Merkle Mountain Ranges are Optimal: On witness update frequency for cryptographic accumulators

Joseph Bonneau¹⁴, Jessica Chen¹, Miranda Christ², and Ioanna Karantaidou¹³

¹ New York University jcb@cs.nyu.edu, jessicachen@nyu.edu ² Columbia University mchrist@cs.columbia.edu ³ George Mason University ikaranta@gmu.edu ⁴ a16z crypto

Abstract. We study append-only set commitments with efficient updates and inclusion proofs, or cryptographic accumulators. In particular, we examine how often the inclusion proofs (or witnesses) for individual items must change as new items are added to the accumulated set. Using a compression argument, we show unconditionally that to accumulate a set of n items, any construction with a succinct commitment ($O(\lambda \text{ polylog } n)$ storage) must induce at least $\omega(n)$ total witness updates as n items are sequentially added. In a certain regime, we strengthen this bound to $\Omega(n \log n / \log \log n)$ total witness updates. These lower bounds hold not just in the worst case, but with overwhelming probability over a random choice of the accumulated set. Our results show that a close variant of the Merkle Mountain range, an elegant construction that has become popular in practice, is essentially optimal.

1 Introduction

Consider the problem of a maintaining a cryptographic commitment to an ever-growing set of elements, for example the set of all users with a registered public key, all issued x.509 certificates on the Internet, or all transactions in a blockchain. In all of these applications, elements can only be added to the set and never deleted. The commitment itself should be succinct (at most logarithmic in the size of the set), and the scheme should support short (again at most logarithmic in the size of the set) inclusion proofs (or *witnesses*⁵) that any individual element is included in the set. These requirements are met by classic cryptographic *accumulators* [5]. However, in many practical applications there is an additional goal, that witnesses of already-included elements do not change too often as new elements are added. Unfortunately, for most known accumulators (e.g., Merkle trees [31], RSA accumulators [11], and bilinear accumulators [35]), adding a single new element to the set requires (except with negligible probability) changing *every* existing element's witness.

As a result, several works (e.g., [44,39,45,21]) whose applications benefit from low witness update frequency have proposed tailored accumulator constructions. Curiously, these constructions all appear to reinvent the same data structure: the *Merkle Mountain Range* (MMR).⁶ An MMR is a list of Merkle trees. New elements are added to the smallest tree, and whenever there are two complete trees of the same size, we merge them. An element's witness is only updated when its tree is merged. This results in a $\Theta(\lambda \log n)$ -sized commitment and a total of $\Theta(n \log n)$ witness changes when accumulating a set of size n.

Despite a large body of work on cryptographic accumulators, no construction has surpassed the witness update efficiency of MMRs, suggesting a fundamental barrier. This leads us to the natural question:

What is the optimal witness update frequency of an append-only accumulator?

⁵ We will use the terms "witness" and "inclusion proof" interchangeably.

⁶ We use the name Merkle Mountain Range as coined by Todd in the first proposal we are aware of [44] and most widely known in practical implementations, though it was not used in later academic work [39,21].

Witness update frequency has rarely been studied, despite its importance in practice. Existing lower bounds on witness updates [10,13] apply only to data structures that support deletion and, critically, use the power of deletion in their proofs. To now, it was not known whether analogous lower bounds held for append-only data structures, let alone lower bounds approaching the $\Theta(n \log n)$ efficiency of the MMR construction.

Furthermore, known lower bounds are quite weak in showing only that at least $\Omega(n/\log n)$ elements must have their witness changed at least once. Our results are stronger in showing that most elements' witnesses change $\omega(1)$ times. This is relevant in practical settings when, for example, every user holding an element whose witness changes must retrieve the new witness with each update. If every element's witness changed at most some constant number of times, users could go offline after this number of changes occurred, as after that point their witness would remain valid indefinitely. In contrast, our lower bound shows that most elements' witnesses will *never stop changing* as the committed set grows.

Our results We show that any accumulator of size $O(\lambda \operatorname{polylog} n)$ requires $\omega(n)$ total witness updates during the course of accumulating a set of n elements in sequence, implying at least $\omega(1)$ updates on average per element. For stronger accumulators accommodating sets of size $2^{\lambda^{\beta}}$ for constant $\beta > 0$, we show a stronger bound of $\Theta(\beta n \log n / \log \log n)$ total updates. Our proof leverages a compression argument, which is from first principles but is quite involved. We observe that our lower bound is asymptotically tight, as the optimal parameter setting for k-ary MMRs results in $O(n \log n / \log \log n)$ total updates.

2 Related Work

Authenticated data structures An authenticated data structure is a succinct commitment to a data structure which facilitates efficient correctness proofs of READ and sometimes WRITE operations for verifiers not storing the entire data structure. Different constructions exist for different underlying data structures, including set commitments (commonly known as accumulators) [11,29,35,16,2], i.e. elements stored without a position; vector commitments [12,27,46,22,37], i.e. elements stored in specified total order; and key-value (also sometimes called dictionary or map) commitments [36,47,48,1,49,20], i.e. element values placed in flexible slots addressed by a key. More constructions are possible; indeed, Miller et al. show that *any* classical acyclic pointer-based data structure can be generically transformed into an authenticated data structure assuming only collision-resistant hash functions [32]. While different constructions are often developed for each setting, they are interrelated, with generic transformations between them [7,12,20]. Our work focuses exclusively on accumulators, the simplest authenticated data structure, but our lower bounds apply to most other constructions (e.g. vector or map commitments) as these can be used to instantiate a set commitment.

Memory checkers A memory checker, defined and first constructed by Blum et al. [6], is a natural type of authenticated data structure that has received attention from both theoretical and applied communities. A memory checker allows a client with a small amount of trusted storage to outsource storage of a large database to untrusted memory. The memory checker sits between the client and the large untrusted memory, and allows the client to make WRITE and READ operations, whose correctness it can verify using its trusted local storage. Two standard efficiency measures of memory checkers are *space complexity*, or the amount of local storage used by the client, and *query complexity*, the maximum number of entries in untrusted storage the memory checker must access to make and validate an operation.

Follow-up works have studied the efficiency of memory checking. Dwork et al. [18] showed that the query complexity of deterministic memory checkers using a sublinear amount of local storage is at least $\frac{\log n}{\log \log n}$ for databases of size n. Boyle et al. [8] further generalized this result to also describe non-deterministic, adaptive memory checkers. Recently, Wang et al. [50] investigated the locality of memory checkers, i.e. the maximum number of non-contiguous public memory regions queried across all index queries. They showed that the locality of memory checkers is also lower bounded by $\frac{\log n}{\log \log n}$. The space complexity of online memory checkers requires computation assumptions and specifically one-way functions [6,34] in order to get meaningful space-query complexity guarantees, with local space proportional to the security parameter.

Memory checker lower bounds are similar in spirit to our accumulator lower bounds, though they are formally different. We discuss these differences in depth after recalling accumulators and their existing lower bounds.

Accumulators Accumulators are authenticated data structures that store a set, and enable verifying inclusion of any element in that set. We give a precise definition lf accumulators in Section 4 (Definition 1). The best-known and still most widely used accumulator is the classic Merkle tree [31], although it is often not considered an accumulator (and indeed provides the stronger abstraction of a vector commitment). Benaloh and de Mare [5] first proposed the first algebraic construction, based on groups of unknown order (RSA groups), which offers constant-sized inclusion proofs and constant-size update costs to add an element. Later advances extended the functionality to include *dynamic* accumulators [11] (supporting efficient deletion as well as addition) and *universal* accumulators [29] (supporting exclusion as well as inclusion proofs). Note that in our work we only consider basic accumulators (e.g. not necessarily dynamic or universal). Again, our lower bounds apply as well to dynamic or universal accumulators, as these offer strictly more functionality and hence can simulate basic accumulators.

Among all constructions, Merkle trees [31] provide optimal query and locality efficiency [18,50] while most non-tree based, algebraic constructions [5,11,29,35] offer optimal proof size (constant).

Existing accumulator lower bounds The efficiency of witness updates in dynamic accumulators was studied by Camacho and Hevia [10], who proved information-theoretically that information at least linear in the number of deleted elements needs to be communicated to update all witnesses. Christ and Bonneau [13] proved that when deleting *n* elements from a dynamic accumulator with a succinct commitment, $\Omega(n/\log n)$ proofs must change in total. They further show this result applies to any universal accumulator (supporting both inclusion and exclusion proofs) even without explicitly supporting deletion of elements, as adding a new element effectively requires "deleting" that element's previous exclusion proof. Notably, we consider accumulators that are not necessarily dynamic or universal, to which these prior lower bounds do not apply.

The trade-off between the size of the global state and the frequency with which proofs must be updated is the basic idea behind *buffering* which we discuss further in Section 3.2. Among prior work, for example, Aardvark [28] uses a buffer of uncommitted new items to efficiently support concurrent data updates.

Accumulators versus memory checkers. Dwork et al. and Boyle et al. [18,8] use compression arguments to prove lower bounds on the complexity of memory checkers. These bounds are incomparable to our accumulator lower bounds for several reasons. An append-only accumulator can be thought of as a restricted-functionality memory checker with a rigid structure, where each entry i in the memory checker's untrusted storage stores either \perp or a membership proof of element i. Trusted storage contains only the commitment.

First, the functionality that an append-only accumulator provides is far narrower than that of memory checkers, and therefore is a priori easier to realize. Specifically, append-only accumulators store only binary strings, and write operations may change 0s to 1s but never 1s to 0s. The accumulator enables attestation to the 1 entries, but does not allow one to verify that an entry is 0. In contrast, memory checkers store arbitrary data, authenticate all entries, and allow arbitrary write operations. As memory checkers must implement stronger functionalities, their lower bounds do not imply accumulator lower bounds.

Second, we consider a slightly different notion of complexity. While the size of our accumulator corresponds to the amount of trusted storage (i.e., *local space*) in a memory checker, our notion of witness update complexity is incomparable to the memory checker notion of write complexity. In our setting, each client (which one can think of as an entry in unreliable storage) maintains its own membership proof of a stored element. We measure the *total number of times the clients' membership proofs must change* as *n* elements are added to the stored set. On the other hand, memory checker lower bounds measure the maximum number of untrusted database locations that must be accessed during a query. Thus, our witness change complexity reflects only a number of changes, whereas memory checker query complexity reflects the pattern of changes.

Finally, accumulators essentially require that it is possible to distribute the entries in untrusted storage, such that each entry authenticates a 1 in the stored string.

Optimizations outside our setting When the accumulator is managed by a trusted party that maintains a secret state, efficient constructions enabling low witness update frequency exist. Baldimtsi et al. [4] and later Karantaidou and Baldimtsi [26] propose accumulators that remain static in additions, requiring no proof updates. Tas and Boneh propose vector commitments with efficient updates that rely on a tree structure [43]. However, this secret state allows the manager to violate soundness of the commitment, whereas we consider the more traditional trustless setting.

