# Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains

Dan Boneh, Benedikt Bünz, Benjamin Fisch
Stanford University

**Abstract**

We present batching techniques for cryptographic accumulators and vector commitments in groups of unknown order. Our techniques are tailored for decentralized 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 a positional vector commitment with constant sized openings and constant sized public parameters. As a core building block for our batching techniques we develop several succinct proofs for groups of unknown order. These include a proof that an exponentiation was done correctly and a zero-knowledge proof of knowledge of an integer discrete logarithm between two group elements. We use these new constructions to design a stateless blockchain, where nodes only need a constant amout of storage. Further we show that our vector commitment can be used to significantly 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/non-membership proofs for any element in the set. These proofs can be publicly verified against the commitment. In a dynamic accumulator, the commitment and proofs can be updated efficiently as elements are added or removed from the set. The typical way in which an accumulator 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 proofs and efficiently verify the integrity of the database. Accumulators have been suggested for many applications within this realm, including accountable certificate management [NN98, BLL00], timestamping [Bd94], revocable anonymous credentials [CL02], computations on authenticated data [ABC$^+$12], privacy-preserving data outsourcing [Sla12], updatable signatures [PS14, CJ10], anonymous e-cash [STS99, MGGR13], and decentralized bulletin boards [FVY14, GGM14].

Several accumulator variants exist. One major distinction is between a *static* and *dynamic* accumulator. A dynamic accumulator supports addition and deletion of elements at unit cost, independent of the number of accumulated elements. A *universal* accumulator is one that is dynamic and also supports both membership and non-membership proofs.

Our present work is motivated by two specific applications of accumulators: stateless blockchains and short interactive oracle proofs.

**Stateless blockchains.** The Bitcoin blockchain operates 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 all nodes in the network. Every transaction has one or more input source addresses and one or more output target addresses. The transaction completely transfers all the funds associated with the source addresses to the output addresses. It is only valid if every source address is the output of a previous transaction that has not been used before as the input to another transaction. These are called the *unspent transaction outputs* (UTXOs). Therefore, checking the validity of a transaction amounts to verifying that every input address in the transaction is a member of the current UTXO set. It is important that all nodes agree on the this UTXO set, which evolves with every new transaction. An ADS is useful as a communication efficient way to verify that all nodes in the network agree on the UTXO set by only comparing the commitment to the set.

Currently Bitcoin uses a hash chain as a short commitment to the transaction history. Every node in the system stores the entire UTXO set in order to verify incoming transactions. An ADS that additionally uses an accumulator for membership proofs would alleviate this need. A node validating a transaction would not need to store the entire UTXO set. Instead, it would verify that a transaction is in the current UTXO set by verifying a membership proof. In fact, given the advanced capabilities of dynamic accumulators, no single node in the network would be required to maintain the entire UTXO set. As long as all the nodes agree on the accumulator commitment, individual nodes can maintain UTXOs of interest along with their membership proofs. They can efficiently update the UTXO set commitment and membership proof with every new batch of transactions. The same idea can be applied to blockchains with more complex data representations, such as a key-value store (e.g. in Ethereum), given an ADS with similar capabilities for a key-value store as accumulators offer for a set. This design concept is referred to as a *stateless blockchain* [Tod16].

**Interactive oracle proofs (IOPs).** Our second application is construct short 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 proof* [BCS16]. In an IOP the prover sends multiple proof oracles to a verifier which responds with challenges. The verifier then queries the oracles for small subsets of the proof and afterwards accepts or rejects. If the oracle is replaced with a Merkle tree commitment and the verifier is public coin then an IOP can be transformed into a short non-interactive proof of knowledge that is secure in the random oracle model [BCS16]. For every oracle query to the proof oracles the prover provides a Merkle inclusion path which ensures that the returned value was indeed a particular part of the proof.

A Merkle tree [Mer88] is a special example of an accumulator: a commitment to a set is constant size (i.e., independent of the size of the set) and membership/non-membership proofs are logarithmic in the size of the set. A Merkle tree can also be used as a *vector commitment* (VC). A VC is similar to an accumulator, but it is a *position binding* commitment (i.e. to a vector or ordered list rather than a set). A VC does not have membership/non-membership proofs per se, but can be opened at any position to a unique value with a short opening proof. Both IOPs and stateless blockchains can be constructed using a Merkle tree: IOPs use Merkle trees as vector commitments and stateless UTXO-blockchains use them as accumulators. However, Merkle trees have

two significant drawbacks for these applications. The first is that the membership proofs (equiv. position openings) are not constant size. This is an issue for large UTXO sets and contributes substantial overhead to IOP proof size. The second is that the multiple membership proofs (equiv. position openings) cannot be compressed into a single proof. The ability to aggregate multiple proofs would have clear advantages for both applications: it would amortize the communication cost for verifying a large batch of transactions and verifying many positions of an IOP.

**Our contributions.** In this paper we provide several batching and aggregation techniques for accumulators and vector commitments. We generally refer to *batching* when a single action is applied to $n$ items instead of one action per item. For example $n$ proofs can be batch verified 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, $n$ membership proofs can be aggregated to a single constant sized proof.

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 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 first expand this technique to provide a new proof of knowledge of exponent, i.e. 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. We prove security in the generic group model. The use of generic groups seems necessary to bypass certain impossibility results for proofs in groups of unknown order [BCK10, TW12]. We also extend the protocol to obtain a zero-knowledge $\Sigma$-Protocol that is an argument of knowledge of a pre-image of a homomorphism from $\mathbb{Z}^q$ to a group of unknown order. This protocol is the first succinct $\Sigma$-protocol for these groups.

Next, we extend current RSA-based accumulators [CL02, LLX07] to create a universal accumulator for a decentralized setting where no single trusted accumulator manager exists and where updates are processed in batches. Despite this we show how 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, is a candidate group of unknown order without a trusted setup[BH01].

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] 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 allows us to create sparse vector commitments which can be used as a key-value map commitment.

Our techniques have applications to a new blockchain design where users provide correctness proofs for their transactions and verifiers are not required to store any state. Additionally, we propose to use our vector commitment as a replacement for Merkle trees in interactive oracle proofs (IOPs). This significantly reduces the proof size for several IOP constructions. Additionally,

we can leverage the proof of exponentiation to reduce the verification time. This comes at the cost of increasing the prover's work.

**Soundness lower bounds in groups of unknown order.** [BCK10] proved that a certain family of sigma protocols in a generic group of unknown order can achieve at most soundness 1/2 per challenge. [TW12] further extend these impossibility results. Our work gives protocols in a generic group of unknown order that have negligible soundness error, appearing to contradict these lower bounds. The reason there is no contradiction is two-fold. First, our protocols are not in the family of protocols considered by [BCK10]. Second, and more importantly, the lower bound proof of [BCK10] shows that an extractor operating in a generic group cannot extract a witness from a successful prover. In contrast, we show how to extract from a successful prover that is restricted to the generic group model. While proving extraction from an arbitrary prover is preferable, the lower bounds suggest that this cannot be done.

**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 very naturally support batching of membership proofs. Vector commitments based on similar techniques [LY10, CF13, LRY16] have constant size openings, but large setup parameters. The ability to aggregate constant-size position openings of vector commitments, or non-membership proofs in accumulators, has not yet been thoroughly explored.

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] first considered the setting of accumulators without a trusted setup from groups of unknown order. His construction does not require an accumulator manager but it provides only a static accumulator. We extend the construction to get a universal accumulator without trusted setup. In concurrent work [CPZ18] also observed that accumulators and vector commitments can be used to build stateless blockchains. They propose a new vector commitment based on bilinear maps and multivariate polynomials. Their scheme, however, requires linear public parameters, does not have a trusted setup and does not support batching of inclusion proofs. Class groups as a group of unknown order, have recently regained attention in the study of verifiable delay functions [Pie18b, Wes18, BBF18].

## 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 $i$th component
- $[\ell]$ denotes the set of integers $\{0, 1, \ldots, \ell - 1\}$.
- $\mathsf{negl}(\lambda)$ is a negligible function of the security parameter $\lambda$
- $\mathsf{Primes}(\lambda)$ is the set of integer primes less than $2^\lambda$

- $x \xleftarrow{\$} S$ 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 $\lambda$.

## 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 $\mathsf{state}$. Then, a random prime $\ell$ 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 $(\mathcal{A}_0, \mathcal{A}_1)$:*

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

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.

