Tight Lower Bound on Witness Update Frequency in Additive Positive Accumulators

Wei Qi

Bocconi University, Department of Computing Sciences, Milan, Italy

Abstract. We study additive positive accumulators, which maintain a short digest of a growing set such that each value in the set can prove membership via a generated witness. Due to the compactness of the digest, previously added values may require updated witnesses as the set grows.
In this paper, we establish a trade-off between the bit-length of the accumulator value and the number of witness updates. Specifically, we show that if the accumulator value has bit-length $\mathsf{poly}(\log n)$, where n is the number of accumulated values, then some values must incur $\Omega(\log n/\log\log n)$ witness updates. This improves upon the recent $\omega(1)$ lower bound of [BCCK25] and matches the upper bound in [MQ23].
Building on the framework of [MQR22], we introduce a new combinatorial structure that removes the fixed-update-time assumption. Our approach also applies to Registration-based Encryption [GHMR18], thereby resolving the open problem left in [MQR22]: it shows that the tight lower bound on decryption-update frequency continues to hold even without any fixed-update-time assumption.

Keywords: Cryptographic Accumulators · Lower Bounds · Witness Updates

1 Introduction

A cryptographic accumulator scheme [BdM94, CL02, Ngu05, BP97] generates a *compact* digest acc_S, called the accumulator value, for a set of values $S = \{x_1, \ldots, x_n\}$. If each value $x_i \in S$ can prove its membership using a witness w_i, the accumulator is called *positive*. If a value $x \notin S$ can prove its non-membership via a non-membership witness, the accumulator is called *negative*. In this paper, we focus exclusively on positive accumulators.

An accumulator is called *dynamic* if the set S can evolve over time. In this work, we study *additive* positive accumulators, where the set S grows monotonically: new elements are added over time, and the accumulator value is updated accordingly. Figure 1 presents an example execution of such an additive positive accumulator. Furthermore, we assume that the underlying data universe $\mathcal{X}$ is exponentially large (e.g., $|\mathcal{X}| = 2^\lambda$ for security parameter λ).

Compactness and witness update frequency. We emphasize that it is required that the accumulator value acc_n remain compact with respect to n, the number of accumulated values. Typically, this means the bit-length satisfies $|\mathsf{acc}_n| = o(n)$. Otherwise, a trivial construction becomes possible: one can simply define the accumulator value as the concatenation of all elements in the set $S = \{x_1, \ldots, x_n\}$, that is, $\mathsf{acc}_S = x_1 \| \cdots \| x_n$. As a consequence of the compactness requirement, a previously accumulated value may require its membership witness to be updated after new elements are added to the set, in order to ensure that its membership can still be correctly verified against the updated accumulator value.

E-mail: weiqicsu@outlook.com (Wei Qi)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-01-01.

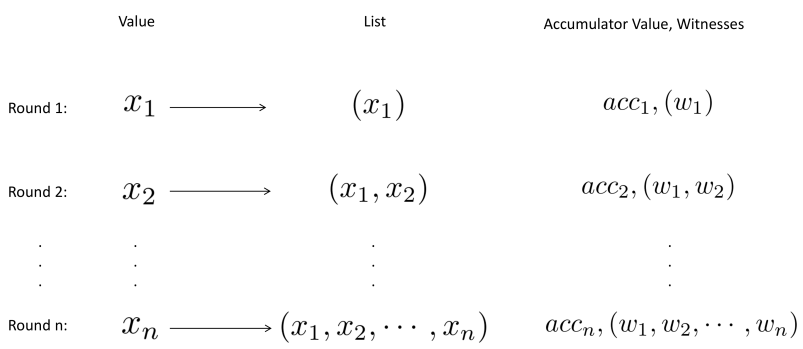

Figure 1: At round i, a new value $x_i \notin S_{i-1}$ is added to the set S_{i-1} to form the updated set $S_i = S_{i-1} \cup \{x_i\}$. The accumulator value acc_{i-1} is then updated to acc_i to reflect the new set. A list of i membership witnesses is generated, corresponding to the i elements in S_i.

Collection of Merkle Trees. A classical construction of cryptographic accumulators is based on Merkle trees [Mer88]. In this approach, a binary hash tree is constructed with the elements of the accumulated set as its leaves, and the root hash is published as the accumulator value. A membership witness for a value consists of the path from the corresponding leaf to the root.

However, a single Merkle tree is poorly suited for a growing set: adding a new value requires rebuilding the entire tree, which invalidates all existing witnesses. A more scalable solution is to maintain a collection of Merkle trees. Specifically, in each round, a new value is added as a singleton tree. Whenever two trees of the same height exist, they are merged to form a tree of greater height. This merging process continues until no further pairs of equal-height trees remain. The accumulator value is then defined as the list of root hashes of the remaining trees.

This design ensures that the accumulator value remains compact, with bit-length $\mathsf{poly}(\log n)$, and that each element requires at most $O(\log n)$ witness updates over the course of n additions.

Tight bounds in Registration-based Encryption. The paradigm of maintaining a collection of Merkle trees has also been applied in other contexts, such as distributed PKI [RY16] and Registration-based Encryption (RBE) [GHMR18, GHM+19, GV20, CES21, GKMR23, FKP23, DKL+23].

Registration-based Encryption is an advanced public key encryption scheme where each user generates her own public key and secret key and registers to the system to generate a public parameter pp. More specifically, at each round i, a user sends her public key pk_i and identity id_i to a transparent key curator (KC) who updates the current public parameter pp_{i-1} to pp_i. One can encrypt to the i-th user using only pp_j (any $j \geq i$) and id_i and the i-th user can decrypt using her own secret key sk_i. Since the public parameter is compact, from time to time, previously registered users might require a decryption update u to successfully run the decryption algorithm. A lower bound on the frequency of decryption updates was established in [MQR22], and a matching upper bound [MQ23] was later achieved using a generalization of Merkle trees, in which nodes may have more than two children.

Beyond the fixed update time assumption. The fixed-update-time assumption is convenient for analysis and is satisfied by many known RBE constructions: their internal

data structures (such as Merkle trees [GHMR18, GHM+19, GV20, DKL+23, DMQ25], crit-bit trees [CES21], or vector-commitment structures [GKMR23]) evolve in ways that do not depend on the specific identities or keys being registered, so the times at which membership or decryption updates are needed depend only on the position i of a user and not on her actual identity and key.

However, this behavior is not inherent to the model. In some designs, the internal data structure used to index identities may evolve in a value-dependent manner. For example, constructions that employ cuckoo-hash–based indexing (as in [FKP23]) may trigger relocations of previously inserted elements when collisions occur; such relocations can, in principle, induce value-dependent update patterns depending on how the underlying building blocks are instantiated. Whether this value dependence arises in practice is scheme-specific, and it can often be avoided. For instance, one may apply the generic transformation in [DKL+23] to decouple update scheduling from the inserted values.

Crucially, our lower bound shows that allowing update times to depend on the registered values does not reduce the number of updates required for soundness.

Our contribution. Our work generalizes the lower-bound framework of [MQR22] by removing the fixed-update-time assumption through a new combinatorial structure called the *falling-step sequence*. As a consequence, we prove the same asymptotic lower bound as [MQR22], but in the more general context of additive positive accumulators. In particular, we establish the following trade-off between the accumulator length and the number of witness updates—a bound that is tight, as it matches the upper bound of [MQ23].

Theorem 1 (Informal)**.** *Let Λ be an additive positive accumulator scheme in which, after n values have been accumulated, each value requires at most $d(n)$ witness updates. Suppose the bit-length of the accumulator value acc_n is non-decreasing in n. Then the following bound holds:*

$$\binom{|\mathsf{acc}_n| + d(n) + 1}{d(n) + 1} > n.$$

In particular, we have that:

- *If the accumulator value satisfies $|\mathsf{acc}_n| \leq \mathsf{poly}(\log n)$, then it must hold that*

$$d(n) = \Omega\left(\frac{\log n}{\log \log n}\right).$$

- *If $d(n) \leq d$ for some fixed positive integer d, then it must hold that*

$$|\mathsf{acc}_n| = \Omega\left(n^{\frac{1}{d+1}}\right).$$

This result improves upon the recent $\omega(1)$ lower bound of [BCCK25] for the case where $|\mathsf{acc}_n| \leq \mathsf{poly}(\log n)$. Moreover, since the same combinatorial framework applies directly to Registration-based Encryption (RBE), it resolves the open problem posed in [MQR22]: the tight lower bound continues to hold even when update times may depend arbitrarily on the identities and keys.

1.1 Related Works

Comparison with algebraic accumulators. Most practical accumulators are either tree-based or algebraic. The motivation for this study comes from tree-based accumulators, such as Merkle trees [Mer88] and the Merkle-based constructions used in RBE. However, our trade-off applies to any public additive accumulator, so it is natural to also discuss algebraic schemes such as RSA-based accumulators [CL02] and pairing-based accumulators [Ngu05].

Compared to tree-based constructions, algebraic accumulators achieve extremely short representations: both the accumulator value and each membership witness are single group elements. However, in the classic dynamic constructions [CL02, Ngu05], the witness for an element is an algebraic function of all previously accumulated values; consequently, inserting a new value typically requires all (or nearly all) previously issued witnesses to be recomputed.

There also exist trapdoor-based algebraic accumulators (e.g., [BCD+17, KB21]) in which a trusted party maintains secret state that enables witness updates to be avoided *entirely* after each addition. Such constructions fall outside the scope of our model, which concerns public accumulators whose update behavior is fully determined by the sequence of additions and does not rely on hidden trapdoor information. To the best of our knowledge, no public-state algebraic accumulator achieves a number of witness updates that is sublinear in the number of added values.

Lower bounds on witness update frequency in additive positive accumulators. Recently, [BCCK25] established a lower bound on the witness update frequency in additive positive accumulators using a compression argument. Specifically, the authors showed that after adding n elements, the total number of witness updates *across all values* must be $\omega(n)$, which implies an $\omega(1)$ lower bound on the number of updates for individual elements.

Lower bound on decryption update frequency in RBE. In the context of RBE, [MQR22, MQ23] proved a tight trade-off between the bit-length of the public parameter (which corresponds to the accumulator value) and the number of decryption updates (analogous to witness updates). However, the proof in [MQR22] relies on the assumption that update times are independent of the accumulated values (see Section 2.2 for further discussion). In this work, we strengthen the techniques of [MQR22] by introducing a new combinatorial structure, called the falling-step sequence, which enables us to establish the same lower bound without imposing any assumptions on how updates are scheduled.

2 Technical Overview

An additive positive accumulator scheme over an exponentially large data universe $\mathcal{X}$ (e.g., $|\mathcal{X}| = 2^\lambda$ for security parameter λ) maintains a compact digest acc_n for a growing list of distinct values $S = (S[1], \cdots, S[n])$, such that for any $i \in [n]$, the membership of $S[i]$ can be efficiently verified using a correctly generated witness w.

We require that the accumulator remain compact, meaning that its size grows only polylogarithmically with the number of accumulated elements; specifically, $|\mathsf{acc}_n| = \mathsf{poly}(\log n)$. In addition to compactness, we require the accumulator to be both sound and complete. Soundness ensures that no PPT adversary can produce a polynomially sized list S, an element $x \notin S$, and a witness w such that the verification $\mathsf{Ver}(\mathsf{acc}_S, x, w)$ returns true, where acc_S denotes the accumulator value corresponding to S.

Standard completeness requires that for any element $S[j] \in S$ and any $i \geq j$, the verification $\mathsf{Ver}(\mathsf{acc}_i, S[j], w)$ succeeds when w is the latest witness for $S[j]$, i.e., the one generated for $S[j]$ immediately after $S[i]$ has been added. However, in known constructions, it is not always necessary to update witnesses immediately after every addition. In this paper, we study how frequently witnesses need to be updated in order to maintain correct membership proofs. To this end, we introduce a relaxed notion of completeness. While standard completeness requires verification to succeed with the latest witness, our relaxed version allows verification to succeed with potentially stale witnesses, as long as updates occur at specific points determined by a function $\mathsf{NeedUpd}$. Specifically, if $\mathsf{NeedUpd}(j, S[1:i]) = 1$, where $S[1:i]$ denotes the sub-list $(S[1], \cdots, S[i])$, then the witness for the j-th value in S, i.e., $S[j]$, must be updated immediately after the i-th

addition. We require that membership verification succeeds as long as each value is updated at every point prescribed by $\mathsf{NeedUpd}$.

2.1 Impossibility of One Immediate Update

We begin with a toy example to illustrate the impossibility of overly restrictive update policies. Specifically, we show that no accumulator can satisfy soundness if each value receives only one immediate witness update. In this setting, every value $S[i]$ receives a witness w immediately after being added to the accumulator—formally, $\mathsf{NeedUpd}(i, S[1:i]) = 1$—and does not receive any further updates thereafter, i.e., $\mathsf{NeedUpd}(i, S[1:j]) = 0$ for all $j > i$.

The attack is information theoretical and relies on the *compactness* requirement of the accumulator value. We begin by defining some random variables. Let $\mathsf{S} \leftarrow U_{\mathsf{List}_n(\mathcal{X})}$, where $\mathsf{List}_n(\mathcal{X})$ denotes the set of all lists of length n containing distinct elements from the data universe $\mathcal{X}$, and $U_{\mathsf{List}_n(\mathcal{X})}$ is the uniform distribution over this set. This defines n jointly distributed random variables $\mathsf{S}[1], \cdots, \mathsf{S}[n]$, where $\mathsf{S}[i]$ is the i-th element in the list S. Let Acc_n denote the random variable representing the accumulator value computed over the list S. We analyze the information that Acc_n reveals about $\mathsf{S}[1], \cdots, \mathsf{S}[n]$.

