be added and deleted more than once, and every element has a count. In other words, it is a commitment to a mapping from items to non-negative integer counters. It has the following properties:

- Each element in the domain is implicitly in the mapping with a counter of 0.
- **Add** increments the counter of the added element by 1
- **Del** decrements the counter of the added element by 1
- A membership witness for an element x and a counter k proves that the counter of x is at least k
- A membership witness for x^k and a non-membership witness for $A^{x^{-k}}$ proves that the counter for x is exactly k. Note that $A^{x^{-k}}$ is exactly the membership witness for x^k.

To build the multi-set accumulator we again employ a hash function mapping an arbitrary domain to an exponentially large set of primes. The **Add** and **Del** algorithms are as described in Section 4.2. The membership witness change in that they now also contain a counter of how many times a certain element has been added. That is if an element x is k times in the accumulator the membership witness is the x^kth root of the accumulator as well as k. **VerMem**, **MemWitCreate**, **MemWitUpAdd**, **MemWitUpDel** are changed accordingly. The completeness definition also needs to be updated to reflect the new multi-set functionalities.

C Security Proofs

C.1 Preliminary lemmas

In the following lemmas, which all concern the generic group model, we restrict ourselves to adversaries that do not receive any group elements as input. This is sufficient to prove our theorems. For our proof protocols we require that the adversary itself outputs the instance after receiving a description of the group. We require this in order to prevent that the instance itself encodes a trapdoor, such as the order of the group.

Lemma 2 (Element representation [Sho97])**.** *Using the notation of Section 2.2, let $\mathbb{G}$ be a generic group and $\mathcal{A}$ a generic algorithm making q_1 queries to $\mathcal{O}_1$ and q_2 queries to $\mathcal{O}_2$. Let $\{g_1, \ldots, g_m\}$ be the outputs of $\mathcal{O}_1$. There is an efficient algorithm Ext that given as input the transcript of $\mathcal{A}$'s interaction with the generic group oracles, produces for every element $u \in \mathbb{G}$ that $\mathcal{A}$ outputs, a tuple $(\alpha_1, \ldots, \alpha_m) \in \mathbb{Z}^m$ such that $u = \prod_{i=1}^m g_i^{\alpha_i}$ and $\alpha_i \leq 2^{q+2}$.*

Lemma 3 (Computing multiple of orders of random elements)**.** *Let $\mathbb{G}$ be a generic group where $|\mathbb{G}|$ is a uniformly chosen integer in $[A, B]$. Let $\mathcal{A}$ be a generic algorithm making q_1 queries to $\mathcal{O}_1$ and q_2 queries to $\mathcal{O}_2$. The probability that $\mathcal{A}$ succeeds in computing $0 \neq k \in \mathbb{N}$ such that for a g which is a response to an $\mathcal{O}_1$ query $g^k = 1$ is at most $\frac{(q_1+q_2)^3}{M}$, where $1/M$ is negligible whenever $|B - A| = \exp(\lambda)$. When $\mathcal{A}$ succeeds we say that event* Root *happened.*

We denote $\mathrm{ord}_{\mathbb{G}}(g)$ as the order of $g \in \mathbb{G}$. By definition $g^k = 1 \wedge 0 \neq k \in \mathbb{Z} \leftrightarrow k \bmod \mathrm{ord}_{\mathbb{G}}(g) = 0$.

Proof. This lemma is a direct corollary of Theorem 1 from [DK02]. That theorem shows that an adversary that interacts with the two generic group oracles cannot solve the strong RSA problem with probability greater than $(q_1+q_2)^3/M$, where M is as in the statement of the lemma. Recall that a strong RSA adversary takes as input a random $g \in \mathbb{G}$ and outputs (u, x) where $u^x = g$ and x is an odd prime. Let $\mathcal{A}$ be an adversary from the statement of the lemma, that is, $\mathcal{A}$ outputs $0 < k \in \mathbb{Z}$ where $k \equiv 0 \bmod |\mathbb{G}|$ with some probability ϵ. This $\mathcal{A}$ immediately gives a strong RSA adversary that also succeeds with probability ϵ: run $\mathcal{A}$ to get k and g such that $g^k = 1 \in \mathbb{G}$. Then find an odd prime x that does not divide k, and output (u, x) where $u = g^{(x^{-1} \bmod k)}$. Clearly $u^x = g$ which is a solution to the given strong RSA challenge. It follows by Theorem 1 from [DK02] that $\epsilon \leq (q_1 + q_2)^3/M$, as required. □

Lemma 4 (Discrete Logarithm)**.** *Let $\mathbb{G}$ be a generic group where $|\mathbb{G}|$ is a uniformly chosen integer in $[A, B]$, where $1/A$ and $1/|B - A|$ are negligible in λ. Let $\mathcal{A}$ be a generic algorithm and let $\{g_1, \ldots, g_m\}$ be the outputs of $\mathcal{O}_1$. Then if $\mathcal{A}$ runs in polynomial time, it succeeds with at most negligible probability in outputting $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m \in \mathbb{Z}$ such that $\prod_{i=1}^m g_i^{\alpha_i} = \prod_{i=1}^m g_i^{\beta_i}$ and $\alpha_i \neq \beta_i$ for some i. We call this event* DLOG.

Proof sketch. We follow the structure of Shoup's argument [Sho97]. By Lemma 2 every group element $u \in \mathbb{G}$ that the adversary obtains in response to an $\mathcal{O}_2$ query can be written as $u = \prod_{i=1}^m g_i^{\alpha_i}$ for some known $\alpha_i \in \mathbb{Z}$. Let $g = \prod_{i=1}^m g_i^{\alpha_i}$ and

$h = \prod_{i=1}^m g_i^{\beta_i}$ be two such group elements. If there is some i for which $\alpha_i \not\equiv \beta_i \pmod{\mathrm{ord}_{\mathbb{G}}(g_i)}$ then the probability that $g = h$ is at most negligible, as shown in [DK02]. Hence, if $g = h$ then with overwhelming probability we have that $\alpha_i \equiv \beta_i \pmod{\mathrm{ord}_{\mathbb{G}}(g_i)}$ for all i. From this it follows by Lemma 3 that $\alpha_i = \beta_i \in \mathbb{Z}$ with overwhelming probability, since otherwise one obtains a multiple of $|\mathbb{G}|$. Since $\mathcal{A}$ constructs at most polynomially many group elements, there are at most polynomially many pairs of such elements. Therefore, a union bound over all pairs shows that the probability that event DLOG happens is at most negligible, as required. □

Lemma 5 (Dlog extraction)**.** *Let $\mathbb{G}$ be a generic group where $|\mathbb{G}|$ is a uniformly chosen integer in $[A, B]$ and g an output of a query to $\mathcal{O}_1$. Let $\mathcal{A}$ be a generic algorithm that outputs $w \in \mathbb{G}$ and then runs the interactive protocol* Protocol PoKE* *with g in the CRS. Let (ℓ_1, Q_1, r_1) and (ℓ_2, Q_2, r_2) two accepting transcripts for* Protocol PoKE* *generated one after the other. If $1/A$ and $1/|B - A|$ are negligible in λ, then with overwhelming probability there exist integers α and β such that $\alpha \cdot l_1 + r_1 = \beta \cdot l_2 + r_2$ and $g^{\alpha \cdot l_1 + r_1} = w$. Further if $\mathcal{A}$ makes q queries to $\mathcal{O}_2$ then $|\alpha|, |\beta|$ are bounded by 2^q.*

