

The Geometry of Witness-Update Tradeoffs in Additive Positive Accumulators

Wei Qi
Bocconi University
Department of Computing Sciences

October 5, 2026

Abstract

An additive positive accumulator represents a growing set by a short public digest. Each inserted element has a membership witness, which may have to be refreshed after later insertions. An execution of n insertions therefore defines a labeled update vector $d = (d_1, \dots, d_n)$, where d_i is the number of strict post-insertion refreshes of the witness created at time i . Previous lower bounds concern scalar quantities such as $\max_i d_i$ and $\sum_i d_i$. Even when both are known, they do not determine how the updates are distributed among the labeled witnesses.

Let $\mathcal{T}_w([n])$ be the set of left-depth vectors of inorder binary trees on $[n]$ whose right-depth is less than w , and define the convex upward region

$$\mathcal{G}_w^{(n)} := \text{conv}(\mathcal{T}_w([n])) + \mathbb{R}_{\geq 0}^n.$$

Our main result constrains the entire expected update vector. For polynomial-time update specifications, let $m = \Theta(1 + L/\log \lambda)$, where L bounds the accumulator length. For every fixed $\delta > 0$ and all sufficiently large λ ,

$$\mathbb{E}[d] \in (1 - \delta) \frac{m}{m-1} \mathcal{G}_{m-1}^{(n)}.$$

Equivalently, there is one distribution $\Pi_{\lambda, \delta}$ over feasible trees such that

$$\mathbb{E}[d_i] \geq (1 - \delta) \frac{m}{m-1} \mathbb{E}_{T \leftarrow \Pi_{\lambda, \delta}}[\ell_T(i)] \quad \text{for every } i.$$

Thus the same tree distribution constrains all labeled coordinates.

The region $\mathcal{G}_w^{(n)}$ is characterized by its nonnegative linear inequalities. If

$$V_w(a) := \min_{T: \max_i \rho_T(i) < w} \sum_i a_i \ell_T(i),$$

then $x \in c\mathcal{G}_w^{(n)}$ exactly when $a \cdot x \geq cV_w(a)$ for every $a \geq 0$. For arbitrary history-dependent update specifications, we establish the corresponding inequality for every predetermined efficiently computable nonnegative weight vector a :

$$\mathbb{E} \left[\sum_i a_i d_i \right] \geq (1 - o(1)) \frac{m}{m-1} V_{m-1}(a),$$

together with a corresponding lower-tail bound. Indicator vectors give position-independent cohort bounds; in particular, every predetermined cohort of size k has benchmark at least $(k - m + 1)_+$ up to the same factor.

The combinatorial basis of these results is that the temporal-blocking vectors used in replay attacks and the vectors in $\mathcal{T}_w([n])$ have the same upward closure. To transfer this characterization to accumulator executions, we use fractional tuple packing, interval rounding, and conditional resampling controlled by the L -bit accumulator state. For the vector theorem, a separating direction is computed from independent executions before the replay attack is applied to a fresh execution.

At fixed width, the same tree vectors describe the coordinatewise lower boundary of online-merger update profiles. Every feasible tree vector is realized by an online merger and, through the standard Merkle-forest construction, by an additive positive accumulator.

1 Introduction

An *additive positive accumulator* maintains a short public digest of a set that grows only through insertions, while allowing each accumulated element to carry a membership witness. As new elements are added, a witness issued earlier may cease to verify against the new digest and must then be refreshed. Succinctness therefore comes with a maintenance cost: reducing the amount of public state may force previously issued witnesses to be updated repeatedly over the lifetime of the system.

The resulting digest-length–witness-update tradeoff has so far been understood mainly through scalar measures of update complexity. Bonneau, Chen, Christ, and Karantaidou [4] prove a tight lower bound on the aggregate number of witness updates in the succinct regime. Qi [10] gives a tight worst-case tradeoff between accumulator length and the maximum lifetime update frequency of a witness.

These results leave open a qualitatively different question. They constrain how large some update count must be, or how many updates must occur in total, but they do not determine *which witnesses* must incur those updates. For an execution of n insertions, let

$$d = (d_1, \dots, d_n),$$

where d_i is the number of strict post-insertion refreshes of the witness created at time i . The previously studied quantities

$$\max_i d_i \quad \text{and} \quad \sum_i d_i$$

are only two scalar projections of d . Even knowing both does not determine the labeled update profile. For example,

$$(2, 1, 0, 0) \quad \text{and} \quad (0, 2, 1, 0)$$

have the same maximum and the same total, but the first insertion incurs two updates in the former vector and none in the latter.

The natural object is therefore the labeled update vector itself, rather than any particular scalar summary. We ask what coordinatewise constraints security imposes on the full update profile:

What geometric constraints does security impose on the full labeled update profile?

Our answer is governed by a family of bounded-right-depth ordered binary trees. Each such tree determines a vector whose i th coordinate is the left-depth of node i . At the combinatorial level, these tree vectors describe the coordinatewise lower boundary exactly: the temporal-blocking vectors that arise in replay arguments have the same upward closure as the feasible tree vectors. Thus the tree family does not merely optimize one particular update statistic; it describes which labeled update profiles are minimal under the coordinatewise order.

The cryptographic statement is a convex version of this geometry. For efficient update specifications, the expected labeled update vector lies in a scaled convex upward closure of the same tree vectors. Equivalently, there is a single distribution over feasible trees whose average left-depth vector is coordinatewise dominated by the expected update vector. The same distribution works simultaneously for every coordinate. This common geometric witness is stronger than separately bounding different scalar summaries of the execution.

Scalar update measures can then be recovered by probing this region in different directions. For a predetermined nonnegative vector

$$a = (a_1, \dots, a_n) \in \mathbb{R}_{\geq 0}^n,$$

the corresponding linear probe is

$$\text{Cost}_a(d) := \sum_{i=1}^n a_i d_i. \tag{1}$$

Equal weights recover total update complexity, while indicator weights $a_i = \mathbf{1}[i \in C]$ measure the update burden of a predetermined cohort $C \subseteq [n]$. More generally, these linear functionals form the supporting inequalities of the convex upward tree region. For arbitrary history-dependent update specifications, where we do not obtain simultaneous vector containment, we still prove each predetermined efficiently computable supporting inequality separately.

This viewpoint also yields concrete consequences. In particular, sufficiently large predetermined cohorts must themselves bear a quantitative update burden, independently of where their members occur in the insertion order; the burden cannot be shifted entirely to witnesses outside the cohort. The same linear probes can also model heterogeneous communication, computation, availability, or delegated-maintenance costs [5, 11]. These interpretations are consequences of the underlying labeled geometry rather than separate optimization problems.

Prior lower-bound arguments provide the architecture for studying this geometry. They combine the limited information in a short public state with combinatorial restrictions on witness updates, represented through forward update graphs or related online-merger abstractions [8, 7]. We study the labeled geometry of this combinatorial update process directly.

We prove two cryptographic lower bounds. When the update specification is efficient, the expected labeled update vector coordinatewise dominates a scaled average of bounded-right-depth tree vectors, where a single distribution over trees works for all coordinates. For arbitrary history-dependent update specifications, we prove the corresponding supporting inequality for every predetermined efficiently computable nonnegative weight vector.

The underlying combinatorial result identifies the possible blocking vectors. Every temporal-blocking vector coordinatewise dominates the left-depth vector of a bounded-right-depth ordered tree, and every such tree vector is realized by a blocker. Thus the blocking and tree families have the same upward closure and the same coordinatewise-minimal elements. Bounded-depth tree resource tradeoffs have classical antecedents [3, 2, 9]; our result connects this tree structure to temporal blocking.

To transfer this characterization to accumulator executions, we relax integral blocking to fractional tuple packing and use interval rounding to control the loss. A conditional resampling argument removes the dependence between the replayed block and the endpoint digest, with the L -bit digest bounding the resulting KL divergence. For the vector theorem, a separating direction is computed from independent executions and then held fixed while the replay attack is applied to a fresh soundness execution.

1.1 Our Results

Let T be a binary tree whose nodes are labeled $1, \dots, n$ in inorder. For a node i , let $\ell_T(i)$ and $\rho_T(i)$ denote the numbers of left and right edges, respectively, on the root-to- i path. Define

$$\mathcal{T}_w([n]) := \left\{ (\ell_T(1), \dots, \ell_T(n)) : \max_i \rho_T(i) < w \right\}$$

and the convex upward region

$$\mathcal{G}_w^{(n)} := \text{conv}(\mathcal{T}_w([n])) + \mathbb{R}_{\geq 0}^n.$$

This region is the object constrained by our main theorem.