Another optimization that does not address our problem is *batching*, which broadly describes efficient operations for groups of elements, for example adding a subset of elements all at once to an accumulator. Boneh et al. [7] modify RSA accumulators to enable a single membership witness that holds for a large set of elements. Srinivasan et al. [41] develop an algorithm that updates all elements' witnesses more computationally efficiently than updating each witness individually. No batching techniques allow witnesses to be updated *less frequently*. In [43] the data is organized in a tree, a trusted manager is taking advantage of the duplicate nodes across different paths (witnesses) and according to the position of the newly added elements, the manager precomputes node updates that are shared by many old elements.

Merkle Mountain Ranges and similar constructions Todd [44] proposed and named⁷ Merkle Mountain Ranges in 2013, looking for an efficient data structure for committing to all blocks in the Bitcoin blockchain. Reyzin and Yakoubov [39] appear to have re-discovered essentially the same construction in 2016, motivated by the application of distributed PKI. Garg et al. again rediscovered the construction 2018 [21], motivated by the application of registration-based encryption. Prior to Todd, Crosby and Wallach proposed Merkle History Trees in 2009 [15] (in the context of tamper-evident logging), a similar notion but with all sub-trees included as nodes of a larger tree. Google's Transparency team proposed a similar concept, *compact ranges* in 2022 [14].

Applications MMRs have seen many proposed applications within the space of blockchains and cryptocurrencies. Todd's motivation in the original MMR proposal was the OpenTimestamps project, seeking to implement an efficient timestamping service on top of Bitcoin by building an MMR of all past Bitcoin blocks. Todd later proposed using MMRs to commit to all UTXOs (unspent transaction outputs) in each Bitcoin block, enabling more efficient proofs of transaction validity [45]. Bünz et al. [9] proposed including an MMR of past blocks within each block header to enable FlyClient, an ultra-light client implementation for proof-of-work cryptocurrencies. Liang et al. [30] proposed extending to k-ary MMRs with applications to IoT-focused blockchains.

Several of these applications have seen practical deployment: FlyClient's use of MMRs for compact history proofs has been adopted by the Grin [24], Minima [33], Nervos [52], Neptune [42] and Zcash [19] blockchains. Other projects build an MMR on top of existing chains for compact proofs of data inclusion, including Axiom [3], Herodotus [25] and Picasso [38]. Finally, in line with Todd's original motivation applications, several projects use MMRs as part of a data timestamping service, including DataTrails [17], OpenTimestamps [44], and Witness [51].

3 Technical overview

An append-only accumulator provides the following functionality: Given a set X, it outputs a commitment A and a membership witness π_i for each element $x_i \in X$. For example, A may be a Merkle tree root, in which case each π_i would be a Merkle inclusion proof for the x_i stored at each leaf. The accumulator provides a verification function that should output true when given any tuple (A, x_i, π_i) of an honestly generated commitment, included element, and a properly-generated witness for that element. It should output false given any element not included in the set corresponding to the given commitment. We provide a formal definition in Section 4.

⁷ We note that later academic rediscoveries [39,21] did not use the name "mountain range."

To add an element x' to the accumulated set X, the naive approach is to compute from scratch a succinct commitment A' to $X \cup \{x'\}$, along with membership witnesses for all elements in $X \cup \{x'\}$. Most practical accumulator constructions offer more efficient algorithms for update, which in some cases don't require changing the membership witnesses for all elements.

3.1 Witness update frequency in append-only accumulators

We consider the following problem: Elements in the set $X = \{x_1, \ldots, x_n\}$ are added to an accumulator in sequence. After x_1 is added, its initial membership witness (which we denote $\pi_1^{(1)}$) is only guaranteed to be valid for the current commitment, which we denote A_1 . After adding x_2 , the commitment changes to A_2 and it is possible (and indeed overwhelmingly likely for most schemes) that verification fails for $\pi_1^{(1)}$ with respect to A_2 , even though x_1 is still in the accumulated set. Therefore, $\pi_1^{(1)}$ must be updated to a new, valid witness $\pi_1^{(2)}$. As additional elements are added, the witness for x_1 may continue to evolve through multiple values; we use $\pi_1^{(i)}$ to denote the witness of x_1 with respect to the commitment A_i .

The number of witness changes for x_j is the number of changes its witness undergoes throughout this sequence of additions, or more precisely the number of indices $j \leq i \leq n-1$ such that $\pi_j^{(i)} \neq \perp$ and $\pi_j^{(i)} \neq \pi_j^{(i+1)}$. Note that element j can have its witness change at most n-j times, its witness is undefined (\perp) before it is added to the accumulator and we don't count the initial assignment of a witness as a change.⁸

We are mostly concerned with the *total* number of witness changes across all elements, though these results easily translate into an average number of witness updates per element. The total number of witness updates can be at most $\frac{n(n-1)}{2} = O(n^2)$ (if all added elements' witnesses change at every step), or O(n) per element, as is achieved by most classical accumulators.

We could also consider the *worst-case* number of witness updates (the maximum number of changes per element, over all elements added to the accumulator). However, in all natural cases that we know of, the worst case is asymptotically equivalent to the average case.⁹

Finally we note that our results do not say anything about the best-case, or *minimum* number of witness updates across all elements. This question is not particularly interesting in our setting, as by definition element x_n (the last accumulated) always has no witness updates. It is also trivial to design an accumulator that avoids witness updates for *any* particular element, for example the first element, by storing it directly and giving it a null witness.

3.2 Comparison of known constructions

We can now compare the efficiency of several constructions for accumulating a set on both commitment size and witness update frequency. The simplest approach is a crude *database*, in which the the commitment is simply the complete list of elements. This approach requires $\Theta(n\lambda)$ storage¹⁰, but each element's witness is constant (the element itself). By contrast, the classical cryptographic accumulator constructions require just $\Theta(\lambda)$ storage, but at the cost of $O(n^2)$ witness updates in total. Between these two extreme points, there are several interesting intermediate trade-offs, as shown in Table 1.

One natural idea, dating to the early days of computing [23], is *buffering*. The commitment now consists of a classical accumulator commitment plus a temporary buffer with space to store up to k elements directly. New elements are written directly to the buffer and not immediately added to the accumulator. After k

⁸ Alternately one could count the initial assignment of a witness as a "change." This would have no impact on our lower bounds; it would simply add one to the witness change count for all elements.

⁹ One could easily construct a contrived accumulator that achieved better average results but poor performance for an arbitrarily chosen element, but we don't know of a natural construction that would exhibit this behavior.

¹⁰ While writing elements in the buffer naively requires logarithmically many bits in the size of the data universe U, one can instead write a $\Theta(\lambda)$ -length collision-resistant hash of each element.

Table 1. Comparison of approaches for constructing an append-only set commitment with n elements. We list the total number of witness updates across all elements, in terms of n and k. For simplicity, we omit the security parameter λ as all commitment sizes are additionally linear in λ and it does not affect the number of witness updates.

Construction	Commitment size	Total $\#$ witness updates
Simple database	$\Theta(n)$	0
Classical accumulator [31,5,35]	$\Theta(1)$	$O(n^2)$
k-buffering + accumulator	$\Theta(k)$	$O(\frac{n^2}{k})$
k-bucketed accumulators	$\Theta(\frac{n}{k})$	O(kn)
k-buffering + k -bucketed accumulators	$\Theta\left(\frac{n}{k}+k\right)$	O(n)
k-ary Mountain Range (§6)	$O(k \log_k n)$	$O\left(n\log_k n\right)$
$(\log n)$ -ary Mountain Range (§6)	$O(\log^2 n)$	$O(n\log n / \log\log n)$

members are written this buffer must be flushed, with every new member added to the committed data structure (and every existing member's witness changing as a result). This approach offers a linear trade-off, requiring $\Theta(k\lambda)$ global storage in exchange for $O(\frac{n^2}{k})$ witness updates in total.

Another natural trade-off can be achieved by *bucketing* the items into a growing list of accumulator commitments each storing up to k elements. New items are added to an accumulator with fewer than k elements, if one exists; otherwise, they are inserted into a new accumulator which is added to the list. With this approach, n elements require $\Theta(\frac{n}{k})$ set commitments, but there are only O(kn) witness updates in total.

Bucketing and buffering can also be combined: New elements are appended to the buffer and given null witnesses. After the buffer is full, the items in the buffer are accumulated into a constant-sized commitment, which is appended to the commitment list. After accumulating n elements, there are at most n/k constant-sized commitments and a buffer of size at most k. Therefore, this approach uses $\Theta(\lambda \cdot (\frac{n}{k} + k))$ storage, and requires at most one update per witness, or O(n) witness updates in total. Setting $k = \sqrt{n}$, one can achieve $\Theta(\lambda\sqrt{n})$ storage.

Can we do better? An elegant solution which achieves both sublinear storage and witness updates is the *Mountain Range*, shown in Figure 1. We provide a complete definition in Section 6, but in brief the approach can be seen as a natural evolution of the bucketing + buffering approach. Instead of using a single buffer and a set of of fixed accumulators of size k, a mountain range consists of a series of buckets (each committed as an accumulator) of size exactly k^m for some m.¹¹ Any time there are k equal-sized buckets of size k^m , they are merged into a new commitment to a bucket of size k^{m+1} (which may trigger additional merges if there are now k buckets of size k^{m+1} and so on). This approach breaks through the linear trade-offs above, requiring $O(\log_k n)$ updates per witness with only $\Theta(k \log_k n)$ global storage.

Merkle Mountain Ranges This idea has typically been proposed specifically as a Merkle Mountain Range (MMR), using (binary) Merkle trees as the underlying accumulator and k = 2, leading to $O(n \log_2 n)$ total witness updates with $\Theta(\log_2 n)$ storage. In an MMR, the commitment consists of a list of roots of Merkle trees decreasing size. We add new elements as leaves in the smallest tree, and every time we obtain two trees of equal size, we merge them. This leads to the visual image of a series of increasingly smaller "mountains" and the name "mountain range." The MMR construction affords a very simple implementation as well as concretely efficient merging: Merges in an MMR always involve two trees of equal size, which are easily merged by adding a new element as the root with the two existing trees as children. Witness updates are similarly concretely efficient, consisting of appending just one new neighbor-node value to the existing witness.

3.3 Our lower bound

We consider the Mountain Range design and ask the natural question: is this the optimal accumulator construction for balancing global storage costs with witness update frequency?

¹¹ The singleton items (conceptually buckets of size k^0) can also be stored directly rather than in an accumulator.



Fig. 1. A binary Merkle Mountain Range with three mountains, storing seven elements (l_0, l_1, \ldots, l_6) . The commitment consists of (r_0, r_1, r_2) .