Proof. W.l.o.g. let $g_1 = g$ be encoded in the PoKE* CRS. The PoKE* verification equations give us $w = Q_1^{\ell_1} g^{r_1} = Q_2^{\ell_2} g^{r_2}$. We can write $Q_1 = \prod_{i=1}^m g_i^{\alpha_i}$ and $Q_2 = \prod_{i=1}^m g_i^{\beta_i}$. This implies that $Q_1^{\ell_1} g^{r_1} = g^{\alpha_1 \cdot \ell_1 + r_1} \prod_{i=2}^m g_i^{\alpha_i \cdot \ell_1} = g^{\beta_1 \cdot \ell_2 + r_2} \prod_{i=2}^m g_i^{\beta_i \cdot \ell_2}$. By Lemma 4, $\alpha_i \ell_1 = \beta_i \ell_2 \in \mathbb{Z}$ for all $i \neq 1$ with overwhelming probability (i.e. unless event DLOG occurs), and therefore $\ell_2 | \alpha_i \ell_1$. The primes ℓ_1 and ℓ_2 are co-prime unless $\ell_1 = \ell_2$, which happens with probability $\frac{\ln(2)\lambda}{2^\lambda}$. Thus, with overwhelming probability $\ell_2 | \alpha_i$. However, $\alpha_i \leq 2^{q_2}$ and α_i is chosen before ℓ_2 is sampled, hence the probability that $\ell_2 | \alpha_i$ for $\alpha_i \neq 0$ is at most $\frac{q_2 \lambda \ln(2)}{2^\lambda}$. We conclude that with overwhelming probability $\alpha_i = \beta_i = 0$ for all $i \neq 1$. It follows that except with probability $\Pr[\mathsf{DLOG}] + \frac{2 q_2 \lambda \ln(2)}{2^\lambda}$, we can express $w = g^{\alpha_1 \ell_1 + r_1} = g^{\beta_1 \ell_2 + r_2}$ for integers $\alpha_1, r_1, \beta_1, r_2$ such that $\alpha_1 \ell_1 + r_1 = \beta_1 \ell_2 + r_2$. □

In what follows we will use the following notation already introduced in Section 3: for generators $g_1, \ldots, g_n \in \mathbb{G}$ we let $Rep : \mathbb{Z}^n \to \mathbb{G}$ be the homomorphism

$$Rep(\boldsymbol{x}) = \prod_{i=1}^n g_i^{x_i}.$$

Lemma 6 (Representation extraction)**.** *Let $\mathbb{G}$ be a generic group where $|\mathbb{G}|$ is a uniformly chosen integer in $[A, B]$ and let $g_1, \ldots, g_n \in \mathbb{G}$ be responses to queries to oracle $\mathcal{O}_1$. Let $\mathcal{A}$ be a generic algorithm that outputs $w \in \mathbb{G}$ and then runs the interactive protocol* Protocol PoKRep *on input w with $g_1, ..., g_n$ in the CRS. Let $(\ell_1, Q_1, \mathbf{r}_1)$ and $(\ell_2, Q_2, \mathbf{r}_2)$ be two accepting transcripts for* Protocol PoKRep*. If $1/A$ and $1/|B - A|$ are negligible in λ, then with overwhelming probability there exist integer vectors $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^n$ such that $\boldsymbol{\alpha} l_1 + \boldsymbol{r}_1 = \boldsymbol{\beta} l_2 + \boldsymbol{r}_2$ and $Rep(\boldsymbol{\alpha} l_1 + \boldsymbol{r}_1) = w$. Further if $\mathcal{A}$ makes q queries to $\mathcal{O}_2$ then each component α_j and β_j of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are bounded by 2^q.*

Proof. The proof is a direct generalization of the argument in Lemma 5 above. From the verification equations of the protocol we have $Q_1^{\ell_1} Rep(\boldsymbol{r}_1) = Q_2^{\ell_2} Rep(\boldsymbol{r}_2) = w$. With overwhelming probability, the generic group adversary knows $\alpha_1, ..., \alpha_m$ and $\beta_1, .., \beta_m$ for $m > n$ such that it can write $Q_1 = \prod_{i=1}^m g_i^{\alpha_i}$ and $Q_2 = \prod_{i=1}^m g_i^{\beta_i}$. From the verification equation and Lemma 4, with overwhelming probability $\alpha_i \ell_1 + \boldsymbol{r}_1[i] = \beta_i \ell_2 + \boldsymbol{r}_2[i]$ for each $i \leq n$ and $\alpha_i \ell_1 = \beta_i \ell_2$ for each $i > n$. As explained in the proof of Lemma 5, this implies that with overwhelming probability $\alpha_i = \beta_i = 0$ for each $i > n$, in which case $w = \prod_{i=1}^n g_i^{\alpha_i \ell_1 + r_1[i]} = \prod_{i=1}^n g_i^{\beta_i \ell_2 + r_2[i]}$. Setting $\boldsymbol{\alpha} := (\alpha_1, ..., \alpha_n)$ and $\boldsymbol{\beta} := (\beta_1, ..., \beta_n)$, we conclude that with overwhelming probability $w = Rep(\boldsymbol{\alpha} \ell_1 + \boldsymbol{r}_1) = Rep(\boldsymbol{\beta} \ell_2 + \boldsymbol{r}_2)$ and $\boldsymbol{\alpha} \ell_1 + \boldsymbol{r}_1 = \boldsymbol{\alpha} \ell_2 + \boldsymbol{r}_2$. Finally, if $\mathcal{A}$ has made at most q queries to $\mathcal{O}_2$ then $\alpha_i < 2^q$ and $\beta_i < 2^q$ for each i.

□

The next two corollaries show that the adaptive root problem and the known order element problem are intractable in a generic group.

Corollary 1 (Adaptive root hardness)**.** *Let $\mathbb{G}$ be a generic group where $|\mathbb{G}|$ is a uniformly chosen integer in $[A, B]$ such that $1/|A|$ and $1/|B - A|$ are negligible in λ. Any generic adversary $\mathcal{A}$ that performs a polynomial number of queries to oracle $\mathcal{O}_2$ succeeds in breaking the adaptive root assumption on $\mathbb{G}$ with at most negligible probability in λ.*