**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 $\ell$ 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 $\mathsf{Pgen}$ takes an (implicit) security parameter $\lambda$ and outputs a common reference string (crs) $\mathsf{pp}$. If the setup algorithm uses only public randomness we say that the setup is transparent and that the crs is unstructured. The prover $\mathsf{P}$ takes as input a statement $x \in \mathcal{X}$, a witness $w \in \mathcal{W}$, and the crs $\mathsf{pp}$. The verifier $\mathsf{V}$ takes as input $\mathsf{pp}$ and $x$ and 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 $\mathsf{V}$ 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* $(x, w) \in \mathcal{R}$*:*

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

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* $\mathsf{Pgen}$ is run, and $\mathcal{A}_1$ runs the interactive protocol with the verifier using a $\mathsf{state}$ 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 $\mathsf{p}$ (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* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*:*

$$\Pr\big[\, \langle \mathsf{V}(\mathit{pp}, x), \mathcal{A}_1(\mathit{pp}, x, \mathit{state}) \rangle = 1 \text{ and } \nexists w \;\; (x, w) \in \mathcal{R} : \mathit{pp} \xleftarrow{\$} \mathsf{Pgen}(1^\lambda), (x, \mathit{state}) \leftarrow \mathcal{A}_0(\mathit{pp}) \big] = \mathsf{negl}(\lambda)\,.$$

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

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

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** (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* $\mathsf{Sim}$ *such that for* $(x, w) \in \mathcal{R}$ *the following two distribution are statistically indistinguishable:*

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

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

**Definition 6** (Non interactive arguments)**.** *A* ***non-interactive argument system*** *is an argument system where the interaction between* $\mathsf{P}$ *and* $\mathsf{V}$ *consists of only a single round. We then write the prover* $\mathsf{P}$ *as* $\pi \xleftarrow{\$} \mathsf{Prove}(pp, x, w)$ *and the verifier as* $\mathsf{Vf}(pp, x, \pi)$*.*

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:

Protocol PoE (Proof of exponentiation) for Relation $\mathcal{R}_{\mathsf{PoE}}$ [Wes18]

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 $Q \leftarrow g^q \in \mathbb{G}$ 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 $g^x \in \mathbb{G}$ takes $O(\log x)$ group operations in $\mathbb{G}$. Hence, for the current groups of unknown order, computing $g^x$ takes $\lambda^3$ times as long as computing $x \bmod \ell$. Concretely, when $\ell$ 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.

### 3.2 A succinct proof of homomorphism preimage

Next, we observe that the protocol PoE above 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 Relation $\mathcal{R}_{\phi,\mathsf{PoHP}}$

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 2** (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 $\ell$, and outputting $\hat{u} := Q/\phi(\mathbf{q}) \in \mathbb{G}$, where $Q$ is the value output by the adversary. □

### 3.3 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:

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

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:

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

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 that $Q^\ell g^r = w$ 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 5 below.

**Theorem 3.** 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 $\ell$ 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) for relation $\mathcal{R}_{\mathsf{PoKE}}$

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 5) 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) for Relation $\mathcal{R}_{\mathsf{PoKE}}$

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 $\alpha$. 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.

**Theorem 4** (PoKE Argument of Knowledge)**.** *Protocol PoKE and Protocol PoKE2 are arguments of knowledge for relation $\mathcal{R}_{\mathsf{PoKE}}$ in the generic group model.*

*Proof.* See Appendix A. □

### 3.4 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 $\phi : \mathbb{Z}^n \to \mathbb{G}$.

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

For a general homomorphism $\phi$ 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) for relation $\mathcal{R}_\phi$

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) for relation $\mathcal{R}_\phi$ where $\phi := Rep$

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 5** (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 A. □

**Theorem 6** (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 A. □

### 3.5 A succinct zero-knowledge proof of discrete-log

Our succinct proof of knowledge for exponents and homomorphism preimages 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.

It is well known that the classical Schnorr protocol for hidden order groups is honest verifier statistical zero-knowledge (HVSZK) 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.

**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 $\ell$ 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 $\rho$ 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 $\phi := \mathit{Rep}$

Params: $(g_1, \dots g_n) \in \mathbb{G}$, $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$; Inputs: $w \in \mathbb{G}$, $B > 2^\lambda |\mathbb{G}|$;
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 $Q^\ell \prod_{i=1}^n g_i^{r_i} = A w^c$.

**Theorem 7** (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 A. □

Finally, we use the protocol above to obtain a ZK protocol for the relation $\mathcal{R}_{\mathsf{PoKE}}$. The protocol applies (in parallel) the $\Sigma$-protocol for PoKRep to a Pedersen commitment $g^x h^\rho$ for $g$ and $h$ specified in the CRS. The extractor for this protocol will invoke the PoKRep extractor to open the commitment. The protocol works as follows:

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

Params: $(g, h) \in \mathbb{G}$, $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$; Inputs: $u, w \in \mathbb{G}$, $B > 2^{\lambda}|\mathbb{G}|$;
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 $r_x, r_\rho$ 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 8** (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 A. □

### 3.6 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). This can also be made zero-knowledge using the classic GQ protocol, which is a special case of a homomorphism pre-image Σ-protocol (i.e., the $x$th root of an element is a pre-image of the homomorphism $g \mapsto g^x$). We describe here a simple aggregation technique for providing a succinct proof of knowledge for multiple *coprime* roots $x_1, ..., x_n$ simultaneously. If the roots are all for the same element $\alpha$ then this is trivial: the witness is a root $\alpha^{1/x^*}$ where $x^* = x_1 \cdots x_n$. From this witness one can publicly extract the $x_i$th root of $\alpha$ for each $i$. However, this method does not generalize when the elements are distinct. 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:

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

The proof is simple: it is the product of witnesses, $w \leftarrow w_1 \cdots w_n$. We show that from this product and the public $x_i$'s and $\alpha_i$'s it is possible to extract an $x_i$th root of each $\alpha_i$. (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: $\mathbf{w} \in \mathbb{Z}^n$ 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** $\alpha$
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.*

*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 $x_i$th root of $\alpha_i$ for all $i$. For each $i$ and $j \neq i$ let $z_{ij} = x^*/(x_i x_j)$. For each $i$, let $A_j = \prod_{j \neq i} \alpha_i^{z_{ij}}$, 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 $x_i$th 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 $x_i$th root of $\alpha_i$ as $(u^a \alpha_i^b)^{x_i} = \alpha_i^{(ax^*/x_i)+bx_i} = \alpha_i$. □

**Non-interactive proofs** All of the protocols PoE,PoKE*,PoKE, PoKE2 and ZKPoKE can be made non-interactive using the Fiat-Shamir heuristic. It is these non-interactive, succinct, and efficiently verifiable proofs that are most useful for the applications discussed later in this paper. Figure 1 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 $\ell$th 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. This only works if there isn't a collision among the primes for distinct inputs, which happens with negligible probability.

# 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.

**NI-PoE**

$\{x, u, w : u^x = w\}$

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

$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(x, u, w)$

$q \leftarrow \lfloor x/\ell \rfloor$

$r \leftarrow x \bmod \ell$

$Q \leftarrow u^q$

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

$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(x, u, w)$

$r \leftarrow x \bmod \ell$

Check: $Q^\ell u^r = w$

**NI-PoKE2**

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

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

$g \leftarrow \mathsf{H}_{\mathbb{G}}(u, w), z = g^x$

$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(u, w, z), \alpha = \mathsf{H}(u, w, z, \ell)$

$q \leftarrow \lfloor x/\ell \rfloor, r \leftarrow x \bmod \ell$

$\pi \leftarrow \{z, (ug^\alpha)^q, r\}$

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

$\{z, Q, r\} \leftarrow \pi$

$g \leftarrow \mathsf{H}_{\mathbb{G}}(u, w)$

$\ell \leftarrow \mathsf{H}_{\mathsf{prime}}(u, w, z), \alpha \leftarrow \mathsf{H}(u, w, z, \ell)$

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

Figure 1: Non-interactive succinct proofs for hidden order groups.