Compression arguments Like many existing data structure lower bounds (e.g., [18,10,13,8]), ours leverages a *compression argument*: Alice chooses a random subset of the data universe, commits to this set using the authenticated data structure and communicates this commitment to Bob. Bob attempts to reconstruct the set by computing initial witnesses for each element and testing if they hold with respect to the received commitment from Alice. If the data structure is too efficient in some sense (e.g., has no witness changes), Bob can decode Alice's random set despite having received fewer bits from Alice than is information theoretically required.

Even if there are some witness changes, Alice can send Bob some extra information to encode which witnesses have changed. If there are few witness changes, this only requires Alice to send a little additional information, and she can still communicate her set with impossible efficiency.

Our main result We show that any set commitment of size O(polylog n) requires at least $\omega(n)$ total witness updates to accumulate a set of n elements. Furthermore, for set commitments that accommodate superpolynomial-sized sets, as do Merkle Mountain Ranges, we show a stronger bound of $\Omega(n \log n / \log \log n)$ witness changes.

Our proof is from first principles, and involves two steps. The first is a compression argument that shows that within any given subsequence added to the accumulator, many witnesses change at least once. This yields a bound of ϵn total witness changes for $\epsilon \in (0, 1)$; recall we need to show $\omega(n)$ total changes. The second step is therefore to use our compression argument to show that many witnesses change a super-constant number of times. We do so by applying our compression argument many times to different subsequences of the accumulated sequence.

We obtain our stronger lower bound by applying our compression argument roughly $\log n$ times; this necessitates considering superpolynomial-sized sets. MMRs provide an asymptotically tight upper bound.

Warm-up: many witnesses change at least once To build intuition for our compression argument and familiarity with witness changes, we begin with a warm-up application of our techniques to show a far weaker statement, that at least $\Theta(n/\log n)$ witnesses change when a sequence of n elements is accumulated (Section 3.3). Strengthening this lower bound to show that a constant fraction of witnesses change at least once is fairly immediate (Lemma 1), but showing that some elements' witnesses change multiple times is more involved (Lemma 2). We'll show a simpler version of this extension to multiple witness changes in Section 3.3.

We apply our compression argument to show that $\Omega(n \log n)$ witnesses change at least once when accumulating *n* elements. We follow a similar template to existing information theoretic lower bounds for data structures [10,18,13,8]. Alice chooses a random set, which she wishes to communicate to Bob. Alice uses the

data structure to encode her set, and if the data structure is too efficient, her encoding is shorter than is information theoretically possible.

For our argument, Alice and Bob have agreed on an ordered subset $X = \{x_1, \ldots, x_{2n}\}$ of the data universe, a uniform random string $r \in \{0, 1\}^*$, and public parameters $pp \leftarrow \text{Setup}(1^{\lambda})$. Alice chooses a random *n*-sized subset $S \subseteq X$ and accumulates S in some canonical ordering (using randomness r, if necessary) to obtain a final commitment A. As Alice does this accumulation, she maintains a list L of included elements whose witnesses changed. Alice sends A and L to Bob.

Bob will attempt to accumulate x_1, x_2, \ldots in sequence using the randomness r, maintaining his own set S' and commitment A' to the elements he has added thus far. We say *attempt* because as Bob goes, he will test whether Alice included each x_i in her set S; if not, he will omit x_i from his accumulator and move on. This test is nontrivial, as we describe below.

In each step, Bob will add x_i to A' to obtain a membership witness π_i and the updated commitment A'_i . He will then test if π_i verifies with respect to Alice's commitment A. If so, he updates A' to equal A'_i and adds x_i to S', as soundness ensures that Alice must have included x_i in her set. If π_i does not verify, there are two possibilities:

- 1. Alice did not include x_i , or
- 2. Alice included x_i , but its witness changed.

The list L allows Bob to distinguish between these two cases when π_i does not verify. If x_i is in L, Bob updates A' to equal A'_i and adds x_i to S'. If x_i is not in L, Bob does not update A' or S' and moves on.

Note that because Alice and Bob use the same random tape, Bob exactly reconstructs the intermediate witnesses and commitment values Alice obtained in computing A. At the end of this process, Bob will have successfully reconstructed S' = S with all but negligible probability.¹²

The number of bits Alice sends is |A| + |L|. If at most $n/(2\log(2n))$ witnesses change, $|L| \le n/2$, since we write down $n/(2\log(2n))$ indices of length $\log 2n$. Since the accumulator is succinct (i.e., its size is sublinear in n), |A| + |L| < n. Therefore, Alice sends far fewer than n bits, yet communicates an element from a uniform distribution over a set of size nearly 2^n .

We have thus shown that if the accumulator is succinct, at least $\Omega(n/\log n)$ elements' witnesses change at least once. We call these one-shot witness updates.

Increasing the fraction of changed witnesses One can immediately strengthen this warm-up result to show that for any constant $\epsilon \in (0, 1)$, at least ϵn witnesses change. One does so by encoding L more cleverly, rather than writing the index of each element whose witness changed (which required $|L| = \Omega(f \log n)$, where fis the number of elements whose witnesses change). Instead, Alice simulates exactly when Bob will consult L and writes a single bit for each of these consultations, indicating whether the element in question was included in S. For appropriate choices of X and Alice's distribution of chosen set S, this encoding of L is short enough to obtain the desired bound.

We prove this result and describe this encoding in Lemma 1, which also shows that the same result applies to any subsequence of added elements. That is, for any sufficiently large d, many witnesses of $S = \{s_i, \ldots, s_{i+d}\}$ change between when they are initially added and the final commitment after s_{i+d} is added.

This ϵn bound is already stronger than the lower bound of [13], which shows that $\Omega(n/\log n)$ witnesses change at least once.

¹² With negligible probability, soundness of the commitment may be violated—that is, a witness may verify for an element that Alice did not add, resulting in Bob erroneously including this element.

Showing that witnesses change multiple times Thus far, we've only shown that a large fraction of elements' witnesses change at least once. While encoding L more efficiently allows us to show that a larger fraction of witnesses change, there is no clear way to alter the warm-up proof to show that any element's witness changes *more than once*. Doing so will require new techniques, as we'll see here.

Our key idea is to apply the warm-up result iteratively, to subsequences. This is helpful because now we can show that some x_i 's witness changes in *multiple* subsequences, and therefore changes multiple times. In this compression game, Alice will choose a random bit vector $v \in \{0, 1\}^n$ to communicate to Bob. She'll encode it by accumulating a much larger set. Now we suppose Alice and Bob agree on 2n disjoint *n*-sized sets $X_1, \ldots, X_n, Y_1, \ldots, Y_n$, and an ordering of their elements. At each step $1 \le i \le n$, Alice chooses either X_i or Y_i based on the next bit v_i in her vector v and accumulates all of the elements in either X_i or Y_i .

Let $s = [s_1, \ldots, s_{n^2}]$ denote the sequence of elements Alice accumulated. Like in the communication game from the warm-up proof, Alice computes a commitment A to s, which she sends to Bob. Bob will attempt to reconstruct Alice's vector v. He computes his own vector v', determining each bit by trying to see if Alice chose X_i or Y_i at each step by checking whether any witness of any element in X_i holds with respect to A.

That is, Bob maintains a current accumulator A' and adds all of X_i to obtain A_i and witnesses for all of its elements. If any proof holds with respect to A, Bob knows that X_i was included. He therefore updates A'to equal A_i and sets $v'_i = 0$. If no witness of X_i holds, either all of X_i 's witnesses changed or Alice chose Y_i . Similarly to in the warm-up, in order to help Bob distinguish between these two cases, Alice sends a list L of indices i for which all witnesses changed between their initial accumulation into A_i and the end accumulator value A. Therefore, if Bob sees that i is not in this list, Alice must have accumulated Y_i ; he adds Y_i and updates A' accordingly.

By encoding L carefully and counting the amount of information sent, we show that for at least a constant fraction of $i \in [n]$, X_i (or Y_i , if it was chosen) must have all of its elements' witnesses change between A_i and A.

What have we accomplished that differs from the warm-up? The warm-up and Lemma 1 show that if one accumulates a sequence x_1, \ldots, x_n , many elements' witnesses change between when they are added and the end of this sequence. But those statements say nothing about what happens to the witnesses of x_1, \ldots, x_n as one *continues adding more elements* x_{n+1}, \ldots In contrast, consider the proof we just sketched, taking a random X_i (or Y_i) to be x_1, \ldots, x_n . We've now shown that after adding X_i to obtain A_i , as more sequences are added $(X_{i+1} \text{ or } Y_{i+1}, \text{ etc.})$, with constant probability all witnesses of elements in X_i change. Lemma 2 formalizes this "multi-shot" argument, stating that for any constant $\epsilon \in (0,1)$, for sufficiently large n, at least ϵn sets i have all of their witnesses change after X_i (or Y_i) is added.

Recall that our key idea is to apply (an extension of) the warm-up argument iteratively. So far, we've applied it once, to show that for most $i \in [n]$, the set Alice chose (without loss of generality, X_i) has all of their witnesses change between X_i 's last element being added, and the final accumulator. We now apply the warmup argument to show that many elements x_j of X_i have their witnesses change between when x_j is added, and the last element of X_i is added. Observe that this time period over which we count witness changes in this second application is disjoint from the time period we consider in the first application. Recall that the strengthened warm-up argument says that if X_i is chosen to be a random set of size n, at least ϵn of its witnesses change while X_i is added.

Putting this all together, we've shown that:

- For each i, at least ϵn witnesses of X_i change within X_i being added, and
- For at least ϵn values of *i*, all of X_i 's witnesses change after X_i has been added.

Therefore, in total there are at least $n(\epsilon n) + (\epsilon n)n = 2\epsilon n^2$ witness changes that occur throughout the course of adding n^2 elements. Recalling that ϵ can be any constant in (0, 1), we've succeeded in showing that some witnesses change multiple times.

Iterating this argument We've essentially shown two (simplified) iterations of our argument, which involved adding n^2 elements and showing there must be $2\epsilon n$ witness changes. One can continue iterating our argument any constant c number of times, requiring adding n^c elements and showing $c\epsilon n$ witness changes. This is how we obtain our main result:

(Theorem 1). For any constant $\alpha > 0$, for sufficiently large n, if one accumulates a random set of size n there are at least αn witness updates with overwhelming probability.

The proof of Theorem 1 simply applies Lemma 2 many times for appropriately chosen parameters, taking care to ensure we do not double count witness changes.