Proof. Recall that in the adaptive root game the adversary outputs $w \in \mathbb{G}$, the challenger then responds with a prime $\ell \in [2, 2^\lambda]$, and the adversary succeeds if it outputs u such that $u^\ell = w$. According to Lemma 2 we can write $u = \prod_{i=1}^m g_i^{\alpha_i}$ and $w = \prod_{i=1}^m g_i^{\beta_i}$, where $g_1, \ldots, g_m$ are the responses to oracle $\mathcal{O}_1$ queries. By Lemma 4 we know that $\alpha_i \ell = \beta_i \bmod |\mathbb{G}|$ for all $i = 1, \ldots, m$ with overwhelming probability, namely $1 - \Pr[\mathsf{DLOG}]$. Therefore, $\alpha_i \ell = \beta_i + k \cdot |\mathbb{G}|$ for some $k \in \mathbb{Z}$. By Lemma 3, an efficient adversary can compute a multiple of the order of the group with at most negligible probability $\Pr[\mathsf{Root}]$. It follows that $k = 0$ and $\alpha_i \ell = \beta_i \in \mathbb{Z}$ with probability greater than $1 - \Pr[\mathsf{DLOG}] - \Pr[\mathsf{Root}]$, since otherwise $\alpha_i \ell - \beta_i$ is a multiple of $\mathbb{G}$. Now, because $\alpha_i \ell = \beta_i$ we know that ℓ must divide β_i. However, β_i is chosen before ℓ and if $\mathcal{A}$ makes q_2 generic group queries then $\beta_i \leq 2^{q_2}$. The probability that ℓ divides β_i, for $\beta_i \neq 0$, is bounded by the probability that a random prime in $\mathsf{Primes}(\lambda)$ divides a number less than 2^{q_2}. Any such number has less than q_2 distinct prime factors and there are more than $2^\lambda/\lambda$ primes in $\mathsf{Primes}(\lambda)$. Therefore, the probability that ℓ divides $\beta_i \neq 0$ is at most $\frac{q_2 \cdot \lambda}{2^\lambda}$. Overall, we obtain that a generic adversary can break the adaptive root assumption with probability at most $\frac{(q_1+q_2)^2}{A} + 2 \cdot \frac{(q_1+q_2)^3}{M} + \frac{q_2 \cdot \lambda}{2^\lambda}$, which is negligible if A and $B - A$ are exponential in λ and q_1, q_2 are bounded by some polynomial in λ. □

Corollary 2 (Non-trivial order hardness)**.** *Let $\mathbb{G}$ be a generic group where $|\mathbb{G}|$ is a uniformly chosen integer in $[A, B]$ such that $1/|A|$ and $1/|B - A|$ are negligible in λ. Any generic adversary $\mathcal{A}$ that performs a polynomial number of queries to oracle $\mathcal{O}_2$ succeeds in finding an element $h \neq 1 \in \mathbb{G}$ and an integer d such that $h^d = 1$ with at most negligible probability in λ.*

Proof. We can construct an adaptive root adversary that first uses $\mathcal{A}$ to obtain h and d, and then computes the ℓth root of h by computing $c = \ell^{-1} \bmod d$ and $h^c = h^{1/\ell}$. Since the adaptive root assumption holds true in the generic group model (Corollary 1), we can conclude that $\mathcal{A}$ succeeds with negligible probability. □

Fact 1 (Chinese Remainder Theorem (CRT))**.** *Let $\ell_1, \ldots, \ell_n$ be coprime integers and let $r_1, \ldots, r_n \in \mathbb{Z}$, then there exists a unique $0 \leq x < \prod_{i=1}^n \ell_i$ such that $x = r_i \bmod \ell_i$ and there is an efficient algorithm for computing x.*

C.2 Proofs of the main theorems

Proof of Theorem 7.

> Protocol PoKRep is an argument of knowledge for the relation $\mathcal{R}_\phi$ where $\phi := Rep$, in the generic group model.

Fix $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$ and $\mathbf{g} = (g_1, ..., g_n) \in \mathbb{G}$. Let $\mathcal{A}_0, \mathcal{A}_1$ be poly-time generic adversaries where $(w, \mathsf{state}) \xleftarrow{\$} A_0(\mathbf{g})$ and $\mathcal{A}_1(\mathsf{state})$ runs Protocol PoKRep with a verifier $V(\mathbf{g}, w)$. We need to show that for all $\mathcal{A}_1$ there exists a poly-time Ext such that for all $\mathcal{A}_0$ the following holds: if $\mathcal{A}_1$ convinces $V(\mathbf{g}, w)$ to accept with probability $\epsilon \geq 1/\mathrm{poly}(\lambda)$, then Ext outputs a vector $\boldsymbol{x} \in \mathbb{Z}^n$ such that $Rep(\boldsymbol{x}) = w$ with overwhelming probability.

> **Subclaim** In Protocol PoKRep, for any polynomial number of accepting transcripts $\{(\ell_i, Q_i, \boldsymbol{r}_i)\}_{i=1}^{\mathsf{poly}(\lambda)}$ obtained by rewinding $\mathcal{A}_1$ on the same input (w, state), with overwhelming probability there exists $\boldsymbol{x} \in \mathbb{Z}^n$ such that $\boldsymbol{x} = \boldsymbol{r}_i \bmod \ell_i$ for each i and $Rep(\boldsymbol{x}) = w$. Furthermore, $x_j \leq 2^q$ for each jth component x_j of $\boldsymbol{x}$, where q is the total number of queries that $\mathcal{A}$ makes to the group oracle.

The subclaim follows from Lemma 6. With overwhelming probability there exists $\boldsymbol{\alpha}, \boldsymbol{\beta}$, and $\boldsymbol{x}$ in $\mathbb{Z}^n$ such that $\boldsymbol{x} = \boldsymbol{\alpha}\ell_1 + \boldsymbol{r}_1 = \boldsymbol{\beta}\ell_2 + \boldsymbol{r}_2$ and $Rep(\boldsymbol{x}) = w$, and each component of $\boldsymbol{x}$ is bounded by 2^q. Consider any third transcript, w.l.o.g. $(\ell_3, Q_3, \boldsymbol{r}_3)$. Invoking the lemma again, there exists $\boldsymbol{\alpha}'$, $\boldsymbol{\beta}'$, and $\boldsymbol{x}'$ such that $\boldsymbol{x}' = \boldsymbol{\alpha}'\ell_2 + \boldsymbol{r}_2 = \boldsymbol{\beta}'\ell_3 + \boldsymbol{r}_3$. Thus, with overwhelming probability, $\boldsymbol{x}' - \boldsymbol{x} = (\boldsymbol{\alpha}' - \boldsymbol{\beta})\ell_2$. However, since ℓ_2 is sampled randomly from an exponentially large set of primes independently from $\boldsymbol{r}_1, \boldsymbol{r}_3, \ell_1$, and ℓ_3 (which fix the value of $\boldsymbol{x}' - \boldsymbol{x}$) there is a negligible probability that $\boldsymbol{x}' - \boldsymbol{x} \equiv 0 \pmod{\ell_2}$, unless $\boldsymbol{x}' = \boldsymbol{x}$. By a simple union bound over the $\mathsf{poly}(\lambda)$ number of transcripts, there exists a single $\boldsymbol{x}$ such that $\boldsymbol{x} = \boldsymbol{r}_i \bmod \ell_i$ for all i.

To complete the proof of Theorem 7 we describe the extractor Ext:

1. run $\mathcal{A}_0$ to get output (w, state)
2. let $R \leftarrow \{\}$
3. run Protocol PoKRep with $\mathcal{A}_1$ on input (w, state), sampling fresh randomness for the verifier
4. if the transcript $(\ell, Q, \boldsymbol{r})$ is accepting set $R \leftarrow R \cup \{(\boldsymbol{r}, \ell)\}$, and otherwise return to Step 3
5. use the CRT algorithm to compute $\boldsymbol{x}$ such that $\boldsymbol{x} = \boldsymbol{r}_i \bmod \ell_i$ for each $(\boldsymbol{r}_i, \ell_i) \in R$
6. if $Rep(\boldsymbol{x}) = w$ output $\boldsymbol{x}$ and stop
7. return to Step 3

It remains to argue that Ext succeeds with overwhelming probability in a $\mathsf{poly}(\lambda)$ number of rounds. Suppose that after some polynomial number of rounds the extractor has obtained M accepting transcripts $\{\ell_i, Q_i, \boldsymbol{r}_i\}$ for independent values of $\ell_i \in \mathsf{Primes}(\lambda)$. By the subclaim above, with overwhelming probability there exists $\boldsymbol{x} \in \mathbb{Z}^n$ such that $\boldsymbol{x} = \boldsymbol{r}_i \bmod \ell_i$ and $Rep(\boldsymbol{x}) = w$ and $x_j < 2^q$ for each component of $\boldsymbol{x}$. Hence, the CRT algorithm used in Step 5 will recover the required vector $\boldsymbol{x}$ once $|R| > q$.

