Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains

Dan Boneh, Benedikt Bünz, Ben Fisch

Stanford University

Abstract. We present batching techniques for cryptographic accumulators and vector commitments in groups of unknown order. Our techniques are tailored for distributed settings where no trusted accumulator manager exists and updates to the accumulator are processed in batches. We develop techniques for non-interactively aggregating membership proofs that can be verified with a constant number of group operations. We also provide a constant sized batch non-membership proof for a large number of elements. These proofs can be used to build the first positional vector commitment (VC) with constant sized openings and constant sized public parameters. As a core building block for our batching techniques we develop several succinct proof systems in groups of unknown order. These extend a recent construction of a succinct proof of correct exponentiation, and include a succinct proof of knowledge of an integer discrete logarithm between two group elements. We circumvent an impossibility result for Sigma-protocols in these groups by using a short trapdoor-free CRS. We use these new accumulator and vector commitment constructions to design a stateless blockchain, where nodes only need a constant amount of storage in order to participate in consensus. Further, we show how to use these techniques to reduce the size of IOP instantiations, such as STARKs. The full version of the paper is available online [BBF18b]

1 Introduction

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

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

The typical way in which an accumulator or VC is used is as a communication-efficient authenticated data structure (ADS) for a remotely stored database where users can retrieve individual items along with their membership proofs in the data structure. Accumulators have been used for many applications within this realm, including accountable certificate management [BLL00,NN98], timestamping [Bd94], group signatures and anonymous credentials [CL02], computations on authenticated data [ABC$^+$12], anonymous e-cash [STS99b,MGGR13], privacy-preserving data outsourcing [Sla12], updatable signatures [PS14,CJ10], and decentralized bulletin boards [FVY14,GGM14].

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

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

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

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

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

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

This technique has been generalized to the broader class of *interactive oracle proofs* (IOPs)[BCS16]. In an IOP the prover sends multiple proof oracles to a verifier. The verifier uses these oracles to query a small subsets of the proof, and afterwards accepts or rejects the proof. If the proof oracle is instantiated with a Merkle tree commitment and the verifier is public coin, then an IOP can be compiled into a non-interactive proof secure in

the random oracle model [BCS16]. In particular, this compiler is used to build short non-interactive (zero-knowledge) proof of knowledge with a quasilinear prover and polylogarithmic verifier. Recent practical instantiations of proof systems from IOPs include Ligero [AHIV17], STARKs [BBHR18], and Aurora [BSCR$^+$18].

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

1.1 Summary of contributions

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

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

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

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

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

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

1.2 Overview of techniques

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

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

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

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

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

Batching non-membership proofs. We next show how our techniques can be amplified to create a succinct and efficiently verifiable batch membership and batch non-membership proofs. We then use these batch proofs to create the first vector commitment construction with constant sized

batch openings (recently called subvector commitments [LM18]) and $O(1)$ setup. This improves on previous work [CF13,LRY16] which required superlinear setup time and linear public parameter size. It also improves on Merkle tree constructions which have logarithmic sized non-batchable openings. The efficient setup also allows us to create sparse vector commitments which can be used as a key-value map commitment.

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

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

2 Preliminaries

Notation.

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

2.1 Assumptions

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

Definition 1. *We say that the* ***adaptive root assumption*** *holds for GGen if there is no efficient adversary* $(\mathcal{A}_0, \mathcal{A}_1)$ *that succeeds in the following task. First,* $\mathcal{A}_0$ *outputs an element* $w \in \mathbb{G}$ *and some* state*. Then, a random prime* ℓ *in* $\mathsf{Primes}(\lambda)$ *is chosen and* $\mathcal{A}_1(\ell, \mathsf{state})$ *outputs* $w^{1/\ell} \in \mathbb{G}$*. More precisely, for all efficient* $(\mathcal{A}_0, \mathcal{A}_1)$*:*

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

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

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

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

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

2.2 Generic group model for groups of unknown order

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

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

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

2.3 Argument systems

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

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

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

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* Pgen is run, and $\mathcal{A}_1$ runs the interactive protocol with the verifier using a state output by A_0. In our soundness definition the adversary $\mathcal{A}_1$ succeeds if he can make the verifier accept when no witness for x exists. For the stronger *argument of knowledge* definition we require that an extractor with access to $\mathcal{A}_1$'s internal state can extract a valid witness whenever $\mathcal{A}_1$ is convincing. We model this by enabling the extractor to rewind $\mathcal{A}_1$ and reinitialize the verifier's randomness.

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

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

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

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

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*.