A stronger bound for stronger accumulators We can increase the number of witness changes by iterating our argument more times, but this increases the size of the committed set. Some accumulators, such as MMRs, accommodate superpolynomially sized sets, and for such accumulators we show a stronger lower bound. In particular, if our accumulator accommodates sets of size $2^{\lambda^{\beta}}$ for a constant $\beta > 0$, we have:

(Corollary 2). For sufficiently large $n \ge 2^{\lambda^{\beta}}$, if one accumulates a random set of size *n* there are at least $\Theta\left(\frac{\beta n \log n}{\log \log n}\right)$ witness updates with overwhelming probability.

Merkle Mountain Ranges are nearly optimal Our stronger lower bound is asymptotically tight for $(\log n)$ -ary Merkle Mountain Ranges when accumulating large sets of size $2^{\lambda^{\beta}}$ for constant $\beta > 0$. We remark that MMRs indeed securely accumulate such large sets.

4 Preliminaries

Notation. Let λ denote the security parameter. We write $\operatorname{poly}(\lambda)$ to mean a polynomial function in λ . We say a function f of λ is negligible if $f(\lambda) = O(\frac{1}{\operatorname{poly}(\lambda)})$ for every polynomial $\operatorname{poly}(\cdot)$, and we write $f(\lambda) \leq \operatorname{negl}(\lambda)$ to mean that f is negligible. We say that a probability is overwhelming if it is at least $1 - f(\lambda)$ for some negligible function f. We write "p.p.t." to mean probabilistic polynomial time.

We use log to mean binary logarithm. We write polylog(n) to mean any function that is $O(\log^c n)$ for some positive constant c. When writing the asymptotic complexity of a data structure storing n elements, we often omit any dependence on the security parameter λ ; e.g., we write $O(\log n)$ instead of $O(\lambda \log n)$. We use boldface to denote a vector \boldsymbol{x} . We use wt(\boldsymbol{x}) to denote the Hamming weight of a vector $\boldsymbol{x} \in \{0,1\}^*$. We use [n] to denote the set $\{1,\ldots,n\}$.

Unless otherwise specified, we refer to append-only accumulators when we mention accumulator in this paper. There also exist accumulators that support deletions (dynamic accumulators) and non-membership witnesses (universal accumulators), but we do not consider them here.

Fact 1 (Chernoff bound) Let X_1, \ldots, X_n be independent Bernoulli random variables, and let $\mu = \mathbb{E}[\sum_{i=1}^n X_i]$. For any $\delta \ge 0$,

$$\Pr\left[\sum_{i=1}^{n} X_i \le (1-\delta)\mu\right] \le \exp\left(\frac{-\mu\delta^2}{2}\right).$$

Definition 1 (Append-only accumulator [5]). An append-only accumulator is a tuple of p.p.t. algorithms $\Pi = (\text{Setup, Acc, Verify})$ together with a data universe \mathcal{U} , such that for any ordering function ord that takes as input a set and outputs a sequence of all elements in the set, and any subset $S \subseteq \mathcal{U}$:

 $\mathsf{Setup}(1^{\lambda}, \mathsf{ord}) \to \mathsf{pp:} \mathsf{Setup}$ is a randomized algorithm that takes as input the security parameter and an ordering function, and outputs public parameters pp .

- $Acc(pp, S) \to A, (\pi_1, \dots, \pi_{|S|})$: Acc is a (possibly randomized) polynomial-time algorithm that takes as input the public parameters pp and a subset $S \subseteq U$. It returns an accumulator value A and a membership proof π_i for each $s_i \in S$.
- Verify(pp, A, u, π) \rightarrow {true, false}: Verify is a (possibly randomized) polynomial-time algorithm that takes as input the public parameters pp, an accumulator value A, an element $u \in U$, and a membership proof π . Verify should output true if and only if π is a valid proof of membership of u in the set that A accumulates.

We let Acc and Verify implicitly take 1^{λ} as input, though we omit this for ease of notation.

Remark 1. Note that this syntax implicitly defines an Add(pp, S, A, s) function that takes as input public parameters, a set S of elements added thus far, an accumulator A of these elements, and an element s' to be added. It calls Acc(pp, $S \cup \{s\}$) to output a new accumulator value A', a (possibly updated) membership witness π_i for each $s_i \in S$, and a membership witness for s'.

Remark 2. Traditionally, accumulators are not defined with respect to an ordering. We include this ordering to strengthen our lower bound: the accumulator value and membership proofs output by Add can be computed based not only on the set accumulated thus far, but also on the *order* in which they were accumulated and on the order in which future elements will be added.

An accumulator must satisfy soundness and correctness.

Definition 2 (Soundness). An accumulator $\Pi = (\text{Setup}, \text{Acc}, \text{Verify})$ is sound if it is computationally infeasible to find a verifying membership proof for an element not included in a set. More precisely, there exists a negligible function negl such that for all p.p.t. adversaries A:

$$\Pr\left[\begin{array}{c} S, \mathsf{ord}, u, \pi \leftarrow \mathcal{A}(1^{\lambda}) \\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, \mathsf{ord}) \\ \mathsf{Verify}(\mathsf{pp}, A, u, \pi) = \mathsf{true} : & u \notin S \\ A, (\pi_1, \dots, \pi_k) \leftarrow \mathsf{Acc}(\mathsf{pp}, S) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Definition 3 (Correctness). An accumulator $\Pi = (\text{Setup, Acc, Verify})$ is correct if an up-to-date membership proof π corresponding to an element u in accumulator A can always be used to verify the membership of u in A. More precisely, for all security parameter λ , any polynomially sized subset $S \subseteq U$, all elements $u \in U$ and any ordering function ord:

$$\Pr_{\mathsf{pp}\leftarrow\mathsf{Setup}(1^{\lambda},\mathsf{ord})}\left[\begin{array}{c}\mathsf{Verify}(\mathsf{pp},A,u,\pi)=\mathsf{true}:\\A,(\pi_{1},\ldots,\pi_{|S|+1})\leftarrow\mathsf{Acc}(\mathsf{pp},S\cup\{u\})\end{array}\right]=1.$$

where π is the membership proof corresponding to u with regard to the accumulator A.

One typically desires that an accumulator satisfies some notion of succinctness; that is, the commitment A grows sublinearly in n, the size of the committed set. We introduce some notation to capture the commitment size and define the succinctness requirement considered in this work, specifically that |A| is polylogarithmic in n.

Definition 4 (Accumulator size). Let Π = (Setup, Acc, Verify) be an accumulator with data universe \mathcal{U} . We write Size(Π , λ , n) to denote the maximum bit size of the accumulator output by Acc given a set of size n. That is,

$$\mathsf{Size}(\Pi, \lambda, n) := \max_{\substack{\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, *) \\ S \subseteq \mathcal{U} \\ |S| = n}} |A|, \text{ where } A, (\pi_1, \dots, \pi_n) \leftarrow \mathsf{Acc}(\mathsf{pp}, S).$$

Definition 5 (Succinct Accumulator). An accumulator $\Pi = (\text{Setup, Acc, Verify})$ is considered succinct if the maximum size of any accumulator output by Acc is polylogarithmic in the size of the accumulated set n. In other words, Π is succinct if there exist polynomials p and q such that for all n, $\text{Size}(\Pi, \lambda, n) \leq p(\lambda)q(\log n)$.

We also consider a slightly nonstandard notion of an accumulator for sets of superpolynomial size.

Definition 6 (Accumulator for superpolynomial-sized sets). An accumulator Π = (Setup, Acc, Verify) accommodates sets of size $f(\lambda)$ if it follows the syntax of a standard accumulator, and satisfies all of the following:

- Its algorithms Setup, Verify run in time $poly(\lambda)$. Its algorithm Acc runs in time $poly(\lambda, f(\lambda))$.
- Correctness (Definition 3) holds for all sets S of size at most $f(\lambda)$.
- Soundness (Definition 2) holds, even for all adversaries running in time $poly(\lambda, f(\lambda))$, producing sets S of size at most $f(\lambda)$; we formally define this property below.

Definition 7 (Soundness for accumulator of superpolynomial-sized sets). An accumulator is sound for sets of size $f(\lambda)$ if there exists a negligible function $negl(\lambda)$ such that for all p.p.t. adversaries A:

$$\Pr\left[\begin{array}{cc} S, \operatorname{ord}, u, \pi \leftarrow \mathcal{A}(1^{\lambda}, 1^{f(\lambda)}) \\ |S| \leq f(\lambda) \\ \operatorname{Verify}(\operatorname{pp}, A, u, \pi) = \operatorname{true} : \operatorname{pp} \leftarrow \operatorname{Setup}(1^{\lambda}, \operatorname{ord}) \\ u \notin S \\ A, (\pi_1, \dots, \pi_k) \leftarrow \operatorname{Acc}(\operatorname{pp}, S) \end{array} \right] \leq \operatorname{negl}(\lambda + f(\lambda)).$$

We emphasize that in the above soundness definition, the adversary can run in time polynomial in the maximum set size $f(\lambda)$, and the adversary's probability of success must be negligible in $f(\lambda)$.

When we write "accumulator," we mean a standard accumulator (Definition 1) unless we specify that it accommodates larger sets. Observe that an accumulator that accommodates sets of size $f(\cdot) \in poly(\cdot)$ under Definition 6, is simply a standard accumulator.

5 Our lower bound

5.1 Bounding the number of one-shot witness updates

Our ultimate goal is to lower bound the total number of witness changes across all elements. However, in our proof it will be useful to reason about a slightly different quantity: the number of elements whose witnesses change *at least once*. To do so, we define a number of *one-shot witness updates*; that is, the number of elements in a sequence whose witnesses generated upon initial addition no longer verify against the accumulator at the end of the sequence.

Definition 8 (One-shot witness updates). Let $\Pi = (\text{Setup}, \text{Acc}, \text{Verify})$ be an accumulator for a data universe \mathcal{U} . Let ord be an arbitrary ordering function and public parameters $pp \leftarrow \text{Setup}(1^{\lambda}, \text{ord})$. Let $S \in \mathcal{U}$ be a set of size n and $\text{ord}(S) = [s_1, \ldots, s_n]$. We say that S has k one-shot witness updates under pp if for at least k values of $i \in [n]$:

$$\begin{split} A_j, (\pi_1^{(j)}, \dots, \pi_j^{(j)}) &\leftarrow \mathsf{Acc}(\mathsf{pp}, \{s_1, \dots, s_j\}) \quad \forall j \in [n] \\ \mathsf{Verify}(\mathsf{pp}, A_n, s_i, \pi_i^{(i)}) &= \mathsf{false} \end{split}$$

Lemma 1 (One-shot witness changes). Let $\Pi = (\text{Setup}, \text{Acc}, \text{Verify})$ be a succinct accumulator for a data universe \mathcal{U} , and let ord be an arbitrary ordering function. Let $U \subseteq \mathcal{U}$ be any subset of size $n = \Omega(\text{poly}(\lambda))$. Let $S \subseteq U$ be a random subset. For any constant $c \in (0, 1)$,

$$\Pr_{\substack{\mathsf{pp}\leftarrow\mathsf{Setup}(1^{\lambda},\mathsf{ord})\\S \subseteq U}} [S \text{ has fewer than } c|S| \text{ one-shot witness updates given } \mathsf{pp}] \leq \mathsf{negl}(\lambda).$$

Proof. We'll show this via a compression argument. Alice will choose a random set $S \subseteq U$; we'll let k denote the size of S. If S has fewer than ck one-shot witness updates, Alice will be able to communicate S to Bob using fewer bits than possible.