$\lambda$: 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
$\mathsf{pp}$: Public parameters implicitly available to all methods
**upmsg**: Information used to update proofs
**Setup**$(\lambda, z) \to \mathsf{pp}, A_0$ Generate the public parameters
**Add**$(A_t, x) \to \{A_{t+1}, \textbf{upmsg}\}$ Update the accumulator
**Del**$(A_t, x) \to \{A_{t+1}, \textbf{upmsg}\}$ Delete a value from the accumulator
**MemWitCreate**$(A_t, S, x) \to w_x^t$ Create an membership proof
**NonMemWitCreate**$(A_t, S, x) \to u_x^t$ Create a non-membership proof
**MemWitUp**$(A_t, w_x^t, x, \textbf{upmsg}) \to w_x^{t+1}$ Update an membership proof
**NonMemWitUp**$(A_t, w_x^t, x, \textbf{upmsg}) \to u_x^{t+1}$ Update a non-membership proof
**VerMem**$(A_t, x, w_x^t) \to \{0, 1\}$ Verify membership proof
**VerNonMem**$(A_t, x, u_x^t) \to \{0, 1\}$ Verify non-membership proof

Figure 2: A trapdoorless universal accumulator.

Our presentation of a trapdoorless universal accumulator mostly follows the definitions and naming conventions of [BCD+17]. Figure 4 summarizes the accumulator syntax and list of associated operations. One notable difference in our syntax is the presence of a common reference string $\mathsf{pp}$ generated by the **Setup** algorithm in place of private/public keys.

**Correctness and security** We omit a correctness definition as it is natural and equivalent to previous schemes. It ensures that additions and deletions correspond to additions and deletions of items in the underlying set and that valid (non)membership witnesses can be created and updated at the appropriate times.

For our security definition we follow [Lip12], who formulates an *undeniability* security property. [DHS15] later showed that for universal accumulators this property implies the collision free property from [CL02]. 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 valid proofs $w_x^t$ and $u_x^t$ where $w_x^t$ shows that $x$ is in the accumulator and $w_x^t$ shows that it is not.

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

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

**ShamirTrick**$(\pi_1, \pi_2, x, y)$: [Sha83]

1. **if** $\pi_1^x \neq \pi_2^y$ **return** $\perp$
2. $a, b \leftarrow$ **Bezout**$(x, y)$
3. **return** $\pi_1^b \pi_2^a$

$\mathsf{H}_{\mathsf{prime}}(x)$:

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

Figure 3: Sub-procedures used in the accumulator construction.

### 4.1 Accumulator construction

Figure 4.1 summarizes several subprocedures that are used heavily in the construction. **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 $y$th root of $g$. **RootFactor** is a new 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.

**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 an RSA group, which may require a trusted setup to generate the modulus, or a class group which eliminates the trusted setup.

**The basic RSA accumulator.** Figure 4.1 presents the classic RSA accumulator [CL02, Lip12], where we omit all the procedures that require trapdoor information, such as the size of the group. All accumulated values are small odd primes. Arbitrary data values can be hashed to small primes, e.g. using the algorithm in Figure 4.1. It is also assumed that no item is added twice to the accumulator.

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.

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

**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**$(x, s^*)$
3. $d \leftarrow g^a$
4. **return** $u_x^t \leftarrow \{d, b\}$

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

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

Figure 4: The basic RSA accumulator

- 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 $(a, b) \leftarrow$ **Bezout**$(x, \prod_{s \in S} s)$ are therefore a valid membership witness. The actual witness is the pair $(g^a, b)$ which is short because $|b| \approx |x|$.
- Membership and non-membership witnesses can be efficiently updated as in [LLX07]

**Theorem 9** (Security accumulator [Lip12])**.** *Assume that the strong RSA assumption (Definition 2) holds in $\mathbb{G}_{RSA}$. Then the accumulator satisfies the deniability definition and is therefore secure.*

**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 discuss separately how this can be done efficiently for additions and deletions:

- **BatchAdd** Anyone who knows the current state $A_t$ can add an element $x$ to the accumulator and update its state appropriately to $A_{t+1} \leftarrow A_t^x$ without knowing any other information about the accumulated set. Furthermore, a network participant who sees $(A_t, A_{t+1}, x)$ can verify that the update was done correctly. 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 need only check that $x^* = \prod_i x_i$ and verify the proof rather than compute the $m$ exponentiations.

- **BatchDel** Deleting elements is not as simple, however we describe how it can be done given membership witnesses. The membership witness for an element $x$ is equivalent to the value of the accumulator without $x$. Therefore, updating the state of the accumulator to delete a single element given its member witness is trivial. A set of elements could be removed in sequence in this manner, where after each removal all other membership witnesses are updated using the standard witness update algorithm. A more efficient method is to use the **ShamirTrick** to a compute $A_t^{1/\prod_{i=1}^{n} x_n}$ given the set of tuples $\{(x_1, w_{x_1}^t), \ldots, (x_n, w_{x_n}^t)\}$ and an accumulator $A_t$ such that $(w_t^{x_i})^{x_i} = A_t$. This algorithm works as long as all the $x_i$ are co-prime. Finally, an NI-PoE proof improves the verification efficiency of this batch update.

**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** $A_{t+1}$, NI-PoE$(x^*, A_t, A_{t+1})$

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

**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** $A_{t+1}$, NI-PoE$(x^*, A_{t+1}, A_t)$

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

**Aggregating membership witnesses** Creating a single membership witness for many elements at once is straightforward given either the entire set or membership witnesses for each item (and using **ShamirTrick**). However, verification of this membership witness uses a linear in the number of group operations. We utilize the succinct proof of exponentiation (NI-PoE) for hidden order groups to produce a single membership witness for a set of elements that can be verified in constant time.

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 $y \in A_y$ to create an inclusion proof $\pi$. The verification checks $\pi^{x \cdot y} = A_1^y A_2^x$. If $x$ and $y$ are coprime[1] then this suffices to directly recover $w_x$ and $w_y$ from the witness. In particular $w_x =$ **ShamirTrick**$(A_1^y, A_1, \pi^y A_2^{-1}, y, x)$ and $w_y =$ **ShamirTrick**$(A_2^x, A_2, \pi^x A_1^{-1}, x, y)$. The witness aggregation methods are presented in Figure 4.1.

**Batching non-membership witnesses** A non-membership witness $u_x$ for $x$ in an accumulator with state $A_t$ for a set $S$ is $u_x = \{g^a, b\}$ such that $ax + b\prod_{s \in S} s = 1$. The verification checks $g^{ax} A_t^b = g$. Since $gcd(x, s) = 1 \wedge gcd(y, s) = 1 \leftrightarrow gcd(xy, s) = 1$, to batch non-membership witnesses we could simply construct an exclusion proof for $xy$. A prover computes $a', b' \leftarrow$ **Bezout**$(xy, \prod_{s \in S} s)$ and sets $u_{xy} \leftarrow g^{a'}, b'$. Unfortunately, $|a'| \approx |xy|$ so the size of the witness is therefore no smaller than giving non-membership witnesses for $x$ and $y$ separately. A natural idea is to set $u_{xy} = (v, d) \leftarrow (g^{a'}, A_t^{b'}) \in \mathbb{G}^2$ instead of $(g, b') \in \mathbb{G} \times \mathbb{Z}$ as the former has constant size. The verification

[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 7.2).

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

**MemWitCreate***$(A, x)$ :
1. $w_x \leftarrow$ **MemWitCreate**$(A, x)$
2. **return** $x, \mathsf{PoE}(x, w_x, A)$

**MemWitX**$(A_1, A_2, w_x, w_y, x, y)$ :
1. **return** $\pi \leftarrow w_x \cdot w_y$

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

Figure 5: Witness aggregation techniques. **MemWitX**,**VerMemWitX** are the prover and the verifier of the PoKCR Protocol

would check that $d^{xy}v = g$. This idea doesn't quite work as an adversary can simply set $v = gd^{-xy}$ without knowing a discrete logarithm between $A_t$ and $d$. Our solution is to use the NI-PoKE2 protocol to ensure that $v$ was created honestly. Intuitively, soundness is achieved because the knowledge extractor for the NI-PoKE2 can extract $b'$ such that $(v, b')$ is a standard non-membership witness for $xy$.