Expected-vector theorem. Let L upper-bound the accumulator length after n insertions and, for any fixed constant $c > 1$, set

$$m_L := \max \left\{ 2, \left\lceil c \frac{L+1}{\log_2 \lambda} \right\rceil \right\}, \quad \alpha_L := \frac{m_L}{m_L - 1}.$$

Suppose that the update specification is efficient, and write

$$\mu_\lambda := \mathbb{E}[d_G].$$

For every fixed $\delta \in (0, 1)$ and all sufficiently large λ ,

$$\mu_\lambda \in (1 - \delta)\alpha_L \mathcal{G}_{m_L-1}^{(n)}.$$

Equivalently, there is a distribution $\Pi_{\lambda,\delta}$ over ordered trees of right-depth less than $m_L - 1$ such that

$$\mathbb{E}[d_G(i)] \geq (1 - \delta)\alpha_L \mathbb{E}_{T \leftarrow \Pi_{\lambda,\delta}}[\ell_T(i)] \quad \text{for every } i.$$

The content of the theorem is the existence of one distribution that works simultaneously for all coordinates. Separate lower bounds obtained from different trees or different linear objectives would not imply this statement.

Directional characterization. For a nonnegative weight vector a , define

$$V_w(a) := \min_{T: \max_i \rho_T(i) < w} \sum_{i=1}^n a_i \ell_T(i). \quad (2)$$

The value $V_w(a)$ gives the supporting inequality of $\mathcal{G}_w^{(n)}$ in direction a . More precisely, for every $x \in \mathbb{R}_{\geq 0}^n$ and every $\gamma > 0$,

$$x \in \gamma \mathcal{G}_w^{(n)} \iff a \cdot x \geq \gamma V_w(a) \quad \text{for every } a \in \mathbb{R}_{\geq 0}^n.$$

The expected-vector theorem therefore implies all nonnegative linear lower bounds associated with the tree region using one common point of its convex hull.

For arbitrary history-dependent update specifications, we prove the corresponding inequality separately for every predetermined efficiently computable nonnegative weight vector a :

$$\mathbb{E}[\text{Cost}_a(G)] \geq (1 - o(1)) \frac{m_L}{m_L - 1} V_{m_L-1}(a),$$

with a lower-tail guarantee whenever the benchmark is nonzero. This statement does not assume that the update specification is efficient, but its quantifiers are weaker: the direction a must be fixed independently of the execution. The simultaneous theorem requires a separating direction that may depend on the unknown vector μ_λ ; efficiency allows that direction to be determined from independent executions before a fresh soundness execution is attacked.

Cohort consequences. Taking $a_i = \mathbf{1}[i \in C]$ gives a lower bound on the updates incurred by a predetermined efficiently computable cohort $C \subseteq [n]$. If $k = |C|$ and

$$\Phi_w(k) := \sum_{r \geq 1} \left(k + 1 - \binom{w+r}{r} \right)_+,$$

then

$$\mathbb{E} \left[\sum_{i \in C} d_G(i) \right] \geq (1 - o(1)) \frac{m_L}{m_L - 1} \Phi_{m_L-1}(k).$$

The right-hand side depends only on the size of the cohort, not on the positions of its members. Since

$$\Phi_{m_L-1}(k) \geq (k - m_L + 1)_+,$$

whenever $k \geq m_L$ we also have

$$\mathbb{E} \left[\sum_{i \in C} d_G(i) \right] \geq (1 - o(1)) \frac{m_L}{m_L - 1} (k - m_L + 1).$$

Combinatorial characterization. Let $\mathcal{B}_{w+1}$ be the degree vectors of forward graphs that block every increasing $(w+1)$ -tuple. We prove

$$\mathcal{T}_w \subseteq \mathcal{B}_{w+1} \subseteq \uparrow \mathcal{T}_w, \quad \uparrow \mathcal{B}_{w+1} = \uparrow \mathcal{T}_w.$$

Thus every blocking vector dominates a feasible tree vector, and every feasible tree vector is realized by a blocker. In particular, the blocking vectors and tree vectors have the same coordinatewise-minimal elements. This proves the directional identity

$$\min_{G: G \text{ blocks every } (w+1)\text{-tuple}} \text{Cost}_a(G) = V_w(a),$$

but the underlying statement concerns the full upward-closed family rather than a particular choice of a . A root-decomposition dynamic program computes $V_w(a)$ in polynomial time.

Equal weights and realization. For equal weights, we determine the complete left-depth distribution. Every feasible tree satisfies

$$|\{i : \ell_T(i) \geq r\}| \geq \left(n + 1 - \binom{w+r}{r} \right)_+,$$

and one tree attains equality for every r simultaneously. The maximum and total update bounds follow by taking different scalar functions of this same vector, and the extremal tree minimizes every symmetric coordinatewise nondecreasing objective.

The tree vectors also describe the lower boundary of width- w online-merger profiles. More precisely,

$$V_w(a) = \min_{G: G \text{ blocks every } (w+1)\text{-tuple}} \text{Cost}_a(G) = \min_{M: \text{width}(M) \leq w} \sum_i a_i M(i)$$

for every $a \geq 0$. A bounded-right-depth tree yields a width- w merger whose per-item post-insertion merge vector is its left-depth vector. The standard Merkle-forest transformation realizes the same vector in an additive positive accumulator.

Scope. The simultaneous theorem constrains the expected update vector and requires an efficient update specification. For arbitrary history-dependent specifications, we establish each predetermined nonnegative linear inequality separately. At the combinatorial and online-merger width level, the entire lower boundary is generated by the same tree vectors. Translating width into public-state bits introduces the representation cost of live cryptographic roots: the Merkle construction uses $O(\sigma w + \log n)$ bits, while the lower bound yields an effective width of order $L/\log \lambda$.

1.2 Technical Overview

The proof of the expected-vector theorem proceeds by separation. If the expected update vector lies outside the claimed convex upward region, then some nonnegative linear functional separates it from that region. We first prove the required lower bound for a direction fixed independently of the attacked execution. We then explain how, for efficient update specifications, a violating direction can be computed from independent executions and used in a fresh soundness attack.

The integral tree region. For a forward graph G , define the reach of a vertex by

$$R_G(i) := \max(\{i\} \cup \{t : (i, t) \in E(G)\}).$$

Suppose that G blocks every increasing $(w+1)$ -tuple, and choose

$$k := \min_i R_G(i)$$

as the root of an ordered tree. For every $i < k$, minimality gives $R_G(i) \geq k$, so restricting G to the interval left of k removes at least one outgoing edge from each remaining vertex. This decrease in out-degree accounts for the additional left edge created when the recursively constructed left subtree is attached to k . On the interval right of k , the degrees do not decrease, but the restricted graph blocks every increasing w -tuple. This corresponds to using one unit of right-depth when the recursion enters the right subtree. Recursing on the two intervals produces an ordered tree T satisfying

$$\ell_T(i) \leq d_G(i) \quad \text{and} \quad \max_i \rho_T(i) < w.$$

Conversely, given such a tree, place an edge from each node to every ancestor whose left subtree contains it. The resulting graph has

$$d_G(i) = \ell_T(i)$$

and blocks every increasing $(w + 1)$ -tuple. Hence the blocking vectors and feasible tree vectors have the same upward closure. The function $V_w(a)$ is the minimum of the linear functional $a \cdot x$ over this region.

A fixed separating direction. Fix a predetermined nonnegative weight vector a . The update graph of an accumulator execution need not block every tuple, so the integral characterization cannot be applied to it directly. We instead relax the blocking problem to a fractional cover of increasing m -tuples by possible update edges. Its dual assigns mass to increasing m -tuples subject to one capacity constraint for each possible edge.

After rescaling the dual solution and choosing one of the m tuple positions uniformly, we obtain a distribution on pairs (S, A) such that every edge $e = (i, t)$ satisfies

$$\Pr[e \text{ blocks } (S, A)] \leq \frac{a_i}{Z},$$

where Z is the total rescaled dual mass. Therefore, for every realized update graph G ,

$$\Pr[\text{Bad} \mid G] \leq \min \left\{ 1, \frac{\text{Cost}_a(G)}{Z} \right\},$$

where **Bad** is the event that some realized update edge blocks the sampled pair.

The relation between Z and the tree region follows from interval rounding. For a fixed source vertex i , the possible endpoints of edges blocking a given tuple position form an interval. Each row of a fractional blocker can therefore be rounded as an interval-stabbing instance. The consecutive-ones property gives an integral solution with a loss of at most $m - 1$. LP duality and the integral characterization then give

$$Z \geq \frac{m}{m-1} V_{m-1}(a).$$

Replay and resampling. Let

$$S = (s_1 < \dots < s_m)$$

be sampled from the tuple packing, and let A be a uniform position. The tuple divides the relevant part of the execution into consecutive blocks $B_1, \dots, B_m$. In an ideal experiment, we replay the genuine block B_A from the genuine state immediately preceding it. If **Bad** does not occur, one of the witnesses produced during replay is the witness retained for the target element at the endpoint. Thus a small value of $a \cdot d_G = \text{Cost}_a(G)$ gives a noticeable ideal replay probability.