Preprocessing. If Π is randomized, Alice and Bob agree on the random coins \$ they are going to use. Alice chooses an arbitrary ordering function ord and set $U \subseteq \mathcal{U}$. Let $[u_1, \ldots, u_n]$ denote the elements of U ordered according to ord. Alice generates public parameters $pp \leftarrow Setup_{\$}(1^{\lambda}, ord)$ and sends pp to Bob.

Then Alice chooses a secret random subset $S \subseteq U$; we denote its elements $[s_1, \ldots, s_k] := \operatorname{ord}(S)$. Let s be the length-n bit-encoding of S, whose 1's encode exactly which elements in $[u_1, \ldots, u_n]$ are included in S.

Encoding (Figure 2). Let $\epsilon \in (0, 1/2)$ be a small constant. If $k < n/2 - \epsilon n$, Alice simply sends (0, s) to Bob and halts.

Otherwise, Alice computes the accumulator:

$$A_k^S, (\tilde{\pi}_1, \dots, \tilde{\pi}_k) \leftarrow \mathsf{Acc}_{\$}(\mathsf{pp}, S)$$

She also needs to compute some additional information to send, which encodes the elements whose final witnesses are different from their witnesses when they were first added. To do so, she does the following:

She initializes an empty bit vector \boldsymbol{v} . For each $i \in [n]$, she computes

$$A_i, (\pi_1^{(i)}, \dots, \pi_i^{(i)}) \leftarrow \mathsf{Acc}_{\$}(\mathsf{pp}, (\{u_1, \dots, u_{i-1}\} \cap S) \cup \{u_i\})$$

and tests whether $\pi_i^{(i)}$ verifies against the final accumulator, i.e. $\operatorname{Verify}_{\$}(\operatorname{pp}, A_k^S, u_i, \pi_i^{(i)}) = \operatorname{true}$. If $u_i \notin S$ but it verifies, Alice has observed a soundness violation; she simply sends Bob (0, s) and halts. If $u_i \notin S$ and it does not verify, Alice appends a 0 to v. Otherwise, if $u_i \in S$ but it does not verify, Alice appends a 1 to v. Observe that the number of 1's in v is the number of elements in S whose witnesses change.

If there were at most ck witness changes, Alice sends $(1, \mathbf{sz}, A_k^S, \boldsymbol{v})$ to Bob, where $\mathbf{sz} := |A_k^S|$ helps Bob parse this string. Otherwise, Alice sends $(0, \boldsymbol{s})$.

Decoding (Figure 3). Let the information Bob receives from Alice be msg. Note that msg is either (0, s) or $(1, sz, A_k^S, v)$.

Bob first checks if msg begins with 0. If so, he interprets the rest as s and outputs the subset $S' \subseteq U$ that is represented by s.

Otherwise if msg begins with 1, Bob interprets the following sz bits as A_k^S and the rest as v. Bob initializes an empty sequence S' and repeats Alice's process to reconstruct S'.

That is, he iterates through each $i \in [n]$ while maintaining a pointer to v. Given the current portion of S' computed up to index i, he computes $\pi_i^{(i)}$ as Alice did and checks if it holds against A_k^S . If it verifies, he adds u_i to S', and moves onto i+1 maintaining his pointer to v. If it does not verify, he adds u_i to S' only if the current bit of v is 1. Regardless of the value of the current bit of v, he increments his pointer to v before moving onto i+1.

Correctness of decoding. If Alice chose a small subset, observed that soundness was violated, or observed too many witness changes, she sent (0, s). In this case, Bob interprets the encoding correctly.

Encoding algorithm. Given Π , \$, pp, $U, S, k := |S|, \epsilon, s \in [0, 1]^n$: 1: if $k < n/2 - \epsilon n$: // the chosen subset S is too small 2: return (0, s) $A_k^S, (\tilde{\pi}_1, \ldots, \tilde{\pi}_k) \leftarrow \mathsf{Acc}_{\$}(\mathsf{pp}, S)$ 3: $v \leftarrow []$ 4:for i in [1, ..., n]: 5: $A_i, (\pi_1^{(i)}, \dots, \pi_i^{(i)}) \leftarrow \mathsf{Acc}_{\$}(\mathsf{pp}, (\{u_1, \dots, u_{i-1}\} \cap S) \cup \{u_i\})$ 6: $ver_i \leftarrow \mathsf{Verify}_{\$}(\mathsf{pp}, A_k^S, u_i, \pi_i^{(i)}))$ 7:if $u_i \notin S$: 8: 9: if $ver_i == true :$ return $(0, s) \not \parallel$ soundness error else : v.append(0) 10:else if $u_i \in S$: 11: $\mathbf{if} \ ver_i == \mathbf{true} : continue \quad /\!\!/ \ u_i \in S'$ 12:else : v.append(1) // the witness of u_i changed during accumulation 13:if $\operatorname{wt}(v) > ck$: // weight of v is equal to number of one-shot witness updates 14: return (0, s)15: $\mathsf{sz} \leftarrow \left| A_k^S \right|$ 16: 17: return $(1, sz, A_k^S, v)$

Fig. 2. Encoding algorithm for Lemma 1.

```
Decoding algorithm. Given \Pi, $, pp, U, msg:
1: if msg[0] == 0:
          return bitparse(msg[1:]) // interprets the rest as s
 2:
3: parse msg to obtain A_k^S, \boldsymbol{v}
 4: S' \leftarrow []
 5: vptr \leftarrow 1
 6: for i in [1, \ldots, n]:
         A_i, (\pi_1^{(i)}, \ldots, \pi_i^{(i)}) \leftarrow \mathsf{Acc}_{\$}(\mathsf{pp}, S' \cup u_i)
 7:
         if Verify<sub>$</sub>(pp, A_k^S, u_i, \pi_i^{(i)})) :
 8:
9:
             S'.append(u_i)
10:
          else :
             if v[vptr] == 1:
11:
12:
                S'.append(u_i)
             vptr \leftarrow vptr + 1
13:
      return S'
14:
```

Fig. 3. Decoding algorithm for Lemma 1.

Otherwise, if Alice chose a subset of size at least $n/2 - \epsilon n$ and there were at most ck witness changes, she sent $(1, \mathbf{sz}, A_k^S, \mathbf{v})$.

Observe that Bob reconstructs S' correctly, since he exactly repeats Alice's process.

Amount of information. Let p be the probability that S has at most ck one-shot witness updates given pp. We'll bound Alice's communication in terms of p.

First, we remark that by a Chernoff bound, only with negligible probability do we have $k < n/2 - \epsilon n$. Let B_1, \ldots, B_n be independent Bernoulli random variables that are 1 with probability $\frac{1}{2}$. Each B_i represents whether Alice chose u_i to be in S; since S is uniformly random each u_i is in S with independently with probability $\frac{1}{2}$. Therefore, $\mathbb{E}[\sum_{i=1}^{n} B_i] = \frac{n}{2} := \mu$. Letting $\delta = 2\epsilon$, we have by Fact 1:

$$\Pr\left[k < n/2 - \epsilon n\right] = \Pr\left[\sum_{i=1}^{n} B_i \le (1-\delta)\mu\right] \le \exp\left(\frac{-\mu\delta^2}{2}\right) = \exp(-n\epsilon^2) \le \mathsf{negl}(\lambda).$$

Second, we remark that because Alice is efficient, she observes that soundness is violated with only negligible probability over **pp** and S. Otherwise, her encoding strategy would constitute an attack on soundness of the accumulator. Therefore, with probability at least $p - \operatorname{negl}(\lambda)$, S has at most ck witness updates and soundness holds. In this case, at least $(1-c)k \ge (1-c)(n/2 - \epsilon n)$ witnesses never changed. Observe that \boldsymbol{v} does not contain a corresponding bit for any element in S whose witnesses never changed. Therefore, $|\boldsymbol{v}| \le n - (1-c)(n/2 - \epsilon n)$, which is at most n - c'n for some constant $c' \in (0, 1)$.

Finally, observe that in any case the length of v is at most n. Slightly abusing notation, we let sz denote the size of the commitment sent by Alice. In expectation, Alice sends a message of length at most:

$$(p - \mathsf{negl}(\lambda))(1 + \mathsf{sz} + n - c'n) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{negl}(\lambda))(1 + n) \le n + 1 - (p - \mathsf{negl}(\lambda))(c'n - \mathsf{sz}) + (1 - p + \mathsf{sz}) + (1 - p + \mathsf{sz})(c'n - \mathsf{sz}) + (1 - p + \mathsf{s$$

Since p is non-negligible and Π is succinct, this is strictly less than n.

Impossible compression. We've thus shown that Bob always reconstructs Alice's sequence successfully, and if p is non-negligible, Alice sends strictly fewer than n bits in expectation. However, Alice communicated a uniformly random subset of n elements. Therefore, she must send Bob at least n bits in expectation by Shannon's Coding Theorem [40]; we've thus arrived at a contradiction.

5.2 Boosting from one-shot to many changes

Lemma 2 (Multi-shot witness changes). Let $\Pi = (\text{Setup}, \text{Acc}, \text{Verify})$ be a succinct accumulator for a data universe \mathcal{U} and ord be an arbitrary ordering function. Let the public parameters be $pp \leftarrow \text{Setup}(1^{\lambda}, \text{ord})$.

Let $X \subseteq \mathcal{U}$ be a random polynomial-sized subset, and let $[x_1, x_2, \ldots] \leftarrow \operatorname{ord}(X)$ denote the elements of X according to the ordering function. Let $A_j, (\pi_1^{(j)}, \ldots, \pi_j^{(j)}) \leftarrow \operatorname{Acc}(\operatorname{pp}, \{x_1, \ldots, x_j\})$ denote the accumulator and proofs after adding j elements from X according to the given ordering.