The new membership witness is $d, v, \pi \leftarrow \mathsf{NI\text{-}PoKE(A,v;b)}$. 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: $\mathsf{NI\text{-}PoE}(x \cdot y, d, g \cdot v^{-1})$. Lastly, recall from Section 3 that the two independent 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***$(A, x)$ : // ($|x|$ much greater than $\lambda$)
1. $a, b \leftarrow$ **Bezout**$(x, \prod_{s\in S} s)$
2. $d \leftarrow g^a, v \leftarrow A^b$
3. $\pi_d \leftarrow \mathsf{NI\text{-}PoKE2}(A, v; b), \pi_g \leftarrow \mathsf{NI\text{-}PoE}(x, d, g \cdot v^{-1})$
4. **return** $\{d, v, \pi_d, \pi_g\}$

**Soundness of batch non-membership witnesses** We will prove that the batch non-membership witnesses are sound, as they have a different structure from the standard non-membership witnesses. The security proof is against generic group adversaries. The generic group model subsumes the Strong-RSA assumption (i.e. Strong-RSA holds in the generic group model [DK02]). Our proof of security applies to the interactive forms of PoE and PoKE. The membership witness for an element $x$ and an accumulator $A$ therefore is $w_x$ and an interactive $\mathsf{PoE}(x, w_x, A)$. The non-membership witness $u_x$ consists of $d, v \in \mathbb{G}$ an interactive $\mathsf{PoKE}(A, v; b)$ and an interactive $\mathsf{PoE}(x, d, g \cdot v^{-1})$

**Theorem 10.** *The accumulator with batch non-membership witnesses satisfies the undeniability definition and is secure against generic efficient adversaries.*

*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 challenge $c \in \mathbb{G}_{\mathsf{RSA}}$. **Setup** sets $g \leftarrow c$. $\mathcal{A}_{Acc}$ outputs a tuple $(A, x, w_x, u_x)$ and performance an interactive PoKE. Using the efficient extractor from Theorem 4 we can with overwhelming probability extract a $b \in \mathbb{Z}$ such that $A^b = v$. By Corollary 1 and

Theorem 1 we have that PoE is overwhelmingly sound. We therefore have that $d^x A^b = g$ and $w_x^x = A$. As in Theorem 4.1, $d \cdot w_x^b$ is an $x$th root of $g$ and by definition $x$ is an odd prime. This contradicts the Strong RSA assumption (Definition 2) which holds against generic adversaries [DK02]. □

**Aggregating non-membership witnesses** Unlike for membership witnesses we do not have an efficient algorithm for aggregating non-membership witnesses without knowing the committed set $S$. The question is whether given $A$, $x_1, d_1, b_1$ and $x_2, d_2, b_2$ such that $d_1^{x_1} A^{b_1} = d_2^{x_2} A^{b_2} = g$ we can find $d_3, b_3$ such that $d_3^{x_1 \cdot x_2} A^{b_3} = g$. Aggregating non-membership witnesses has important applications to aggregating openings for our vector and hash-map commitment from Section 5.

**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.

**Multiset accumulator** The stateless accumulator assumes that no item is added twice. This is necessary because a stateless add algorithm cannot ensure that no duplicates are added. Usually a centralized accumulator manager is responsible for this. In our scheme there is no such entity. Despite this, there are several ways to circumvent this limitation. For some applications it may be impossible to add the same item twice. For example, in many cryptocurrencies each transaction refers to previous transaction outputs that are being spent. Creating a valid transaction that spends a previously unspent output would require creating a collision in a cryptographic hash function. In a similar generic transformation, we can use a collision resistant hash function $\mathsf{H}_{\mathsf{prime}}$ to ensure that each added item is unique. To do that the hash function hashes an element from an arbitrary domain and the current update counter $t$ to a large odd prime. Adding the same item twice, again requires finding a collision for the hash function. Unfortunately, this procedure would render exclusion proofs less useful as an exclusion proof can only be given for a fixed update point $t$. The final proposal is to explicitly allow adding the same item twice. The resulting accumulator is an accumulator for a mapping from items to a counter instead of a single set and 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 4.1. 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 $x^k$th 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.

# 5 Batchable Vector Commitments with Small Parameters

While cryptographic accumulators can be viewed as a commitment to a set, they are not (in general) position binding. That is, one cannot claim that the $i$th element of the set is a particular value. It is tempting to reason that elements could be inserted into the accumulator paired with a position, i.e. $(x, i)$ as the element $x$ at position $i$. However, the problem is that one could insert many elements $(x, i)$ consisting of different values $x$ paired with the same index $i$. A prover who claims that the accumulator commits to a unique value at the $i$th index would need to provide not only an inclusion proof of $(x, i)$ but also a "range" exclusion proof that no other $(x', i)$ is in accumulator any $x' \neq x$. Otherwise, the accumulator does not bind the prover to a specific value, and it can reveal any one of a (polynomial) number of values that it inserted.

A vector commitment (VC) [LY10, CF13, LRY16] is similar to an accumulator but is position binding. More precisely, it is a commitment scheme with a Commit procedure that outputs a compact commitment $\Phi$ to an ordered sequence of $m$ values $(x_1, ..., x_m)$ and an Open procedure that outputs a succinct opening $\pi$ at any given index, i.e. a proof that $x_i$ is the $i$th committed value. The sizes of $\Phi$ and $\pi$ are sublinear in $m$ (ideally an independent constant). A VC is *position binding*, which means that an adversary should not be able to open a commitment to two different values at the same position. Vector commitments have many applications including IOPs [BCS16] and proof-of-retrievability (PoR).

A Merkle tree is a simple example of a VC that is position binding. The size of a Merkle tree commitment is constant and the opening proof at a given index for a vector of length $m$ is $O(\log m)$. To be a bit more precise, the commitment is $\lambda$ bits and the opening is $\lambda \log_2 m$ where $\lambda$ is the security parameter for a collision resistant hash function. Catalano and Fiore [CF13] were the first to give constructions of VCs achieving constant size openings, one based on the RSA assumption and the other based on CDH in bilinear groups. These constructions were further improved upon in [LRY16]. Another highly advantageous property of these constructions over a Merkle tree is that the openings at several indices can be batched. This means that the prover can produce a constant size proof $\pi$ (i.e. the size of a proof for a single index) that a subset of $k \leq n$ indices open to $k$ particular values. While the communication between the prover and verifier may still be linear in $k$ if the prover needs to send the opened values, the communication is still greatly reduced due to the fact that the size of a proof is generally larger than the value at a specific index. Furthermore, in some applications the verifier may already have the values (e.g., proving that several VCs all open at certain indices to specific values held by the verifier).

However, unlike a Merkle tree, these VC constructions each involve large public parameters (linear in the length of the committed vector for RSA and quadratic for the CDH variant). This

requires the verifier of any commitment opening to either use at least linear storage or perform linear work to generate the parameters on the fly during verification. Time/space tradeoffs are possible as an optimization depending on the particular resource constraints of the verifier. While reasonable for applications with short vectors, this quickly becomes impractical as the vector length grows (e.g. a database application with a GB size vector). Naturally, this motivates the question as to whether there is a VC scheme with $O(1)$ openings and also $O(1)$ parameters. We present a new VC scheme that achieves this. Moreover, our new VC scheme also has batchable opening proofs that use a constant number of expensive group operations. The scheme is based on the classical RSA accumulator and uses several new techniques, including the succinct proofs for hidden order groups presented in Section 3.

Lastly, we show how our new VC scheme can be used to realize a key-value map commitment as a generalization of a vector commitment. A key-value map pairs with each key $k$ a unique value $v_k$. A vector of fixed length $N$ is a special case of a key-value map where the keys are the integers in $[0, N)$. While a standard vector commitment of exponential length could be used to (inefficiently) implement a key-value map, to the best of our knowledge our scheme is the first efficient construction.

### 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 $\lambda$, length $n$ of the vector, and message space of vector components $\mathcal{M}$, output public parameters $\mathsf{pp}$, 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 $\tau$.

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 $\tau$.

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 $\tau$ output an opening $\pi$ that proves $m$ is the $i$th 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 $\pi$ 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 $\mathsf{VC}$ is correct if for all $\boldsymbol{m} \in \mathcal{M}^n$ and $i \in [1, n]$:*

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

The correctness definition for dynamic vector commitments also incorporates updates. Concretely whenever $\mathsf{VC.Update}$ is invoked the underlying commited 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 $\mathsf{VC}$ is position binding if for all $O(\mathsf{poly}(\lambda))$-time adversaries $\mathcal{A}$ the probability over $pp \leftarrow \mathsf{VC.Setup}(\lambda, n, \mathcal{M})$ and $(com, i, m, m', \pi, \pi') \leftarrow \mathcal{A}(pp)$ 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 $\lambda$.*

## 5.2 VC construction

For the following section we will disambiguate accumulator and vector commitment methods that have the same name by writing $A$.**Setup** to signify an accumulator method and $\mathsf{VC}$.**Setup** to signify a vector commitment method. 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. We first provide an overview of the bit vector construction and the full details in Figure 5.2.