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

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 $\mathcal{R}_{\mathsf{PoE}}$ [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 $Q \leftarrow u^q \in \mathbb{G}$ to the Verifier.
3. Verifier computes $r \leftarrow (x \bmod \ell) \in [\ell]$ and accepts if $Q^\ell u^r = w$ holds in $\mathbb{G}$.

The protocol above is a minor generalization of the protocol from [Wes18] in that we allow an arbitrary exponent $x \in \mathbb{Z}$, where as in [Wes18] the exponent was restricted to be a power of two. This does not change the soundness property captured in the following theorem, whose proof is given in [Wes18, Prop. 2] (see also [BBF18a, Thm. 2]) and relies on the adaptive root assumption for *GGen*.

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

For the protocol to be useful the verifier must be able to compute $r = x \bmod \ell$ faster than computing $u^x \in \mathbb{G}$. The original protocol presented

by Wesolowski assumed that $x = 2^T$ is a power of two, so that computing $x \bmod \ell$ requires only $\log(T)$ multiplications in $\mathbb{Z}_\ell$ whereas computing u^x requires T group operations.

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

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

3.2 A succinct proof of knowledge of a discrete-log

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

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

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:

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

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 $Q^\ell g^r = w$ 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 full version of the paper[BBF18b] we present a generalization of Protocol PoKE* to group representations in terms of bases $\{g_i\}_{i=1}^n$ included in the CRS, i.e. a proof of knowledge of an integer vector $\mathbf{x} \in \mathbb{Z}^n$ such that $\prod_i g_i^{x_i} = w$. We will prove that this protocol is an argument of knowledge against a generic group adversary. The security of Protocol PoKE* above follows as a special case. Hence, the following theorem is a special case of Theorem 7 in the full version.

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

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

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

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

Extending PoKE for general bases. To obtain a protocol for the relation $\mathcal{R}_{\mathsf{PoKE}}$ we start by modifying protocol PoKE* so that the prover first sends $z = g^x$, for a fixed base g, and then executes two PoKE* style protocols, one base g and one base u, in parallel, showing that the

discrete logarithm of w base u equals the one of z base g. We show that the resulting protocol is a secure argument of knowledge (in the generic group model) for the relation $\mathcal{R}_{\mathsf{PoKE}}$. The transcript of this modified protocol now consists of two group elements instead of one.

Protocol PoKE (Proof of knowledge of exponent)

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

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

The intuition for the security proof is as follows. The extractor first uses the same extractor for Protocol PoKE* 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. The protocol is presented in the full version[BBF18b].

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

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

We can also construct a (honest-verifier) zero-knowledge version of the PoKE argument of knowledge protocol using a method similar to the classic Schnorr Σ-protocol for hidden order groups. This is covered in the full version [BBF18b].

3.3 Aggregating Knowledge of Co-prime Roots

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

In the full version of the proof we describe the PoKCR. It is a proof for the relation:

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

4 Trapdoorless Universal Accumulator

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

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

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

The following definition states that an accumulator is secure if an adversary cannot construct an accumulator, an element x and a valid membership witness w_x^t and a non-membership witness u_x^t where w_x^t shows that x is in the accumulator and u_x^t shows that it is not. Lipmaa [Lip12] also defines undeniability without a trusted setup. In that definition the adversary has access to the random coins used by **Setup**.

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

Fig. 1. A trapdoorless universal accumulator.

Definition 6 (Accumulator Security (Undeniability)).

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

4.1 Accumulator construction

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

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

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

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

Groups of unknown order The accumulator requires a procedure $GGen(\lambda)$ which samples a group of unknown order in which the strong root assumption (Definition 2) holds. One can use the quotient group $(\mathbb{Z}/N)^*/\{-1, 1\}$, where N is an RSA modulus, which may require a trusted setup to generate the modulus N. Alternatively, one can use a class group which eliminates the trusted setup. Note that the adaptive root assumption requires that these groups have no known elements of low order, and hence the group $(\mathbb{Z}/N)^*$ is not suitable because $(-1) \in (\mathbb{Z}/N)^*$ has order two [BBF18a]. Given an element of order two it is possible to convince a PoE-verifier that $g^x = -y$ when in fact $g^x = y$.

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

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

- **Setup** generates a group of unknown order and initializes the group with a generator of that group.
- **Add** takes the current accumulator A_t, an element from the odd primes domain, and computes $A_{t+1} = A_t$.
- **Del** does not have such a trapdoor and therefore needs to reconstruct the set from scratch. The **RootFactor** algorithm can be used for precomputation. Storing 2^k elements and doing $n \cdot k$ work, the online removal will only take $(1 - \frac{1}{2}^k) \cdot n$ steps.
- A membership witness is simply the accumulator without the aggregated item.
- A membership non-witness, proposed by [LLX07], uses the fact that for any $x \notin S$, $\gcd(x, \prod_{s \in S} s) = 1$. The Bezout coefficients $(a, b) \leftarrow$ **Bezout**$(x, \prod_{s \in S} s)$ are therefore a valid membership witness. The actual witness is the pair (a, g^b) which is short because $|a| \approx |x|$.