Consider any starting index $m_0 + 1$, subsequence size m, and degree d, and consider the following witnesses:

- The witnesses of $x_{m_0+1}, \ldots, x_{i+m}$ with respect to A_{m_0+m} . Call these witnesses $\pi^*_{m_0+1}, \ldots, \pi^*_{m_0+m}$
- The witnesses of $x_{m_0+m+1}, \ldots, x_{m_0+2m}$ with respect to A_{m_0+2m} . Call these witnesses $\pi^*_{m_0+m+1}, \ldots, \pi^*_{m_0+2m}$.
- ...
- The witnesses of $x_{m_0+(d-2)m+1}, \ldots, x_{m_0+(d-1)m}$ with respect to $A_{m_0+(d-1)m}$. Call these witnesses $\pi^*_{m_0+(d-2)m+1}, \ldots, \pi^*_{m_0+(d-1)m}$.
- The witnesses of $x_{m_0+(d-1)m+1}, \ldots, x_{m_0+dm}$ with respect to A_{m_0+dm} . Call these witnesses $\pi^*_{m_0+(d-1)m+1}, \ldots, \pi^*_{m_0+dm}$.

If $d = \omega(\text{Size}(\Pi, \lambda, m_0 + dm))$, then for any positive constant c < 1, at least a fraction c of these witnesses no longer hold with respect to A_{m_0+dm} with overwhelming probability¹³. That is, there is an overwhelming probability (over choice of **pp** and any randomness in the algorithms of the accumulator) that for at least cdm values of $j \in [dm]$,

Verify(pp, $A_{m_0+dm}, x_{m_0+j}, \pi^*_{m_0+j}$) = false.

Proof. This proof largely follows that of Lemma 1.

Preprocessing. Similar to the preprocessing phase in the proof of Lemma 1, if Π is randomized, Alice and Bob agree on the random coins \$ beforehand. Alice chooses an arbitrary ordering function ord. Alice generates public parameters $pp \leftarrow Setup_{\$}(1^{\lambda}, ord)$ and sends pp to Bob. Alice and Bob agree on some positive value m.

Alice randomly chooses a starting subset $X_0 \subseteq \mathcal{U}$ of size m_0 , and sends X_0 to Bob. Let $[x_1, \ldots, x_{m_0}] \leftarrow \operatorname{ord}(X_0)$. Alice then randomly chooses a secret uniform vector $s \leftarrow \{0, 1\}^d$. She then selects two random subsets $P, Q \subseteq \mathcal{U}$, each containing dm elements disjoint from X_0 and each other; let $[p_1, \ldots, p_{dm}] \leftarrow \operatorname{ord}(P), [q_1, \ldots, q_{dm}] \leftarrow \operatorname{ord}(Q)$. She sends P and Q to Bob.

To summarize, Bob has received pp (containing the ordering information), the random coins \$ if Π is randomized, and subsets X_0, P, Q .

Encoding (Figure 4). Alice computes:

$$X_{j} = x_{m_{0}+(j-1)m+1}, \dots, x_{m_{0}+jm} = \begin{cases} p_{(j-1)m+1}, \dots, p_{jm} \text{ if } s_{j} = 0\\ q_{(j-1)m+1}, \dots, q_{jm} \text{ if } s_{j} = 1 \end{cases} \quad \forall j \in [d]$$
$$A_{m_{0}+dm}^{s}, (\tilde{\pi}_{1}, \dots, \tilde{\pi}_{m_{0}+dm}) \leftarrow \mathsf{Acc}\left(\mathsf{pp}, \bigcup_{j=0}^{d} X_{j}\right)$$

She also needs to compute some additional information to send, which encodes the elements whose witnesses changed. To do so, Alice does the following:

She initializes an empty bit vector \boldsymbol{v} . For each $j \in [d]$, she computes

$$A_{m_{0}+jm}^{s0}, (\pi_{1}^{(m_{0}+jm,0)}, \dots, \pi_{m_{0}+jm}^{(m_{0}+jm,0)}) = \mathsf{Acc}\left(\mathsf{pp}, \bigcup_{i=0}^{j-1} X_{i} \cup \{p_{(j-1)m+1}, \dots, p_{jm}\}\right)$$

and tests whether any of the membership proofs verifies against the final accumulator, i.e.

Verify(pp, $A_{m_0+dm}^s, p_\ell, \pi_\ell^{(m_0+jm,0)}$) = true for some $\ell \in [m_0 + (j-1)m + 1, m_0 + jm]$. If $s_j \neq 0$, which means the current segment X_j should be from Q and not P, but some proofs verify, Alice has observed a soundness violation; she simply sends Bob (0, s) and halts. If $s_j \neq 0$ and none of the proofs verify, Alice appends a 0 to v. Otherwise, if $s_j = 0$, which means the current segment X_j is indeed from P, but none of the proofs verify, Alice appends a 1 to v.

Observe that the number of 1's in v is the number of segments that had *all* of their witnesses change. If at most *cd* segments had *all* of their witnesses change (i.e. there were at most *cdm* witness changes, $d - |S'| \leq cd$), Alice sends $(1, \mathsf{sz}, A^s_{m_0+dm}, v)$ to Bob (where $\mathsf{sz} = |A^s_{m_0+dm}|$ helps Bob parse this message). Otherwise, Alice just sends (0, s) to Bob.

Decoding (Figure 5). Let the information Bob receives from Alice be msg. Note that msg is either (0, s) or $(1, A^s_{m_s+dm}, v)$.

Bob first checks if msg begins with 0. If so, he interprets the rest as $s \in \{0,1\}^d$ and simply outputs that.

¹³ This requirement on d is met if, for example, $m_0 + dm$ is polynomial in λ , Π is succinct, and $d = \omega(\lambda)$.

Encoding algorithm. Given: Π , pp, U, d, m, s, P, Q, (X_0, \ldots, X_d) 1: $v \leftarrow []$ 2: $m_0 := |X_0|$ 3: $A^s_{m_0+dm}, (\tilde{\pi}_1, \dots, \tilde{\pi}_{m_0+dm}) \leftarrow \mathsf{Acc}(\mathsf{pp}, \bigcup_{j=0}^d X_j)$ 4: **for** j **in** [1, ..., d]: 5: $A_{m_0+jm}^{s0}, (\pi_1^{(m_0+jm,0)}, ..., \pi_{m_0+jm}^{(m_0+jm,0)}) = \mathsf{Acc}(\mathsf{pp}, \bigcup_{i=0}^{j-1} X_i \cup \{p_{(j-1)m+1}, ..., p_{jm}\})$ $ver_j \leftarrow \bigvee_{\ell=m_0+(j-1)m+1}^{m_0+jm} \mathsf{Verify}(\mathsf{pp}, A^s_{m_0+dm}, p_\ell, \pi_\ell^{(m_0+jm,0)})$ 6: if $s_j \neq 0$: // the current segment should not be from P 7:if $ver_j ==$ true : return (0, s) // soundness error 8: 9: else : v.append(0) 10: else if $s_j == 0$: // the current segment should be from P if $ver_j == \mathbf{true} : continue \ // j \in S'$ 11: else : v.append(1) // all witnesses of this segment changed during accumulation 12: if wt(v) > cd: 13:return (0, s)14: $\mathsf{sz} \leftarrow |A^s_{m_0+dm}|$ 15:return $(1, \mathbf{sz}, A^s_{m_0+dm}, \boldsymbol{v})$ 16:

Fig. 4. Encoding algorithm for Lemma 2.

Otherwise if msg begins with 1, Bob extracts sz and interprets the following sz bits as $A_{m_0+dm}^s$ and the rest as v. Bob initializes an empty bit vector s' and repeats Alice's process to reconstruct s'.

That is, Bob iterates through each $j \in [d]$ while maintaining a pointer to v. For each $j \in [d]$, Bob computes

$$A_{m_0+jm}^{s0}, (\pi_1^{(m_0+jm,0)}, \dots, \pi_{m_0+jm}^{(m_0+jm,0)}) = \mathsf{Acc}(\mathsf{pp}, \bigcup_{i=0}^{j-1} X_i \cup \{p_{(j-1)m+1}, \dots, p_{jm}\})$$

and tests whether $\operatorname{Verify}(\operatorname{pp}, A^s_{m_0+dm}, p_\ell, \pi_\ell^{(m_0+jm,0)})$ holds true for any $\ell \in [m_0 + (j-1)m + 1, m_0 + jm]$. If any of them verifies, Bob appends 0 to s', stores $X_j \leftarrow \{p_{(j-1)m+1}, \ldots, p_{jm}\}$, maintains his pointer to v, and moves onto j + 1. If none of them verify, Bob checks if the current bit of v is 1 and increments the pointer to v: if so, he appends 0 to s' and stores $X_j \leftarrow \{p_{(j-1)m+1}, \ldots, p_{jm}\}$; otherwise, he appends 1 to s' and stores $X_j \leftarrow \{p_{(j-1)m+1}, \ldots, p_{jm}\}$; otherwise, he appends 1 to s' and stores $X_j \leftarrow \{q_{(j-1)m+1}, \ldots, q_{jm}\}$.

Decoding algorithm. Given Π , pp, d, m, P, Q, X_0 , msg 1: $m_0 \leftarrow |X_0|$ 2: **if** msg[0] = 0: return msg[1 :] // interprets info received as s 3:parse msg to obtain $A^s_{m_0+dm}, \boldsymbol{v}$ 4:5: $s' \leftarrow []$ 6: $vptr \leftarrow 0$ 7:for $j \in [1, ..., d]$: $A_{m_0+jm}^{s0}, (\pi_1^{(m_0+jm,0)}, \dots, \pi_{m_0+jm}^{(m_0+jm,0)}) = \mathsf{Acc}(\mathsf{pp}, \bigcup_{i=0}^{j-1} X_i \cup \{p_{(j-1)m+1}, \dots, p_{jm}\})$ 8: $\text{if} \; \bigvee_{\ell=m_0+(j-1)m+1}^{m_0+jm} \text{Verify}(\text{pp}, A^s_{m_0+dm}, p_\ell, \pi_\ell^{(m_0+jm,0)}):$ 9: 10: s'.append(0) // Some witness verifies, indicating this segment is from P $X_j \leftarrow \{p_{(j-1)m+1}, \ldots, p_{jm}\}$ 11:else : 12: if $v_{vptr} == 1$: // This segment is from P but all its witnesses changed during accumulation 13:s'.append(0) 14: $X_j \leftarrow \{p_{(j-1)m+1}, \dots, p_{jm}\}$ 15: elseif $v_{vptr} == 0$: // This segment is from Q 16: s'.append(1) 17: $X_j \leftarrow \{q_{(j-1)m+1}, \ldots, q_{jm}\}$ 18: $\mathsf{vptr} \leftarrow \mathsf{vptr} + 1$ 19:return s'20:

Fig. 5. Decoding algorithm for Lemma 2.

Correctness of decoding. If Alice observed that soundness was violated, or observed too many witness changes, she sent exactly the *d*-bit vector (0, s). In this case, Bob interprets the vector correctly.

Otherwise, if no soundness violation was observed and at most cd segments had all of their witnesses changed, Alice sends $(1, A^s_{m_0+dm}, v)$. Bob can correctly parse this information given the Size he received in the preprocessing phase. Observe that Bob reconstructs s' correctly, since he exactly repeats Alice's process.