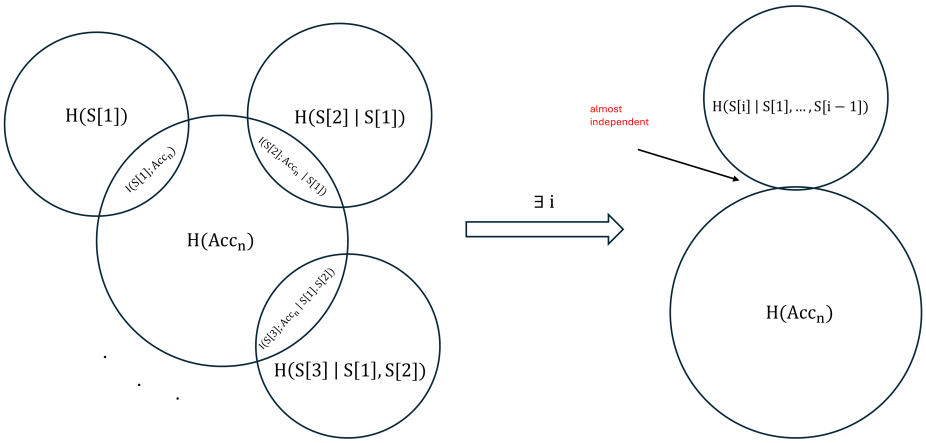

Figure 2: There must exist an index i where $I(\mathsf{S}[i]; \mathsf{Acc}_n \mid \mathsf{S}[1], \cdots, \mathsf{S}[i-1])$ is low.

As shown in Figure 2, each circle represents the entropy of a random variable while the overlap is the mutual information. Since the n-th accumulator value is completely determined by the n values, using the chain rule of mutual information, we have

$$I(\mathsf{S}[1], \cdots, \mathsf{S}[n]; \mathsf{Acc}_n) = I(\mathsf{S}[1]; \mathsf{Acc}_n) + \sum_{j=2}^{n} I(\mathsf{S}[j]; \mathsf{Acc}_n \mid \mathsf{S}[1], \cdots, \mathsf{S}[j-1]) \leq H(\mathsf{Acc}_n).$$

Since the accumulator is required to be compact, its length—and thus its entropy—is bounded by $\mathsf{poly}(\log n)$. Therefore, either

$$I(\mathsf{S}[1]; \mathsf{Acc}_n) \leq \frac{\mathsf{poly}(\log n)}{n},$$

or there exists some $i \in [2 : n]$ such that

$$I(\mathsf{S}[i]; \mathsf{Acc}_n \mid \mathsf{S}[1], \cdots, \mathsf{S}[i-1]) \leq \frac{\mathsf{poly}(\log n)}{n}.$$

We focus on the second case, as it is more general. Suppose that for some index $i \in [2 : n]$, the mutual information satisfies

$$I(\mathsf{S}[i]; \mathsf{Acc}_n \mid \mathsf{S}[1], \cdots, \mathsf{S}[i-1]) \leq \frac{\mathsf{poly}(\log n)}{n}.$$

This implies that the accumulator value Acc_n reveals very little information about $\mathsf{S}[i]$, even when $\mathsf{S}[1], \cdots, \mathsf{S}[i-1]$ are known.

Let us consider the following two distributions.

- *The true joint distribution*, where $(\mathsf{S}[1], \cdots, \mathsf{S}[i], \mathsf{Acc}_n)$ are sampled according to the original generative process (i.e., sample $\mathsf{S} \sim U_{\mathsf{List}_n(\mathcal{X})}$, then compute Acc_n deterministically from S).
- *The hybrid distribution*, where we first sample $(\mathsf{S}[1], \cdots, \mathsf{S}[i-1])$, then sample $\mathsf{S}[i]$ conditioned on $(\mathsf{S}[1], \cdots, \mathsf{S}[i-1])$, and finally sample Acc_n independently of $\mathsf{S}[i]$, conditioned only on $(\mathsf{S}[1], \cdots, \mathsf{S}[i-1])$.

By Pinsker's inequality, the total variation distance between these two distributions is bounded by

$$\sqrt{\frac{\ln 2}{2} \cdot I(\mathsf{S}[i]; \mathsf{Acc}_n \mid \mathsf{S}[1], \cdots, \mathsf{S}[i-1])} \leq \sqrt{\frac{\ln 2}{2} \cdot \frac{\mathsf{poly}(\log n)}{n}}.$$

Suppose the attacker correctly guesses the index i, which happens with probability at least $\frac{1}{n}$. The attacker proceeds as follows:

1. Sample $S \leftarrow \mathsf{S}$ to obtain the corresponding accumulator value acc.
2. Sample $x' \leftarrow \mathsf{S}[i] \mid \mathsf{S}[1 : i-1] = S[1 : i-1]$.
3. Compute a witness w' for x' with respect to the list $S[1 : i-1] || x'$.
4. Output x' and w'.

Note that the sampling in the first two steps follows the hybrid distribution defined earlier:

- The prefix $S[1 : i-1]$ is sampled first.
- Then, the accumulator value acc is sampled conditioned only on $S[1 : i-1]$. This is implemented by sampling the remaining elements $S[i], \cdots, S[n]$ conditioned on $S[1 : i-1]$, and then computing acc from the full list S.
- Separately, the value x' is sampled from the conditional distribution $\mathsf{S}[i] \mid \mathsf{S}[1 : i-1] = S[1 : i-1]$.

If x' and acc were instead jointly sampled from the true distribution conditioned on $S[1 : i-1]$, then—by the completeness property and the assumption that witnesses are computed immediately after addition—the verification $\mathsf{Ver}(\mathsf{acc}, x', w')$ would succeed with probability 1.

Since the hybrid and true distributions are statistically close—within distance at most $\delta := \sqrt{\frac{\ln 2}{2} \cdot \frac{\mathsf{poly}(\log n)}{n}}$—the success probability of verification under the hybrid distribution is at least $1 - \delta$.

Finally, we need to ensure that the sampled value x' does not appear in the list S. Since S consists of n distinct elements drawn uniformly from a data universe $\mathcal{X}$ of exponential size (e.g., $|\mathcal{X}| = 2^\lambda$ for security parameter λ), the probability that x'—sampled independently of the rest of S conditioned on $S[1 : i-1]$—is already in S is at most $n/|\mathcal{X}|$, which is negligible in λ for polynomial n.

Thus, with overwhelming probability, the attacker outputs a valid membership proof w' for an element $x' \notin S$, and the verification succeeds with probability at least $1 - \delta$. This contradicts soundness.

2.2 Impossibility Assuming Fixed Update Times

In this subsection, we consider a more general setting where witness updates are allowed to occur at designated update times. However, we impose the constraint that the update schedule must be independent of the actual accumulated values. Specifically, we assume that the function $\mathsf{NeedUpd}(i, S[1 : j])$—which indicates whether the witness for the i-th element must be updated after the j-th element is added—is determined solely by the indices i and j, and not by the specific values in the list $S[1], \cdots, S[j]$. Therefore, in this setting, we treat $\mathsf{NeedUpd}$ as a function taking two integers i and j (with $i \leq j$).

At a high level, the key idea is to partition consecutive values into blocks, treating each block as a "long value." This abstraction allows us to view each block as a single unit, enabling us to adapt the attack strategy from the simpler setting discussed in the previous subsection to this more general case. However, to successfully adapt the attack, the partitioning must be done carefully: we need to ensure that the witnesses for values in the i-th block—or at least the witness for one value at some position within the block—can be computed using only the previous i blocks. Recall that in the simpler setting, we resampled the i-th value conditioned on the first $i-1$ values, and completeness ensured that a witness for the i-th value could be computed from the first i values. In this more general setting, we aim to resample the i-th block conditioned on the first $i-1$ blocks, and we must ensure that the required witness can still be computed using only the first i blocks. To this end, we need the following combinatorial tool, which was introduced and used in the context of RBE as well [MQR22].

Update graph and step sequence. To analyze update times, we introduce a combinatorial tool based on graph theory. Given the function $\mathsf{NeedUpd}$, we define an update graph over the vertex set $[n]$, where each vertex corresponds to an added value. The graph contains a directed edge from vertex i to vertex j if and only if $\mathsf{NeedUpd}(i, j) = 1$. That is, an edge (i, j) indicates that the witness for the i-th element has been updated immediately after the j-th element is added. Observe that the out-degree of vertex i corresponds to the number of times the witness for value $S[i]$ is updated. Consequently, the maximum out-degree of the update graph reflects the largest number of updates required by any single value.

We now introduce the concept of a step sequence in an update graph. A step sequence of length k in G is a sequence of pairs of *vertices*

$$(u_1, v_1), (u_2, v_2), \cdots, (u_k, v_k)$$

satisfying the following conditions:

- The last pair satisfies $u_k = v_k$, which we call the *end* of the step sequence.
- For each $i \in [k-1]$, the vertex v_i is defined as follows: if there exists an edge from u_i to some vertex $w \leq u_k$, then v_i is the largest such vertex w; otherwise, if no such edge exists, set $v_i = u_i$. **It is then required that $v_i < u_{i+1}$ for $i \in [k-1]$.**

Existence of long step sequence. We will show later in the technical portion of the paper that if the parameters n, k and d satisfy $n \geq \binom{k+d}{d+1}$, then any update graph G with maximum out-degree at most d contains a step sequence of length k.

Blocks. We are now ready to define the block partition. Suppose that no element requires more than d updates. Then, by the above combinatorial result, the update graph contains a step sequence of length k, where k is the largest integer satisfying $n \geq \binom{k+d}{d+1}$. Given such a step sequence

$$(u_1, v_1), (u_2, v_2), \cdots, (u_k, v_k),$$

we can partition any list $S \in \mathsf{List}_n(\mathcal{X})$ into k consecutive blocks. The first block is defined as $B_1(S) = S[1 : v_1]$, i.e., it contains the first v_1 elements. For each $i \in [2 : k]$, the i-th block is defined as $B_i(S) = S[v_{i-1} + 1 : v_i]$.

This partitioning has the following crucial property: **the membership witness for the element $S[u_i]$ with respect to the accumulator value acc_{u_k} can be computed using only the blocks $B_1(S), \cdots, B_i(S)$.**

Attack. As before, let $\mathsf{S} \sim U_{\mathsf{List}_n(\mathcal{X})}$ be uniformly sampled from the set of all lists of n distinct elements. This induces a distribution over the blocks:

$$\mathsf{B}_1 = B_1(\mathsf{S}), \mathsf{B}_2 = B_2(\mathsf{S}), \cdots, \mathsf{B}_k = B_k(\mathsf{S}).$$

By the same information-theoretic argument as in the previous subsection, we conclude that either:

$$I(\mathsf{B}_1; \mathsf{Acc}_{u_k}) \leq \frac{\mathsf{poly}(\log n)}{k},$$

or there exists some $i \in [2 : k]$ such that

$$I(\mathsf{B}_i; \mathsf{Acc}_{u_k} \mid \mathsf{B}_1, \cdots, \mathsf{B}_i - 1) \leq \frac{\mathsf{poly}(\log n)}{k}.$$

It is straightforward to adapt the previous attack strategy to this setting by targeting the u_i-th value, where i is the index of a block whose mutual information with acc_{u_k}—conditioned on the preceding blocks—is small. Since the witness for $S[u_i]$ can be computed using only the first i blocks, the attacker can mimic the earlier attack by resampling B_i conditioned on $\mathsf{B}_1, \cdots, \mathsf{B}_{i-1}$ and constructing a convincing membership proof for the new u_i-th value in the freshly sampled i-th block.

2.3 Impossibility without Assumptions

In this subsection, we move on to the general case where no assumptions are made about the witness update pattern. While each list $S \in \mathsf{List}_n(\mathcal{X})$ induces an update graph $G(S)$—where there is an edge from i to j if and only if $\mathsf{NeedUpd}(i, S[1 : j]) = 1$—the update graphs for different lists S and S' may differ significantly. This introduces a new layer of complexity, as we can no longer rely on a fixed graph structure across instances. Nonetheless, we begin with the same combinatorial insight: if the parameters n, k, and d satisfy $n \geq \binom{k+d}{d+1}$, then any update graph G with maximum out-degree at most d contains a step sequence of length k. However, the variability of the update graphs across different input lists raises two key issues:

Step sequences have different ends. In the previous setting, the step sequence was defined with respect to a fixed graph and had a known endpoint u_k. In the general case, however, the update graph $G(S)$ depends on the particular list S, and thus the step sequence—and in particular its endpoint—can vary from instance to instance. This makes it challenging to define a uniform position at which the witness must be verified or to identify a single target value for attack.

To address this, we observe that there are only n possible choices for the endpoint of a step sequence (one for each position in the list). Therefore, by a simple averaging argument, there must exist some index $\mathsf{end} \in [n]$ such that at least $1/n$ fraction of all lists in $\mathsf{List}_n(\mathcal{X})$ admit a step sequence that ends at end. In what follows, we condition on the event that the sampled list yields a step sequence ending at this fixed index end, allowing us to carry out the attack strategy uniformly over this subset of inputs.

Conditionally sampling blocks. As in previous attacks, we need to resample the i-th block B_i conditioned on the previously sampled blocks $B_1, \ldots, B_{i-1}$. However, directly sampling the blocks is challenging because the update function $\mathsf{NeedUpd}$ used by the accumulator may be unknown and thus we only know how to sample the full list S. A natural approach is as follows: given a sampled list $S \leftarrow \mathsf{S}$, we guess the endpoint of block B_{i-1} with probability at least $\frac{1}{n}$, then resample $S' \leftarrow \mathsf{S}$ conditioned on having $B_1||\cdots||B_{i-1}$ as its prefix, and finally guess the endpoint of the new block B_i' within S'.

A caveat is that since the step sequence depends on the entire list, **there is no guarantee that the first $i-1$ blocks of the resampled S' remain exactly $B_1, \ldots, B_{i-1}$.**

To address this challenge, we introduce a stronger variant of step sequences called *falling-step sequences*. Unlike ordinary step sequences, falling-step sequences can be shown to have the property that their block partitioning is *stable under prefix conditioning*. Concretely, this means that if we fix the prefix of a list S' to be $B_1||\cdots||B_{i-1}$, then the first $i-1$ blocks induced by the falling-step sequence on S' remain exactly $B_1, \ldots, B_{i-1}$. This stability property is crucial because it allows us to correctly condition on the earlier blocks when resampling later blocks, even though the update graph depends on the entire list. This combinatorial tool enables us to perform the conditional resampling of blocks accurately and thus adapt our attack strategy to the general setting where update times depend arbitrarily on the accumulated values.

In Section 4, we detail how the above ideas combine to carry out the attack in the general setting. In Section 5, we prove the existence of falling-step sequences and establish the combinatorial properties that make them suitable for our purposes.