Since a single round of interaction with $\mathcal{A}_1$ results in an accepting transcript with probability $\epsilon \geq 1/\mathrm{poly}(\lambda)$, in expectation the extractor obtains $|R| > q$ accepting transcripts for independent primes ℓ_i after $q \cdot \mathrm{poly}(\lambda)$ rounds. Hence, Ext outputs a vector $\boldsymbol{x}$ such that $Rep(\boldsymbol{x}) = w$ in expected polynomial time, as required. □

Proof of Theorem 3.

> Protocol PoKE and Protocol PoKE2 are arguments of knowledge for relation $\mathcal{R}_{\mathsf{PoKE}}$ in the generic group model.

Fix $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$ and $g \in \mathbb{G}$. Let $\mathcal{A}_0, \mathcal{A}_1$ be poly-time adversaries where $(u, w, \mathsf{state}) \xleftarrow{\$} \mathcal{A}_0(g)$ and $\mathcal{A}_1$ runs Protocol PoKE or Protocol PoKE2 with the verifier $\mathsf{V}(g, u, w)$. We need to show that for all $\mathcal{A}_1$ there exists a poly-time Ext such that for all $\mathcal{A}_0$ the following holds: if $\mathsf{V}(g, u, w)$ outputs 1 with non-negligible probability on interaction with $\mathcal{A}_1(g, u, w, \mathsf{state})$ then Ext outputs an integer x such that $u^x = w$ in $\mathbb{G}$ with overwhelming probability.

Proof for Protocol PoKE. Protocol PoKE includes an execution of Protocol PoKE* on $g \in \mathbb{G}$ and input z (the first message sent by the prover to the verifier), and the prover succeeds in Protocol PoKE only if it succeeds in this subprotocol for Protocol PoKE*. Since Protocol PoKE* is a special case of Protocol PoKRep, by Theorem 7 there exists Ext^* for $\mathcal{A}_1$ that outputs $x^* \in \mathbb{Z}$ such that $g^{(x^*)} = z$. Furthermore, as already shown in the analysis of Theorem 7, once Ext^* has obtained x^* it can continue to replay the protocol, sampling a fresh prime $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$, and in each fresh round that produces an accepting transcript it obtains from the Prover a triple (Q, Q', r) such that $r = x^* \bmod \ell$ with overwhelming probability. This is due to the fact that the adversary outputs Q' such that $Q'^{\ell} g^r = z = g^{x^*}$, and the generic group adversary can write $Q' = g^q \prod_{i>1} g_i^{q_i}$ (Lemma 2) such that $q\ell + r = x^*$ with overwhelming probability (Lemma 4).

The extractor Ext will simply run Ext^* to obtain x^*. Now we will show that either $u^{x^*} = w$, i.e. Ext^* extracted a valid witness, or otherwise the adaptive root assumption would be broken, which is impossible in the generic group model (Corollary 1). To see this, we construct an adaptive root adversary $\mathcal{A}_{AR}$ that first runs Ext^* with $\mathcal{A}_0, \mathcal{A}_1$ to obtain x^* and provides $h = w/u^{x^*} \in \mathbb{G}$ to the challenger. When provided with $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ from the challenger, $\mathcal{A}_{AR}$ rewinds $\mathcal{A}_1$, passes ℓ to $\mathcal{A}_1$, and with overwhelming probability obtains Q, r such that $x^* = r \bmod \ell$ and $Q^\ell u^r = w$. Finally, $\mathcal{A}_{AR}$ outputs $v = \frac{Q}{u^{\lfloor \frac{x^*}{\ell} \rfloor}}$, which is an ℓth root of h:

$$v^\ell = \left(\frac{Q}{u^{\lfloor \frac{x^*}{\ell} \rfloor}}\right)^\ell = \left(\frac{Q}{u^{\lfloor \frac{x^*}{\ell} \rfloor}}\right)^\ell \frac{u^r}{u^r} = \frac{w}{u^{x^*}} = h$$

If $w \neq u^{x^*}$ so that $h \neq 1$, then $\mathcal{A}_{AR}$ succeeds in the adaptive root game. In conclusion, the value x^* output by Ext satisfies $w = u^{x^*}$ with overwhelming probability.

Proof for protocol PoKE2 Showing that $\mathsf{Protocol\ PoKE2}$ requires a fresh argument (similar to the analysis in Theorem 7) since the protocol no longer directly contains $\mathsf{Protocol\ PoKE}^*$ as a subprotocol. Ext first obtains u, w from $\mathcal{A}_0$ and runs the first two steps of $\mathsf{Protocol\ PoKE2}$ with $\mathcal{A}_1$ playing the role of the verifier, sampling $g \xleftarrow{\$} \mathbb{G}$ and receiving $z \in \mathbb{G}$ from $\mathcal{A}_1$. Ext is a simple modification of the extractor for $\mathsf{Protocol\ PoKE}$:

1. Set $R \leftarrow \{\}$ and sample $\alpha \xleftarrow{\$} [0, 2^\lambda]$.
2. Sample $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ and send α, ℓ to $\mathcal{A}_1$.
3. Obtain output Q, r from $\mathcal{A}_0$. If $Q^\ell u^r g^{\alpha r} = wz^\alpha$ (i.e. the transcript is accepting) then update $R \leftarrow R \cup \{(r, \ell)\}$. Otherwise return to step 2.
4. Use CRT to compute $x = r_i \mod \ell_i$ for each $(r_i, \ell_i) \in R$. If $u^x = w$ then output x, otherwise return to step 2.

Note that the extractor samples a fresh prime challenge ℓ each time it rewinds the adversary but keeps the challenge α fixed each time. Since these are independently sampled in the real protocol, keeping α fixed while sampling a fresh prime does not change the output distribution of the adversary. This subtle point of the rewinding strategy is important.

There is a negligible probability that the random g sampled by the extractor was contained in the group oracle queries from $\mathcal{A}_0$ to $\mathcal{O}_1$. Thus, by Lemma 2, $\mathcal{A}_0$ knows representations $w = \prod_i g_i^{\omega_i}$ and $u = \prod_i g_i^{\mu_i}$ such that $g_i \neq g$ for all i. $\mathcal{A}_0$ also knows a representation $z = g^\zeta \prod_i g_i^{\zeta_i}$ and for each Q obtained $\mathcal{A}_0$ knows a representation $Q = g^q \prod_i g_i^{q_i}$, which it can pass in state to $\mathcal{A}_1$. If $Q^\ell u^r g^{\alpha r} = wz^\alpha$, then $\mathcal{A}_1$ obtains an equation $g^{q\ell + \alpha r} \prod_i g_i^{q_i \ell + \mu_i r} = g^{\zeta\alpha} \prod_i g_i^{\zeta_i \alpha + \omega_i}$.