Amount of information. Let p be the probability that at most cd segments among X_1, \ldots, X_d had all of their witnesses updated given pp. We will bound Alice's communication in terms of p.

First, we remark that because Alice is efficient, she observes that soundness is violated with only negligible probability over **pp** and **s**. Otherwise, her encoding strategy would constitute an attack on soundness of the accumulator. Therefore, with probability at least $p - \operatorname{negl}(\lambda)$, at most cd segments among X_1, \ldots, X_d had all of their witnesses updated *and* soundness holds. In this case, at least $(1 - c)\operatorname{wt}(s)$ segments have at least one witness that verifies against A_{i+dm}^s . Therefore, v has length at most d - (1 - c)d = cd.

Finally, observe that in any case the length of v is at most d. Slightly abusing notation, we let sz denote the size of $A^s_{m_s+dm}$, the commitment sent by Alice. In expectation, Alice sends a message of length at most:

 $(p - \mathsf{negl}(\lambda))(\mathsf{sz} + cd) + (1 - p + \mathsf{negl}(\lambda))(d) \le d - (p - \mathsf{negl}(\lambda))(d - cd - \mathsf{sz})$

Observe that if p is non-negligible, this is strictly less than d if d - cd - sz > 0. This is satisfied, due to the requirement that $d = \omega(\text{Size}(\Pi, \lambda, m_0 + dm))$.

Impossible compression. We've thus shown that Bob always reconstructs Alice's sequence successfully, and if p is non-negligible, Alice sends strictly fewer than d bits in expectation. However, Alice communicated a uniformly random string of size d. Therefore, she must send Bob at least d bits in expectation by Shannon's Coding Theorem [40]; we've thus arrived at a contradiction.

Corollary 1. If Π is an accumulator for superpolynomial $f(\lambda)$ -sized sets (Definition 6), Lemma 2 holds for sets $X \subseteq \mathcal{U}$ of size at most $f(\lambda)$, where the "overwhelming probability" with which fewer than cdm witnesses change is $1 - \operatorname{negl}(\lambda + f(\lambda))$.

Proof. This follows from observing that the probability of a soundness violation is at most $\operatorname{negl}(\lambda + f(\lambda))$, by definition of an accumulator for superpolynomial-sized sets (Definition 7).

Theorem 1. Let $\Pi = (\text{Setup}, \text{Acc}, \text{Verify})$ be a succinct accumulator for a data universe \mathcal{U} . Let ord be an arbitrary ordering function. For any constant $\alpha \geq 0$, for sufficiently large n,

$$\Pr_{\substack{X \subseteq \mathcal{U}: |X| = n \\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, \mathsf{ord})}} [there \ are \ at \ least \ \alpha n \ witness \ updates \ given \ \mathsf{pp}] \ge 1 - \mathsf{negl}(\lambda)$$

Proof. We set parameters $d \ge \lambda^2$, $h = \alpha/0.99$, and $n = d^h$. Observe that n is polynomial in λ .

Let ord be an arbitrary ordering function. Select a random *n*-sized subset $X \subseteq \mathcal{U}$ and let $\operatorname{ord}(X) = [x_1, \ldots, x_n]^{.14}$ Consider breaking X into segments according to the given order, which are contiguous subsets of size d^i for $i \in \{0, \ldots, h\}$. We will use $X_{d^i}^{(\ell)}$ to denote the ℓ^{th} segment of size d^i . That is, for all $\ell \in [\frac{n}{d^i}], X_{d^i}^{(\ell)} = \{x_{(\ell-1)d^i+1}, \ldots, x_{\ell d^i}\}$. It will be helpful to think of these segments as belonging to levels, with Level *i* containing the segments of length d^i . We define $|L_i|$ to mean the number of segments in Level *i*; by construction we have that $|L_i| = \frac{n}{d^i}$.

Let A_m denote the accumulator after adding m elements from X according to the given ordering. Let Level i contain the accumulators that accumulate d more segments from the Level i - 1; in other words, Level i contains $A_{d^i}, A_{2d^i}, A_{3d^i}, \ldots$, where A_{jd^i} accumulates $\bigcup_{\ell=1}^{j} X_{d^{i-1}}^{(\ell)}$. The number of accumulators per level is trivially equal to the number of segments.

Let L_0 be the bottom-most level and L_h the top-most. At the topmost level L_h , there is only one accumulator A_n which accumulates all elements in X. See Figure 6 for a visual illustration of this hierarchical structure.

 $^{^{14}}$ We now use sets and sequences interchangeably; assume that a given set is ordered according to ord.



Fig. 6. Illustration of the levels in the proof for Theorem 1

For any $i \in \{0, \ldots, h-1\}$ and any $j \in |L_i|$, consider $A_{jd^{i+1}}$, the j^{th} accumulator in Level i + 1. Notice that $A_{jd^{i+1}}$ accumulates d more lower-level segments, namely the segments of size d^i , compared to $A_{(j-1)d^{i+1}}$. We consider how many of the witnesses from the corresponding d segments and accumulators in the lower level L_i no longer hold with respect to $A_{jd^{i+1}}$. Let $m_0 := (j-1)d^{i+1}$. These segments are disjoint, and we take care not to double count their witness changes. Precisely, the witnesses we consider are the following:

- The witnesses of segment $x_{m_0+1}, \ldots, x_{m_0+d^i}$ with respect to $A_{m_0+d^i}$. Call these witnesses $\pi^*_{m_0+1}, \ldots, \pi^*_{m_0+d^i}$.
- The witnesses of segment $x_{m_0+d^i+1}, \ldots, x_{m_0+(j-1)d^i}$ with respect to $A_{m_0+2d^i}$. Call these witnesses $\pi^*_{m_0+d^i+1}, \ldots, \pi^*_{m_0+(j-1)d^i}$.
- ...
- The witnesses of segment $x_{m_0+(d-1)d^{i+1}}, \ldots, x_{jd^{i+1}}$ with respect to $A_{m_0+(d)d^i} = A_{jd^{i+1}}$. Call these witnesses $\pi^*_{m_0+(d-1)d^{i+1}}, \ldots, \pi^*_{jd^{i+1}}$.

Succinctness implies that the size of each of these accumulators is $O(\operatorname{polylog}(n))$. Since $n = O(\operatorname{poly}(d))$, these accumulators have size at most $O(\operatorname{polylog}(d))$ and are therefore succinct in terms of the current subsequence length. Therefore, we can apply Lemma 2 with c = 0.99, recalling that c is the fraction of elements whose witnesses change. We obtain that with only negligible probability do fewer than cd^{i+1} witnesses of the lower-level d^i -sized segments change with regard to $A_{jd^{i+1}}$. By a union bound, we know the probability that fewer than cd^{i+1} witnesses in the corresponding d^i -sized segments changed for any $j \in |L_{i+1}|$ is at most negligible. Therefore, by another union bound, there is overwhelming probability that the total number of witness changes observed on L_i is $cd^{i+1}|L_{i+1}| = cd^{i+1}\left(\frac{n}{d^{i+1}}\right) = cn$ for every $i \in [h]$.

We now compute the number of witness changes that we've shown across all levels. As there are h levels, we have that at least 0.99hn total witness changes throughout the entire process of accumulation. Recalling that $h = \alpha/0.99$, we have completed the proof that at least αn witnesses change with overwhelming probability.

Corollary 2. Let $\Pi = (\text{Setup, Acc, Verify})$ be a succinct accumulator for sets of superpolynomial size n (as in Definition 6), for a data universe \mathcal{U} . Let ord be an arbitrary ordering function. There exists some constant α such that for any constant $c \in (0, 1)$, for sufficiently large n,

$$\Pr_{\substack{X \subseteq \mathcal{U}: |X| = n \\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, \mathsf{ord})}} \left[\text{there are at least } \frac{c\beta n \log n}{\alpha \log \log n} \text{ witness updates given } \mathsf{pp} \right] \ge 1 - \mathsf{negl}(\lambda).$$

Proof. Since Π is succinct, there must exist some polynomials p, q such that the size of the commitment to a set of size n is at most $p(\lambda)q(\log n)$ (Definition 5). Let $d = (p(\lambda)q(\log n))^2$, $h = \log n/\log d$, and therefore $n = d^h$. Note that here $n = f(\lambda)$ for some superpolynomial function f. Repeat the proof of Theorem 1, invoking Corollary 1 rather than Lemma 2. (Corollary 1 is simply the analogue of Lemma 2, for accumulators

accommodating large sets). The fact that the probability of error in Corollary 1 is negligible in $n = f(\lambda)$ rather than just negligible in λ allows us to take a union bound over a polynomial in n number of events and obtain a probability that is still negligible. Observe that

$$\log d = 2 \log(p(\lambda)q(\log n))$$

= 2(\alpha_p \log \log \lambda + \alpha_q \log \log \log n)
= (2\alpha_p/\beta) \log \log n + 2\alpha_q \log \log n
\$\leq (\alpha/\beta) \log \log n\$

where α_p, α_q , and α are constants that depend only on p and q. We conclude that the total number of witness updates is at least $chn = \frac{cn \log n}{\log d} \ge \frac{c\beta n \log n}{\alpha \log \log n}$.

6 Merkle Mountain Ranges are essentially optimal

In this section we present generalized Merkle Mountain Ranges with any degree k, and show that they are essentially optimal given our lower bound. k-ary Merkle Mountain Ranges have been considered in previous work [30], which we recall here for completeness.

6.1 k-ary Merkle Mountain Range

To accumulate a sequence of elements $X = x_1, \ldots, x_n$, one computes a list of complete k-ary Merkle trees storing disjoint subsequences of S at their leaves. The number of Merkle trees of each size is equal to the corresponding digit of the k-ary representation of n. The accumulator is defined to be the list of these Merkle roots, in decreasing order of sizes of their trees. The membership witness of an element consists of the sibling nodes along the path to its Merkle root. To verify a membership witness, one checks whether it is a valid Merkle inclusion proof for any root in the commitment.

To add a new element, one first computes a (trivial) tree containing only that element. One then iteratively checks if there are any k Merkle roots of the same size; if there are, they are merged into a single tree and these new sibling nodes are added to their membership witnesses. One continues this merging process until there are no more tuples of k trees to merge.

6.2 Upper bounding the number of witness updates

We show in Theorem 2 that a k-ary Mountain Range is succinct and has a relatively small asymptotic upper bound to the number of total witness updates. Reyzin and Yakoubov proved a similar result for the special case of k = 2 [39, Theorem 2].

Theorem 2. A k-ary Merkle Mountain Range is succinct and requires $O(n \log_k n)$ witness updates when accumulating n elements.