The ideal experiment is not yet a soundness attack because the genuine block B_A may be correlated with the endpoint accumulator Y . Since Y has length at most L ,

$$H(Y) < L + 1.$$

For every fixed tuple, the chain rule gives

$$\sum_{j=1}^m I(B_j; Y \mid B_{<j}) \leq H(Y) < L + 1.$$

A uniformly selected block therefore has conditional mutual information less than $(L + 1)/m$ with the endpoint. We replace B_A by an independent sample from its conditional distribution given the preceding blocks. If P denotes the ideal law and Q the resampled law, then

$$D_{\text{KL}}(P \parallel Q) \leq \Gamma_m := \frac{(L + 1) \ln 2}{m}.$$

The adversary does not need to evaluate the possibly inefficient update specification. It collects all candidate witnesses produced during replay and uses the public verification algorithm to select one with nontrivial acceptance probability. If $\mathbb{E}[\text{Cost}_a(G)]$ were a constant factor below Z , the corresponding verification event would have constant probability under P . A rare-event bound for KL divergence shows that it retains inverse-polynomial probability under Q whenever $\Gamma_m = O(\log \lambda)$. The resampled experiment is efficiently implementable, and its target is outside the genuine accumulated set except with negligible probability. It would therefore contradict soundness.

Choosing

$$m = \Theta\left(1 + \frac{L}{\log \lambda}\right)$$

makes $\Gamma_m = O(\log \lambda)$. Together with the lower bound on Z , this proves the supporting inequality

$$a \cdot \mathbb{E}[d_G] \geq (1 - o(1)) \frac{m}{m-1} V_{m-1}(a)$$

for every predetermined efficiently computable direction a .

Learning a separating direction. The preceding argument proves one supporting inequality at a time. To establish containment in $\mathcal{G}_w^{(n)}$, suppose instead that

$$\mu := \mathbb{E}[d_G]$$

lies outside the claimed scaled region. Convex separation then gives a nonnegative direction a for which $a \cdot \mu$ is smaller than the corresponding multiple of $V_w(a)$. This direction depends on μ and cannot be obtained from the execution to which the soundness attack is applied.

When the update specification is efficient, we estimate μ from independent canonical executions. A clipping argument allows the separator to be normalized so that

$$0 \leq a_i \leq 1, \quad V_w(a) = 1.$$

Sufficiently accurate coordinatewise estimation of μ then controls the error in every candidate direction. The dynamic program for $V_w(a)$ gives a separation oracle for the constraint $V_w(a) \geq 1$, allowing a violating direction to be computed in polynomial time from the empirical mean.

After the direction and its tuple packing have been computed, the replay attack is run on an independent soundness execution. Conditional on the training transcript, the chosen direction and packing are fixed and independent of this execution. The fixed-direction replay argument therefore applies conditionally on the transcript and rules out failure of the vector containment.

Realizing the tree boundary. Process an ordered tree in inorder and, when a node is inserted, merge the components associated with its left subtree together with the new singleton. The maximum right-depth bounds the number of live components, while the left-depth of an item equals the number of post-insertion rounds in which its component is merged. Hence every vector in $\mathcal{T}_w([n])$ is realized as the per-item merge vector of a width- w online merger.

Conversely, a width- w online merger induces a blocking graph. If an increasing $(w + 1)$ -tuple were unblocked, its items would remain in distinct components through the final selected round, contradicting the width bound. The online-merger profiles therefore have the same coordinatewise lower boundary as the blocking vectors. The standard Merkle-forest construction implements each tree profile as the update vector of an additive positive accumulator.

1.3 Related Work

From update graphs to accumulator tradeoffs. Mahmoodi, Qi, and Rahimi [8] study decryption updates in registration-based encryption. Their lower-bound argument separates the information-theoretic effect of a short public parameter from a combinatorial constraint on the update schedule: updates are encoded by a forward DAG, and long skipping sequences provide the combinatorial structure used by the attack. Their main tradeoff assumes fixed update times, with extensions that allow some additional dependence on the execution.

Mahmoodi and Qi [7] subsequently introduced the online-merger abstraction and used its width–maximum-depth tradeoff for registration-based encryption and public-state accumulators, together with matching-style positive constructions. For additive positive accumulators, Qi [10] establishes a tight worst-case tradeoff between accumulator length and maximum lifetime witness-update frequency, even for history-dependent update schedules. Bonneau, Chen, Christ, and Karantaidou [4] study the aggregate number of witness updates and prove a tight total-update lower bound in the succinct regime, together with strong concentration guarantees. These works extract scalar information from the update process—most notably $\|d\|_\infty$ or $\|d\|_1$. Our characterization acts before either scalar functional is chosen: it identifies the full labeled coordinatewise frontier of d , from which these quantities and other monotone readouts can be recovered. In particular, neither scalar result, nor their conjunction, implies our cohort-weighted lower bounds or the simultaneous containment of the expected labeled update vector.

Abusalah, Anthoine, Avitabile, and Giunta [1] study a complementary time-local notion for short additive accumulators and updatable vector commitments: the expected number of membership proofs invalidated by a single append or update. For constant-size digests approaching the concrete security level, they show that this number becomes nearly maximal. Our object is instead the labeled vector of lifetime update counts across insertion times.

Classical tree resource tradeoffs and merge schedules. Static-to-dynamic transformations and their combinatorial resource tradeoffs go back to Bentley and Saxe [3]. Bentley and Brown [2] developed a general tree-based framework for resource tradeoffs in which binomial trees play a central role. More directly, Mathieu, Staelin, Young, and Yousefi [9] give a bijection between Bigtable merge-compaction schedules and binary search trees: stack size is $1 +$ right-depth, while the number of times a file is merged is $1 +$ left-depth. These works provide classical antecedents for the bounded-right-depth tree optimization used here. Our combinatorial result starts from a different constraint—temporal blocking—and shows that the blocking and bounded-right-depth tree families have the same upward closure, and hence the same coordinatewise-minimal elements.

The online-merger model of Mahmoodi and Qi [7] gives the corresponding positive accumulator construction. Retaining per-item merge counts, rather than only their maximum, yields the weighted identity

$$V_w(a) = \min_{G: G \text{ blocks every } (w+1)\text{-tuple}} \text{Cost}_a(G) = \min_{M: \text{width}(M) \leq w} \sum_i a_i M(i).$$

Thus the blocking lower-bound problem and the operational merger problem meet at the same weighted tree value.

Heterogeneous witness-maintenance costs. Witness maintenance is itself a meaningful system cost in accumulator-based credential and revocation settings [5], and recent work considers delegating proof updates to external servers [11]. Different holders may therefore face different communication, computation, availability, or delegated-maintenance costs. Our objective

$$\sum_i a_i d_i$$

is a simple first model of this heterogeneity. The coordinatewise characterization makes these nonuniform costs exact linear probes of the same labeled update geometry, rather than a separate combinatorial problem.

1.4 Organization

The remainder of the paper is organized as follows. We first define additive positive accumulators, witness-update specifications, and the forward update graph induced by an execution. We then prove the exact ordered-tree characterization and derive its optimization consequences, including the dynamic program for $V_w(a)$. Next we introduce fractional blocking and tuple packing and relate the fractional and integral problems through interval rounding. We then develop the replay and KL-resampling argument, prove the weighted cryptographic lower bound, and strengthen it for efficient update specifications to geometric containment of the expected labeled update vector.

After establishing the general result, we specialize to equal weights, evaluate the extremal tree exactly, and recover maximum, total, and more general symmetric update costs from the same left-depth geometry. Finally, we give the exact online-merger interpretation of $V_w(a)$ and its Merkle-forest realization.

2 Preliminaries

We write

$$[n] := \{1, \dots, n\}, \quad (z)_+ := \max\{z, 0\}.$$

Let $\mathcal{X} = \mathcal{X}_\lambda$ be an efficiently uniformly samplable universe satisfying

$$|\mathcal{X}| \geq 2^{\Omega(\lambda)}.$$

For $n \geq 0$, let $\text{List}_n(\mathcal{X})$ denote the set of ordered lists of n distinct elements of $\mathcal{X}$.

2.1 Additive Positive Accumulators

Definition 2.1 (Additive positive accumulator). An *additive positive accumulator* consists of PPT algorithms

$$\Lambda = (\text{Setup}, \text{Add}, \text{Ver})$$

with the following syntax:

•

$$\text{Setup}(1^\lambda) \rightarrow \text{crs};$$

•

$$\text{Add}(\text{crs}, S, \text{acc}, x; r) \rightarrow (\text{acc}', W),$$

where S is the current ordered list of accumulated elements, $x \notin S$, and W contains a fresh membership witness for every element of $S \parallel x$;

•

$$\text{Ver}(\text{crs}, \text{acc}, x, w) \rightarrow \{0, 1\}.$$

For a fixed setup string crs , an execution of n additions is specified by an ordered list

$$S = (x_1, \dots, x_n) \in \text{List}_n(\mathcal{X})$$

and addition randomness

$$R = (r_1, \dots, r_n).$$

We write

$$(\text{acc}_t, W_t) \leftarrow \text{Add}(\text{crs}, S[1 : t - 1], \text{acc}_{t-1}, x_t; r_t), \quad \text{acc}_0 = \perp, \quad (3)$$

for the resulting sequence of accumulator states and fresh witness lists.

A polynomial-time computable nondecreasing function

$$\ell(\lambda, n)$$

is an *accumulator-length bound* if every legal execution of at most n additions satisfies

$$|\text{acc}_t| \leq \ell(\lambda, t)$$

with probability one. For an n -step execution, we write

$$L := \ell(\lambda, n).$$

2.2 Witness Updates and Security

We allow the witness-update schedule to depend arbitrarily on the realized execution prefix, as in [10].