By Lemma 4, with overwhelming probability $q\ell + \alpha r = \zeta\alpha$, which implies $\alpha | q\ell$. Since $gcd(\alpha, \ell) = 1$ with overwhelming probability, it follows that $\alpha | q$ and setting $a = q/\alpha$ shows that $\zeta = a\ell + r$, i.e. $\zeta = r \bmod \ell$. Also for the same reasoning $q_i \ell + \mu_i r = \zeta_i \alpha + \omega_i$ with overwhelming probability. Repeating the argument for a different ℓ' sampled by the extractor yields a similar equation $\zeta = a'\ell' + r'$, hence $a\ell + r = a'\ell' + r'$ for some $a' = q'/\alpha$. Also $q_i \ell + \mu_i r - \zeta_i \alpha = q_i' \ell' + \mu_i r' - \zeta_i \alpha$. Substituting for r and r' gives $q_i \ell + \mu_i(\zeta - a\ell) = q_i' \ell' + \mu_i(\zeta - a'\ell')$ implying:

$$(q_i - \mu_i a)\ell = (q_i' - \mu_i a')\ell'$$

(This is where it was important that α is fixed by the extractor, as otherwise we could not cancel the $\zeta_i \alpha$ term on each side of the equation). Now since $\ell \neq \ell' \neq 0$ with overwhelming probability, it follows that $\ell | q_i' - \mu_i a'$ and $\ell' | q_i - \mu_i a$. However, $q_i - \mu_i a$ was fixed independently before ℓ' was sampled, hence there is a negligible probability that it has ℓ' as a factor unless $q_i - \mu_i a = 0$, in which case $q_i' - \mu_i a' = 0$ as well. We conclude that with overwhelming probability $q_i \ell + \mu_i r = q_i' \ell' + \mu_i r' =$

$\mu_i\zeta$. In other words, for each ℓ sampled, as long as $Q^\ell u^r g^{\alpha r} = wz^\alpha$ then with overwhelming probability:

$$wz^\alpha = g^{q\ell+\alpha r}\prod_i g_i^{q_i\ell+\mu_i r} = g^{\zeta\alpha}\prod_i g_i^{\mu_i\zeta} = g^{\zeta\alpha}u^\zeta$$

Finally, if $u^\zeta \neq w$ then $g^\zeta/z \neq 1$ and yet $(g^\zeta/z)^\alpha = u^\zeta/w$. Since α is sampled independently from u, w, g, and ζ, this relation can only hold true with non-negligible probability over the choice of α if both g^ζ/z and u^ζ/w are elements of a small (i.e. $\mathsf{poly}(\lambda)$ size) subgroup generated by g^ζ/z. In other words, g^ζ/z is an element of low order, and it is possible to compute its order in polynomial time. This would be a contradiction in the generic group model since it is hard to find a non-trivial element and its order (Corollary 2). In conclusion, with overwhelming probability $u^\zeta = w$.

Repeating this analysis for each accepting transcript (ℓ_i, Q_i, r_i) shows that $\zeta = r_i \bmod \ell_i$ with overwhelming probability. The remainder of the analysis is identical to the last part of the proof of Theorem 7. Namely, since $\zeta < 2^q$ where $q < \mathsf{poly}(\lambda)$ is an upper bound on the number of queries the adversary makes to the group oracle, we can show there exists a polynomial number of rounds after which Ext would succeed in extracting ζ with overwhelming probability.

□

Proof of Theorem 8.

> For any homomorphism $\phi : \mathbb{Z}^n \to \mathbb{G}$, Protocol PoKHP for relation $\mathcal{R}_\phi = \{(w; \boldsymbol{x}) : \phi(\boldsymbol{x}) = w\}$ is an argument of knowledge in the generic group model.

The proof is a direct generalization of the proof of Theorem 3 for Protocol PoKE. As usual, fix $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$ and $\mathbf{g} = (g_1, ..., g_n) \in \mathbb{G}$. Let $\mathcal{A}_0, \mathcal{A}_1$ be poly-time generic adversaries where $(w, \mathsf{state}) \xleftarrow{\$} A_0(\mathbf{g})$ and $\mathcal{A}_1(\mathsf{state})$ runs Protocol PoKHP with the verifier $V(\mathbf{g}, w)$. We need to show that for all $\mathcal{A}_1$ there exists a poly-time Ext such that for all $\mathcal{A}_0$ the following holds: if $\mathcal{A}_1$ convinces $V(\mathbf{g}, w)$ to accept with probability at least $1/\mathrm{poly}(\lambda)$ then Ext outputs $\boldsymbol{x} \in \mathbb{Z}^n$ such that $\phi(\boldsymbol{x}) = w$ with overwhelming probability.

Protocol PoKHP includes an execution of Protocol PoKRep on $g_1, ..., g_n \in \mathbb{G}$ and input z (the first message sent by the prover to the verifier), and the prover succeeds in Protocol PoKHP only if it succeeds in this subprotocol for Protocol PoKRep. By Theorem 7 there exists Ext^* for each $\mathcal{A}_1$ that outputs $\boldsymbol{x}^*$ such that $Rep(\boldsymbol{x}^*) = z$. Furthermore, as shown in the analysis of Theorem 7, once Ext^* has obtained $\boldsymbol{x}^*$ it can continue to replay the protocol, sampling a fresh prime $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$, and in each fresh round that produces an accepting transcript it obtains from the Prover values Q, Q' and $\boldsymbol{r}$ such that $\boldsymbol{r} = x^* \bmod \ell$ with overwhelming probability.

The extractor Ext simply runs Ext^* to obtain $\boldsymbol{x}^*$. Now we will show that either $\phi(\boldsymbol{x}^*) = w$, i.e. Ext^* extracted a valid witness, or otherwise the adaptive root assumption would be broken, which is impossible in the generic group model (Corollary 1). To see this, we construct an adaptive root adversary $\mathcal{A}_{AR}$ that first runs Ext^* with $\mathcal{A}_0, \mathcal{A}_1$ to obtain $\boldsymbol{x}^*$ and provides $h = w/\phi(\boldsymbol{x}^*) \in \mathbb{G}$ to the challenger. When provided with $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ from the challenger, $\mathcal{A}_{AR}$ rewinds $\mathcal{A}_1$, passes ℓ to $\mathcal{A}_1$, and with overwhelming probability obtains $Q, \boldsymbol{r}$ such that $\boldsymbol{x}^* = \boldsymbol{r} \bmod \ell$ and $Q^\ell\phi(\boldsymbol{r}) = w$. Finally, define $\lfloor \boldsymbol{x}^*/\ell \rfloor$ to be the vector obtained by replacing each component x_i with the quotient $\lfloor x_i/\ell \rfloor$. $\mathcal{A}_{AR}$ outputs $v = \frac{Q}{\phi(\lfloor \boldsymbol{x}^*/\ell \rfloor)}$. Using the fact that ϕ is a group homomorphism we can show that this is an ℓth root of h:

$$v^\ell = \Big(\frac{Q}{\phi(\lfloor \frac{\boldsymbol{x}^*}{\ell} \rfloor)}\Big)^\ell = \frac{Q^\ell}{\phi(\ell \cdot \lfloor \frac{\boldsymbol{x}^*}{\ell} \rfloor)} = \frac{Q^\ell}{\phi(\boldsymbol{x}^* - \boldsymbol{r})}\frac{\phi(\boldsymbol{r})}{\phi(\boldsymbol{r})} = \frac{w}{\phi(\boldsymbol{x}^*)} = h$$

If $w \neq \phi(\boldsymbol{x}^*)$ so that $h \neq 1$, then $\mathcal{A}_{AR}$ succeeds in the adaptive root game. In conclusion, the value $\boldsymbol{x}^*$ output by Ext satisfies $w = \phi(\boldsymbol{x}^*)$ with overwhelming probability.