**Setup** $\mathsf{VC.Setup}(\lambda, \mathsf{n}, \{0, 1\})$ sets up an accumulator $\mathsf{acc}$ as described in Section 4. It also fixes a deterministic collision resistant (or free) function, $\mathsf{PrimeGen}$ from $[0, n]$ to odd primes. 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).[2] Using this function a $k$ bit input will map to a $k + 2\log(k)$ bit value. Yet another approach would map to large integers such that each integer has at least one unique prime factor.

Both the accumulator's CRS as well as $\mathsf{PrimeGen}$ can be represented in constant space independent of $n$. This means that the public parameters for the vector commitment are also independent of $n$, unlike the previous vector commitments with $O(1)$ size openings [CF13, LRY16].

**Commit** For any $i \geq 1$ let $p_i \leftarrow \mathsf{PrimeGen}(i)$. $\mathsf{VC.Com}(\mathsf{m})$ adds $p_i$ to $\mathsf{acc}$ for each $m_i \in \boldsymbol{m}$ where $m_i = 1$. Concretely the accumulator state after $\mathsf{VC.Com}$ is $A = g^{\prod_{i=1, m_i=1}^{n} p_i} \in \mathbb{G}_{\mathsf{RSA}}$ where $\mathbb{G}_{\mathsf{RSA}}$ is a hidden order group. Initializing a commitment is linear in the number of 1 bit's in $\boldsymbol{m}$, i.e. $\sum_{i=1}^{n} m_i$. It is, therefore, feasible to commit to a long (even exponentially long) vector if only a polynomial number of bit's in $\boldsymbol{m}$ are set. We use this fact in the key-value map construction in Section 5.4. The commitment can be updated later by adding/deleting $p_i$'s from the accumulator.

**Open & verify** In order to show that the $i$th bit of the VC is 1 we generate a membership witness for $p_i$ in $\mathsf{acc}$. If $m_i$ is 0 then we generate a non-membership witness $p_i$. The verification

[2] This map is collision free as long as there is alway a prime number between $f(i)$ and $f(i + 1)$ for all $i$. Note the difference $f(i + 1) - f(i) > 2\log(f(i))^2$. Asymptotically, Cramer's conjecture states that the distance between adjacent primes $p_n$ and $p_{n+1}$ converges to $\log(p_n)^2$, hence for sufficiently large $i$ the interval $f(i)$ to $f(i + 1)$ contains a prime. It can be verified experimentally that this is true from 0 to 100 million. Furthermore, for polynomial size $N$ this can be verified by brute force, and for very large $N$ one might as well use $\mathsf{H_{prime}}$ instead.

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

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

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** $A.\mathbf{Del}(p_i)$

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

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

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

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

Figure 6: Vector commitment scheme from accumulator with batchable membership and non-membership witnesses.

of an opening checks these non-membership/membership witnesses using the accumulator witness verification algorithms.

**Extension to arbitrary message space** To support arbitrary an message $\mathcal{M}$ it suffices to consider the message space $\{0,1\}^\lambda$ (for sufficiently large $\lambda$) and use a collision resistant hash function $H : \mathcal{M} \to \{0,1\}^\lambda$. The bit vector commitment can be generalized to $\{0,1\}^\lambda$ by associating primes $\{p_j : j \in [i\lambda, (i+1)\lambda)$ with the $i$th index of the vector. To set the $i$th position to a $\lambda$-bit message $m = m_1 | \cdots | m_\lambda$ the primes $p(j)$ are added for each $j$ where $m_j = 1$. On an update (where the current value at the $i$th position is non-zero) then it is also necessary to delete the primes $p(j')$ for each $j'$ where $m_{j'} = 0$. The opening of a message at the $i$th index could provide membership witnesses for all the $p_j$ and non-membership witnesses for all the $p_{j'}$. However, if done naively in this way then the opening of a $\lambda$-bit value is a factor $\lambda$ larger than the bit vector opening proofs, and considerably larger than Merkle openings for $\lambda > \log(n)$. This is where the batching of membership witnesses come into play, and in particular, the new features of our accumulators from Section 4. In classical RSA accumulators the membership witness for an integer product of primes is independent from the size of the integer, but the non-membership witness grows linearly.

To open the $i$th index to a message $m \in \{0,1\}^\lambda$, we first set $p_i^{\intercal}$ to the product of all $p_{\lambda i + j}$ where $m_j = 1$ and $p_i^{\perp}$ to the product of all $p_{\lambda i + j'}$ where $m_{j'} = 0$ and then compute a membership witness $\pi_I$ for $p_i^{\intercal}$ using $A.\mathbf{MemWitCreate}(p_i)$ and a non-membership witness $\pi_E$ for $p_i^{\perp}$ using $A.\mathbf{NonMemWitCreate*}(p_i)$. Both of these witnesses are constant size, independent of $|\mathbf{m}|$, and take only a constant number of group operations to verify. Concretely $\pi_I$ consists of 2 elements in $\mathbb{G}_{\mathsf{RSA}}$ and $\pi_E$ of 5 elements in $\mathbb{G}_{\mathsf{RSA}}$ and 1 in $\mathbb{Z}_{2^\lambda}$. In fact, we can use the same proofs to

prove inclusion for a whole vector. The verification time scales linear only in the number of $\mathbb{Z}_{2^\lambda}$ operations.

**Optimization** The number of group elements can be reduced by utilizing a PoKCR for all of the PoE and PoKE roots. 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 $4 \in \mathbb{G}_{\mathsf{RSA}}$ and 1 in $\mathbb{Z}_{2^\lambda}$

## 5.3 Comparison

Table 5.3 compares the performance of our new VC scheme, the Catalone-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 $\lambda$. We note that the **MultiExp** algorithm from Section 3.6 also applies to the CF scheme. In particular it can improve the $\mathsf{Setup}$ and $\mathsf{Open}$ time. The comparison reflects these improvements.

| Metric | This Work | Catalono-Fiore [CF13] | Merkle Tree |
|---|---|---|---|
| | **Setup** | | |
| $\mathsf{Setup}$ | O(1) | $O(n \cdot \log(n) \cdot \lambda)$ $\mathbb{G}_{\mathsf{RSA}}$ | O(1) |
| $\lvert\mathsf{pp}\rvert$ | O(1) | O(n) $\mathbb{G}_{\mathsf{RSA}}$ | O(1) |
| $\mathsf{Com}(\mathbf{m}) \to c$, $\lvert\mathbf{m}\rvert = n$ | $O(n \cdot \log(n) \cdot k)$ $\mathbb{G}_{\mathsf{RSA}}$ | $O(n \cdot k)$ $\mathbb{G}_{\mathsf{RSA}}$ | $O(n)$ $\mathsf{H}$ |
| $\lvert\mathbf{c}\rvert$ | 1 $\mathbb{G}_{\mathsf{RSA}}$ | 1 $\mathbb{G}_{\mathsf{RSA}}$ | 1 $\lvert\mathsf{H}\rvert$ |
| | **Proofs** | | |
| $\mathsf{Open}(m, i) \to \pi$ | $O(n \log(n) \cdot k)$ $\mathbb{G}_{\mathsf{RSA}}$ | $O(n \cdot (k + \lambda))$ $\mathbb{G}_{\mathsf{RSA}}$. | $O(\log(n))$ $\mathsf{H}$ |
| $\mathsf{Verify}(m, i, \pi)$ | $O(\lambda)$ $\mathbb{G}_{\mathsf{RSA}}$+ $\log(n) \cdot k$ $\mathbb{Z}_{2^\lambda}$ | $O(k + \lambda)$ $\mathbb{G}_{\mathsf{RSA}}$ | $O(\log(n))$ $\mathsf{H}$ |
| $\lvert\pi\rvert$ | $O(1)$ $\lvert\mathbb{G}_{\mathsf{RSA}}\rvert$ | 1 $\lvert\mathbb{G}_{\mathsf{RSA}}\rvert$ | $O(\log(n))$ $\lvert\mathsf{H}\rvert$ |
| $\mathsf{Open}(\mathbf{m}, i)$, $\lvert\mathbf{m}\rvert = t$ | $O(n(\log(n) \cdot k))$ $\mathbb{G}_{\mathsf{RSA}}$ | $O(n \log(n) \cdot (k + \lambda))$ $\mathbb{G}_{\mathsf{RSA}}$ | $O(t \cdot \log(n))$ $\mathsf{H}$ |
| $\mathsf{Verify}(\mathbf{m}, i, \pi_{\mathbf{m}})$ | $O(\lambda)$ $\mathbb{G}_{\mathsf{RSA}}$ + $O(t \log(n)k)$ $\mathbb{Z}_{2^\lambda}$ | $O(tk)$ $\mathbb{G}_{\mathsf{RSA}}$ | $O(t \cdot \log(n))$ $\mathsf{H}$ |
| $\lvert\pi_{\mathbf{m}}\rvert$ | 4 $\lvert\mathbb{G}_{\mathsf{RSA}}\rvert$+ 1 $\lvert\mathbb{Z}_{2^\lambda}\rvert$ | 1 $\lvert\mathbb{G}_{\mathsf{RSA}}\rvert$ | $O(t \cdot \log(n))$ $\lvert\mathsf{H}\rvert$ |