Definition 2.2 (Update specification). An *update specification* is an arbitrary function

$$\text{NeedUpd}_\lambda : \text{CRS} \times \mathbb{N} \times \text{List}_{\leq |\mathcal{X}|}(\mathcal{X}) \times \mathcal{R}^* \rightarrow \{0, 1\}.$$

Definition 2.3 (Efficient update specification). An update specification is *efficient* if there is a deterministic polynomial-time algorithm that, on input

$$(1^\lambda, \text{crs}, i, S[1 : t], R[1 : t]),$$

computes

$$\text{NeedUpd}_\lambda(\text{crs}, i, S[1 : t], R[1 : t]).$$

The weighted lower bound proved below allows arbitrary update specifications. The stronger vector-level geometric containment additionally assumes that the update specification is efficient.

For a fixed execution (crs, S, R) and times $i < t$, we use the shorthand

$$\text{NeedUpd}(i, t) := \text{NeedUpd}_\lambda(\text{crs}, i, S[1 : t], R[1 : t]). \quad (4)$$

Thus the decision whether to update the witness created at time i after addition t may depend arbitrarily on the entire realized prefix through time t .

For every insertion time i , define its *retained witness* recursively. At insertion,

$$w_{i,i} := W_i[i].$$

For $t > i$, set

$$w_{i,t} := \begin{cases} W_t[i], & \text{NeedUpd}(i, t) = 1, \\ w_{i,t-1}, & \text{NeedUpd}(i, t) = 0. \end{cases} \quad (5)$$

Hence $\text{NeedUpd}(i, t) = 1$ means precisely that, after processing addition t , the retained witness for x_i is replaced by the fresh witness $W_t[i]$.

Both experiments below use the following interaction convention. The experiment samples

$$\text{crs} \leftarrow \text{SetUp}(1^\lambda),$$

gives crs to the adversary, and initializes

$$S = \emptyset, \quad \text{acc}_0 = \perp.$$

On each request for a fresh insertion x_t , it samples fresh independent addition randomness r_t , invokes **Add** as in Eq. (3), appends x_t to S , and returns acc_t and W_t to the adversary. In the retained-witness experiment, witnesses are retained or refreshed according to Eq. (5).

Definition 2.4 (NeedUpd-completeness). Consider the retained-witness experiment in which a computationally unbounded adversary adaptively chooses polynomially many fresh insertions. After any round t , it may stop and designate any previously inserted element x_i , $i \leq t$. The experiment then verifies the retained witness $w_{i,t}$ against the current accumulator:

$$\text{Ver}(\text{crs}, \text{acc}_t, x_i, w_{i,t}).$$

The accumulator is *NeedUpd-complete* if, for every fixed polynomial-round adversary, the probability that this verification rejects is negligible in λ .

Soundness.

Definition 2.5 (Soundness). In the soundness experiment, a PPT adversary adaptively requests polynomially many additions of fresh elements. After each addition it receives the resulting public accumulator value and the corresponding fresh witness list.

If the interaction terminates with accumulated list S and public accumulator acc , the adversary wins by outputting

$$(x^*, w^*)$$

such that

$$x^* \notin S$$

and

$$\text{Ver}(\text{crs}, \text{acc}, x^*, w^*) = 1.$$

The accumulator is *sound* if every PPT adversary wins with negligible probability in λ .

2.3 Update Graphs and Weighted Cost

Every fixed execution (crs, S, R) induces a forward graph

$$G = G(\text{crs}, S, R)$$

on vertex set $[n]$. For $i < t$, the graph contains an edge

$$(i, t)$$

exactly when

$$\text{NeedUpd}(i, t) = 1.$$

Thus an edge (i, t) records an update, at time t , to the witness created at insertion time i . The initial witness $w_{i,i} = W_i[i]$ is not counted as an update.

For each $i \in [n]$, define

$$d_G(i) := |\{t > i : (i, t) \in E(G)\}|.$$

Thus $d_G(i)$ is the total number of subsequent updates to the witness created at time i , and

$$d_G = (d_G(1), \dots, d_G(n))$$

is the *labeled update vector* of the execution.

Definition 2.6 (Weighted update cost). For a predetermined vector

$$a = (a_1, \dots, a_n) \in \mathbb{R}_{\geq 0}^n,$$

where a_i is the cost of one update to the witness created at time i , define

$$\text{Cost}_a(G) := \sum_{(i,t) \in E(G)} a_i = \sum_{i=1}^n a_i d_G(i). \quad (6)$$

The random execution. For the lower-bound argument, we use the following canonical random execution.

Definition 2.7 (Random n -step execution). The experiment

$$\text{Exec}_{\Lambda, n}$$

is defined as follows:

1. sample

$$\text{crs} \leftarrow \text{SetUp}(1^\lambda);$$

2. sample

$$x_1, \dots, x_n$$

uniformly without replacement from $\mathcal{X}$;

3. independently sample the randomness

$$r_1, \dots, r_n$$

used by the successive invocations of **Add**;

4. generate

$$(\text{acc}_t, W_t)_{t=1}^n, \quad (w_{i,t})_{1 \leq i \leq t \leq n}, \quad G$$

according to (3), (4), and (5).

Unless stated otherwise, all execution-dependent random variables in the lower-bound analysis are taken with respect to

$$\text{Exec}_{\Lambda, n}.$$

The weight vector

$$a = a(\lambda)$$

is fixed independently of this execution randomness. We write

$$U_a := \mathbb{E}_{\text{Exec}_{\Lambda, n}} [\text{Cost}_a(G)]. \quad (7)$$

When

$$a = \mathbf{1},$$

this is the expected total number of witness updates.

3 Combinatorial Geometry of Update Vectors

The previous section associates every accumulator execution with a forward graph whose out-degrees form its labeled update vector. We now forget the cryptographic origin of this graph and study the resulting combinatorial object abstractly.

The central question is the following. Suppose a forward graph satisfies a blocking condition strong enough to rule out certain replay intervals. What coordinatewise lower bounds does this impose on its update vector? We show that the blocking vectors and the left-depth vectors of bounded-right-depth ordered binary trees have the same upward closure, and hence the same coordinatewise-minimal elements. This gives a single structural description from which maximum-update, total-update, heterogeneous weighted, and more general monotone objectives can all be recovered.

Fix an integer interval

$$I = [p : q] = \{p, p + 1, \dots, q\}.$$

A forward graph on I is a directed graph with vertex set I whose edges have the form

$$(i, t), \quad i < t.$$

For such a graph, we use the same notation as in [Section 2](#):

$$d_G(i) := |\{t > i : (i, t) \in E(G)\}|,$$

and

$$d_G = (d_G(i))_{i \in I}.$$

Thus d_G is the labeled out-degree vector of the abstract forward graph, and for an execution-induced graph it coincides with the labeled update vector defined in the preliminaries.

Two familiar scalar statistics of this vector are

$$\max_{i \in I} d_G(i)$$

and

$$\sum_{i \in I} d_G(i),$$

corresponding respectively to the largest number of updates incurred by any one witness and the total number of witness updates. Rather than optimize these statistics separately, we characterize the coordinatewise lower boundary of the entire feasible set.

3.1 Temporal Blocking

The blocking relation grows out of the update-graph and skipping-sequence viewpoint used in earlier lower-bound arguments for registration-based encryption [\[8\]](#) and later accumulator update bounds [\[7, 10\]](#). Consider an increasing tuple

$$S = (s_1 < \dots < s_m)$$

of vertices in I . For each $j < m$, the relevant interval is

$$[s_{j+1} : s_m].$$

An edge leaving s_j and ending inside this interval records precisely the type of later update that prevents replay at position j .

Definition 3.1 (Tuple blocking). Let G be a forward graph on I , and let

$$S = (s_1 < \dots < s_m)$$

be an increasing m -tuple of vertices in I . An edge

$$e = (s_j, t)$$

blocks S at position j if $j < m$ and

$$s_{j+1} \leq t \leq s_m.$$

We say that G blocks S if some edge blocks S at some position.

In these replay-based arguments, the absence of an update edge across a selected interval is what allows old information to be reused. Here we take the resulting temporal blocking condition itself as a combinatorial constraint and ask what it forces about the complete labeled vector d_G .

For each $i \in I$, define the reach

$$R_G(i) := \max(\{i\} \cup \{t : (i, t) \in E(G)\}).$$

Thus $R_G(i) = i$ if i has no outgoing edge, and otherwise $R_G(i)$ is the largest endpoint of an edge leaving i .

Since the object of interest is a vector, we use the coordinatewise order. For

$$x, y \in \mathbb{N}^I,$$

write

$$x \leq y$$

if

$$x_i \leq y_i \quad \text{for every } i \in I.$$

For a set

$$A \subseteq \mathbb{N}^I,$$

define its coordinatewise upward closure by

$$\uparrow A := \{x \in \mathbb{N}^I : \text{there exists } y \in A \text{ with } y \leq x\}.$$

The structured vectors that describe this lower boundary arise from ordered binary trees.