□

Proof of Theorem 10.

> Protocol ZKPoKRep is an honest-verifier statistical zero-knowledge argument of knowledge for relation $\mathcal{R}_{\mathsf{Rep}}$ in the generic group model.

Part 1: HVZK To show that the protocol is honest-verifier zero-knowledge we build a simulator Sim. Sim samples $(\tilde{A}, \tilde{c}, \tilde{\ell}, \tilde{\boldsymbol{r}}, \tilde{Q})$ as follows. Let $\mathbb{G}_i$ denote the subgroup of $\mathbb{G}$ generated by the base g_i.

1. $\tilde{c} \xleftarrow{\$} [0, 2^\lambda]$, $\tilde{\ell} \xleftarrow{\$} \mathsf{Primes}(\lambda)$
2. $\tilde{\boldsymbol{q}} \xleftarrow{\$} [B]^n$
3. $\tilde{\boldsymbol{r}} \xleftarrow{\$} [\ell]^n$
4. $\tilde{Q} \leftarrow \prod_{i=1}^n g_i^{\tilde{q}_i}$
5. $\tilde{A} \leftarrow \tilde{Q}^{\tilde{\ell}} (\prod_{i=1}^n g_i^{\tilde{r}_i})^{-1} w^{-\tilde{c}}$.

We now argue that $(\tilde{A}, \tilde{c}, \tilde{\ell}, \tilde{\boldsymbol{r}}, \tilde{Q})$ is statistically indistinguishable from a transcript between an honest prover and verifier: $(A, c, \ell, \boldsymbol{r}, Q)$. Sim chooses $\tilde{\ell}$ and $\tilde{c}$ identically to the honest verifier in the real protocol. It also solves for $\tilde{A}$ uniquely from the other values such that the verification holds. Therefore, it remains only to show that $\tilde{\boldsymbol{r}}$ and $\tilde{Q}$ have the correct distribution. We must show that in the real protocol, independent of ℓ and c, $\boldsymbol{r}$ has statistical distance less than $2^{-\lambda}$ from the uniform distribution over $[\ell]^n$ and each $g_i^{q_i}$ has statistical distance less than $2^{-\lambda}$ from uniform over $\mathbb{G}_i$ (recall that $Q = \prod_i g_i^{q_i}$). In addition we must argue that Q and $\boldsymbol{r}$ are independent.

For this we use the following facts, which are easy to verify:

1. Fact 1: If Z is uniform random variable over N consecutive integers and $m < N$ then $Z \bmod m$ has statistical distance at most m/N from the uniform distribution over $[m]$.

2. Fact 2: For independent random variables X_1, X_2, Y_1, Y_2, the distance between the joint distributions (X_1, X_2) and (Y_1, Y_2) is at most the sum of statistical distances of X_1 from Y_1 and X_2 from Y_2. Similarly, if these variables are group elements in $\mathbb{G}$, the statistical distance between $X_1 \cdot X_2$ and $Y_1 \cdot Y_2$ is no greater than the sum of statistical distances of X_1 from Y_1 and X_2 from Y_2.

3. Fact 3: Consider random variables X_1, X_2, Y_1, Y_2 with statistical distances $s_1 = \Delta(X1, X2)$ and $s_2 = \Delta(Y_1, Y_2)$, where $Pr(X_1 = x | Y_1 = y) < Pr(X_1 = x) + \epsilon_1$ and $Pr(X_2 = x | Y_2 = y) < Pr(X_1 = x) + \epsilon_2$ for all values x, y. Then the joint distributions (X_1, X_2) and (Y_1, Y_2) have statistical distance at most $s_1 + s_2 + \epsilon_2 |supp(X_1)| + \epsilon_1 |supp(Y_1)|$, where *supp* is the support.

Consider fixed values of c, x_i and ℓ. In the real protocol, for each $i \in [n]$ the prover computes $s_i = cx_i + k_i$ where k_i is uniform in $[-B, B]$ and sets $r_i = s_i \bmod \ell$ and $q_i = \lfloor \frac{s_i}{\ell} \rfloor$. The value of s_i is distributed uniformly over a range of $2B+1$ consecutive integers, thus r_i has statistical distance at most $\ell/(2B+1)$ from uniform over $[\ell]$. This bounds the distance between r_i and the simulated $\tilde{r}_i$, which is uniform over $[\ell]$.

Next we show that each $g_i^{q_i}$ is statistically indistinguishable from uniform in $\mathbb{G}_i$. Consider the distribution of $\lfloor \frac{s_i}{\ell} \rfloor$ over the consecutive integers in $[\lfloor \frac{cx_i - B}{\ell} \rfloor, \lfloor \frac{cx_i + B}{\ell} \rfloor]$. Denote this by the random variable Z_i. The distribution of $g_i^{q_i}$ over $\mathbb{G}_i$ is determined by the distribution of $q_i \bmod |\mathbb{G}_i|$. The probability that $q_i = z$ is the probability that s_i falls in the interval $[z\ell, (z+1)\ell - 1]$. This probability is $\ell/(2B+1)$ for all points where $z\ell \geq cx_i - B$ and $(z+1)\ell \leq cx_i + B$, which includes all points except possibly the two endpoints $z = \lfloor \frac{cx_i - B}{\ell} \rfloor$ and $z = \lfloor \frac{cx_i + B}{\ell} \rfloor$. Call this set of points Y. The distance of q_i from a uniform random variable U_Y over Y is largest when $cx_i - B = 1 \bmod \ell$ and $cx_i + B = 0 \bmod \ell$. In this case, q_i is one of the two endpoints outside Y with probability $1/B$. For each $z \in Y$, $Pr[q_i = z] = \ell/(2B+1)$. As $|Y| = 2(B-1)/\ell$, the statistical distance of q_i from U_Y is at most: $\frac{1}{2}[Y(\frac{1}{Y} - \frac{\ell}{2B}) + \frac{1}{B}] = \frac{1}{2}(1 - \frac{B-1}{B} + \frac{1}{B}) = \frac{1}{B}$. Moreover, the statistical distance of $q_i \bmod |\mathbb{G}_i|$ from $U_Y \bmod |\mathbb{G}_i|$ is no larger.