3 Preliminaries

Notations. We denote the security parameter by λ. We write $\mathsf{poly}(\lambda)$ to denote a generic polynomial function in λ, and more generally, $\mathsf{poly}(x, y)$ to denote a polynomial in x and y. A function $f(\lambda)$ is said to be negligible if $f(\lambda) = O(1/\mathsf{poly}(\lambda))$ for every polynomial function $\mathsf{poly}(\lambda)$. We write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to indicate that $f(\lambda)$ is negligible. A probability $p(\lambda)$ is called overwhelming if $1 - p(\lambda)$ is negligible. We use PPT to refer to probabilistic polynomial time.

For a randomized algorithm A, we sometimes write $\mathsf{A}(x; r)$ to make the internal randomness r explicit. For any positive integer n, we let $[n]$ denote the set $1, 2, \ldots, n$, and for $i \leq j$, we write $[i : j] := i, i+1, \ldots, j$. [1]

Let $\mathcal{X}$ be any set. We write $|\mathcal{X}|$ for its cardinality. For a set $\mathcal{X}$ and an integer $n \leq |\mathcal{X}|$, we let $\mathsf{List}_n(\mathcal{X})$ denote the set of all (ordered) lists of n distinct elements from $\mathcal{X}$. We also

[1] If $i > j$, then we define $[i : j] := \emptyset$.

define $\mathsf{List}_{\leq n}(\mathcal{X}) := \bigcup_{i=1}^{n} \mathsf{List}_i(\mathcal{X})$. For a list $S = (s_1, \ldots, s_n)$ and indices $i \leq j \leq n$, we write $S[i] := s_i$ and $S[i:j] := (s_i, \ldots, s_j)$. [2]

For lists S and S', we write $S||S'$ for their concatenation. Namely, if $S = (s_1, \ldots, s_n)$ and $S' = (s'_1, \ldots, s'_m)$, then $S||S' = (s_1, \ldots, s_n, s'_1, \ldots, s'_m)$. For convenience, we also write $S||x$ to denote $S||(x)$. We say $S \sqsubseteq S'$ if S is a prefix of S', i.e., $S = S'[1 : |S|]$.

Throughout the paper, we denote by $\mathcal{X}$ the data universe—the set of values that may be accumulated; by $\mathcal{RD}$ the set of randomness strings used by PPT algorithms; and by $\mathcal{CRS}$ the set of possible CRS (common reference string) values. Without loss of generality, we assume $\mathcal{X} = \mathcal{RD} = \mathcal{CRS} = \{0,1\}^\lambda$.

3.1 Additive Positive Accumulator

Definition 1 (Additive positive accumulator)**.** An *additive positive accumulator* for a data universe $\mathcal{X}$ consists of three PPT algorithms $(\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$:

- $\mathsf{SetUp}(1^\lambda) \to \mathsf{crs}$: A randomized algorithm that takes as input the security parameter 1^λ and outputs a common reference string crs.
- $\mathsf{Add}(\mathsf{crs}, S, \mathsf{acc}, x) \to (\mathsf{acc}', W)$: A (possibly randomized) algorithm that takes as input a crs, a list S of distinct values, an accumulator state acc, and a new value $x \notin S$, and outputs an updated accumulator acc' and a list W of witnesses for all values in $S||x$. (If $S = \emptyset$, we require $\mathsf{acc} = \perp$.)
- $\mathsf{Ver}(\mathsf{crs}, \mathsf{acc}, x, w) \to \{\mathsf{true}, \mathsf{false}\}$: A (possibly randomized) algorithm that takes as input a crs, an accumulator value acc, a value x, and a purported witness w, and returns true if w is a valid membership witness for x in acc, and false otherwise.

We assume that the security parameter 1^λ is included in the CRS. Therefore, the algorithms Add and Ver do not take 1^λ as input. We also define the following helper algorithms Eval and Wit for convenience:

- $\mathsf{Eval}(\mathsf{crs}, S) \to \mathsf{acc}$: A (possibly randomized) algorithm that evaluates the accumulator on a list $S = (s_1, \ldots, s_n) \in \mathsf{List}_n(\mathcal{X})$ by performing:
  1. Initialize $\mathsf{acc} \leftarrow \perp$, $W \leftarrow \emptyset$.
  2. For $i \in [n]$, update $(\mathsf{acc}, W) \leftarrow \mathsf{Add}(\mathsf{crs}, S[1 : i-1], \mathsf{acc}, s_i)$.
  3. Output acc.
- $\mathsf{Wit}(\mathsf{crs}, S, i) \to w$: A (possibly randomized) algorithm that outputs a membership witness for the i-th value in a list $S = (s_1, \ldots, s_n) \in \mathsf{List}_n(\mathcal{X})$:
  1. Initialize $\mathsf{acc} \leftarrow \perp$, $W \leftarrow \emptyset$.
  2. For $j \in [n]$, update $(\mathsf{acc}, W) \leftarrow \mathsf{Add}(\mathsf{crs}, S[1 : j-1], \mathsf{acc}, s_j)$.
  3. Output $W[i]$.

Importantly, we require that the accumulator value acc be compact with respect to the number n of accumulated elements. Without this requirement, the definition becomes trivial: for example, if the size of acc grows linearly with n, one could simply store all accumulated values directly as part of the accumulator (e.g., by concatenating them), defeating the purpose of succinct accumulation.

In this work, we impose a strong compactness requirement by demanding that $|\mathsf{acc}| \leq \mathsf{poly}(\log n)$—that is, the size of the accumulator should be at most polylogarithmic in the number of accumulated elements. We formalize this requirement below.

[2] If $i > j$, we define $S[i:j] := \emptyset$.

Definition 2 (Compactness)**.** An additive positive accumulator scheme $\Lambda = (\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ is said to have a *length function* $\ell(\lambda, n)$[3] if for all security parameters λ, all $\mathsf{crs} \in \mathcal{CRS}$, all $n \leq |\mathcal{X}|$, and all $S \in \mathsf{List}_n(\mathcal{X})$, it holds that

$$\Pr\left[|\mathsf{acc}| \leq \ell(\lambda, n) : \mathsf{acc} \leftarrow \mathsf{Eval}(\mathsf{crs}, S)\right] = 1.$$

We say that Λ is *compact* if $\ell(\lambda, n) = \mathsf{poly}(\lambda, \log n)$.

Next, we define completeness. The standard definition requires that every accumulated value can always be verified using its latest (possibly updated) witness—and that this witness must be updated after every new addition. While standard, this definition is often too restrictive. In practice, some values may not require updated witnesses after certain additions. Since this paper studies how frequently witnesses need to be updated, we introduce a relaxed notion of completeness that lets us reason about selective update behavior. Specifically, we define a function $\mathsf{NeedUpd}_\lambda$ that determines, after a new value is added, which existing values require updated witnesses. The completeness guarantee will then require only that **if witnesses are updated according to $\mathsf{NeedUpd}_\lambda$, verification succeeds**. Note that if $\mathsf{NeedUpd}_\lambda$ outputs 1 for all values after every addition, then this reduces to the standard definition of completeness.

To model flexible update behavior, we allow $\mathsf{NeedUpd}_\lambda$ to depend on *everything that has happened so far*. Formally, the function

$$\mathsf{NeedUpd}_\lambda : \mathcal{CRS} \times \mathbb{N} \times \mathsf{List}_{\leq|\mathcal{X}|}(\mathcal{X}) \times \mathcal{RD}^* \to \{0, 1\}$$

takes as input:

- the common reference string crs,
- an index $i \in [n]$;
- the list of accumulated values $(x_1, \ldots, x_n)$
- and the list of randomness values $(r_1, \ldots, r_n)$ used during each addition.

It outputs 1 if x_i requires a new witness after the addition of x_n, and 0 otherwise.

Although the accumulator value acc is not explicitly passed to $\mathsf{NeedUpd}_\lambda$, its value is *fully determined* by crs, $(x_1, \ldots, x_n)$, and $(r_1, \ldots, r_n)$. Hence, the function has access to all necessary information for determining whether an update is needed.

Definition 3 (Completeness)**.** Let

$$\mathsf{NeedUpd}_\lambda : \mathcal{CRS} \times \mathbb{N} \times \mathsf{List}_{\leq|\mathcal{X}|}(\mathcal{X}) \times \mathcal{RD}^* \to \{0, 1\}$$

be any update function. We define a game $\mathsf{Comp}_{\mathsf{Adv}}(\lambda, \mathsf{NeedUpd}_\lambda)$ between a computationally unbounded adversary Adv and a challenger Chal, where Adv is restricted to at most $\mathsf{poly}(\lambda)$ rounds of interaction:

1. The challenger initializes:

$$t \leftarrow 1, \quad S_{\mathcal{X}} \leftarrow \emptyset, \quad S_{\mathcal{RD}} \leftarrow \emptyset, \quad \mathsf{acc} \leftarrow \perp, \quad W \leftarrow \emptyset, \quad W_{\mathsf{upd}} \leftarrow \emptyset.$$

2. The challenger samples $\mathsf{crs} \leftarrow \mathsf{SetUp}(1^\lambda)$ and sends crs to Adv.
3. The following steps are repeated for at most $\mathsf{poly}(\lambda)$ rounds:
   (a) The adversary sends a value $x_t \notin S_{\mathcal{X}}$ to Chal.

[3]Without loss of generality, we assume ℓ is non-decreasing in n.

(b) The challenger samples $r_t \leftarrow \mathcal{RD}$ and computes:

$$(\mathsf{acc}, W) \leftarrow \mathsf{Add}(\mathsf{crs}, S_{\mathcal{X}}, \mathsf{acc}, x_t; r_t).$$

(c) The challenger updates the state:

$$S_{\mathcal{X}} \leftarrow S_{\mathcal{X}} || x_t, \quad S_{\mathcal{RD}} \leftarrow S_{\mathcal{RD}} || r_t, \quad W_{\mathsf{upd}} \leftarrow W_{\mathsf{upd}} || \perp.$$

(d) For each $i \in [t]$, if $\mathsf{NeedUpd}_\lambda(\mathsf{crs}, i, S_{\mathcal{X}}, S_{\mathcal{RD}}) = 1$, then update:

$$W_{\mathsf{upd}}[i] \leftarrow W[i].$$

(e) Increment $t \leftarrow t + 1$.

4. The adversary outputs an index $i < t$ and wins the game if:

$$\mathsf{Ver}(\mathsf{crs}, \mathsf{acc}, x_i, W_{\mathsf{upd}}[i]) = \mathsf{false}.$$

We say that an additive positive accumulator scheme $(\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ is $\mathsf{NeedUpd}_\lambda$-complete if

$$\Pr[\mathsf{Adv} \text{ wins in } \mathsf{Comp}_{\mathsf{Adv}}(\lambda, \mathsf{NeedUpd}_\lambda)] = 0.$$

Equipped with the above completeness definition, we can now formalize the number of updates required by an accumulator scheme with respect to a given update function $\mathsf{NeedUpd}_\lambda$. For reasons that will become clear later, we refer to this as the *degree function*.

Definition 4 (Degree Function)**.** Let $\Lambda = (\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ be an additive positive accumulator that is $\mathsf{NeedUpd}_\lambda$-complete. We say that a function $\deg : \mathbb{N} \to \mathbb{N}$ is a *degree function* for Λ with respect to $\mathsf{NeedUpd}_\lambda$ if, for every security parameter λ and every $n \leq 2^\lambda$, the following holds:

$$\deg(n) \geq \max_{\mathsf{crs},\, i,\, S,\, \mathcal{R}} \sum_{j=i}^{n} \mathsf{NeedUpd}_\lambda(\mathsf{crs}, i, S[1:j], \mathcal{R}[1:j]),$$

where the maximum is taken over all $\mathsf{crs} \in \mathcal{CRS}$, $i \in [n]$, $S \in \mathsf{List}_n(\mathcal{X})$, and $\mathcal{R} \in \mathcal{RD}^n$.

Finally, the soundness property requires that no PPT adversary can produce a valid witness for a value that was not accumulated. In other words, it should be computationally infeasible to forge a membership proof for any value outside the accumulated set.

Definition 5 (Soundness)**.** For any interactive PPT adversary Adv, consider the following game $\mathsf{Sec}_{\mathsf{Adv}}(\lambda)$ between Adv and a challenger Chal:

1. The challenger initializes:

$$S \leftarrow \emptyset, \quad \mathsf{acc} \leftarrow \perp, \quad W \leftarrow \emptyset.$$

2. The challenger samples $\mathsf{crs} \leftarrow \mathsf{SetUp}(1^\lambda)$ and sends crs to Adv.

3. The following steps are repeated while Adv continues (for at most $\mathsf{poly}(\lambda)$ rounds):

(a) The adversary sends a value $x \notin S$ to the challenger.

(b) The challenger computes $(\mathsf{acc}, W) \leftarrow \mathsf{Add}(\mathsf{crs}, S, \mathsf{acc}, x)$.

(c) The challenger updates $S \leftarrow S || x$.

(d) The challenger sends (acc, W) to Adv.

4. Eventually, the adversary outputs a value $x \notin S$ and a witness w, and wins the game if:
$$\mathsf{Ver}(\mathsf{crs}, \mathsf{acc}, x, w) = \mathsf{true}.$$

We say that an additive positive accumulator $(\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ is *sound* if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all PPT adversaries Adv,

$$\Pr[\mathsf{Adv} \text{ wins in } \mathsf{Sec}_{\mathsf{Adv}}(\lambda)] \leq \mathsf{negl}(\lambda).$$

3.2 Information-Theoretic Tools

Notation. For a random variable X, we write $x \leftarrow \mathsf{X}$ to denote sampling x from X, and we let P_{X} denote the distribution (or law) of X, written as $\mathsf{X} \sim P_{\mathsf{X}}$. For jointly distributed random variables $(\mathsf{X}, \mathsf{Y}) \sim P_{\mathsf{X},\mathsf{Y}}$, we write P_{X} to denote the marginal distribution of X, and $\mathsf{Y} \mid \mathsf{X}$ to denote the conditional random variable of Y given X. We write $P_{\mathsf{Y}|\mathsf{X}}$ for the corresponding conditional distribution, and $\mathsf{Y} \mid \mathsf{X} = x$ to denote the distribution of Y conditioned on the event $\mathsf{X} = x$. For any function f, we write $\mathbb{E}_{\mathsf{X}}[f(\mathsf{X})]$ to denote the expectation of $f(\mathsf{X})$ when X is distributed as P_{X}. When the context is clear, we omit the subscript and simply write $\mathbb{E}[f(\mathsf{X})]$. We write $\mathsf{sprt}(\mathsf{X})$ to denote the support of the random variable X, and $\mathsf{sprt}(P)$ to denote the support of the distribution P. For a finite set $\mathcal{X}$, we write $U_{\mathcal{X}}$ for the uniform distribution over $\mathcal{X}$.