Definition 3.2 (Ordered-tree depth vector). An ordered binary tree on I is a binary tree whose inorder traversal visits the labels of I in increasing order. For a node $i \in I$, let

$$\ell_T(i) \quad \text{and} \quad \rho_T(i)$$

denote the numbers of left and right edges, respectively, on the root-to- i path. We call

$$(\ell_T(i))_{i \in I}$$

the left-depth vector of T .

For $m \geq 2$, define

$$\mathcal{B}_m(I) := \{(d_G(i))_{i \in I} : G \text{ is a forward graph on } I \text{ blocking every increasing } m\text{-tuple}\},$$

and for $w \geq 1$ define

$$\mathcal{T}_w(I) := \left\{ (\ell_T(i))_{i \in I} : T \text{ is an ordered binary tree on } I, \max_{i \in I} \rho_T(i) < w \right\}.$$

Our combinatorial problem is therefore to identify the coordinatewise minimal elements of

$$\mathcal{B}_{w+1}(I).$$

We will show that they are exactly the coordinatewise minimal elements of

$$\mathcal{T}_w(I).$$

3.2 The Ordered-Tree Characterization

We first show that every feasible blocking vector dominates an ordered-tree vector.

Theorem 3.3 (Blocking-to-tree domination). *Let $w \geq 1$, and let G be a forward graph on an interval*

$$I = [p : q].$$

Suppose G blocks every increasing $(w + 1)$ -tuple in I . Then there exists an ordered binary tree T on I satisfying

$$\max_{i \in I} \rho_T(i) < w$$

and

$$\ell_T(i) \leq d_G(i) \quad \text{for every } i \in I. \quad (8)$$

Proof. We induct on w , with an inner induction on $|I|$. Empty intervals are trivial. Assume I is nonempty. Let

$$k := \min_{i \in I} R_G(i),$$

and choose

$$p_0 \in I$$

with

$$R_G(p_0) = k.$$

We use label k as the root.

The choice of k produces the two recursions needed for the tree. To the left of k , every vertex loses at least one outgoing edge when we restrict the graph to the left interval. To the right of k , the degree bound is unchanged, but the blocking parameter decreases by one.

Left interval. For every $i < k$, minimality of k gives

$$R_G(i) \geq k > i.$$

Hence i has an outgoing edge (i, t) with

$$t \geq k.$$

Let G_L be the restriction of G to

$$[p : k - 1].$$

At least one outgoing edge is removed from each $i < k$, so

$$d_{G_L}(i) \leq d_G(i) - 1 \quad (i < k). \quad (9)$$

Moreover, G_L blocks every increasing $(w + 1)$ -tuple contained in $[p : k - 1]$. Indeed, if an edge blocks such a tuple, then its first endpoint is one of the tuple vertices and its second endpoint is no larger than the final tuple vertex. Both endpoints therefore lie in $[p : k - 1]$.

By the inner induction, there is an ordered tree T_L on $[p : k - 1]$ with local right-depth less than w and

$$\ell_{T_L}(i) \leq d_{G_L}(i).$$

Attach T_L as the left subtree of root k . Every node in T_L gains one left edge, and hence by (9),

$$\ell_T(i) = \ell_{T_L}(i) + 1 \leq d_G(i).$$

Right interval. Let G_R be the restriction of G to

$$[k + 1 : q].$$

If $w = 1$, then this interval must be empty. Indeed, if $s_1 > k$, then the pair

$$(p_0, s_1)$$

cannot be blocked, because no edge from p_0 reaches s_1 :

$$R_G(p_0) = k < s_1.$$

This contradicts the hypothesis.

Now suppose $w \geq 2$. We claim that G_R blocks every increasing w -tuple in $[k+1 : q]$. Otherwise let

$$s_1 < \dots < s_w$$

be an unblocked w -tuple in G_R . Since

$$p_0 \leq k < s_1 \quad \text{and} \quad R_G(p_0) = k,$$

no edge from p_0 reaches s_1 .

Furthermore, an edge leaving one of the s_j blocks

$$(p_0, s_1, \dots, s_w)$$

exactly when it blocks

$$(s_1, \dots, s_w).$$

Thus

$$(p_0, s_1, \dots, s_w)$$

is an unblocked $(w+1)$ -tuple in G , contradicting the hypothesis.

By the outer induction with parameter $w-1$, there exists an ordered tree T_R on $[k+1 : q]$ of local right-depth less than $w-1$ satisfying

$$\ell_{T_R}(i) \leq d_{G_R}(i) \leq d_G(i).$$

Attach T_R as the right subtree of root k . Every node in T_R gains one right edge and no left edge, so its global right-depth remains less than w , while its left-depth remains at most $d_G(i)$.

Finally, the root k has left-depth zero. This completes the induction. $\square$

Thus

$$\mathcal{B}_{w+1}(I) \subseteq \uparrow \mathcal{T}_w(I).$$

For the converse, we show that every feasible tree vector is itself realized by a blocking forward graph.

Proposition 3.4 (Trees induce exact blockers). *Let $w \geq 1$, and let T be an ordered binary tree on I satisfying*

$$\max_{i \in I} \rho_T(i) < w.$$

Construct a forward graph G_T by placing an edge

$$(i, k)$$

whenever k is an ancestor of i and i lies in the left subtree of k . Then

$$d_{G_T}(i) = \ell_T(i)$$

for every $i \in I$, and G_T blocks every increasing $(w+1)$ -tuple in I .

Proof. The out-degree identity is immediate: the ancestors $k > i$ contributing an edge (i, k) are exactly those for which the root-to- i path enters the corresponding subtree by a left edge.

We prove the blocking statement by induction on w , with an inner induction on $|I|$. Empty intervals are trivial.

For $w = 1$, every nonempty ordered tree of right-depth less than one is a left chain. Hence for every $i < j$, the node j is an ancestor of i , and therefore

$$(i, j) \in E(G_T).$$

Thus every increasing pair is blocked.

For the inductive step, let k be the root, with left and right subtrees

$$T_L, T_R,$$

and let

$$S = (s_1 < \cdots < s_{w+1})$$

be an increasing tuple.

If S lies entirely in T_L , apply the inner induction with parameter w .

If S lies entirely in T_R , then T_R has local right-depth less than $w - 1$. Apply the outer induction to the first w vertices of S . Any edge blocking those w vertices also blocks S .

It remains to consider tuples crossing the root.

Suppose first that

$$k = s_r \in S.$$

If $r = 1$, then the remaining w tuple vertices lie in T_R and are blocked by the outer induction. If $r > 1$, then

$$s_{r-1} \in T_L,$$

so

$$(s_{r-1}, k) \in E(G_T).$$

Since $k = s_r$, this edge blocks S at position $r - 1$.

Finally, suppose

$$k \notin S,$$

and let r be the number of tuple vertices in T_L . The case $r = 0$ was already handled. If $r = 1$, the remaining w tuple vertices lie in T_R and are blocked by the outer induction. If $r \geq 2$, then s_{r-1} and s_r are the final two tuple vertices in T_L . The edge

$$(s_{r-1}, k)$$

exists and satisfies

$$s_r < k \leq s_{w+1},$$

so it blocks S at position $r - 1$.

These cases exhaust all possibilities. □

The two directions give the main combinatorial characterization.

Theorem 3.5 (Coordinatewise blocking characterization). *For every interval I and every $w \geq 1$,*

$$\mathcal{T}_w(I) \subseteq \mathcal{B}_{w+1}(I) \subseteq \uparrow \mathcal{T}_w(I).$$

Consequently,

$$\uparrow \mathcal{B}_{w+1}(I) = \uparrow \mathcal{T}_w(I). \tag{10}$$

Moreover, $\mathcal{B}_{w+1}(I)$ and $\mathcal{T}_w(I)$ have exactly the same coordinatewise minimal elements.

Proof. The inclusion

$$\mathcal{B}_{w+1}(I) \subseteq \uparrow \mathcal{T}_w(I)$$

is [Theorem 3.3](#), while

$$\mathcal{T}_w(I) \subseteq \mathcal{B}_{w+1}(I)$$

follows from [Proposition 3.4](#). Taking upward closures gives (10).

Now let x be coordinatewise minimal in $\mathcal{B}_{w+1}(I)$. The first inclusion gives some

$$y \in \mathcal{T}_w(I)$$

with

$$y \leq x.$$

Since

$$\mathcal{T}_w(I) \subseteq \mathcal{B}_{w+1}(I),$$

minimality of x forces $y = x$. Thus x is also minimal in $\mathcal{T}_w(I)$.

Conversely, let y be minimal in $\mathcal{T}_w(I)$ and suppose

$$x \in \mathcal{B}_{w+1}(I), \quad x \leq y.$$

By the first inclusion there exists

$$z \in \mathcal{T}_w(I)$$

with

$$z \leq x \leq y.$$

Minimality of y in $\mathcal{T}_w(I)$ gives $z = y$, and hence $x = y$. Thus y is also minimal in $\mathcal{B}_{w+1}(I)$. $\square$

Hence bounded-right-depth ordered-tree vectors generate exactly the same upward-closed region as the integral blocking vectors, and the two families have the same coordinatewise-minimal elements.