As noted in the Fact 1 above, $U_Y \bmod |\mathbb{G}_i|$ has statistical distance at most $|\mathbb{G}_i|/|Y| \leq \ell|\mathbb{G}|/2(B-1) < 1/(n2^{\lambda+1})$ for $B > n2^{2\lambda}|\mathbb{G}|$. By the triangle inequality, the statistical distance of $q_i \bmod |\mathbb{G}_i|$ from uniform is at most $1/B + 1/n2^{\lambda+1} < 1/(n2^{\lambda})$. This also bounds the distance of $g_i^{q_i}$ from uniform in $|G|_i$. The simulated $g_i^{\tilde{q}_i}$ has distance at most $1/(n2^{2\lambda})$ from uniform in G_i since $\tilde{q}_i \bmod |G|_i$ has distance $B/|G|_i < 1/(n2^{2\lambda})$ from uniform (again by the Fact 1 above). By the triangle inequality, the distance between $g_i^{\tilde{q}_i}$ and $g_i^{q_i}$ is at most $1/B + 1/(n2^{\lambda+1}) + 1/(n2^{2\lambda}) < (1/2^{\lambda} + 1/2 + 1/2^{\lambda})1/(n2^{\lambda}) < 1/(n2^{\lambda})$.

We have shown that each r_i is statistically indistinguishable from the simulated $\tilde{r_i}$ and each g^{q_i} is statistically indistinguishable from the simulated $g^{\tilde{q}_i}$. However, we must consider the distances between the joint distributions. Since q_i and r_i are not independently distributed, arguing about the joint distributions requires more work. The simulated $\tilde{q}_i$ and $\tilde{r}_i$ are independent on the other hand.

Consider the conditional distribution of $q_i|r_i$ (i.e. the distribution of the random variable for q_i conditioned the value of r_i). Note that $q_i = z$ if $(s_i - r_i)/\ell = z$. We repeat a similar argument as above for bounding the distribution of q_i from uniform. For each possible value of z, there always exists a unique value of s_i such that $s_i//\ell = z$ and $s_i = 0 \bmod \ell$, except possibly at the two endpoints of the range of q_i (i.e. $e_1 = \lfloor \frac{cx_i - B}{\ell} \rfloor$ and $e_2 = \lfloor \frac{cx_i + B}{\ell} \rfloor$). When r_i disqualifies the two points e_1 and e_2, then each of the remaining points $z \notin \{e_1, e_2\}$ still have equal probability mass, and thus the probability $Pr(q_i = z|r_i)$ increases by at most $[Pr(q_i = e_1) + Pr(q_i = e_2)]/(2\lfloor \frac{B}{\ell} \rfloor) < 1/B^2$. The same applies to the variable $q_i|r_i \bmod |G|_i$ and hence the variable $g^{q_i}|r_i$.

We can compare the joint distribution $X_i = (g^{q_i}, r_i)$ to the simulated $Y_i(g^{\tilde{q}_i}, \tilde{r_i})$ using Fact 3 above. Setting $\epsilon_1 = 1/B^2$ and $\epsilon_2 = 0$, the distance between these joint distributions is at most $1/(n2^{\lambda}) + \ell/(2B+1) + \ell/B^2$. Moreover, as each $X_i = (g^{q_i}, r_i)$ is independent from $X_j = (g^{q_j}, r_j)$ for $i \neq j$, we use Fact 2 to bound the distances between the joint distributions $(g^{q_1}, ..., g^{q_n}, r_1, ..., r_n)$ and $(g^{\tilde{q_1}}, ..., g^{\tilde{q_n}}, \tilde{r_1}, ..., \tilde{r_n})$ by the sum of individual distances between each X_i and Y_i, which is at most $1/2^{\lambda} + n\ell/(2B+1) + n\ell/B^2 < 2^{-\lambda+1}$. Finally, this also bounds the distance between $(Q, \boldsymbol{r})$ and $(\tilde{Q}, \tilde{\boldsymbol{r}})$ where $Q = \prod_i g^{q_i}$ and $\tilde{Q} = \prod_i g^{\tilde{q}_i}$.

Part 2: PoK For extraction we describe an efficient extractor Ext. Ext randomly samples two random challenges c and c', and $c \neq c'$ with probability $\frac{1}{2^{\lambda}}$. Ext then uses the extractor from *Theorem* 7 to extract $\boldsymbol{s}$ and $\boldsymbol{s}'$ such that $\prod_{i=1}^n g_i^{s_i} = Aw^c$ and $\prod_{i=1}^n g_i^{s_i'} = Aw^{c'}$. We now compute $\Delta s_i = s_i - s_i'$ for all $i \in [1, n]$ and $\Delta c = c - c'$. This gives us $\prod_{i=1}^n g_i^{\Delta s_i} = w^{\Delta c}$. We now claim that $\Delta c \in \mathbb{Z}$ divides $\Delta s_i \in \mathbb{Z}$ for each $i \in [1, n]$ with overwhelming probability and that $\prod_{i=1}^n g_i^{\Delta s_i/\Delta c} = w$. By Lemma 2, we can write $w = \prod_{i=1}^m g_i^{\alpha_i}$, for integers $\alpha_i \in \mathbb{Z}$ that can be efficiently computed from $\mathcal{A}$'s queries to the generic group oracle. Since $\prod_{i=1}^n g_i^{\Delta s_i} = w^{\Delta c}$ it follows by Lemma 4 that, with overwhelming probability, $\alpha_j = 0$ for all $j > n$ and $\Delta s_i = \alpha_i \Delta c$ for all $i \in [1, n]$.

Furthermore, if $\mu = \prod_{i=1}^n g_i^{\Delta s_i/\Delta c} \neq w$, then since $\mu^{\Delta c} = \prod_{i=1}^n g_i^{\Delta s_i} = w^{\Delta c}$ it would follow that μ/w is an element of order $\Delta c > 1$. As Δc is easy to compute this would contradict the hardness of computing a non-trivial element and its order in the generic group model (Corollary 2). We can conclude that $\mu = w$ with overwhelming probability. The extractor outputs $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$ where $\alpha_i = \Delta s_i/\Delta c$. □

Proof of Theorem 11.

> $\mathsf{Protocol\ ZKPoKE}$ is an honest-verifier statistically zero-knowledge argument of knowledge for relation $\mathcal{R}_{\mathsf{PoKE}}$ in the generic group model.

To prove that the protocol is honest-verifier zero-knowledge we build a simulator Sim which generates valid transcripts that are statistically indistinguishable from honestly generated ones. The simulator generates a transcript as follows:

1. $\tilde{c} \xleftarrow{\$} [0, 2^{\lambda}], \tilde{\ell} \xleftarrow{\$} \mathsf{Primes}(\lambda)$
2. $\tilde{z} \leftarrow h^{\tilde{\rho}}, \rho \xleftarrow{\$} [B]$

3. $\tilde{q}_x, \tilde{q}_r \xleftarrow{\$} [B]^2$
4. $\tilde{r}_x, \tilde{r}_\rho \in [\ell]^2$
5. $\tilde{Q}_g \leftarrow g^{\tilde{q}_x} h^{\tilde{q}_\rho}, \tilde{Q}_u \leftarrow u^{\tilde{q}_x}$
6. $\tilde{A}_g \leftarrow \tilde{Q}_g^\ell g^{\tilde{r}_x} h^{\tilde{r}_\rho} z^{-\tilde{c}}, \tilde{A}_u \leftarrow \tilde{Q}_u^\ell u^{\tilde{r}_x} w^{-\tilde{c}}$

We now argue that the transcript $(\tilde{z}, \tilde{A}_g, \tilde{A}_u, \tilde{c}, \tilde{\ell}, \tilde{Q}_g, \tilde{Q}_u, \tilde{r}_x, \tilde{r}_\rho)$ is statistically indistinguishable from a transcript between an honest prover and verifier: $(z, A_g, A_u, c, \ell, Q_g, Q, u, r_x, r_\rho)$ $\tilde{\ell}, \tilde{c}$ are identically chosen as by the random verifier and $\tilde{A}_g, \tilde{A}_u$ are uniquely defined by the rest of the transcript and the verification equations. It thus suffices to argue that $\tilde{z}, \tilde{Q}_g, \tilde{Q}_u, \tilde{r}_x, \tilde{r}_\rho$ as well as z, Q_g, Q_u, r_x, r_ρ are statistically indistinguishable from uniform in their respective domain.