– Membership and non-membership witnesses can be efficiently updated as in [LLX07]

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

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

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

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

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

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

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

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

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

4.2 Batching and aggregation of accumulator witnesses

Aggregating membership witnesses Aggregating membership witnesses for many elements into a single membership witness for the set is

straightforward using **ShamirTrick**. However, verification of this membership witness is linear in the number of group operations. Note that the individual membership witnesses can still be extracted from the aggregated witness as $w_x = w_{xy}^y$. Security, therefore, still holds for an accumulator construction with aggregated membership witnesses. The succinct proof of exponentiation (NI-PoE) enables us to produce a single membership witness that can be verified in constant time. The verification **VerAggMemWit** simply checks the proof of exponentiation.

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

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

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

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

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

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

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

We observe that the basic accumulator functions do not require a trapdoor or knowledge of the entire state, summarized in Figure 2. In particular, deleting an item requires knowledge of the item's current membership

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

witness (the accumulator state after deletion is this witness). Moreover, operations can be performed in batches as follows:

The techniques are summarized as follows:

- **BatchAdd** An NI-PoE proof can be used to improve the amortized verification efficiency of a batch of updates that add elements $x_1, ..., x_m$ at once and update the accumulator to $A_{t+1} \leftarrow A_t^{x^*}$. A network participant would check that $x^* = \prod_i x_i$ and verify the proof rather than compute the m exponentiations.
- **BatchDel** Deleting elements in a batch uses the **AggMemWit** function to a compute the aggregate membership witness from the individual membership witnesses of each element. This is the new state of the accumulator. A NI-PoE proof improves the verification efficiency of this batch update.
- **CreateAllMemWit** It is possible for users to update membership and non-membership witnesses [LLX07]. The updates do not require knowledge of the accumulated set S but do require that every accumulator update is processed. Since this is cumbersome some users may rely on service providers for maintaining the witness. The service provider may store the entire state or just the users witnesses. Creating all users witnesses naively requires $O(n^2)$ operations. Using the **RootFactor** algorithm this time can be reduced to $O(n \log(n))$ operations or amortized $O(\log(n))$ operations per witness.
- **CreateManyNonMemWit** Similarly to **CreateAllMemWit** it is possible to create m non-membership witness using $O(\max(n, m) + m \log(m))$ operations. This stands in contrast to the naive algorithm that would take $O(m \cdot n)$ operations. The algorithm is in Figure 4.2.

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

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

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

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

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

Fig. 2. Distributed and stateless accumulator functions.

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

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

Fig. 3. Algorithm for creating multiple non membership witnesses

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

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

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

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

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

Definition 7 (Batch Accumulator Security (Undeniability)).

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

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

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

For the security proof see the full version[BBF18b].

Aggregating non-membership witnesses In the full version of the paper [BBF18b] we show how non-membership witnesses can be aggregated non-interactively. Multiple independently created non-membership witnesses can be aggregated into a single witness. We can use similar batching techniques as discussed above to make this witness constant sized.

5 Batchable Vector Commitments with Small Parameters

5.1 VC Definitions

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

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

1. $\mathsf{VC.Setup}(\lambda, n, \mathcal{M}) \rightarrow \mathsf{pp}$ Given security parameter λ, length n of the vector, and message space of vector components $\mathcal{M}$, output public parameters pp, which are implicit inputs to all the following algorithms.
2. $\mathsf{VC.Com}(\boldsymbol{m}) \rightarrow \tau, com$ Given an input $\boldsymbol{m} = (m_1, ..., m_n)$ output a commitment com and advice τ.
3. $\mathsf{VC.Update}(com, m, i, \tau) \rightarrow \tau, com$ Given an input message m and position i output a commitment com and advice τ.
4. $\mathsf{VC.Open}(com, m, i, \tau) \rightarrow \pi$ On input $m \in \mathcal{M}$ and $i \in [1, n]$, the commitment com, and advice τ output an opening π that proves m is the ith committed element of com.
5. $\mathsf{VC.Verify}(com, m, i, \pi) \rightarrow 0/1$ On input commitment com, an index $i \in [n]$, and an opening proof π output 1 (accept) or 0 (reject).