Table 1: Comparison between Merkle trees, Catalone-Fiore RSA VCs and the new VCs presented in this work. $|\mathbb{G}_{\mathsf{RSA}}|, |\mathbb{Z}_{2^\lambda}|, |\mathsf{H}|$ are the size of a hidden order group element, a $\lambda$ bit field element and a hash. $\mathbb{G}_{\mathsf{RSA}}$ is a group operation,$\mathbb{Z}_{2^\lambda}$ a multiplication in a field of size roughly $2^\lambda$ and $\mathsf{H}$ a hash operation. Group operations are generally far more expensive than hashes which are more expensive than multiplication in $\mathbb{Z}_{2^\lambda}$

## 5.4 Key-Value Map Commitment

An accumulator is a commitment to a set. A vector-commitment is a commitment to a positional vector. We will now show how we can use the vector-commitment to build a commitment to a key-value map. A key-value map is an associative data structure where elements from a key space $\mathcal{K}$ are mapped to a value space $\mathcal{V} \cup \{\perp\}$. We say that a key $k \in \mathcal{K}$ is in the map if it does not map to $\perp$. 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. Note that if the key-space is large then we need a *sparse* vector. A sparse vector is a vector whose complexity is only dependent on the number of non-zero elements in it. A sparse vector commitment has the same property with

respect to the number of elements that were committed. We can use a sparse vector commitment and two collision resistant hash function $\mathsf{H}_{\mathcal{K}}, \mathsf{H}_{\mathcal{V}}$. $\mathsf{H}_{\mathcal{K}}$ maps from $\mathcal{K}$ to $[0, 2^\lambda]$ and $\mathsf{H}_{\mathcal{V}}$ from $\mathcal{V}$ to $\{0,1\}^\lambda$. We set up a vector commitment $\mathsf{VC}$ with message space $\{0,1\}^\lambda$ and length $2^\lambda$. In order to add a mapping from a key $k$ to a value $v$ we $\mathsf{update}$ the vector commitment at position $\mathsf{H}_{\mathcal{K}}(k)$ and set the value to $\mathsf{H}_{\mathcal{V}}(v)$. $\perp$ is represented by 0. In order to prove that the committed map contains a commitment $k \to v$ we open the vector at the position $\mathsf{H}_{\mathcal{K}}(k)$ to the value $\mathsf{H}_{\mathcal{V}}(v)$. The key-value map inherits its properties from the vector commitment. In particular, it has batch openings with efficient verification and supports distributed updates.

# 6 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 (See Figure 1). In Figure 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 $\mathsf{H}$ with domain $\mathbb{Z}_{2^\lambda}$. The hash function iteratively hashes a message and a counter until the output is a prime. If we model $\mathsf{H}$ 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 context: 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 $\ell$ is generated by hashing the previous transcript to a prime. The protocol can be modified by having the prover provide a nonce such that $H(nonce||transcript) = \ell$ with $\ell \in \mathsf{Primes}(\lambda)$. While this allows an adversary to produce different challenges 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 $\lambda$. 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 $nonce||element$ is accumulated instead of just the element. This of course allows an adversary to accumulate the same element twice but this can be prevented by additionally hashing in the current state of the accumulator. Also in some applications, such as stateless blockchains

it is guaranteed that no element is accumulated twice(see Section 7). 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 $\mathsf{H}(nonce||element)$ 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 $\lambda$ 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 7.2, 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.

# 7 Applications

## 7.1 Stateless Blockchains

Camenisch and Lysanskia [CL02] originally considered the application of accumulators to distributed databases. In the application a database manager would store and update a database and

users storing only the short accumulator value could verifiably convince other users that an item was stored in the database. We consider a similar application of a blockchain acting as a decentralized ledger. In a blockchain users submit transactions and miners validate and aggregate the transactions in a block. The block is then appended to the ledger. Decentralized and permissionless blockchains such as Bitcoin allow any user to be a miner and employ a consensus mechanism to agree on which blocks are valid and will be added to the chain. In this setting, having a trusted database manager is not a reasonable assumption.

**UTXO commitments** We first consider a simplified blockchain design which closely corresponds to Bitcoin's UTXO[3] 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 ([MGGR13, BCG+14] are notable exception) the miners store the whole UTXO set and use it to verify whether a coin was unspent. This design allows users to issue transactions without knowing the whole UTXO set. Several authors and members of the blockchain community have proposed committing to the UTXO set in every block [TMA13, Tod16] using a Merkle tree based construction. This has several advantages: A user can efficiently proof to another user that his coins are unspent or that a certain transaction has not yet happened. Additionally, Todd [Tod16] proposed using the UTXO commitment to prove that a certain transaction output is indeed unspent. This has the advantage that the miners only need to store the commitment to the UTXOs not the UTXOs themselves. Todd proposes this mostly for old transaction outputs as "the $\log_2(n)$ bandwidth overhead per transaction is substantial". He further remarks that checking that the commitment has been properly updated incurs a significant overhead.
We propose to replace the Merkle tree based accumulator with our decentralized accumulator which will resolve many of the problems of Merkle based UTXO commitments. The basic design is equivalent to the one proposed by [TMA13, Tod16, Dra]. Each block contains an accumulator which represents the current state of the blockchain, i.e. the UTXO set. When a user wants to spend a coin, or more generally change the state, she provides a membership witness for the coin (UTXO) that is being spent. A miner can verify the transactions and use **BatchDel** to properly delete all spent coins from the accumulator. **BatchDel** additionally produces a proof of correctness for the deletions. The miner can, therefore, throw away all of the membership witnesses. For the newly produced and minted coins the miner uses **BatchAdd** to add them to the accumulator and produce a second NI-PoE proof of correctness. Neither of these operations require the miner to have any state beyond the accumulator value. Furthermore, other miners can verify that the accumulator was updated correctly using only a constant number of group operations and highly efficient arithmetic in $\mathbb{Z}_\ell$. Users will need to store membership witnesses for their own coins. No further storage is required. This accumulator based blockchain design therefore does not require even a single participant to the store the whole state of the ledger. It also enables highly efficient verification of transaction inclusion[4].

Unfortunately, the design requires that user update their witnesses for every addition or deletion to the set. The impossibility result of [CH10] suggests that this is unavoidable. We, therefore,

[3] Unspent Transaction Outputs

[4] The signature verification for each transaction is orthogonal. Recent advances [BDN18] have dramatically reduced the signature size and verification time.

envision that some users will use services[5] that provide them with inclusion proofs. These services are not trusted for security, but only for availability. A service stores the whole UTXO set and can compute membership witnesses from scratch. A service may need to produce many membership witnesses at once. It can use the **CreateAllMemWit** algorithm to produce all membership witnesses in just $O(n \log(n))$ time.

**Account based state commitments** The accumulator based state commitment we described works for transactional currencies where the state can be represented as a set. Some currencies such as Ethereum [Woo14] or Stellar [SYB14] use an account based system where the state is a key-value map. Each account has some state, such as the amount of currency that is associated with an account. Ethereum actually does use a Merkle tree based state commitment but currently does not require users to provide inclusion proofs.
A transaction in a state based currency updates the state of the sending and the receiving accounts. For stateless verification a user would have to provide proofs of the current state of the sending and receiving accounts. While an accumulator does not suffice to represent the state, we can use our new batchable hash-map commitment from Section 5.4 to represent the state. The commitment has similar batching properties as the accumulator. The one notable difference is that we do not have an efficient method for aggregating opening proofs. This, however, only applies in the stateless setting. Some aggregation nodes that do store the full state can verifiably aggregate openings. This service could be provided by the same service providers that generate and maintain opening proofs.