Using Fact 1 stated in the proof of Theorem 10 and that $B > 2^\lambda |\mathbb{G}|$ we can see that $\tilde{z}$ is indistinguishable from a uniform element in the subgroup of $\mathbb{G}$ generated by h. Since g and h generate the same subgroup the same argument applies to z. For $\tilde{Q}_g, \tilde{Q}_u, \tilde{r}_x, \tilde{r}_\rho$ and Q_g, Q_u, r_x, r_ρ the same argument as in the proof of *Theorem* 10 apply, showing that all values are nearly uniform. The simulation therefore produces valid, statistically indistinguishable transcripts. Note that the requirement that g, h generate the same group can be relaxed under computational assumptions. The assumption states that it is difficult to distinguish between g, h which generate the same subgroup and g', h' which don't. Given this we can use a hybrid argument which replaces g', h' with g, h and the applies the same simulation argument as above.

For extraction, note that the protocol contains Protocol ZKPoKRep as a sub-protocol on input A_g and bases g, h in the CRS, and therefore we can use the ZKPoKReP and PoKRep extractors to extract x, ρ such that $z = g^x h^\rho$ and s_1, s_2 such that $g^{s_1} h^{s_2} = A_g z^c$ with overwhelming probability. Moreover, as shown in the analysis for the PoKRep extractor, we can rewind the adversary on fresh challenges so that each accepting transcript outputs an r_1, ℓ where $s_1 = r_1 \bmod \ell$ with overwhelming probability. If $u^{s_1} \neq A_u w^c = Q_u^\ell u^{r_1}$ then $\gamma = (r_1 - s_1)/\ell$ is an integer and $Q_u u^\gamma$ is an ℓth root of $A_u w^c / u^{s_1} \neq 1$. This would break the adaptive root assumption, hence by Corollary 1 it follows that $u^{s_1} = A_u w^c$ with overwhelming probability.

Recall from the analysis of Theorem 10 that the extractor obtains a pair of accepting transcripts with $s_1, s_2, s_1', s_2', c, c'$ so that $x = \Delta s_1 / \Delta c = (s_1 - s_1')/(c - c')$ and $\rho = \Delta s_2 / \Delta c = (s_2 - s_2')/(c - c')$. Since $u^{s_1} = A_u w^c$ and $u^{s_1'} = A_u w^{c'}$ with overwhelming probability, we obtain $u^{\Delta s_1} = w^{\Delta c}$ with overwhelming probability. Finally, this implies $(u^x)^{\Delta c} = w^{\Delta c}$. If $u^x \neq w$, then u^x / w is a non-trivial element of order Δc, which would contradict the hardness of computing a non-trivial element and its order in the generic group model (Corollary 2). Hence, we conclude that $u^x = w$ with overwhelming probability.

□

D Non-interactive PoE and PoKE variants

NI-PoE	NI-PoKE2
$\{x, u, w : u^x = w\}$	$\{(u, w; x) : u^x = w\}$
$\mathsf{Prove}(x, u, w)$:	$\mathsf{Prove}(x, u, w)$:
$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(x, u, w)$	$g \leftarrow \mathsf{H}_{\mathbb{G}}(u, w), z = g^x$
$q \leftarrow \lfloor x/\ell \rfloor$	$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(u, w, z), \alpha = \mathsf{H}(u, w, z, \ell)$
$Q \leftarrow u^q$	$q \leftarrow \lfloor x/\ell \rfloor, r \leftarrow x \bmod \ell$
	$\pi \leftarrow \{z, (ug^\alpha)^q, r\}$
$\mathsf{Verify}(x, u, w, Q)$:	$\mathsf{Verify}(u, w, z, Q, r)$:
$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(x, u, w)$	$g \leftarrow \mathsf{H}_{\mathbb{G}}(u, w)$
$r \leftarrow x \bmod \ell$	$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(u, w, z), \alpha \leftarrow \mathsf{H}(u, w, z, \ell)$
Check: $Q^\ell u^r = w$	Check: $Q^\ell (ug^\alpha)^r = wz^\alpha$

NI-PoDDH

$\{(y_1, y_2, y_3); (x_1, x_2) : g^{x_1} = y_1 \wedge g^{x_2} = y_2 \wedge y_1^{x_2} = y_3$

$\mathsf{Prove}(\mathbf{x} = (x_1, x_2), \mathbf{y} = (y_1, y_2, y_3)) :$

$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(\mathbf{y})$

$(q_1, q_2) \leftarrow (\lfloor x_1/\ell \rfloor, \lfloor x_2/\ell \rfloor)$

$(r_1, r_2) \leftarrow (x_1 \bmod \ell, x_2 \bmod \ell)$

$\pi \leftarrow \{(g^{q_1}, g^{q_2}, y_1^{q_2}), r_1, r_2\}$

$\mathsf{Verify}(\mathbf{y}, \pi) :$

$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(\mathbf{y})$

$\{Q_{y_1}, Q_{y_2}, Q_{y_3}, r_1, r_2\} \leftarrow \pi$

Check:

$\mathbf{r} \in [\ell]^2 \wedge Q_{y_1}^{\ell} g^{r_1} = y_1 \wedge Q_{y_2}^{\ell} g^{r_2} = y_2 \wedge Q_{y_3}^{\ell} y_1^{r_2} = y_3$

NI-ZKPoKE

$\{(u, w; x) : u^x = w\}$

$\mathsf{Prove}(x, u, w) :$

$k, \rho_x, \rho_k \xleftarrow{\$} [-B, B]; \quad z = g^x h^{\rho_x}; \quad A_g = g^k h^{\rho_k}; \quad A_u = u^k;$

$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(u, w, z, A_g, A_u); \quad c \leftarrow \mathsf{H}(\ell);$

$q_x \leftarrow \lfloor (k + c \cdot x)/\ell \rfloor; \quad q_\rho \leftarrow \lfloor (\rho_k + c \cdot \rho_x)/\ell \rfloor;$

$r_x \leftarrow (k + c \cdot x) \bmod \ell; \quad r_\rho \leftarrow (\rho_k + c \cdot \rho_x) \bmod \ell;$

$\pi \leftarrow \{\ell, z, g^{q_x} h^{q_\rho}, u^{q_x}, r_x, r_\rho\}$

$\mathsf{Verify}() :$

$\{c, z, Q_g, Q_u, r_x, r_\rho\} \leftarrow \pi$

$c = \mathsf{H}(\ell) \quad A_g \leftarrow Q_g^{\ell} g^{r_x} h^{r_\rho} z^{-c}; \quad A_u \leftarrow Q_u^{\ell} u^{r_x} w^{-c}$

Check: $r_x, r_\rho \in [\ell]; \ \ell = \mathsf{H}_{\mathsf{prime}}(u, w, z, A_g, A_u)$