3.3 Optimization over Update Vectors

The characterization immediately applies to every coordinatewise nondecreasing objective. Let

$$F : \mathbb{N}^n \rightarrow \mathbb{R}$$

be coordinatewise nondecreasing. Then

$$\min_{G: G \text{ blocks every } m\text{-tuple}} F(d_G) = \min_{T: \max_i \rho_T(i) < m-1} F((\ell_T(i))_{i=1}^n). \quad (11)$$

Indeed, [Theorem 3.3](#) shows that every feasible forward-graph vector dominates a feasible tree vector, while [Proposition 3.4](#) realizes every feasible tree vector exactly.

We will primarily use this statement for the weighted update cost

$$\text{Cost}_a(G)$$

defined in [Definition 2.6](#). The weights need not be uniform: they may represent, for example, heterogeneous communication, computation, energy, or delegated-service costs. Choosing

$$a_i = \mathbf{1}[i \in C]$$

isolates the update burden of a designated cohort

$$C \subseteq [n].$$

For $m \geq 2$, define

$$\tau_m(a) := \min_{G: G \text{ blocks every } m\text{-tuple}} \text{Cost}_a(G). \quad (12)$$

On the tree side, define

$$V_w(a) := \min_{T: \max_i \rho_T(i) < w} \sum_{i=1}^n a_i \ell_T(i). \quad (13)$$

Bounded-depth tree resource tradeoffs have classical antecedents [3, 2]. A closely related weighted left-depth objective appears in Bigtable merge compaction, where binary-tree right-depth encodes stack size and left-depth encodes per-file merge count [9]. Here this tree value arises because Theorem 3.5 shows that temporal blocking forces the full labeled vector onto exactly the same frontier.

The update-vector characterization identifies the blocking and tree optimization problems exactly.

Corollary 3.6 (Weighted update cost via ordered trees). *For every $m \geq 2$ and every*

$$\begin{aligned} a &\in \mathbb{R}_{\geq 0}^n, \\ \tau_m(a) &= V_{m-1}(a). \end{aligned} \quad (14)$$

Proof. Let G block every increasing m -tuple. By Theorem 3.3 with

$$w = m - 1,$$

there is a feasible ordered tree T satisfying

$$\ell_T(i) \leq d_G(i)$$

for every i . Since the weights are nonnegative,

$$V_{m-1}(a) \leq \sum_i a_i \ell_T(i) \leq \sum_i a_i d_G(i) = \text{Cost}_a(G).$$

Taking the minimum over G gives

$$V_{m-1}(a) \leq \tau_m(a).$$

Conversely, let T be an optimal tree for $V_{m-1}(a)$. By Proposition 3.4, the graph G_T blocks every increasing m -tuple and satisfies

$$d_{G_T}(i) = \ell_T(i)$$

for every i . Hence

$$\text{Cost}_a(G_T) = V_{m-1}(a),$$

which gives the reverse inequality. $\square$

A direct root decomposition gives a dynamic program for $V_w(a)$. The recurrence reflects the same left-right asymmetry used in the proof of Theorem 3.3: attaching a left subtree increases the left-depth of every node in that subtree by one, whereas attaching a right subtree consumes one unit of right-depth budget. Such root recurrences are standard for bounded-depth tree optimization [2, 9].

For an interval

$$[p : q] \subseteq [n],$$

write

$$A(p, q) := \sum_{i=p}^q a_i,$$

and let

$$V_w(p, q)$$

denote the optimization in (13) restricted to the labels

$$p, \dots, q.$$

We use the conventions

$$V_w(\emptyset) = 0$$

and

$$V_0(p, q) = +\infty$$

for every nonempty interval $[p : q]$.

Proposition 3.7 (Root recursion for the tree value). *For every $w \geq 1$ and every nonempty interval $[p : q]$,*

$$V_w(p, q) = \min_{k=p, \dots, q} \{V_w(p, k-1) + A(p, k-1) + V_{w-1}(k+1, q)\}. \quad (15)$$

Consequently, $V_w(a)$ can be computed by dynamic programming in

$$O(w n^3)$$

arithmetic operations.

Proof. Fix the root label k . The left subtree uses the labels

$$[p : k-1].$$

Attaching it below k adds one left edge to the root path of every node in that subtree, contributing

$$A(p, k-1)$$

in addition to the optimal internal cost

$$V_w(p, k-1).$$

No right edge is consumed when entering the left subtree, so its local right-depth bound remains w .

The right subtree uses the labels

$$[k+1 : q].$$

Attaching it below k adds one right edge and no left edge to every root path in that subtree. Hence its remaining local right-depth budget is $w-1$, and its optimal contribution is

$$V_{w-1}(k+1, q).$$

The root contributes zero left-depth. Minimizing over

$$k = p, \dots, q$$

proves (15). Evaluating the recurrence for all intervals, width parameters, and root choices gives the stated running time. $\square$

The weighted optimization can differ substantially from its unweighted specialization. For example, let

$$n = 8, \quad w = 2, \quad a = (100, 100, 1, 1, 1, 1, 1, 1).$$

One tree minimizing total left-depth has left-depth vector

$$(2, 1, 2, 1, 0, 2, 1, 0).$$

Its total left-depth is 9, but its weighted cost is

$$2 \cdot 100 + 1 \cdot 100 + 2 + 1 + 0 + 2 + 1 + 0 = 306.$$

In contrast, the weighted recurrence returns the feasible left-depth vector

$$(1, 0, 5, 4, 3, 2, 1, 0),$$

whose weighted cost is

$$100 + 5 + 4 + 3 + 2 + 1 = 115.$$

The recurrence certifies that 115 is optimal for this weight vector. The weighted optimum therefore accepts a larger total number of updates in order to protect the expensive coordinates. This illustrates why the labeled update vector contains information that is lost by retaining only the total number of updates.

This completes the integral combinatorial problem. In particular, [Corollary 3.6](#) gives the exact minimum weighted update cost of a forward graph that blocks every increasing m -tuple. A realized accumulator execution, however, need not block every tuple. The next section therefore passes to a fractional blocking problem whose dual produces a distribution over tuples, while preserving a quantitative connection to the exact tree value

$$V_{m-1}(a).$$

4 From Blocking to Replay: Fractional Tuple Packings

By [Corollary 3.6](#), the integral blocking problem is characterized exactly by

$$\tau_m(a) = V_{m-1}(a).$$

The first obstacle in transferring this geometry to a cryptographic execution is that a realized update graph need not block every tuple. We therefore need a randomized counterpart: a distribution over replay tuples for which no individual forward edge accounts for too much probability mass.

This distribution arises from the LP relaxation of the blocking problem. The primal assigns fractional mass to possible forward edges so that every tuple receives total blocking mass at least one, while the dual assigns mass to tuples subject to one capacity constraint per edge. After rescaling, the dual becomes the tuple packing used by the replay argument. We relate its total mass to the exact tree value $V_{m-1}(a)$ by interval rounding and then show that an optimal packing can be computed and sampled in polynomial time.

4.1 Fractional Covers and Tuple Packings

Let

$$\mathcal{E}_n := \{(i, t) : 1 \leq i < t \leq n\}$$

denote the set of possible forward edges on $[n]$. For an edge

$$e = (i, t) \in \mathcal{E}_n$$

and an increasing tuple

$$S = (s_1 < \dots < s_m),$$

write

$$e \triangleright S$$

if e blocks S in the sense of [Definition 3.1](#).

The integral problem defining $\tau_m(a)$ chooses a subset of $\mathcal{E}_n$. We first relax these edge indicators to nonnegative real variables.

Definition 4.1 (Fractional blocker cover). For $m \geq 2$, define

$$\begin{aligned} \tau_m^*(a) := \min \quad & \sum_{(i,t) \in \mathcal{E}_n} a_i x_{i,t} \\ \text{s.t.} \quad & \sum_{e: e \triangleright S} x_e \geq 1 \quad \text{for every increasing } m\text{-tuple } S, \\ & x_e \geq 0 \quad \text{for every } e \in \mathcal{E}_n. \end{aligned} \tag{16}$$

Thus $\tau_m^*(a)$ is the fractional analogue of $\tau_m(a)$: every increasing m -tuple must still be blocked to total weight at least one, but this weight may be distributed fractionally among several possible edges.

The dual LP places a nonnegative variable y_S on every increasing m -tuple:

$$\begin{aligned} \tau_m^*(a) = \max \quad & \sum_S y_S \\ \text{s.t.} \quad & \sum_{S: (i,t) \triangleright S} y_S \leq a_i \quad \text{for every } (i,t) \in \mathcal{E}_n, \\ & y_S \geq 0 \quad \text{for every increasing } m\text{-tuple } S. \end{aligned} \tag{17}$$

The dual has the form needed later: it places as much mass as possible on tuples while limiting, for each possible edge (i,t) , the total mass of tuples blocked by that edge.

The replay argument will additionally choose one of the m tuple positions uniformly. We therefore rescale

$$z_S := m y_S.$$

For a tuple

$$S = (s_1 < \dots < s_m)$$

and a position $A \in [m]$, we say that an edge e *blocks* (S, A) if it blocks S at position A . By definition this can occur only for $A < m$.