We prove the following simple inequality.

Proposition 1 (Double Sampling)**.** *Let* X *and* Y *be jointly distributed random variables, and let* $\mathcal{C} \subseteq \mathsf{sprt}(\mathsf{X}, \mathsf{Y})$ *be an event such that*

$$\Pr[(\mathsf{X}, \mathsf{Y}) \in \mathcal{C}] \geq p.$$

Consider the following sampling procedure:

- *Sample* $(x, y) \leftarrow (\mathsf{X}, \mathsf{Y})$.
- *Independently sample* $y' \leftarrow \mathsf{Y} \mid \mathsf{X} = x$.

Then the probability that both samples fall inside $\mathcal{C}$ *satisfies:*

$$\Pr[(x, y) \in \mathcal{C} \text{ and } (x, y') \in \mathcal{C}] \geq p^2.$$

Proof. For each x, let

$$a(x) := \Pr\big((x, \mathsf{Y}) \in \mathcal{C}\big).$$

Therefore $\Pr[(\mathsf{X}, \mathsf{Y}) \in \mathcal{C}] = \mathbb{E}[a(\mathsf{X})] \geq p$.

In the two-sample procedure, given x the draws y, y' are independent, each falling in $\mathcal{C}$ with probability $a(x)$. Hence

$$\Pr[(x, y) \in \mathcal{C} \text{ and } (x, y') \in \mathcal{C}] = \mathbb{E}[a(\mathsf{X})^2].$$

Applying Jensen's Inequality, we get

$$\mathbb{E}[a(\mathsf{X})^2] \;\geq\; (\mathbb{E}[a(\mathsf{X})])^2 \;\geq\; p^2.$$

□

We recall the following standard definitions from information theory.

Definition 6 (Shannon Entropy and Mutual Information)**.** Let X, Y, and Z be discrete random variables.

- The *Shannon entropy* of X is defined as $H(\mathsf{X}) = -\sum_{x \in \mathsf{sprt}(\mathsf{X})} P_{\mathsf{X}}(x) \log P_{\mathsf{X}}(x)$.
- The *joint entropy* of (X, Y) is $H(\mathsf{X}, \mathsf{Y}) = -\sum_{x \in \mathsf{sprt}(\mathsf{X})} \sum_{y \in \mathsf{sprt}(\mathsf{Y})} P_{\mathsf{X},\mathsf{Y}}(x, y) \log P_{\mathsf{X},\mathsf{Y}}(x, y)$.
- The *mutual information* between X and Y is

$$I(\mathsf{X}; \mathsf{Y}) = \sum_{x \in \mathsf{sprt}(\mathsf{X})} \sum_{y \in \mathsf{sprt}(\mathsf{Y})} P_{\mathsf{X},\mathsf{Y}}(x, y) \log \frac{P_{\mathsf{X},\mathsf{Y}}(x, y)}{P_{\mathsf{X}}(x) P_{\mathsf{Y}}(y)}.$$

- The *conditional mutual information* between X and Y given Z is

$$I(\mathsf{X}; \mathsf{Y} \mid \mathsf{Z}) = \sum_{x \in \mathsf{sprt}(\mathsf{X})} \sum_{y \in \mathsf{sprt}(\mathsf{Y})} \sum_{z \in \mathsf{sprt}(\mathsf{Z})} P_{\mathsf{X},\mathsf{Y},\mathsf{Z}}(x, y, z) \log \frac{P_{\mathsf{X},\mathsf{Y}|\mathsf{Z}}(x, y \mid z)}{P_{\mathsf{X}|\mathsf{Z}}(x \mid z) P_{\mathsf{Y}|\mathsf{Z}}(y \mid z)}.$$

We recall the standard fact that the entropy of a random variable is upper bounded by the logarithm of the size of its support, which in turn is bounded by the maximum bit-length of elements in the support.

Fact 1. *Let X be a random variable. Then, we have $H(\mathsf{X}) \leq \log |\mathsf{sprt}(\mathsf{X})| \leq \max_{x \in \mathsf{sprt}(\mathsf{X})} |x|$, where $|x|$ denotes the bit-length of x.*

We recall the chain rule for mutual information, which expresses the mutual information between a collection of random variables and another variable as a sum of conditional mutual informations.

Proposition 2 (Chain Rule for Mutual Information)**.** *Let $\mathsf{X}_1, \ldots, \mathsf{X}_n$ and Y be jointly distributed random variables. Then $I(\mathsf{X}_1, \ldots, \mathsf{X}_n; \mathsf{Y}) = I(\mathsf{X}_1; \mathsf{Y}) + \sum_{i=2}^{n} I(\mathsf{X}_i; \mathsf{Y} \mid \mathsf{X}_1, \ldots, \mathsf{X}_{i-1})$.*

We also recall the definition of statistical distance and a standard consequence.

Definition 7 (Statistical Distance)**.** Let P and Q be probability distributions over the same domain. Their *statistical distance* is defined as $\mathsf{SD}(P, Q) = \sum_{x \in \mathsf{sprt}(P) \cup \mathsf{sprt}(Q)} \frac{|P(x) - Q(x)|}{2}$. For random variables X and Y, their statistical distance is defined as $\mathsf{SD}(\mathsf{X}, \mathsf{Y}) = \mathsf{SD}(P_{\mathsf{X}}, P_{\mathsf{Y}})$.

The following fact captures the usefulness of statistical distance in bounding probabilities under two distributions.

Fact 2. *Let P and Q be distributions such that $\mathsf{SD}(P, Q) \leq \epsilon$. Then for any event $\mathcal{A}$, it holds that $|P(\mathcal{A}) - Q(\mathcal{A})| \leq \epsilon$.*

Finally, we state the main information-theoretic tool on which our main result relies.

Lemma 1 (Lemma B.5 in [MQR22])**.** *Let $\mathsf{X}, \mathsf{Y}, \mathsf{Z} \sim P_{\mathsf{X},\mathsf{Y},\mathsf{Z}}$ be jointly distributed random variables. Define new random variables $(\mathsf{X}, \mathsf{Y}', \mathsf{Z}')$ with the same marginal on X, and with Y' and Z' conditionally independent given X, as follows:*

- $\mathsf{X} \sim P_{\mathsf{X}}$,
- $\mathsf{Y}' \sim P_{\mathsf{Y}|\mathsf{X}}$,
- $\mathsf{Z}' \sim P_{\mathsf{Z}|\mathsf{X}}$.

Then the statistical distance between the original joint distribution and the modified one is bounded by $\mathsf{SD}\big((\mathsf{X}, \mathsf{Y}, \mathsf{Z}), (\mathsf{X}, \mathsf{Y}', \mathsf{Z}')\big) \leq \sqrt{\frac{\ln 2}{2} \cdot I(\mathsf{Y}; \mathsf{Z} \mid \mathsf{X})}$.

4 Lower Bounds

In this section, we present and prove a lower bound on the frequency with which membership witnesses must be updated in additive positive accumulators. Due to the complexity of the full proof, we proceed incrementally: we begin with simplified settings under helpful assumptions and gradually relax these to arrive at the general case.

Throughout this section, we use both Assumptions and Claims. Assumptions are made solely for exposition and will be systematically removed as the proof progresses. Claims are formal intermediate results and will be rigorously proved.

4.1 Main Theorem and Implications

Theorem 2. *Let* $\Lambda = (\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ *be an additive positive accumulator scheme. Suppose* $\deg(n) = d(n)$ *is a degree function for* Λ *satisfying* $d(n) = o(n)$*. Suppose* $\ell(\lambda, n)$*, which is non-decreasing in* n*, is a length function for* Λ*.*

Let $n = \mathsf{poly}(\lambda)$ *be an arbitrary polynomial in* λ*. Let* $k(n)$ *be the largest integer such that* $n \geq \binom{k(n)+d(n)}{d(n)+1}$*. Then there exists a polynomial-time adversary that breaks the soundness of the accumulator scheme* Λ *with probability at least* $\frac{1}{n^8} \cdot \left(1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda,n)}{k(n)}}\right) - \mathsf{negl}(\lambda)$*.*

We will prove the theorem later. At the moment, we deduce the following corollary, which gives an asymptotic lower bound on the degree, i.e., number of updates, required for soundness.

Corollary 1. *Let* $\Lambda = (\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ *be an additive positive accumulator scheme. Suppose* $\deg(n) = d(n)$ *is a degree function for* Λ *satisfying* $d(n) = o(n)$*. Suppose* $\ell(\lambda, n)$*, which is non-decreasing in* n*, is a length function for* Λ*.*

If $\ell(\lambda, n) = \mathsf{poly}(\log n)$*, then it must hold that*

$$d(n) = \Omega\left(\frac{\log n}{\log\log n}\right).$$

Proof. Let $k = k(n)$ be the largest integer such that $n \geq \binom{k+d}{d+1}$. Then, by the monotonicity of the binomial coefficient in k, we have $\binom{k+d+1}{d+1} > n$. Using the standard upper bound on binomial coefficients $\binom{k+d+1}{d+1} \leq \left(\frac{e(k+d+1)}{d+1}\right)^{d+1}$, we conclude that $n < \left(\frac{e(k+d+1)}{d+1}\right)^{d+1}$. Solving for k, we obtain $k > \frac{d+1}{e} \cdot n^{1/(d+1)} - d - 1$.

Now suppose $d(n) = \frac{\log n}{\log\log n} \cdot \varepsilon(n)$ for some function $\varepsilon(n) \to 0$ as $n \to \infty$. Then, we have $\log\left(n^{1/(d+1)}\right) = \frac{\log n}{d+1} = \frac{\log n}{\frac{\log n}{\log\log n} \cdot \varepsilon(n)+1}$. We then have $\log\left(n^{1/(d+1)}\right) = \frac{\log\log n}{\varepsilon(n)+o(1)} = \frac{\log\log n}{\varepsilon'(n)}$, where $\varepsilon'(n) \to 0$ as $n \to \infty$. Thus, we have $n^{1/(d+1)} = \exp\left(\frac{\log\log n}{\varepsilon'(n)}\right)$, and hence, $k = \omega(\log^c n)$ for any constant $c > 0$.

We now apply Theorem 2. As shown earlier, if $d(n) = o\left(\frac{\log n}{\log\log n}\right)$, then $k(n) = \omega(\log^c n)$ for any such c. Therefore, we have $\frac{\ell(\lambda,n)}{k(n)} \to 0$ as $\lambda \to \infty$. Theorem 2 then guarantees a soundness-breaking adversary with success probability at least $\frac{1}{n^8} \cdot (1 - o(1)) - \mathsf{negl}(\lambda) = \frac{1}{\mathsf{poly}(\lambda)} - o\left(\frac{1}{\mathsf{poly}(\lambda)}\right) - \mathsf{negl}(\lambda)$, which is non-negligible in λ for sufficiently large security parameter. This contradicts the soundness of the scheme unless $d(n) = \Omega\left(\frac{\log n}{\log\log n}\right)$. □

Remark 1 (Tightness of the Binomial Estimate for Corollary 1). In Corollary 1, the proof assumes $d(n) = o(\frac{\log n}{\log\log n})$ and applies the standard estimate $\binom{k+d+1}{d+1} \leq \left(\frac{e(k+d+1)}{d+1}\right)^{d+1}$

to the inequality $\binom{k+d+1}{d+1} > n$ to derive $k(n) = \omega(\log^c n)$ for every constant $c > 0$, which makes the adversary's advantage in Theorem 2 non-negligible.

It is natural to ask whether a sharper approximation of the binomial coefficient could be used in place of the standard estimate above to obtain a stronger lower bound on $d(n)$, i.e., to rule out even the case $d(n) = \Theta(\frac{\log n}{\log\log n})$. This is not possible within the framework of Theorem 2.

Consider the boundary regime

$$d(n) = \frac{\log n}{\log\log n} \qquad \text{and} \qquad \ell(\lambda, n) = \log^3 n,$$

which still satisfies $\ell(\lambda, n) = \mathsf{poly}(\log n)$. In this regime, the constraint

$$n < \binom{k(n) + d(n) + 1}{d(n) + 1}$$

admits solutions with $k(n) = O(\log^2 n)$, as can be verified using the standard lower bound $\binom{k+d+1}{d+1} \geq \left(\frac{k+d+1}{d+1}\right)^{d+1}$. For any such polylogarithmic $k(n)$ we have $\frac{\ell(\lambda,n)}{k(n)} = \Omega(\log n)$, and plugging these parameters into the advantage bound of Theorem 2 yields a negligible success probability for the adversary. Thus our attack cannot rule out the boundary regime $d(n) = \Theta(\frac{\log n}{\log\log n})$, and no sharpening of the binomial estimate can improve the asymptotic lower bound on $d(n)$ beyond $\Omega(\frac{\log n}{\log\log n})$ within this framework.

General trade-offs and comparison with [MQ23]. Theorem 2 gives a general combinatorial relation between the accumulator length $k(n)$ and the number of witness updates $d(n)$ through the constraint $n < \binom{k(n)+d(n)+1}{d(n)+1}$. By choosing different parameter regimes for $(k(n), d(n))$, the theorem yields an entire spectrum of lower bounds. For example, fixing $d(n)$ determines the minimum feasible accumulator length $k(n)$, while fixing $k(n)$ determines the minimum update frequency $d(n)$ compatible with soundness. These trade-offs are matched by the upper-bound framework of [MQ23] (see their Proposition 45), which transforms any static accumulator into an additive one of length $O(\lambda \cdot k(n))$[4] whose witnesses incur at most $d(n)$ updates, provided that $(k(n), d(n))$ satisfy $n \geq \binom{k(n)+d(n)}{d(n)+1}$. Thus, Theorem 2 captures the optimal relationship between accumulator length and update frequency up to constant factors across all parameter settings. Corollary 1 treats the succinct regime $\ell(\lambda, n) = \mathsf{poly}(\log n)$. To illustrate the generality of Theorem 2 (together with the upper bounds of [MQ23]), we record one further concrete regime. This example shows that even with a *constant* number of witness updates, the accumulator length must already grow as a sub-linear function of n. This follows from standard estimates on binomial coefficients.