## 7.2 Short IOPs

Micali [Mic94] showed how PCPs could be used to construct succinct non-interactive arguments. 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. [LM18] recently proposed using the [CF13] vector commitment as a replacement for the Merkle tree. The advantage is that the size of the inclusion proofs is independent of the length of the underlying PCP and the number of positions queried. [LM18] claim that their scheme has both succinct public parameters and efficient verification. This, however, seems to be a mistake as [CF13] either requires linear sized public parameters (in the length of the committed vector) or has linear verification time. Our new vector commitment can be used as a replacement to achieve both of these properties.

[BCS16] generalized PCPs and interactive proofs to interactive oracle proofs. In an IOP the prover sends oracles proofs to a verifier which responds with challenges. The verifier then queries the oracle for small subsets of the proof and afterwards accepts or rejects. [BCS16] show that if the oracle is replaced with a Merkle tree commitment and the verifier is public coin than an IOP can be transformed into a short non-interactive proof of knowledge that is secure in the random oracle model. For every oracle query to the proof oracles the prover provides a Merkle inclusion path which ensures that the returned value was indeed a particular part of the proof. The Merkle inclusion proofs results in a $O(\log(n))$ multiplicative overhead vs. the information theoretic setting. [BCS16] build on top of [Val08]'s extractor which turns Micali's CS proofs into proofs of knowledge. More generally, vector commitments have similar extraction properties. We therefore propose generalizing the IOP transformation to work with other secure vector commitments such as

[5] These services are sometimes referred to as bridge nodes.

the one presented in Section 5. In every round the prover sends a commitment to a vector and then verifiably opens certain elements of the vector. The advantage for using the vector commitment over Merkle trees is that a single constant size inclusion proof suffices for an arbitrary number of queried elements. Further, using the $\mathsf{PoE}$ protocol the inclusion checks can essentially be reduced to multiplication in a $\lambda$-bit prime field. The inclusion proofs are therefore significantly smaller and more efficient to verify than the Merkle inclusion proofs.

There are several different proofs in the IOP Model. These involve classical zero-knowledge proofs such as Ligero [AHIV17], STARKs [BSBHR18] and Aurora [BSCR$^+$18] but also proofs of space [DFKP15, RD16, Pie18a, Fis18]. The proof size for all of these protocols would be reduced by at least a factor of $\log(n)$. However, since for most protocols, such as STARKs or Aurora, the verifier makes $O(\lambda)$ queries per oracle/vector commitment additional space savings can be gained by aggregating the inclusion proofs. The prover will simply send the responses to the $O(\lambda)$ queries plus a constant sized inclusion proof. Additionally, the prover sends the vector commitment in each round which is also of constant size. As we showed in Section 5 it is possible to aggregate vector commitment openings over $k$ different vector commitments such that only $k+1$ elements in $\mathbb{G}_{\mathsf{RSA}}$ and $k$ elements in $\mathbb{Z}_{2^\lambda}$ need to be send. This is independent of how many elements per vector are opened. Using class groups with 1000 bit discriminants the inclusion proofs and vector commitment would only take up 265 bytes per round of the IOP. The improvements are therefore not only asymptotic but also practically relevant. For a circuit of size $2^{20}$ [BSCR$^+$18] report that the Merkle paths take up 154kb of the 222kb proof. Using our vector commitment, the proof size would go down to less than 73kb. For STARKs the benefits are equally significant. For a benchmark circuit of $2^{52}$ gates [BSBHR18] the Merkle paths make up over 400kb of the 600kb proof and the inclusion checks are roughly 80% of the verification time. Note, that current IOPs optimize the proofs to open as few Merkle paths as possible. Our construction potentially opens the development of IOPs with different tradeoffs.

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. Using a Merkle tree with a sufficiently large hash function on the other hand is at least plausibly quantum secure. Instantiating a Vector commitment with similar properties as ours, using plausibly quantum-secure assumptions, remains an interesting open problem. Additionally, constructing a Merkle tree and Merkle inclusion proofs is much faster than generating a vector commitment of equivalent size. The prover for an IOP instantiated with our vector commitment would, therefore, be significantly slower.

# 8 Conclusion

In this paper we discussed new batching techniques for accumulators and how these techniques can be used to build sparse vector commitments, IOPs and succinct and stateless blockchains. As part of our batching techniques we develop new succinct zero-knowledge arguments for groups of unknown order that are of independent interest.

We expect that our techniques and commitments will have more applications beyond what was discussed. Several interesting open questions remain: Is it possible to non-interactively aggregate non-membership witnesses? Can one build an accumulator with constant sized witnesses from a quantum resistant assumption? Additionally, we hope that this research will motivate further study of class groups as a group of unknown order.

## Acknowledgments

This work was partially supported by NSF, ONR, the Simons Foundation, and a Google faculty fellowship.

## 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`.

[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.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. Cryptology ePrint Archive, Report 2018/483, 2018. `https://eprint.iacr.org/2018/483`.

[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.

[BSBHR18] 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`.

[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.

[CH10] Philippe Camacho and Alejandro Hevia. On the impossibility of batch update for cryptographic accumulators. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 178–188. Springer, Heidelberg, August 2010.

[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`.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, August 2015.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 127–144. Springer, Heidelberg, April 2015.

[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`.

[Fis18] Ben Fisch. Tight proofs of space and replication. Cryptology ePrint Archive, Report 2018/702, 2018. `https://eprint.iacr.org/2018/702`.

[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.

[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.

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

[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.

[Pie18a] Krzysztof Pietrzak. Proofs of catalytic space. Cryptology ePrint Archive, Report 2018/194, 2018. `https://eprint.iacr.org/2018/194`.

[Pie18b] Krzysztof Pietrzak. Simple verifiable delay functions. Cryptology ePrint Archive, Report 2018/627, 2018. `https://eprint.iacr.org/2018/627`.

[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.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 262–285. Springer, Heidelberg, October / November 2016.

[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.

[STS99] 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.

[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.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

[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 Security Proofs

## A.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 $\epsilon$. This $\mathcal{A}$ immediately gives a strong RSA adversary that also succeeds with probability $\epsilon$: run $\mathcal{A}$ to get $k$ and $g$ such that $g^k = 1 \in \mathbb{G}_{\mathsf{RSA}}$. 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 $\lambda$. 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 $(\ell_1, Q_1, r_1)$ and $(\ell_2, Q_2, r_2)$ two accepting transcripts for* Protocol PoKE$^*$ *generated one after the other. If $1/A$ and $1/|B-A|$ are negligible in $\lambda$, then with overwhelming probability there exist integers $\alpha$ and $\beta$ 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 $\ell_1$ and $\ell_2$ 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 $\alpha_i$ is chosen before $\ell_2$ 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 $\lambda$, 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 $\alpha_j$ and $\beta_j$ 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 $\lambda$. 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 $\lambda$.*

*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 $\ell$ must divide $\beta_i$. However, $\beta_i$ is chosen before $\ell$ and if $\mathcal{A}$ makes $q_2$ generic group queries then $\beta_i \leq 2^{q_2}$. The probability that $\ell$ divides $\beta_i$, 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 $\ell$ 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 $\lambda$ and $q_1, q_2$ are bounded by some polynomial in $\lambda$. □

**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 $\lambda$. 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 $\lambda$.*

*Proof.* We can construct an adaptive root adversary that first uses $\mathcal{A}$ to obtain $h$ and $d$, and then computes the $\ell$th 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 $x$.*

## A.2 Proofs of the main theorems

**Proof of Theorem 5.**

> Protocol PoKRep is an argument of knowledge for the relation $\mathcal{R}_\phi$ where $\phi := \mathit{Rep}$, 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 $\mathsf{Ext}$ such that for all $\mathcal{A}_0$ the following holds: if $\mathcal{A}_1$ convinces $V(\mathbf{g}, w)$ to accept with probability $\epsilon \geq 1/\text{poly}(\lambda)$, then $\mathsf{Ext}$ 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 $(w, \mathsf{state})$, 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 $j$th 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 $\ell_2$ is sampled randomly from an exponentially large set of primes independently from $\boldsymbol{r}_1, \boldsymbol{r}_3, \ell_1$, and $\ell_3$ (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 5 we describe the extractor $\mathsf{Ext}$:

1. run $\mathcal{A}_0$ to get output $(w, \mathsf{state})$
2. let $R \leftarrow \{\}$
3. run Protocol PoKRep with $\mathcal{A}_1$ on input $(w, \mathsf{state})$, 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 $\mathsf{Ext}$ 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 $\epsilon \geq 1/\text{poly}(\lambda)$, in expectation the extractor obtains $|R| > q$ accepting transcripts for independent primes $\ell_i$ after $q \cdot \text{poly}(\lambda)$ rounds. Hence, $\mathsf{Ext}$ outputs a vector $\boldsymbol{x}$ such that $Rep(\boldsymbol{x}) = w$ in expected polynomial time, as required. □

**Proof of Theorem 4.**

> 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 $\mathsf{Ext}$ such that for all $\mathcal{A}_0$ the following holds: if $\Pr\left[\langle \mathsf{V}(g, u, w), \mathcal{A}_1(g, u, w, \mathsf{state})\rangle = 1\right]$ (i.e. $\mathcal{A}_1$ convinces $\mathsf{V}$ to accept on these inputs) then $\mathsf{Ext}$ 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 5 there exists $\mathsf{Ext}^*$ for $\mathcal{A}_1$ that outputs $x^* \in \mathbb{Z}$ such that $g^{(x^*)} = z$. Furthermore, as already shown in the analysis of Theorem 5, once $\mathsf{Ext}^*$ 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 $\mathsf{Ext}$ will simply run $\mathsf{Ext}^*$ to obtain $x^*$. Now we will show that either $u^{x^*} = w$, i.e. $\mathsf{Ext}^*$ 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 $\mathsf{Ext}^*$ 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 $\ell$ 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 $\ell$th 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 $\mathsf{Ext}$ satisfies $w = u^{x^*}$ with overwhelming probability.

**Proof for protocol PoKE2** Showing that Protocol PoKE2 requires a fresh argument (similar to the analysis in Theorem 5) since the protocol no longer directly contains Protocol PoKE* as a subprotocol. $\mathsf{Ext}$ first obtains $u, w$ from $\mathcal{A}_0$ and runs the first two steps of Protocol PoKE2 with $\mathcal{A}_1$ playing the role of the verifier, sampling $g \xleftarrow{\$} \mathbb{G}$ and receiving $z \in \mathbb{G}$ from $\mathcal{A}_1$. $\mathsf{Ext}$ is a simple modification of the extractor for Protocol PoKE:

1. Set $R \leftarrow \{\}$ and sample $\alpha \xleftarrow{\$} [0, 2^{\lambda}]$.
2. Sample $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ and send $\alpha, \ell$ to $\mathcal{A}_1$.
3. Obtain output $Q, r$ from $\mathcal{A}_0$. If $Q^{\ell} u^r g^{\alpha r} = w z^{\alpha}$ (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 $\ell$ each time it rewinds the adversary but keeps the challenge $\alpha$ fixed each time. Since these are independently sampled in the real protocol, keeping $\alpha$ 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 $\mathsf{state}$ to $\mathcal{A}_1$. If $Q^\ell u^r g^{\alpha r} = w z^\alpha$, 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 $\ell'$ 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 $\alpha$ 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 $\ell'$ was sampled, hence there is a negligible probability that it has $\ell'$ 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 $\ell$ sampled, as long as $Q^\ell u^r g^{\alpha r} = w z^\alpha$ then with overwhelming probability:

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

Finally, if $u^\zeta \neq w$ then $g^\zeta/z \neq 1$ and yet $(g^\zeta/z)^\alpha = u^\zeta/w$. Since $\alpha$ is sampled independently from $u, w, g$, and $\zeta$, this relation can only hold true with non-negligible probability over the choice of $\alpha$ if both $g^\zeta/z$ and $u^\zeta/w$ are elements of a small (i.e. $\mathsf{poly}(\lambda)$ size) subgroup generated by $g^\zeta/z$. In other words, $g^\zeta/z$ 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 $(\ell_i, Q_i, r_i)$ 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 5. 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 $\mathsf{Ext}$ would succeed in extracting $\zeta$ with overwhelming probability.

□

**Proof of Theorem 6.**

> For any homomorphism $\phi : \mathbb{Z}^n \to \mathbb{G}$, $\mathsf{Protocol\ PoKHP}$ 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 4 for $\mathsf{Protocol\ PoKE}$. 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 $\mathsf{Protocol\ PoKHP}$ with the verifier $V(\mathbf{g}, w)$. We need to show that for all $\mathcal{A}_1$ there exists a poly-time $\mathsf{Ext}$ 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 $\mathsf{Ext}$ 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 5 there exists $\mathsf{Ext}^*$ for each $\mathcal{A}_1$ that outputs $\boldsymbol{x}^*$ such that $Rep(\boldsymbol{x}^*) = z$. Furthermore, as shown in the analysis of Theorem 5, once $\mathsf{Ext}^*$ 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 $\mathsf{Ext}^*$ to obtain $\boldsymbol{x}^*$. Now we will show that either $\phi(\boldsymbol{x}^*) = w$, i.e. $\mathsf{Ext}^*$ 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 $\mathsf{Ext}^*$ 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 $\ell$ 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 $\phi$ is a group homomorphism we can show that this is an $\ell$th root of $h$:

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

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 7.**

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

To show that the protocol is honest-verifier zero-knowledge we build a simulator Sim. Sim picks a random $\boldsymbol{s}^* \in [0, B]^n, c \in [1, 2^\lambda]$ and computes $A = \prod_{i=1}^n g_i^{s_i^*} w^{-c}$. Sim then samples a random prime $\ell$ and computes $Q^*$ and $\boldsymbol{r}^*$ from $\boldsymbol{s}^*$ such that $q_i^* \ell + r_i^* = s_i^*$ for each $i$. The values $c, \ell$ are distributed identically to the verifier's challenges in the real protocol transcript and $Q^*, \boldsymbol{r}^*$ are computed from $\boldsymbol{s}^*$ in the same way that $Q, \boldsymbol{r}$ are computed in the real transcript. It remains to show that the distribution of $\boldsymbol{s}^*$ is statically indistinguishable from $\boldsymbol{s}$ in the real protocol. In the real protocol transcript, each $i$th component is $s_i = r_i + cx$ where $c$ and $r_i$ are uniformly distributed over $[0, B]$. For each value of $c$, $s_i$ is distributed uniformly in $[cx, cx + B]$ and thus has a statistical distance of $2cx/B$ from $s_i^*$, which is negligible because $B > 2^{2\lambda} |\mathbb{G}|$ and hence $2cx/B < 2^{-\lambda+1}$. It follows that the statistical distance between $\boldsymbol{s}$ and $\boldsymbol{s}^*$ is negligible in $\lambda$. The simulation therefore produces statistically indistinguishable transcripts and ZKPoKRep is statistically honest-verifier zero-knowledge.

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* 5 to extract $\boldsymbol{s}$ and $\boldsymbol{s}'$ such that $\prod_{i=1}^n g_i^{s_i} = A w^c$ and $\prod_{i=1}^n g_i^{s_i'} = A w^{c'}$. 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 $\mu / w$ is an element of order $\Delta c > 1$. As $\Delta c$ 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 8.**

> 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. Sim samples random challenges $c \xleftarrow{\$} [0, 2^{\lambda}]$ and $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ as well as $s_1, s_2 \in [0, B]$. Sim then computes $A_g = g^{s_1} h^{s_2} z^{-c}$ and $A_u = u^{s_1} w^{-c}$. $Q_g, Q_u, r_1, r_2$ are computed from $s_1, s_2$ as in the real protocol. As shown in the analysis of Theorem 7, due to the fact that $cx/B < 2^{-\lambda}$ the distribution of $s_1, s_2$ is statistically close to the distribution of $s_1, s_2$ in the real protocol transcript, with distance negligible in $\lambda$. The simulation therefore produces valid, statistically indistinguishable transcripts.

For extraction, note that the protocol contains Protocol ZKPoKRep as a subprotocol on input $A_g$ and bases $g, h$ in the CRS, and therefore we can use the ZKPoKReP and PoKRep extractors to extract $x, \rho$ 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 $r_1, \ell$ 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 $\ell$th 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 7 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 $\Delta c$, 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.

□