Since an edge $e = (i, t)$ can block a fixed tuple at at most one position, the dual constraint becomes

$$\frac{1}{m} \sum_S \sum_{A=1}^m \mathbf{1}[e \text{ blocks } (S, A)] z_S \leq a_i \tag{18}$$

for every

$$e = (i, t) \in \mathcal{E}_n.$$

This motivates the following definition.

Definition 4.2 (Tuple packing). An m -tuple packing for a weight vector

$$a \in \mathbb{R}_{\geq 0}^n$$

is a family of nonnegative masses

$$z = (z_S)_S,$$

indexed by increasing m -tuples, such that for every possible edge

$$e = (i, t) \in \mathcal{E}_n,$$

$$\frac{1}{m} \sum_S \sum_{A=1}^m \mathbf{1}[e \text{ blocks } (S, A)] z_S \leq a_i. \quad (19)$$

Its *total mass* is

$$Z := \sum_S z_S.$$

When $Z > 0$, the normalized packing distribution on pairs (S, A) is defined by

$$\Pr[S = s] = \frac{z_s}{Z}, \quad A \leftarrow [m]$$

independently and uniformly.

Under this normalized distribution,

$$\begin{aligned} \Pr[e \text{ blocks } (S, A)] &= \frac{1}{mZ} \sum_S \sum_{A=1}^m \mathbf{1}[e \text{ blocks } (S, A)] z_S \\ &\leq \frac{a_i}{Z} \end{aligned} \quad (20)$$

for every possible edge

$$e = (i, t).$$

Thus the packing converts the deterministic edge capacities a_i into probability bounds: an edge whose first endpoint has weight a_i can block at most an a_i/Z fraction of the normalized pair distribution.

Every feasible solution of the dual LP (17) yields an m -tuple packing by setting

$$z_S = m y_S.$$

For an optimal dual solution, strong LP duality gives

$$Z = m \sum_S y_S = m \tau_m^*(a). \quad (21)$$

The edge-load property (20) is exactly the local guarantee needed later. It remains to lower-bound the total mass Z in terms of the integral quantity

$$\tau_m(a) = V_{m-1}(a).$$

4.2 Interval Rounding

For a general covering problem, the fractional optimum may be much smaller than the integral optimum. The blocking relation has additional interval structure that limits this integrality gap.

Fix a first endpoint i . For a tuple position whose first vertex is i , the possible second endpoints of edges that block that position form an interval. This allows each row of a fractional blocker to be rounded by interval stabbing.

Lemma 4.3 (Interval rounding). *For every $m \geq 2$ and every*

$$a \in \mathbb{R}_{\geq 0}^n,$$

$$\tau_m(a) \leq (m-1) \tau_m^*(a). \quad (22)$$

Consequently, an optimal dual solution yields an m -tuple packing of total mass

$$Z = m \tau_m^*(a) \geq \frac{m}{m-1} V_{m-1}(a). \quad (23)$$

Proof. Fix a feasible fractional cover x . For each

$$i \in [n],$$

consider the family of integer intervals

$$J = [u : v] \subseteq [i + 1 : n]$$

satisfying

$$\sum_{t \in J} x_{i,t} \geq \frac{1}{m-1}. \quad (24)$$

View the points

$$i + 1, \dots, n$$

as an interval-stabbing instance. The fractional point weights

$$(m-1)x_{i,t}$$

cover every interval satisfying (24). Since the interval-point incidence matrix has the consecutive-ones property and is totally unimodular, the fractional and integral interval-stabbing optima coincide. Hence there exists a set

$$H_i \subseteq [i + 1 : n]$$

that intersects every interval satisfying (24) and obeys

$$|H_i| \leq (m-1) \sum_{t > i} x_{i,t}. \quad (25)$$

Construct a forward graph H by including

$$(i, t)$$

for every

$$t \in H_i.$$

We claim that H blocks every increasing m -tuple. Fix

$$S = (s_1 < \dots < s_m).$$

Since x is a feasible fractional cover,

$$\sum_{j=1}^{m-1} \sum_{t=s_{j+1}}^{s_m} x_{s_j,t} \geq 1.$$

There are $m-1$ summands indexed by j , so for some $j < m$,

$$\sum_{t=s_{j+1}}^{s_m} x_{s_j,t} \geq \frac{1}{m-1}.$$

Therefore

$$[s_{j+1} : s_m]$$

is one of the heavy intervals for row s_j . By construction of H_{s_j} , there exists

$$t \in H_{s_j} \cap [s_{j+1} : s_m].$$

The edge

$$(s_j, t)$$

therefore blocks S at position j .

Thus H blocks every increasing m -tuple. Moreover, by (25),

$$\begin{aligned}\text{Cost}_a(H) &= \sum_i a_i |H_i| \\ &\leq (m-1) \sum_{i < t} a_i x_{i,t}.\end{aligned}\tag{26}$$

Taking the minimum over feasible fractional covers x gives

$$\tau_m(a) \leq (m-1)\tau_m^*(a),$$

proving (22).

Finally, by (21),

$$Z = m\tau_m^*(a).$$

Combining (22) with

$$\tau_m(a) = V_{m-1}(a)$$

from Corollary 3.6 yields

$$Z \geq \frac{m}{m-1}\tau_m(a) = \frac{m}{m-1}V_{m-1}(a).$$

□

The preceding lemma provides the second property needed later. An optimal tuple packing simultaneously satisfies

$$\Pr[e \text{ blocks } (S, A)] \leq \frac{a_i}{Z}$$

for every possible edge

$$e = (i, t),$$

and

$$Z \geq \frac{m}{m-1}V_{m-1}(a).$$

The factor

$$\frac{m}{m-1}$$

comes from two complementary operations. Rounding the fractional cover loses a factor $m-1$, while the rescaling

$$z_S = m y_S$$

contributes a factor m before the target position is chosen uniformly. As m grows, this factor approaches one.

4.3 Efficient Packing and Sampling

For the cryptographic application, existence of the packing is not sufficient: the normalized distribution on tuples must be efficiently samplable. Although the dual LP has one variable for every increasing m -tuple, the primal has only

$$\binom{n}{2}$$

variables. We therefore optimize the primal using a separation oracle and recover a sparse optimal dual solution.

Fix a rational vector x . To separate the tuple constraints in (16), fix the final vertex

$$v = s_m$$

of a candidate tuple and define

$$C_v(i, j) := \sum_{t=j}^v x_{i,t} \quad (i < j \leq v).$$

For an increasing tuple

$$S = (s_1 < \cdots < s_{m-1} < v), \quad s_m = v,$$

the total fractional blocking mass is

$$W_x(S) = \sum_{j=1}^{m-1} C_v(s_j, s_{j+1}). \quad (27)$$

This has a shortest-path interpretation. Consider the ordered DAG with vertex set

$$[v] = \{1, \dots, v\}$$

and an edge

$$i \rightarrow j$$

of weight

$$C_v(i, j)$$

for every $i < j$. An increasing m -tuple ending at v corresponds exactly to an increasing path with m vertices, and its path weight is $W_x(S)$. Hence a tuple constraint ending at v is violated exactly when such a path has weight less than one.

The minimum path weight can be computed by dynamic programming. Let

$$D_r(j)$$

denote the minimum weight of an increasing path with exactly r vertices and final vertex j . Set

$$D_1(j) = 0,$$

and for $r \geq 2$ define

$$D_r(j) = \min_{i < j} \{D_{r-1}(i) + C_v(i, j)\}. \quad (28)$$

The minimum blocking mass of an increasing m -tuple ending at v is therefore

$$D_m(v).$$

Equivalently,

$$D_m(v) = \min_{i < v} \{D_{m-1}(i) + C_v(i, v)\}.$$

For fixed v , the dynamic program takes

$$O(mv^2)$$

time. Checking all possible endpoints v therefore takes

$$O(mn^3)$$

time, after the values $C_v(i, j)$ are made available by prefix sums.

Proposition 4.4 (Efficient tuple packing). *Assume*

$$2 \leq m \leq n,$$

where n is polynomially bounded in λ , and let

$$a \in \mathbb{Q}_{\geq 0}^n$$

have polynomial bit complexity. Then $\tau_m^*(a)$ can be computed exactly in polynomial time.

Moreover, one can compute an optimal dual solution to (17) having polynomial-size support and polynomial encoding length. The corresponding tuple packing

$$z_S = m y_S$$

has total mass

$$Z = m \tau_m^*(a),$$

and whenever $Z > 0$, its normalized tuple distribution can be sampled exactly in probabilistic polynomial time.

Proof. The primal LP (16) has

$$N = \binom{n}{2}$$

variables, one for each possible forward edge, and exponentially many tuple constraints. Since replacing any $x_e > 1$ by 1 preserves feasibility and cannot increase the objective, we may restrict to $0 \leq x_e \leq 1$; the feasible region is therefore a bounded rational polytope.

Given a rational candidate vector x , nonnegativity can be checked directly. For every possible final vertex v , compute

$$C_v(i, j) = \sum_{t=j}^v x_{i,t}$$

and evaluate the dynamic program (28). If

$$D_m(v) < 1,$$

the stored minimizing predecessors recover an increasing m -tuple whose covering constraint is violated. If

$$D_m(v) \geq 1$$

for every v , then all tuple constraints are satisfied. This gives a strong separation oracle in polynomial time.