Constant number of updates. If $d(n) = d$ for some fixed positive integer d, then Theorem 2 implies

$$\ell(\lambda, n) = \Omega\left(n^{1/(d+1)}\right).$$

Conversely, the framework of [MQ23] yields an additive accumulator of degree d and length

$$O\left(\lambda \cdot n^{1/(d+1)}\right),$$

showing that this trade-off is tight up to constant factors.

[4]The multiplicative factor λ reflects the bit-length of a single accumulator value, e.g. a Merkle-tree root.

Remark 2 (On λ-dependent accumulator lengths). Corollary 1 isolates the succinct regime in which the accumulator value has bit-length $\mathsf{poly}(\log n)$, and in this regime the trade-off of Theorem 2 yields the sharp lower bound $d(n) = \Omega(\log n / \log\log n)$. This succinct regime also reflects practical settings: Merkle-tree–based accumulators are typically instantiated with a fixed-output hash function, while the number of accumulated values may grow. Corollary 1 thus captures how the required update frequency scales with n under such ultra-succinct representations.

When the accumulator length is allowed to depend polynomially on the security parameter λ, the same trade-off still applies but yields a different quantitative bound. For example, if the accumulator value has length $\ell(\lambda, n) = \lambda^c \cdot \mathsf{poly}(\log n)$, then Theorem 2 implies

$$d(\lambda, n) = \Omega(\log_\lambda n),$$

which is strictly stronger than the previously known $\omega(1)$ lower bound. This behavior is also consistent with the upper-bound framework of [MQ23], which achieves matching trade-offs up to a multiplicative λ factor in the accumulator length.

4.2 Simple Case with Deterministic Addition

In this subsection, we fix a common reference string crs and an arbitrary polynomial function $n = \mathsf{poly}(\lambda)$. As a result, we will omit crs from the inputs of algorithms for simplicity. We consider an arbitrary accumulator scheme $\Lambda = (\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ in which the Add **algorithm is deterministic**. By Definition 2, the accumulator value is compact, and we fix an upper bound $\ell(\lambda, n)$ that is non-decreasing in n. We also fix an arbitrary update function $\mathsf{NeedUpd}_\lambda$ such that Λ is $\mathsf{NeedUpd}_\lambda$-complete according to Definition 3. Since we have fixed both the common reference string and assumed determinism of Add, we may, without loss of generality, treat $\mathsf{NeedUpd}_\lambda$ as a function defined only on $\mathbb{N} \times \mathsf{List}_{\le |\mathcal{X}|}(\mathcal{X})$. Next, we fix an arbitrary degree function deg for Λ with respect to $\mathsf{NeedUpd}_\lambda$ (see Definition 4). Without loss of generality, we assume $\deg(n) = o(n)$. For convenience, we write $d = \deg(n)$, although it should be understood that d is implicitly a function of n. Finally, we define k to be the largest integer such that $n \ge \binom{k+d}{d+1}$, which should also be interpreted as a function of n.

4.2.1 Concepts

We now provide the formal definitions of several core concepts used throughout the proof, which have already been introduced in Subsection 2.2.

Definition 8 (Step Sequence)**.** Let $S = (x_1, \ldots, x_n) \in \mathsf{List}_n(\mathcal{X})$. A *step sequence* of length $k \ge 1$ is a sequence of integer pairs $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ satisfying the following conditions:

1. (**End of sequence.**) $u_k = v_k$. (This defines the endpoint $\mathsf{end} := u_k$ of the sequence.)
2. (**Non-overlapping steps.**) For each $i \in [k-1]$, the step (u_i, v_i) is defined in the following way: $v_i = \max\{j \le u_k : \mathsf{NeedUpd}_\lambda(u_i, S[1:j]) = 1\}$, or, if no such j exists, we define $v_i = u_i$.

   It is then required that $v_i < u_{i+1}$ for $i \in [k-1]$.

We are now ready to state the first structural claim. A more general form of this result will be used and proved later.

Claim 3 (Existence of Step Sequence)**.** *For every list $S \in \mathsf{List}_n(\mathcal{X})$, there exists a step sequence of length k. We denote a fixed such step sequence by* $\mathsf{CANONICAL}(S)$.

We write $\mathsf{CANONICAL}(S) = \{(u_1(S), v_1(S)), \ldots, (u_k(S), v_k(S))\}$.
Then, we define *blocks*.

Definition 9 (Blocks)**.** Let $S = (x_1, \ldots, x_n) \in \mathsf{List}_n(\mathcal{X})$ with

$$\mathsf{CANONICAL}(S) = \{(u_1, v_1), \ldots, (u_k, v_k)\}.$$

The sequence of *blocks* associated with S is defined as follows:

- The first block is $B_1(S) := S[1 : v_1]$.
- For each $i \in [2 : k]$, the i-th block is $B_i(S) := S[v_{i-1} + 1 : v_i]$.

We recall the following observation that explains why blocks are useful in the analysis.

Fact 3 (Use of Blocks)**.** *Suppose that the prefix $S[1 : u_k]$ has been accumulated and let acc_{u_k} denote the corresponding accumulator value. By completeness (Definition 3), for each $i \in [k]$, the witness for the u_i-th added value $S[u_i]$ that is generated after the v_i-th addition will successfully certify its membership with respect to acc_{u_k}.*

Since we have assumed that the Add *algorithm is deterministic*[5]*, the witness produced at that time is uniquely defined as*

$$\mathsf{Wit}(S[1 : v_i], u_i) \;=\; \mathsf{Wit}(B_1(S) \,\|\, \cdots \,\|\, B_i(S), u_i),$$

because $S[1 : v_i] = B_1(S) \,\|\, \cdots \,\|\, B_i(S)$ by construction.

Namely, for every $i \in [k]$, the witness computed using only the blocks $B_1(S), \ldots, B_i(S)$ makes the following verification succeed:

$$\mathsf{Ver}(\mathsf{acc}_{u_k}, S[u_i], \mathsf{Wit}(B_1(S) \,\|\, \cdots \,\|\, B_i(S), u_i)) = \mathsf{true}.$$

4.2.2 Common End and Block-Sampling Oracle

We now analyze the adversarial strategy under a simplified setting that satisfies two key assumptions, which allow for a clean attack. The first assumption is about common end.

Assumption 1 (Common End)**.** *There exists an* end *such that for every $S \in \mathsf{List}_n(\mathcal{X})$, we have $\mathsf{CANONICAL}(S) = \{(u_1, v_1), \ldots, (u_{k-1}, v_{k-1}), (\mathsf{end}, \mathsf{end})\}$.*

We now define the random variables used in the analysis.

Random Variables. Let $\mathsf{S} \sim U_{\mathsf{List}_n(\mathcal{X})}$ be a uniformly random list of n distinct elements from $\mathcal{X}$. We define the canonical step sequence $\mathsf{CANONICAL}(\mathsf{S}) = \{(\mathsf{u}_1, \mathsf{v}_1), \ldots, (\mathsf{u}_k, \mathsf{v}_k)\}$ as the random step sequence associated with S, where $(\mathsf{u}_i, \mathsf{v}_i) := (u_i(\mathsf{S}), v_i(\mathsf{S}))$. Let $\mathsf{B}_1, \ldots, \mathsf{B}_k$ be the sequence of random blocks determined by the random step sequence $\mathsf{CANONICAL}(\mathsf{S})$, i.e., $\mathsf{B}_i := B_i(\mathsf{S})$ for each $i \in [k]$. We also define the random variable $\mathsf{Acc}_{\mathsf{end}} \leftarrow \mathsf{Eval}(\mathsf{S}[1 : \mathsf{end}])$, which is the accumulator value obtained after adding the first end elements of S.

Similar to Subsection 2.2, the key of the proof is the existence of a "good block", which has low (conditional) mutual information with the accumulator value $\mathsf{Acc}_{\mathsf{end}}$. The proof is exactly the same as in Subsections 2.1 and 2.2 and thus omitted.

[5] In Section 4.3, we handle the case where Add is randomized. Briefly, the randomness used during the j-th addition can be incorporated into the description of $S[u_j]$, which renders the resulting addition process deterministic. The accumulator value and the witness for $S[u_i]$ are then derived from the same randomness and therefore remain correlated as required for verification.

Proposition 3 (Good Block)**.** *Either*

$$I(\mathsf{B}_1; \mathsf{Acc}_{\mathsf{end}}) \leq \frac{\ell(\lambda, n)}{k}.$$

or there exists an index $i \in [2:k]$ *such that*

$$I(\mathsf{B}_i; \mathsf{Acc}_{\mathsf{end}} \mid \mathsf{B}_1, \ldots, \mathsf{B}_{i-1}) \leq \frac{\ell(\lambda, n)}{k}.$$

Remark 3. We refer to a block B_i as *good* if its conditional mutual information with $\mathsf{Acc}_{\mathsf{end}}$ is at most $\ell(\lambda, n)/k$, as described in Proposition 3.

Finally, we assume access to the following oracle for block-wise sampling.

Assumption 2 (Block Sampling Oracle)**.** *There exists an oracle* $\mathcal{O}$ *that responds to queries as follows:*

- *If no input is provided, then* $\mathcal{O}()$ *returns a sample* B_1 *drawn from* B_1*.*
- *If* $i < k$ *blocks* $B_1, \ldots, B_i$ *have been provided as input, then* $\mathcal{O}(B_1, \ldots, B_i)$ *returns a sample* B_{i+1} *drawn from the conditional distribution*

$$\mathsf{B}_{i+1} \mid (\mathsf{B}_1 = B_1, \ldots, \mathsf{B}_i = B_i).$$

We now describe an adversary $\mathsf{Adv}^{\mathcal{O}}$ that, using access to the block sampling oracle (Assumption 2) and exploiting the existence of a good block (Proposition 3), succeeds in breaking the soundness of the accumulator scheme (Definition 5).

Attack 1 (Deterministic addition, common end, prefix-consistency, and block sampling oracle)**.** *The adversary* $\mathsf{Adv}^{\mathcal{O}}$ *interacts with the soundness challenger as follows:*

1. *The adversary guesses the number of blocks* k *by sampling* $k \leftarrow [n]$*.*
2. *The adversary uses the oracle* $\mathcal{O}$ *to sample* $B_1, \ldots, B_k \leftarrow \mathsf{B}_1, \ldots, \mathsf{B}_k$*.*
3. *The adversary sends the elements of* $S := B_1 || \cdots || B_k$ *one by one to the challenger* Chal *and receives the resulting accumulator value* $\mathsf{acc}_{\mathsf{end}}$*.*
4. *The adversary guesses the index of the good block by sampling* $i \leftarrow [k]$*.*
   - *If* $i = 1$*, then the adversary resamples* $B'_1 \leftarrow \mathsf{B}_1$ *using* $\mathcal{O}()$*.*
   - *If* $i > 1$*, the adversary resamples* $B'_i \leftarrow \mathsf{B}_i \mid (\mathsf{B}_1 = B_1, \ldots, \mathsf{B}_{i-1} = B_{i-1})$ *using* $\mathcal{O}(B_1, \ldots, B_{i-1})$*.*
5. *The adversary guesses the index* u'_i *by sampling* $u'_i \leftarrow [n]$*, and defines:*

$$x := (B_1 || \cdots || B_{i-1} || B'_i)[u'_i].$$

6. *The adversary computes a witness:*

$$w \leftarrow \mathsf{Wit}(B_1 || \cdots || B_{i-1} || B'_i, u'_i).$$

7. *The adversary outputs* (x, w)*.*

Proposition 4. *Under Assumptions 1 and 2, the PPT adversary* $\mathsf{Adv}^{\mathcal{O}}$ *from Attack 1 breaks the soundness of the accumulator scheme* Λ *with probability at least*

$$\frac{1}{n^3} \cdot \left(1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}\right) - \mathsf{negl}(\lambda).$$

Proof. With probability at least $1/n$, the adversary correctly guesses the value of k. Given this, it guesses the index i of a good block with probability at least $1/n$. Finally, it guesses the corresponding index u'_i with probability at least $1/n$. Thus, the total probability of correctly guessing all three values is at least $1/n^3$.

Conditioned on these guesses, by Proposition 3, Lemma 1, and Fact 2, the verification succeeds with probability at least $1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda,n)}{k}}$.

Moreover, by the argument above, with overwhelming probability $1 - \mathsf{negl}(\lambda)$, the re-sampled value x does not appear in $B_1||\cdots||B_k$, so it was not part of the accumulator input. Therefore, a successful forgery occurs with total probability at least $\frac{1}{n^3} \cdot \left(1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda,n)}{k}}\right) - \mathsf{negl}(\lambda)$, as claimed. □

4.2.3 Subset Block-Sampling Oracle

In this subsubsection, we replace Assumption 1 with the following proposition, which follows from Claim 3.

Proposition 5 (Subset with Common End)**.** *There exists a subset $\mathcal{C} \subseteq \mathsf{List}_n(\mathcal{X})$ such that for every $S \in \mathcal{C}$, the step sequence $\mathsf{CANONICAL}(S) = \{(u_1, v_1), \ldots, (u_k, v_k)\}$ has the same endpoint $u_k = v_k$. Moreover, the cardinality of the subset satisfies $\frac{|\mathcal{C}|}{|\mathsf{List}_n(\mathcal{X})|} \geq \frac{1}{n}$.*

Proof. Every list $S \in \mathsf{List}_n(\mathcal{X})$ has an associated step sequence, and there are only n possible choices for the final step index $u_k = v_k$. By a simple pigeonhole argument, at least $1/n$ fraction of such lists share the same endpoint. □

We now fix such a subset $\mathcal{C}$ and its corresponding endpoint end. We must adapt the relevant random variables and sampling oracle to be defined over the restricted domain $\mathcal{C}$.