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

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

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

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

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

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

5.2 VC construction

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

Our VC construction associates a unique prime[2] integer p_i with each ith index of the bitvector $\boldsymbol{m}$ and uses an accumulator to commit to the set of all primes corresponding to indices where $m_i = 1$. The opening of the ith index to $m_i = 1$ is an inclusion proof of p_i and the opening to $m_i = 0$ is an exclusion proof of p_i. By using our accumulator from Section 4, the opening of each index is constant-size. Moreover, the opening of several indices can be batched into a constant-size proof by aggregating all the membership witnesses for primes on the indices opened to 1 and batching all the non-membership witnesses for primes on the indices opened to 0.

The VC for vectors on a message space of arbitrary bit length is exactly the same, interpreting the input vector as a bit vector. Opening a λ-bit component is then just a special case of batch opening several indices of a VC to a bit vector. The full details are in Figure 4 of the full version[BBF18b].

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 constant-size and independent of n, unlike all previous vector commitments with $O(1)$ size openings [CF13,LRY16,LM18]. The batch opening of several (mixed value) indices consists of 2 elements in $\mathbb{G}$ for the aggregate membership-witness and an additional 5 elements in $\mathbb{G}$ for the batch non-membership witness, plus one λ-bit integer.

[2] Examples include $\mathsf{H}_{\mathsf{prime}}$ (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 $\mathsf{H}_{\mathsf{prime}}$ (in expectation).

Aggregating Openings, Key-Value commitment and Optimizations In the full version [BBF18b] we describe how vector commitment openings can be non-interactively aggregated. We also discuss how a vector commitment with constant sized setup can be used as a commitment to a key-value map as well as several optimizations

5.3 Key-Value Map Commitment

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

6 Applications

6.1 Stateless Blockchains

UTXO commitment We first consider a simplified blockchain design which closely corresponds to Bitcoin's UTXO design where users own coins and issue transaction by spending old coins and creating new coins. We call the set of unspent coins the UTXO set. Updates to the blockchain can be viewed as asynchronous updates to the UTXO set. In most current blockchain designs (with some exceptions[MGGR13,BCG+14]) nodes participating in transaction validation store the whole UTXO set and use it to verify whether a coin was unspent. Instead, we consider a blockchain design where the network maintains the UTXO set in a dynamic accumulator [STS99a,TMA13,Tod16,Dra]. We instantiate this accumulator with our new construction from Section 4.1, taking advantage of our distributed batch updates and aggregate membership proofs.

Each transaction block will contain an accumulator state, which is a commitment to the current UTXO set. To spend a coin, a user provides a membership witness for the coin (UTXO) that is being spent inside a transaction. Any validator (aka miner) may verify the transactions against the latest accumulator state and also uses **BatchDel** to delete all spent coins from the accumulator, derive its new state, and output a proof of correctness for the deletions. The proof is propagated to other validators in the network. For the newly minted coins, the validator uses **BatchAdd**

to add them to the accumulator and produce a second proof of correctness to propagate. Other validators are able to verify that the accumulator was updated correctly using only a constant number of group operations and highly efficient arithmetic over λ-bit integers.

In this design, users store the membership witnesses for their own coins and are required to update their witnesses with every block of transactions. It is plausible that users use third-party services to help with this maintenance. These services are not trusted for integrity, but only for availability. Note that a may produce many (e.g. n) membership witnesses at once in $O(n \log(n))$ time using the **CreateAllMemWit** algorithm

Accounts commitment Some currencies such as Ethereum [Woo14] use an account-based system where the state is a key-value map. A transaction updates the balances of the sending and the receiving accounts. To enable stateless validation in this setting, a user can provide proofs of the balances of the sending and receiving accounts in the current ledger state. Instead of using an accumulator to commit to this state, we use the new key-value map commitment from Section 5.3. This commitment supports batch distributed updates, similar to our new accumulator. Using the aggregation of vector commitment openings a miner or validator can perform the aggregation and batching operations without storing the state providing efficient proofs that the openings are correct. Other nodes can verify these opening proofs efficiently requiring only a constant number of group operations.

6.2 Short IOPs

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

$5 \cdot 2048 + 128 + 3 \cdot 80 = 10608$ bits, or 1.3 KB. This is the shortest (theoretical) setup-free SNARK with sublinear public parameters to date.

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

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

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

Acknowledgments

This work was partially supported by NSF, ONR, the Simons Foundation and the ZCash foundation.

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.

[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.

[BBF18a] 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.

[BBF18b] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. Cryptology ePrint Archive, Report 2018/1188, 2018. https://eprint.iacr.org/2018/1188.

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

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

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

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

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

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

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

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

[BSCR+18] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct ar-

guments 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.

[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.

[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.

[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.

[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.

[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.

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

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

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

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

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

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

[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.

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

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

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

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

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

[STS99a] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 555–572. Springer, Heidelberg, August 1999.

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

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

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

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

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

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

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