Proof. To show that the construction is succinct, we can see that accumulating n elements requires creating trees of at most $\lceil \log_k n \rceil$ unique sizes $(1, k, k^2, \dots, k^{\lfloor \log_k n \rfloor})$. There are at most k - 1 trees of any one unique size, leading to a maximum of $(k-1)\lceil \log_k n \rceil$ trees. This bound is hit exactly when accumulating $n = k^m - 1$ elements for any integer m. Thus, the commitment size is $O(k \log_k n)$, meeting the definition of succinctness.

Any individual element's proof changes only when the tree it is a member of is merged. Each merger takes k trees of size k^m and combines them into a new sub-accumulator of size k^{m+1} , updating all of the included elements' proofs in the process with overwhelming probability. Hence, the items currently in the largest trees have been involved in the most merges (and have had the most witness updates). Since the largest tree is of size $k^{\lfloor \log_k n \rfloor}$, each of the elements in this tree were previously in accumulators of size $1, k, k^2, \ldots k^{\lfloor \log_k n \rfloor - 1}$, and thus have had their witness updated $\lfloor \log_k n \rfloor$ times. Since this is the maximal number of merges for any element and there are n elements, the total number of witness changes is $O(n \log_k n)$.

Corollary 3. A k-ary Merkle Mountain Range with $k = \log^{c} n$ will require $O(\log^{c+1} n)$ storage and $O(\frac{n \log n}{c \log \log n})$ witness updates to accumulate n elements.

Proof. As shown in the proof for Theorem 2, the accumulator size for a k-ary Merkle Mountain Range is $O(k \log_k n) = O(k \log n) = O(\log^{c+1} n)$. The total number of witness updates is

$$O(n \log_{\log^c n} n) = O(\frac{n \log n}{\log \log^c n}) = O(\frac{n \log n}{c \log \log n})$$

6.3 Optimality of Merkle Mountain Ranges

Consider k-ary Merkle Mountain Ranges that accumulate superpolynomial-sized sets of size at most $2^{\lambda^{\beta}}$ for any constant $\beta < 1$ (Definition 6).

Remark 3. Such Merkle Mountain Ranges are secure in the Random Oracle Model; that is, the underlying hash function is represented as a random function with range $\{0,1\}^{\lambda}$. Soundness holds because breaking soundness of the MMR requires finding a hash collision, which happens with probability at most $\mathsf{poly}(2^{\lambda^{\beta}})$.

 $2^{-\lambda}$ by an adversary running in time polynomial in $2^{\lambda^{\beta}}$, satisfying the definition of soundness for accumulator for large sets as in Definition 7.

Recall from Corollary 2, we know the lower bound of the number of witness updates in MMRs accumulating large sets is $\frac{c\beta n \log n}{\alpha \log \log n}$ for some constant α and constant $c \in (0, 1)$ with overwhelming probability. This lower bound is asymptotically tight for $\log n$ -ary Merkle Mountain Ranges, since they require at most $O(\frac{n \log n}{\log \log n})$ witness updates (Corollary 3). We conclude that MMRs are essentially optimal regarding the number of witness updates when accumulating large sets.

Acknowledgments. We would like to thank Leonid Reyzin for inspiring conversations on Merkle Mountain Ranges and the compression argument. We thank Hossein Hafezi for his contributions on the early stages of this work. Joseph Bonneau was additionally supported by DARPA Agreement HR00112020022 and NSF Grant CNS-2239975. Miranda Christ was supported by a Google CyberNYC grant, an Amazon Research Award, and NSF grants CCF-2312242, CCF-2107187, and CCF-2212233. The views and conclusions contained in this material are those of the authors and do not necessarily reflect the official policies or endorsements of the United States Government, DARPA, a16z Crypto, or any other supporting organization.

References

- 1. Agrawal, S., Raghuraman, S.: Kvac: key-value commitments for blockchains and beyond. In: Asiacrypt (2020)
- Au, M.H., Tsang, P.P., Susilo, W., Mu, Y.: Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In: CT-RSA (2009)
- 3. Axiom (1 2024), https://docs.axiom.xyz/
- 4. Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., Yakoubov, S.: Accumulators with applications to anonymity-preserving revocation. In: IEEE EuroS&P (2017)
- 5. Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Eurocrypt (1993)
- Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica 12, 225–244 (1994)
- 7. Boneh, D., Bünz, B., Fisch, B.: Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In: CRYPTO. Cham (2019)
- 8. Boyle, E., Komargodski, I., Vafa, N.: Memory checking requires logarithmic overhead. In: ACM TOC (2024)
- Bünz, B., Kiffer, L., Luu, L., Zamani, M.: Flyclient: Super-light clients for cryptocurrencies. In: IEEE Security & Privacy (2020)
- 10. Camacho, P., Hevia, A.: On the impossibility of batch update for cryptographic accumulators. In: Latincrypt (2010)
- 11. Camenisch, J., Lysyanskaya, A.: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: CRYPTO. Berlin, Heidelberg (2002)
- 12. Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC (2013)
- 13. Christ, M., Bonneau, J.: Limits on revocable proof systems, with applications to stateless blockchains. In: Financial Crypto (2022)
- Compact Ranges. Google Transparency project (2024), https://github.com/transparency-dev/merkle/blob/ main/docs/compact_ranges.md
- 15. Crosby, S.A., Wallach, D.S.: Efficient data structures for tamper-evident logging. In: USENIX Security (2009)
- Damgård, I., Triandopoulos, N.: Supporting Non-membership Proofs with Bilinear-map Accumulators. Cryptology ePrint Archive, Report 2008/538 (2008)
- 17. Data Trails (2024), https://docs.datatrails.ai/developers/developer-patterns/navigating-merklelogs/
- Dwork, C., Naor, M., Rothblum, G.N., Vaikuntanathan, V.: How efficient can memory checking be? In: TCC (2009)
- 19. Explaining FlyClient. Electronic Coin Company blog (3 2020), https://electriccoin.co/blog/ explaining-flyclient-2/
- Fiore, D., Kolonelos, D., de Perthuis, P.: Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces. In: Asiscrypt (2023)
- Garg, S., Hajiabadi, M., Mahmoody, M., Rahimi, A.: Registration-based encryption: removing private-key generator from IBE. In: TCC (2018)
- Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. In: ACM CCS (2020)
- 23. Greenwald, S., Haueter, R., Alexander, S.: Seac. IRE **41**(10) (1953)
- 24. Merkle structures. Grin project (2024), https://docs.grin.mw/wiki/chain-state/merkle-structures%28old% 29/
- 25. Herodotus. Herodotus Project (2024), https://docs.herodotus.dev/herodotus-docs
- 26. Karantaidou, I., Baldimtsi, F.: Efficient constructions of pairing based accumulators. In: IEEE CSF (2021)
- 27. Lai, R.W., Malavolta, G.: Subvector commitments with application to succinct arguments. In: CRYPTO (2019)
- 28. Leung, D., Gilad, Y., Gorbunov, S., Reyzin, L., Zeldovich, N.: Aardvark: An asynchronous authenticated dictionary with applications to account-based cryptocurrencies. In: USENIX Security (2020)
- 29. Li, J., Li, N., Xue, R.: Universal Accumulators with Efficient Nonmembership Proofs. In: ACNS. Berlin, Heidelberg (2007)
- 30. Liang, C., Zhang, J., Ma, S., Zhou, Y., Hong, Z., Fang, J., Zhou, Y., Tang, H.: Study on data storage and verification methods based on improved Merkle mountain range in IoT scenarios. Journal of King Saud University-Computer and Information Sciences 36(6) (2024)
- 31. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Eurocrypt (1987)
- Miller, A., Hicks, M., Katz, J., Shi, E.: Authenticated data structures, generically. In: Jagannathan, S., Sewell, P. (eds.) The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014. pp. 411–424. ACM (2014). https://doi.org/10.1145/2535838.
 2535851, https://doi.org/10.1145/2535838.2535851

- 33. Minima (2024), https://docs.minima.global/docs/about/welcome
- Naor, M., Rothblum, G.N.: The complexity of online memory checking. Journal of the ACM (JACM) 56(1), 1–46 (2009)
- 35. Nguyen, L.: Accumulators from Bilinear Pairings and Applications. In: CT-RSA. Berlin, Heidelberg (2005)
- Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables based on cryptographic accumulators. Algorithmica 74, 664–712 (2016)
- 37. Peikert, C., Pepin, Z., Sharp, C.: Vector and functional commitments from lattices. In: TCC (2021)
- 38. Picasso (2024), https://docs.picasso.network/
- 39. Reyzin, L., Yakoubov, S.: Efficient Asynchronous Accumulators for Distributed PKI. In: SCN (2016)
- 40. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal 27(3), 379-423 (1948). https://doi.org/https://doi.org/10.1002/j.1538-7305.1948.tb01338.x, https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x
- Srinivasan, S., Chepurnoy, A., Papamanthou, C., Tomescu, A., Zhang, Y.: Hyperproofs: Aggregating and maintaining proofs in vector commitments. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 3001– 3018 (2022)
- 42. Szepieniec, A., Værge, T.: Neptune White Paper (1 2024), https://neptune.cash/whitepaper/
- 43. Tas, E.N., Boneh, D.: Vector Commitments with Efficient Updates. In: AFT (2023)
- 44. Todd, P.: Merkle Mountain Ranges. OpenTimestamps (March 2013), https://github.com/opentimestamps/ opentimestamps-server/blob/master/doc/merkle-mountain-range.md
- 45. Todd, P.: Making UTXO Set Growth Irrelevant With Low-Latency Delayed TXO Commitments. Bitcoin Devs mailing list (May 2016), https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2016-May/012715. html
- Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., Khovratovich, D.: Aggregatable subvector commitments for stateless cryptocurrencies. In: Security and Cryptography for Networks: 12th International Conference, SCN 2020, Amalfi, Italy, September 14–16, 2020, Proceedings 12. pp. 45–64. Springer (2020)
- 47. Tomescu, A., Bhupatiraju, V., Papadopoulos, D., Papamanthou, C., Triandopoulos, N., Devadas, S.: Transparency logs via append-only authenticated dictionaries. In: ACM CCS (2019)
- 48. Tomescu, A., Xia, Y., Newman, Z.: Authenticated dictionaries with cross-incremental proof (dis) aggregation. Cryptology ePrint Archive (2020)
- 49. Tyagi, N., Fisch, B., Zitek, A., Bonneau, J., Tessaro, S.: VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries. In: ACM CCS (2022)
- 50. Wang, W., Lu, Y., Papamanthou, C., Zhang, F.: The locality of memory checking. In: ACM CCS (2023)
- 51. Witness (2024), https://docs.witness.co/
- 52. Xie, J.: Nervos CKB: A Common Knowledge Base for Crypto-Economy. Nervos Network (1 2018)