Random Variables. Let $\mathsf{S}_\mathcal{C} \sim U_\mathcal{C}$ be a uniformly random list sampled from the subset $\mathcal{C}$. Define the associated step sequence as $\mathsf{CANONICAL}(\mathsf{S}_\mathcal{C}) = \{(\mathsf{u}_{\mathcal{C},1}, \mathsf{v}_{\mathcal{C},1}), \ldots, (\mathsf{u}_{\mathcal{C},k}, \mathsf{v}_{\mathcal{C},k})\}$, where $(\mathsf{u}_{\mathcal{C},i}, \mathsf{v}_{\mathcal{C},i}) := (u_i(\mathsf{S}_\mathcal{C}), v_i(\mathsf{S}_\mathcal{C}))$. Similarly, define the block decomposition $\mathsf{B}_{\mathcal{C},1}, \ldots, \mathsf{B}_{\mathcal{C},k}$, where $\mathsf{B}_{\mathcal{C},i} := B_i(\mathsf{S}_\mathcal{C})$. Let $\mathsf{Acc}_{\mathcal{C},\mathsf{end}} \leftarrow \mathsf{Eval}(\mathsf{S}_\mathcal{C}[1 : \mathsf{end}])$ denote the corresponding accumulator value.

Proposition 3 naturally extends to the restricted domain:

Proposition 6 (Subset Good Block)**.** *Either*

$$I(\mathsf{B}_{\mathcal{C},1}; \mathsf{Acc}_{\mathcal{C},\mathsf{end}}) \leq \frac{\ell(\lambda, n)}{k}.$$

or there exists an index $i \in [2 : k]$ such that

$$I(\mathsf{B}_{\mathcal{C},i}; \mathsf{Acc}_{\mathcal{C},\mathsf{end}} \mid \mathsf{B}_{\mathcal{C},1}, \ldots, \mathsf{B}_{\mathcal{C},i-1}) \leq \frac{\ell(\lambda, n)}{k}.$$

We also redefine the sampling oracle accordingly:

Assumption 3 (Subset Block-Sampling Oracle)**.** *There exists an oracle $\mathcal{O}_\mathcal{C}$ that responds to queries as follows:*

- *If no input is provided, then $\mathcal{O}_\mathcal{C}()$ returns a sample B_1 from $\mathsf{B}_{\mathcal{C},1}$.*
- *If $i < k$ blocks $B_1, \ldots, B_i$ have been provided, then $\mathcal{O}_\mathcal{C}(B_1, \ldots, B_i)$ returns a sample B_{i+1} drawn from the conditional distribution*

$$\mathsf{B}_{\mathcal{C},i+1} \mid (\mathsf{B}_{\mathcal{C},1} = B_1, \ldots, \mathsf{B}_{\mathcal{C},i} = B_i).$$

The idea of the attack in this setting is fully analogous to that in the previous subsubsection. The only difference is that we now restrict our attention to the subset $\mathcal{C}$. Accordingly, we state the result without proof.

Proposition 7. *Under Assumption 3, there exists a PPT adversary* $\mathsf{Adv}^{\mathcal{O}_\mathcal{C}}$ *with access to the oracle* $\mathcal{O}_\mathcal{C}$ *that breaks the soundness of the accumulator scheme* Λ *with probability at least*

$$\frac{1}{n^3} \cdot \left(1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}\right) - \mathsf{negl}(\lambda).$$

4.2.4 No Additional Assumption

In this subsubsection, we remove all assumptions. More specifically, we want to remove Assumption 3, which postulated the existence of the oracle $\mathcal{O}_\mathcal{C}$. As a result, we can no longer assume direct access to samples from the block random variables $\mathsf{B}_{\mathcal{C},1}, \ldots, \mathsf{B}_{\mathcal{C},k}$. Instead, we will show that it suffices to sample from the global distribution $\mathsf{S} \sim U_{\mathsf{List}_n(\mathcal{X})}$ as well as from the conditional distribution $\mathsf{S} \mid \mathsf{S}[1:i] = S$ for any $i \in [n-1]$ and prefix $S \in \mathsf{List}_i(\mathcal{X})$—that is, the distribution of S conditioned on having a fixed prefix. To do so, we need the following definition and claim, whose full combinatorial proof appears in Section 5.

Definition 10 (Prefix-Consistency)**.** A set $\mathcal{C} \subseteq \mathsf{List}_n(\mathcal{X})$ is said to satisfy *prefix-consistency* if for any $S, S' \in \mathcal{C}$ and any index $i \in [n]$ such that $S[1:i] = S'[1:i]$, it holds that

$$(u_j(S), v_j(S)) = (u_j(S'), v_j(S')) \quad \text{for all } j \in [k] \text{ such that } v_j(S) \le i.$$

Claim 4 (Existence of Step Sequence and Good Subset)**.** *For every list* $S \in \mathsf{List}_n(\mathcal{X})$*, there exists a step sequence of length* k*. We fix one such step sequence and denote it by* $\mathsf{CANONICAL}(S)$*. Moreover, there exists a subset* $\mathcal{C} \subseteq \mathsf{List}_n(\mathcal{X})$ *and an index* $\mathsf{end} \in [n]$ *such that:*

1. *The subset* $\mathcal{C}$ *has density at least* $1/n^2$*, i.e.,* $\frac{|\mathcal{C}|}{|\mathsf{List}_n(\mathcal{X})|} \ge \frac{1}{n^2}$*.*
2. *For every* $S \in \mathcal{C}$*, the final step in the step sequence satisfies* $u_k(S) = v_k(S) = \mathsf{end}$*.*
3. *The set* $\mathcal{C}$ *satisfies prefix-consistency (Definition 10).*

We now fix the subset $\mathcal{C}$ and the index end as given by Claim 4. We then have the following two corollaries, which follow respectively from Proposition 1 and Claim 4.

Corollary 2 (From Proposition 1 and Claim 4)**.** *Let* $i \in [n-1]$*. Suppose we first sample* $S \leftarrow \mathsf{S}$ *and then independently sample* $S' \leftarrow \mathsf{S} \mid \mathsf{S}[1:i] = S[1:i]$*. Then we have* $\Pr[S \in \mathcal{C} \text{ and } S' \in \mathcal{C}] \ge \frac{1}{n^4}$*.*

Corollary 3 (From Claim 4)**.** *Fix any* $i \in [k-1]$*, any* i *blocks* $B_1, \ldots, B_i$*, and any list* $S \in \mathcal{C}$*. Then we have* $B_1 || \cdots || B_i \sqsubseteq S$ *if and only if* $B_j(S) = B_j$ *for all* $j \in [i]$*.*

In particular, let $\ell = |B_1 || \cdots || B_i|$ *be the total number of elements in the concatenated prefix. Then the following two conditional distributions are identical:*

$$\mathsf{S}_\mathcal{C} \mid (B_1(\mathsf{S}_\mathcal{C}), \ldots, B_i(\mathsf{S}_\mathcal{C})) = (B_1, \ldots, B_i) \quad \text{and} \quad \mathsf{S}_\mathcal{C} \mid \mathsf{S}_\mathcal{C}[1:\ell] = B_1 || \cdots || B_i.$$

Moreover, to sample $B_{i+1} \leftarrow \mathsf{B}_{\mathcal{C},i+1} \mid (B_1(\mathsf{S}_\mathcal{C}), \ldots, B_i(\mathsf{S}_\mathcal{C})) = (B_1, \ldots, B_i)$*, it suffices to sample* $S \leftarrow \mathsf{S}_\mathcal{C} \mid \mathsf{S}_\mathcal{C}[1:\ell] = B_1 || \cdots || B_i$ *and return* $B_{i+1}(S)$*.*

It is straightforward to adapt the attack to this setting. We begin by explaining the main idea.

Main Idea. By Proposition 6, there exists at least one good block.

- **Case 1:** Suppose B_1 is the good block.
  1. Sample $S \leftarrow \mathsf{S}$. By Claim 4, with probability at least $1/n^2$, we have $S \in \mathcal{C}$. That is, with probability at least $1/n^2$, we obtain a sample $S \leftarrow \mathsf{S}_{\mathcal{C}}$. Then sample $\mathsf{end} \leftarrow [n]$ and accumulate $S[1:\mathsf{end}]$ to obtain $\mathsf{acc}_{\mathsf{end}}$.
  2. Independently resample $S' \leftarrow \mathsf{S}$. Again, with probability at least $1/n^2$, we have $S' \in \mathcal{C}$. Guess the index j where $B_1(S')$ ends, and guess the index u'_1, denoted tgt, by sampling $j \leftarrow [n]$ and $\mathsf{tgt} \leftarrow [n]$.
  3. If all guesses are correct—i.e., end, j, and tgt match the true values—and both $S, S' \in \mathcal{C}$, then by Lemma 1 and Fact 2, the verification algorithm
  $$\mathsf{Ver}(\mathsf{acc}_{\mathsf{end}}, S'[\mathsf{tgt}], \mathsf{Wit}(S'[1:j], S'[\mathsf{tgt}]))$$
  succeeds with probability at least
  $$1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}.$$
- **Case 2:** Suppose B_t is the good block for some $t \in [2:k]$.
  1. Sample $S \leftarrow \mathsf{S}$, sample $\mathsf{end} \leftarrow [n]$, and accumulate $S[1:\mathsf{end}]$ to obtain $\mathsf{acc}_{\mathsf{end}}$. Then guess the index i where $B_t(S)$ starts by sampling $i \leftarrow [n]$.
  2. Resample $S' \leftarrow \mathsf{S} \mid \mathsf{S}[1:i-1] = S[1:i-1]$. By Corollary 2, with probability at least $1/n^4$ we have $S, S' \in \mathcal{C}$. By Corollary 3, we can then guess the index j where $B_t(S')$ ends and the index u'_t, denoted tgt, by sampling $j \leftarrow [n]$ and $\mathsf{tgt} \leftarrow [n]$.
  3. If end, i, j, and tgt are correctly guessed and $S, S' \in \mathcal{C}$, then the verification
  $$\mathsf{Ver}(\mathsf{acc}_{\mathsf{end}}, S'[\mathsf{tgt}], \mathsf{Wit}(S'[1:j], S'[\mathsf{tgt}]))$$
  succeeds with probability at least
  $$1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}.$$

Note that the two cases described above can be unified by sampling $i \leftarrow [n]$ even in Case 1.

Formally, we have the following attack.

Attack 2 (Deterministic Addition)**.** *The following adversary* Adv *is against Definition 5.*

1. *The adversary samples a guess for the end index:* $\mathsf{end} \leftarrow [n]$.
2. *The adversary samples a list:* $S \leftarrow \mathsf{S}$.
3. *The adversary sends elements of* $S[1:\mathsf{end}]$ *one-by-one to the challenger* Chal *and receives the resulting accumulator value* $\mathsf{acc}_{\mathsf{end}}$.
4. *The adversary guesses the index* i *where the good block starts:* $i \leftarrow [\mathsf{end} - 1]$.
   - *If* $i = 1$*, then the adversary resamples* $S' \leftarrow \mathsf{S}$ *and guesses the end index* j *of the block by sampling* $j \leftarrow [n]$.
   - *Otherwise, the adversary samples* $S' \leftarrow \mathsf{S} \mid \mathsf{S}[1:i-1] = S[1:i-1]$ *and guesses the end index* $j \leftarrow [n]$.

5. *The adversary guesses an index* $\mathsf{tgt} \leftarrow [n]$ *and sets* $x := S'[\mathsf{tgt}]$.

6. *The adversary computes a witness:* $w \leftarrow \mathsf{Wit}(S'[1:j], \mathsf{tgt})$.

7. *The adversary outputs* (x, w).

Proposition 8. *The PPT adversary* Adv *in Attack 2 breaks the soundness of the accumulator scheme* Λ *with probability at least*

$$\frac{1}{n^8} \cdot \left(1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}\right) - \mathsf{negl}(\lambda).$$

Proof. The adversary succeeds if the following events all occur:

- It correctly guesses the indices end, i, j, and tgt, which happens with probability at least $1/n^4$.
- Both S and S' lie in $\mathcal{C}$, which occurs with probability at least $1/n^4$ by Corollary 2.
- Conditioned on the above, the verification succeeds with probability at least

$$1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}.$$

Multiplying the success probabilities, we obtain a total success probability of at least

$$\frac{1}{n^8} \cdot \left(1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}\right).$$

Finally, with overwhelming probability, the sampled element x does not appear in $S[1:\mathsf{end}]$, and thus the adversary successfully forges a proof for a non-accumulated value, breaking soundness. The result follows. □

4.3 General Attack With Randomized Addition

In this subsection, we consider an arbitrary accumulator $\Lambda = (\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$, where the algorithm Add is no longer assumed to be deterministic. The structure of the attack is essentially the same as in Subsubsection 4.2.4, once we treat the randomness used by Add as part of the added value x. We present the attack and corresponding result for completeness.

Attack 3 (General)**.** *The following adversary* Adv *is against Definition 5.*

1. *The adversary samples a guess for the end index:* $\mathsf{end} \leftarrow [n]$.

2. *The adversary guesses the start index* i *of the good block:* $i \leftarrow [\mathsf{end} - 1]$.

3. *The adversary samples a list:* $S \leftarrow \mathsf{S}$.

4. *If* $i > 1$*, the adversary sends elements of* $S[1 : i-1]$ *one-by-one to the challenger* Chal*, receiving the accumulator value* acc_{i-1}.

5. *The adversary then sends the elements of* $S[i : \mathsf{end}]$ *to the challenger and receives the accumulator value* $\mathsf{acc}_{\mathsf{end}}$.

6. *Depending on the value of* i*:*

   - *If* $i = 1$*, the adversary resamples* $S' \leftarrow \mathsf{S}$ *and guesses the end index* j *of the good block by sampling* $j \leftarrow [n]$.

- *If $i > 1$, the adversary resamples $S' \leftarrow \mathsf{S} \mid \mathsf{S}[1 : i-1] = S[1 : i-1]$ and again guesses $j \leftarrow [n]$.*

7. *The adversary guesses an index $\mathsf{tgt} \leftarrow [n]$ and sets $x := S'[\mathsf{tgt}]$.*

8. *The adversary generates a witness w for x as follows:*

   - *If $i = 1$, it locally accumulates $S'[1 : j]$ and extracts the witness for x.*
   - *If $i > 1$, it locally adds the elements of $S'[i : j]$ to the prefix $S[1 : i-1]$ using acc_{i-1} and computes the witness w for x.*

9. *The adversary outputs (x, w).*

Proposition 9. *The PPT adversary* Adv *in Attack 3 breaks the soundness of the accumulator scheme Λ with probability at least*

$$\frac{1}{n^8} \cdot \left(1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda, n)}{k}}\right) - \mathsf{negl}(\lambda).$$

Sketch. The analysis mirrors that of Proposition 8. With probability $\geq 1/n^4$, the adversary correctly guesses the indices end, i, j, and tgt. With an additional $1/n^4$ chance, both S and S' fall into the good subset $\mathcal{C}$. Conditioned on these events, the verification succeeds with probability at least $1 - \sqrt{\frac{\ln 2}{2} \cdot \frac{\ell(\lambda,n)}{k}}$ by Lemma 1. Finally, the element x does not appear in the accumulated prefix $S[1 : \mathsf{end}]$ with overwhelming probability, completing the soundness violation. □

Conclusion. This concludes the proof of Theorem 2.

5 Proof of Claim 4

In this section, we prove Claim 4. The proof is divided into two parts:

1. In the first part, we work with graphs that capture update dependencies. To see why this is useful, observe that for any list $S \in \mathsf{List}_n(\mathcal{X})$, we can construct a directed graph on the vertex set $[n]$, where there is an edge from i to j if and only if the value $S[i]$ requires an update at time j, i.e., $\mathsf{NeedUpd}(i, S[1 : j]) = 1$.

2. In the second part, we deduce Claim 4 using the structural properties established in the graph setting.

This reduction allows us to apply combinatorial tools to the update times structure, yielding the prefix-consistency properties required by the claim.

5.1 Update Graphs

5.1.1 Definitions

All graphs are directed. For a graph $G = (\mathcal{V}_G, \mathcal{E}_G)$, the vertex set is either $\mathcal{V}_G = [n]$ or $\mathcal{V}_G = \mathbb{N}$. We write $(i, j) \in G$ to mean that $(i, j) \in \mathcal{E}_G$.

Definition 11 (Forward Graph)**.** A graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ is called a *forward graph* if every edge respects the ordering, i.e., $i \leq j$ for all $(i, j) \in \mathcal{E}_G$.

For a vertex $u \in \mathcal{V}_G$ and any end $v \in \mathbb{N}$, we define:

- $\mathcal{N}_{G,v}(u)$ as the set of out-neighbors of u that are at most v, i.e.,

$$\mathcal{N}_{G,v}(u) := \{j \in \mathcal{V}_G \mid (u, j) \in \mathcal{E}_G \text{ and } j \leq v\}.$$

- The *degree* of u up to v is given by

$$\deg_{G,v}(u) := |\mathcal{N}_{G,v}(u)|.^{6}$$

- The *latest destination* of u up to v is defined as

$$\mathsf{dst}_{G,v}(u) := \begin{cases} \max \mathcal{N}_{G,v}(u) & \text{if } \mathcal{N}_{G,v}(u) \neq \emptyset, \\ u & \text{otherwise.} \end{cases}$$

We now formalize a combinatorial structure called *falling-step sequence*, which plays a key role in this paper. A falling-step sequence captures a sequence of vertices in the update graph whose required updates form a non-overlapping and degree-decreasing pattern.

Definition 12 (Falling-Step Sequence)**.** A sequence of vertex pairs

$$\{(u_1, v_1), \ldots, (u_k, v_k)\}$$

is a *falling-step sequence of type* $(d_1, \ldots, d_{k-1})$ in a forward graph G if the following conditions hold:

1. Terminal condition:
$$u_k = v_k = v_{k-1} + 1.$$
We refer to u_k as the *end* of the falling-step sequence.

2. Step definition:
$$v_i = \mathsf{dst}_{G,u_k}(u_i) \quad \text{and} \quad d_i = \deg_{G,u_k}(u_i) \quad \text{for all } i \in [k-1].$$

3. Monotonicity:
$$d_i \geq d_{i+1} \quad \text{and} \quad v_i < u_{i+1} \quad \text{for all } i \in [k-2].$$

4. Canonicality:
   - v_1 is the smallest vertex such that there exists a vertex u such that $\deg_{G,u_k}(u) = d_1$ and $\mathsf{dst}_{G,u_k}(u) = v_1$; and u_1 is the smallest such u.
   - For all $i = 2, \ldots, k-1$: v_i is the smallest vertex greater than v_{i-1} such that there exists a vertex u with $\deg_{G,u_k}(u) = d_i$ and $\mathsf{dst}_{G,u_k}(u) = v_i$; and u_i is the smallest such u.

5. Maximality:
   - There is no vertex u with $\deg_{G,u_k}(u) > d_1$.
   - For all $i \in [2 : k-1]$, there is no vertex $u > v_{i-1}$ with $\deg_{G,u_k}(u) > d_i$.

In the next subsection, we prove that such sequences exist in any forward graph with bounded out-degree, which will allow us to construct the subset $\mathcal{C}$ used in Claim 4.

[6] By convention, $|\emptyset| = 0$.

5.1.2 Existence and Algorithm

In this subsection, we prove that a falling-step sequence exists in any sufficiently large forward graph with bounded degree. To do this, we present a recursive algorithm that constructs such a sequence by selecting high-degree vertices in a greedy, non-overlapping manner. We begin with a conceptual explanation of the algorithm and its correctness.

Conceptual explanation. The algorithm relies on the following Hockey-stick identity for binomial coefficients:

$$\binom{k+d}{d+1} = \binom{k+d-1}{d} + \binom{k+d-2}{d} + \cdots + \binom{d}{d}.$$

This identity allows us to divide a forward graph on $n = \binom{k+d}{d+1}$ vertices (with out-degree at most d) into k blocks, where:

- The first block contains the first $\binom{k+d-1}{d}$ vertices,
- The second block contains the next $\binom{k+d-2}{d}$ vertices, and so on.

We illustrate the algorithmic ideas through progressively more general cases:

1. **Ideal case.** Suppose there are exactly $k-1$ vertices with out-degree exactly d, denoted $u_1 < u_2 < \cdots < u_{k-1}$, with each u_i lying in the i-th block. Let v_i be the largest neighbor of u_i (i.e., the endpoint of its last outgoing edge), and suppose that each v_i also lies in the i-th block. Then the sequence

   $$\{(u_1, v_1), \ldots, (u_{k-1}, v_{k-1}), (v_{k-1}+1, v_{k-1}+1)\}$$

   forms a valid falling-step sequence of type $(d, \ldots, d)$ ending at $v_{k-1}+1$.

   See Figure 3 for an illustration.

2. **Partial ideal case.** Suppose we find only $j < k-1$ such vertices $u_1 < \cdots < u_j$ with out-degree exactly d, each lying in the i-th block for $i \in [j]$. Let v_i denote the largest neighbor of u_i, and suppose each v_i also lies in the i-th block.

   - We proceed recursively: observe that the $(j+1)$-th block contains

     $$\binom{k+d-j-1}{d} = \binom{k+d-j-2}{d-1} + \cdots + \binom{d-1}{d-1}$$

     vertices, and that all vertices in this block have out-degree at most $d-1$ within the block. This allows us to recursively construct a falling-step sequence of length $k-j$ within the $(j+1)$-th block.

3. **General case.** We search for the smallest v_1 in the first block such that there exists a vertex u_1 with out-degree exactly d, and v_1 is the largest neighbor of u_1, i.e., the endpoint of its last outgoing edge.

   - If no such pair exists, we recursively construct a falling-step sequence of length k entirely within the first block.
   - Otherwise, we select the lexicographically smallest such vertex u_1, and fix the pair (u_1, v_1) as the first step. We then recursively construct a falling-step sequence of length $k-1$ within the next $\binom{k+d-2}{d}$ vertices following v_1, noting that all such vertices have out-degree at most $d-1$ within this range.

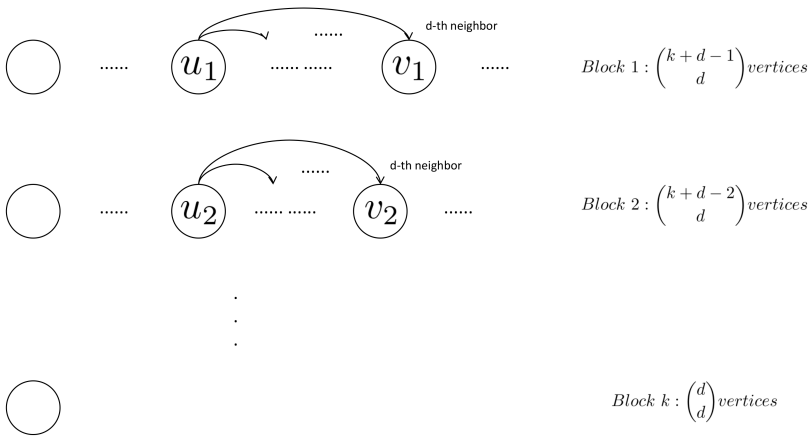

Figure 3: We first divide a forward graph on $n = \binom{k+d}{d+1}$ vertices (with out-degree at most d) into k blocks where the first block contains the first $\binom{k+d-1}{d}$ vertices, the second block contains the next $\binom{k+d-2}{d}$ vertices, and so on. In the ideal case, each block i (except the last one) has exactly one vertex u_i with out-degree d. Furthermore, the largest neighbor v_i of u_i is also inside block i. We simply select $(u_1, v_1), \cdots, (u_{k-1}, v_{k-1})$ and append $(v_{k-1}+1, v_{k-1}+1)$ to form a falling-step sequence.

These ideas are formalized in the recursive algorithm that follows, along with a proposition establishing its correctness.

Algorithm 1 (An Algorithm to Find a Long Falling-Step Sequence)**.** *The following algorithm takes as input a forward graph G, a degree bound d, and a target length k, and outputs a sequence of k pairs of vertices in G satisfying the conditions of a falling-step sequence.*

1. *If there is no vertex $u \leq \binom{k+d-1}{d}$ such that $\deg_{G,\binom{k+d-1}{d}}(u) = d$, then recursively call the algorithm on the induced subgraph G' formed by the first $\binom{k+d-1}{d}$ vertices and all edges among them, with parameters degree $d-1$ and length k. Return the result of this recursive call.*

2. *Otherwise, construct a partial falling-step sequence by iteratively selecting vertices satisfying the required degree and destination conditions. Let i be the largest index such that there exist sequences $\{u_1 < \cdots < u_i\}$ and $\{v_1 < \cdots < v_i\}$ satisfying the following:*

   - *v_1 is the smallest vertex for which there exists a vertex u such that*

     $$\deg_{G,\binom{k+d-1}{d}}(u) = d, \quad v_1 = \mathsf{dst}_{G,\binom{k+d-1}{d}}(u).$$

     Then u_1 is the smallest such vertex that satisfies $\deg_{G,\binom{k+d-1}{d}}(u_1) = d$ and $\mathsf{dst}_{G,\binom{k+d-1}{d}}(u_1) = v_1$.
   - *For each $j \in [i-1]$, define:*

– *v_{j+1} as the smallest vertex larger than v_j for which there exists a vertex $u > v_j$ such that*

$$\deg_{G,v_j+\binom{k-j+d-1}{d}}(u) = d, \quad v_{j+1} = \mathsf{dst}_{G,\binom{k-j+d-1}{d}}(u),$$

and u_{j+1} is the smallest such $u > v_j$.

3. Depending on the value of i:

- *If $i = k-1$, return the sequence*

$$\{(u_1, v_1), \ldots, (u_{k-1}, v_{k-1}), (v_{k-1}+1, v_{k-1}+1)\}.$$

- *Otherwise, recursively call the algorithm on the induced subgraph G' consisting of the next $\binom{k-i+d-1}{d}$ vertices after v_i, with degree $d-1$ and target length $k-i$.*[7] *Let the output of the recursive call be*

$$\{(u_{i+1}, v_{i+1}), \ldots, (u_k, v_k)\}.$$

Then return the full sequence

$$\{(u_1, v_1), \ldots, (u_k, v_k)\}.$$[8]

Proposition 10. *If G is a forward graph on $n \geq \binom{k+d}{d+1}$ vertices with maximum out-degree at most d, then Algorithm 1 terminates on input (G, d, k) and outputs a falling-step sequence of length k.*

Proof. We proceed by induction on d.

Base case ($d = 0$): The only possible vertex degrees are 0, so the algorithm simply returns $\{(1,1),(2,2),\ldots,(k,k)\}$, which trivially satisfies the falling-step sequence conditions.

Inductive step: Assume the claim holds for degrees $0, 1, \ldots, d-1$. We show it also holds for d.

We distinguish two cases:

1. If there is no vertex $u \leq \binom{k+d-1}{d}$ such that $\deg_{G,\binom{k+d-1}{d}}(u) = d$, then the algorithm recurses on degree $d-1$ and length k. By the inductive hypothesis, this recursive call terminates and returns a valid falling-step sequence.

2. Otherwise, the algorithm enters Step 2 and finds the maximal index $i < k$ satisfying the conditions. Then:

   - If $i = k-1$, the algorithm returns a sequence of length k directly, and it satisfies the falling-step conditions by construction.
   - If $i < k-1$, the algorithm recursively calls itself on a graph G' consisting of the next $\binom{k-i+d-1}{d}$ vertices following v_i. To ensure this subgraph is well-defined, observe:

$$v_i + \binom{k-i+d-1}{d} \leq \sum_{j=k-i}^{k} \binom{j+d-1}{d} \leq \binom{k+d}{d+1} \leq n$$

   by the Hockey-stick identity. Hence, the recursive call receives valid input. Furthermore, by the maximality of i, all vertices in G' have degree at most $d-1$, so the degree bound is satisfied. By the inductive hypothesis, the recursive call returns a falling-step sequence of length $k-i$, and by construction, the full sequence of length k satisfies the definition.

[7] We omit the explicit relabeling of vertex indices for clarity, as the procedure is conceptually straightforward.

[8] Relabeling may again be required to merge the results consistently.

Thus, in all cases, the algorithm terminates and returns a valid falling-step sequence of length k. □

We denote by $\mathsf{CANONICAL}_{d,k}(G)$ the falling-step sequence outputted by Algorithm 1 on input (G, d, k).

5.1.3 Prefix-Consistent Subset of Forward Graphs

Definition 13 (Graph Prefix-Consistency)**.** Fix parameters d and k. Let G and G' be two forward graphs on $n \geq \binom{k+d}{d+1}$ vertices, each with maximum out-degree at most d.

Let

$$\mathsf{CANONICAL}_{d,k}(G) = \{(u_1, v_1), \ldots, (u_k, v_k)\},$$

and

$$\mathsf{CANONICAL}_{d,k}(G') = \{(u'_1, v'_1), \ldots, (u'_k, v'_k)\}.$$

We say that G and G' are *prefix-consistent* (with respect to d and k) if the following holds:

- For any $j \in [n]$, if G and G' have the same induced subgraph on $[j]$, and at least one of v_i or v'_i lies in $[j]$, then
$$(u_i, v_i) = (u'_i, v'_i).$$

Proposition 11 (Prefix-Consistency)**.** *Fix d and k, and let $\mathcal{G}$ be a set of forward graphs on $n \geq \binom{k+d}{d+1}$ vertices, each with maximum out-degree at most d. Suppose that for all $G \in \mathcal{G}$, the sequence $\mathsf{CANONICAL}_{d,k}(G)$ has the same end and the same type. Then any two graphs $G, G' \in \mathcal{G}$ are prefix-consistent.*

Proof idea. Let $j \in [n]$ be the largest index such that G and G' have the same induced subgraph on the vertex set $[j]$. Let $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ and $\mathsf{CANONICAL}_{d,k}(G') = \{(u'_1, v'_1), \ldots, (u'_k, v'_k)\}$ be the canonical falling-step sequences in G and G', respectively.

Since both sequences have the same type and endpoint, we know that for each i, the number of neighbors of u_i in G that lie in $[u_k]$ equals the number of such neighbors of u'_i in G'. This guarantees that if $v_i \leq j$, then the pair (u_i, v_i) is a valid choice for the i-th step in G' (due to maximality constraints and by induction). Then, by canonicality, it must be that $(u_i, v_i) = (u'_i, v'_i)$.

This reasoning ensures that all steps of the canonical sequence that lie entirely within $[j]$ must be identical in both G and G', thereby establishing prefix-consistency. We now formalize this argument in the proof below.

Proof. Let

$$\mathsf{CANONICAL}_{d,k}(G) = \{(u_1, v_1), \ldots, (u_k, v_k)\}$$

and

$$\mathsf{CANONICAL}_{d,k}(G') = \{(u'_1, v'_1), \ldots, (u'_k, v'_k)\}.$$

Suppose G and G' have the same induced subgraph on $[j]$. For contradiction, let i be the smallest index such that $v_i \leq j$ and $(u_i, v_i) \neq (u'_i, v'_i)$.

- **Case $i = 1$:** Since the types are the same, we have
$$\deg_{G,u_k}(u_1) = \deg_{G',u'_k}(u'_1).$$

  Step 1: Show $v_1 = v'_1$.

– If $v_1 < v'_1$, then since G and G' agree on $[j]$ and $v_1 < u'_k$, we have

$$\deg_{G',u'_k}(u_1) \geq \deg_{G,u_k}(u_1).$$

By maximality in $S' = \mathsf{CANONICAL}_{d,k}(G')$,

$$\deg_{G',u'_k}(u_1) \leq \deg_{G',u'_k}(u'_1) = \deg_{G,u_k}(u_1).$$

Therefore, we know equality must hold, but $v_1 < v'_1$, which contradicts canonicality in S'.

– The case $v_1 > v'_1$ is symmetric.

Step 2: Show $u_1 = u'_1$.

Since $v_1 = v'_1$ and G, G' agree on $[j]$, we have

$$\deg_{G',u'_k}(u_1) \geq \deg_{G,u_k}(u_1) = \deg_{G',u'_k}(u'_1).$$

Maximality again implies equality, and similarly,

$$\deg_{G,u_k}(u'_1) = \deg_{G,u_k}(u_1).$$

Therefore, both u_1 and u'_1 have equal degree and the same destination under $u_k = u'_k$, which implies $u_1 = u'_1$ by canonicality.

- **Case $i > 1$:** By minimality of i, we have $v_{i-1} = v'_{i-1}$, which in particular implies $u_i > v'_{i-1}$ and $u'_i > v_{i-1}$. (This will allow us to use the maximality condition later.) Since the types are the same, we again have

$$\deg_{G,u_k}(u_i) = \deg_{G',u'_k}(u'_i).$$

Step 1: Show $v_i = v'_i$.

– If $v_i < v'_i$, then since G and G' agree on $[j]$ and $v_i \leq j$, we have

$$\deg_{G',u'_k}(u_i) \geq \deg_{G,u_k}(u_i).$$

Maximality in $S' = \mathsf{CANONICAL}_{d,k}(G')$ implies

$$\deg_{G',u'_k}(u_i) \leq \deg_{G',u'_k}(u'_i) = \deg_{G,u_k}(u_i),$$

so equality holds. But $v_i < v'_i$ contradicts the canonicality of S'.

– The case $v_i > v'_i$ is symmetric.

Step 2: Show $u_i = u'_i$.

As in Case 1, since $v_i = v'_i$ and G, G' agree on $[j]$, we have

$$\deg_{G',u'_k}(u_i) \geq \deg_{G,u_k}(u_i) = \deg_{G',u'_k}(u'_i).$$

By maximality, we get equality. Similarly,

$$\deg_{G,u_k}(u'_i) = \deg_{G,u_k}(u_i).$$

Since both u_i and u'_i share the same degree and destination under the same endpoint $u_k = u'_k$, canonicality implies $u_i = u'_i$.

This contradiction implies that $(u_i, v_i) = (u'_i, v'_i)$ for all i such that $v_i \leq j$, completing the proof. □

5.2 Completing the Proof of Claim 4

As in Subsection 4.2, we fix a crs and an arbitrary polynomial function $n = \mathsf{poly}(\lambda)$. We consider an arbitrary accumulator $\Lambda = (\mathsf{SetUp}, \mathsf{Add}, \mathsf{Ver})$ where the algorithm Add is deterministic, and an arbitrary function $\mathsf{NeedUpd}_\lambda$ such that Λ is $\mathsf{NeedUpd}_\lambda$-complete according to Definition 3. We also fix a degree function deg for Λ with respect to $\mathsf{NeedUpd}_\lambda$ as in Definition 4. Let $d = \deg(n)$, and define k to be the largest integer such that

$$n \geq \binom{k+d}{d+1},$$

which is also a function of n.

It is easy to see that $\mathsf{NeedUpd}_\lambda$ defines a forward graph for any $S \in \mathsf{List}_{\leq n}(\mathcal{X})$: there is an edge from i to j if and only if $\mathsf{NeedUpd}_\lambda(i, S[1:j]) = 1$. Therefore, for any such list S, we write $G(S)$ for the associated forward graph.

For any subset $\mathcal{C} \subseteq \mathsf{List}_{\leq n}(\mathcal{X})$, we define the corresponding set of graphs as

$$\mathcal{G}(\mathcal{C}) := \{G(S) \mid S \in \mathcal{C}\}.$$

Finally, we define the set of canonical falling-step sequences over $\mathcal{C}$ as

$$\mathsf{CANONICAL}_{d,k}(\mathcal{C}) := \{\mathsf{CANONICAL}_{d,k}(G(S)) \mid S \in \mathcal{C}\}.$$

We are now ready to combine these ingredients to complete the proof of Claim 4.

Proposition 12. *There exists a subset $\mathcal{C} \subseteq \mathsf{List}_n(\mathcal{X})$ such that $|\mathcal{C}| \geq \frac{1}{n^2} \cdot |\mathsf{List}_n(\mathcal{X})|$, and such that the set $\mathcal{C}$ is prefix-consistent according to Definition 10 and all sequences in $\mathsf{CANONICAL}_{d,k}(\mathcal{C})$ share the same end.*

Proof. There are at most n possible values for the end u_k of $\mathsf{CANONICAL}_{d,k}(G)$, for any $G \in \mathcal{G}(\mathsf{List}_n(\mathcal{X}))$. By the pigeonhole principle, there exists a subset $\mathcal{C}_1 \subseteq \mathsf{List}_n(\mathcal{X})$ such that $|\mathcal{C}_1| \geq \frac{1}{n} \cdot |\mathsf{List}_n(\mathcal{X})|$, and for all $G(S) \in \mathcal{G}(\mathcal{C}_1)$, the canonical sequence $\mathsf{CANONICAL}_{d,k}(G(S))$ has the same end.

We now bound the number of possible types. Each type is a $(k-1)$-tuple of integers:

- The tuple is non-increasing by construction (see the definition of falling-step sequences).
- Each entry lies in $\{0, 1, \ldots, d\}$, so there are $d+1$ possible values per entry.

This is a standard stars-and-bars counting problem: the number of non-increasing $(k-1)$-tuples with values in $\{0, \ldots, d\}$ is

$$\binom{k+d-1}{d}.$$

Note that this quantity is monotonic in d (for fixed k), and since $n \geq \binom{k+d}{d+1}$ by definition of k, we have

$$n \geq \binom{k+d-1}{d}.$$

Thus, among the $|\mathcal{C}_1|$ lists, there exists a subset $\mathcal{C} \subseteq \mathcal{C}_1$ such that:

$$|\mathcal{C}| \geq \frac{1}{\binom{k+d-1}{d}} \cdot |\mathcal{C}_1| \geq \frac{1}{n} \cdot |\mathcal{C}_1| \geq \frac{1}{n^2} \cdot |\mathsf{List}_n(\mathcal{X})|.$$

Since all graphs in $\mathcal{G}(\mathcal{C})$ share the same type and the same end, Proposition 11 implies that $\mathcal{G}(\mathcal{C})$ is graph prefix-consistent according to Definition 13. Finally, to show that $\mathcal{C}$ satisfies prefix-consistency according to Definition 10, it suffices to observe that for any $S, S' \in \mathcal{C}$ and any index $i \in [n]$ such that $S[1:i] = S'[1:i]$ we have $G(S)$ and $G(S')$ have the same induced subgraph on $[i]$. □

Conclusion. We have completed the proof of Claim 4. By constructing canonical falling-step sequences on forward graphs induced by the update function $\mathsf{NeedUpd}_\lambda$, we showed that a large subset $\mathcal{C} \subseteq \mathsf{List}_n(\mathcal{X})$ exists such that the corresponding update graphs have falling-step sequences with the same type and end. This structural uniformity implies prefix-consistency, which was crucial in the analysis of our attack in Section 4.

Acknowledgement

W.Q. is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019547).

References

[BCCK25] Joseph Bonneau, Jessica Chen, Miranda Christ, and Ioanna Karantaidou. Merkle mountain ranges are optimal: On witness update frequency for cryptographic accumulators. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025: 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part II*, volume 16001 of *Lecture Notes in Computer Science*, page 170–202, Berlin, Heidelberg, 2025. Springer-Verlag. doi:10.1007/978-3-032-01878-6_6.

[BCD+17] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 301–315, 2017. doi:10.1109/EuroSP.2017.13.

[BdM94] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. doi:10.1007/3-540-48285-7_24.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. doi:10.1007/3-540-69053-0_33.

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In Maura B. Paterson, editor, *Cryptography and Coding*, volume 13129 of *Lecture Notes in Computer Science*, pages 129–157, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-92641-0_7.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45708-9_5.

[DKL+23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In Carmit Hazay and Martijn Stam, editors, *Advances in*

Cryptology – EUROCRYPT 2023, volume 14006 of *Lecture Notes in Computer Science*, pages 417–446, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-30620-4_14.

[DMQ25] Jesko Dujmovic, Giulio Malavolta, and Wei Qi. Registration-based encryption in the plain model. In Tibor Jager and Jiaxin Pan, editors, *Public-Key Cryptography – PKC 2025*, volume 15674 of *Lecture Notes in Computer Science*, pages 194–226, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-91820-9_7.

[FKP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, volume 14442 of *Lecture Notes in Computer Science*, pages 166–200, Singapore, 2023. Springer Nature Singapore. doi:10.1007/978-981-99-8733-7_6.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography – PKC 2019*, volume 11443 of *Lecture Notes in Computer Science*, pages 63–93, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-17259-6_3.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from ibe. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography: 16th International Conference, TCC 2018, Panaji, India, November 11–14, 2018, Proceedings, Part I*, volume 11239 of *Lecture Notes in Computer Science*, page 689–718, Berlin, Heidelberg, 2018. Springer-Verlag. doi:10.1007/978-3-030-03807-6_25.

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1065–1079, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3576915.3616596.

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, volume 12170 of *Lecture Notes in Computer Science*, pages 621–651, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-56784-2_21.

[KB21] Ioanna Karantaidou and Foteini Baldimtsi. Efficient constructions of pairing based accumulators. *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021. URL: https://par.nsf.gov/biblio/10397346, doi:10.1109/CSF51468.2021.00033.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg. doi:10.1007/3-540-48184-2_32.

[MQ23] Mohammad Mahmoody and Wei Qi. Online Mergers and Applications to Registration-Based Encryption and Accumulators. In Kai-Min Chung, editor, *4th Conference on Information-Theoretic Cryptography (ITC 2023)*, volume 267 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITC.2023.15, doi:10.4230/LIPIcs.ITC.2023.15.

[MQR22] Mohammad Mahmoody, Wei Qi, and Ahmadreza Rahimi. Lower bounds for the number of decryption updates in registration-based encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 559–587, Cham, 2022. Springer Nature Switzerland. doi:10.1007/978-3-031-22318-1_20.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/978-3-540-30574-3_19.

[RY16] Leonid Reyzin and Sophia Yakoubov. Efficient asynchronous accumulators for distributed pki. In Vassilis Zikas and Roberto De Prisco, editors, *Security and Cryptography for Networks*, volume 9841 of *Lecture Notes in Computer Science*, pages 292–309, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-44618-9_16.