The primal is a rational polyhedron with polynomially many variables and coefficients of polynomial encoding length. By the constructive optimization–separation equivalence for rational polyhedra [6], the separation oracle yields a polynomial-time algorithm for computing the exact optimum

$$\tau_m^*(a)$$

and an optimal solution to the dual LP (17) represented with polynomially many nonzero variables and polynomial encoding length. More explicitly, the constructive reduction records only polynomially many tuple constraints; at termination these contain a subsystem with the same optimum. The finite dual of that subsystem, extended by zero on all other tuple variables, is therefore an optimal solution of (17) with polynomial support.

The sparsity can also be seen directly. The dual has

$$N = \binom{n}{2}$$

nontrivial packing constraints, one for each possible forward edge, in addition to nonnegativity. Hence whenever the optimum is finite, there exists an optimal basic feasible dual solution with at most N positive variables. Such a solution may therefore be written as

$$(S_1, y_{S_1}), \dots, (S_r, y_{S_r}), \quad r \leq \binom{n}{2},$$

with all masses rational and of polynomial encoding length.

Suppose now that

$$Z = m \sum_S y_S > 0.$$

Since

$$z_S = m y_S,$$

normalizing the z_S is equivalent to normalizing the y_S .

Take a common denominator D for the positive masses and write

$$y_{S_j} = \frac{b_j}{D}, \quad b_j \in \mathbb{N}.$$

Because the dual solution has polynomial encoding length, so do the integers b_j and

$$B := \sum_{j=1}^r b_j.$$

Sampling a uniform integer from

$$\{1, \dots, B\}$$

and locating it in the cumulative sums of the b_j samples S_j with probability

$$\frac{b_j}{B} = \frac{y_{S_j}}{\sum_S y_S} = \frac{z_{S_j}}{Z}.$$

Thus the normalized tuple distribution is exactly samplable in probabilistic polynomial time. $\square$

We have therefore transformed the exact integral structure of [Section 3](#) into an efficiently samplable m -tuple packing with two quantitative guarantees:

$$\Pr[e \text{ blocks } (S, A)] \leq \frac{a_i}{Z} \quad \text{for every } e = (i, t),$$

and

$$Z \geq \frac{m}{m-1} V_{m-1}(a).$$

The next section applies these two properties to the forward graph induced by a random accumulator execution. Together with the L -bit bound on the endpoint accumulator, they yield a lower bound on its weighted update cost.

5 The Digest Length–Witness Update Tradeoff

[Section 4](#) constructs an efficiently samplable m -tuple packing of total mass Z with two quantitative properties. Under its normalized distribution on pairs (S, A) ,

$$\Pr[e \text{ blocks } (S, A)] \leq \frac{a_i}{Z} \quad \text{for every } e = (i, t),$$

while

$$Z \geq \frac{m}{m-1} V_{m-1}(a).$$

We now apply these properties to the forward graph induced by the random accumulator execution from [Definition 2.7](#). The proof first considers an ideal replay in which the selected block of the genuine execution can be reproduced exactly. If no edge of the realized update graph blocks the sampled pair (S, A) , this replay produces the witness retained for the target at the endpoint. The packing bound therefore controls the failure probability by the realized weighted update cost.

The second obstacle is information-theoretic. The ideal replay is not itself an implementable soundness attack because the selected block and the genuine endpoint accumulator arise from the same execution and may be correlated. We remove this correlation by independently resampling the selected block from its conditional distribution. Since the endpoint accumulator has length at most L , it carries fewer than $L + 1$ bits of entropy even when its length is variable. This bounds the KL divergence introduced by the resampling step. Public verification then identifies a valid witness among the replay candidates, and a rare-event KL argument transfers noticeable success of the ideal replay to a non-negligible soundness attack.

5.1 Ideal Replay

Sample a genuine execution

$$\omega \leftarrow \text{Exec}_{\Lambda, n},$$

and use the notation

$$(x_t, \text{acc}_t, W_t, w_{i,t}, G)$$

from [Section 2](#). Independently sample a pair (S, A) from the normalized distribution of an m -tuple packing

$$z = (z_S)_S$$

of total mass $Z > 0$. Thus

$$S = (s_1 < \dots < s_m)$$

is sampled with probability

$$\frac{z_S}{Z},$$

and

$$A \leftarrow [m]$$

is uniform and independent. Write

$$i := s_A$$

for the target insertion time.

For a fixed tuple S , partition the portion of the genuine execution from s_1 through s_m into blocks. For $j < m$, let

$$B_j := ((x_{s_j}, r_{s_j}), \dots, (x_{s_{j+1}-1}, r_{s_{j+1}-1})),$$

and let

$$B_m := (x_{s_m}, r_{s_m}).$$

We also write

$$Y := \text{acc}_{s_m}$$

for the genuine endpoint accumulator.

Definition 5.1 (Ideal replay). For a sampled pair (S, A) , the *ideal replay* starts from the genuine accumulated list

$$(x_1, \dots, x_{s_A-1})$$

and genuine accumulator state

$$\text{acc}_{s_A-1},$$

and reruns the successive invocations of **Add** specified by the genuine block B_A , using the same inserted elements and the same addition coins as in the genuine execution.

During this replay, every fresh witness output for the target element x_{s_A} is placed in a candidate list

$$\mathcal{C}(S, A),$$

together with $\perp$.

The ideal replay is an information-theoretic experiment; in particular, it is allowed to reuse the genuine addition coins. Because it begins from the genuine accumulated list and accumulator state and reuses the genuine block exactly, it reproduces the genuine execution throughout the selected block.

The only obstruction to obtaining the endpoint witness is therefore an update to the target after the selected block.

Definition 5.2 (Bad update event). For a realized update graph G and sampled pair (S, A) , the event **Bad** occurs if

$$A < m$$

and there exists

$$(s_A, t) \in E(G)$$

such that

$$s_{A+1} \leq t \leq s_m.$$

For $A = m$, the event **Bad** does not occur.

By [Definition 3.1](#), **Bad** occurs exactly when some edge of the realized update graph blocks the sampled pair (S, A) .

Lemma 5.3 (Correctness of the ideal replay). *Conditioned on $\neg \text{Bad}$, the candidate list $\mathcal{C}(S, A)$ contains the witness*

$$w_{s_A, s_m}$$

retained for the target by the genuine execution at time s_m .

Proof. If $A = m$, the replay consists of the genuine insertion at time s_m . It therefore produces

$$W_{s_m}[s_m] = w_{s_m, s_m},$$

which is the retained endpoint witness.

Suppose now that $A < m$. Since the ideal replay reproduces the genuine execution on

$$[s_A : s_{A+1} - 1],$$

its candidate list contains

$$w_{s_A, s_{A+1}-1}.$$

On $\neg \text{Bad}$ there is no edge

$$(s_A, t) \in E(G)$$

with

$$s_{A+1} \leq t \leq s_m.$$

By the definition of the execution-induced update graph, the retained witness for x_{s_A} is therefore never replaced after time $s_{A+1} - 1$ and before the endpoint. Hence

$$w_{s_A, s_{A+1}-1} = w_{s_A, s_m},$$

and the latter belongs to $\mathcal{C}(S, A)$. □

The tuple packing bounds the probability of this obstruction.

Lemma 5.4 (Packing bound for bad updates). *For every fixed realized update graph G ,*

$$\Pr[\text{Bad} \mid G] \leq \min \left\{ 1, \frac{\text{Cost}_a(G)}{Z} \right\}. \quad (29)$$

Consequently,

$$\Pr[\text{Bad}] \leq \frac{U_a}{Z}. \quad (30)$$

Proof. Fix an edge

$$e = (i, t) \in E(G).$$

By the normalized tuple-packing guarantee from [Section 4](#),

$$\Pr[e \text{ blocks } (S, A)] \leq \frac{a_i}{Z}.$$

Conditioning on the realized graph G , the event Bad implies that at least one edge of G blocks the sampled pair. A union bound therefore gives

$$\Pr[\text{Bad} \mid G] \leq \sum_{(i,t) \in E(G)} \frac{a_i}{Z} = \frac{\text{Cost}_a(G)}{Z}.$$

Combining this with the trivial upper bound 1 proves (29).

Averaging over

$$\omega \leftarrow \text{Exec}_{\Lambda, n}$$

and using the definition

$$U_a = \mathbb{E}_{\text{Exec}_{\Lambda, n}}[\text{Cost}_a(G)]$$

gives (30). $\square$

Thus the combinatorial packing has a direct cryptographic consequence: whenever the realized weighted update cost is small relative to Z , the ideal replay is likely to contain the genuine endpoint witness.

The ideal experiment is nevertheless unavailable to a real adversary, because it reuses the genuine random block B_A . We next replace this block by an independent conditional resample and quantify the resulting change of distribution.

5.2 Decoupling Replay from the Endpoint State

Fix a tuple

$$S = (s_1 < \dots < s_m).$$

Let

$$B_0 := (\text{crs}, (x_1, r_1), \dots, (x_{s_1-1}, r_{s_1-1}))$$

denote the setup string together with the genuine execution data preceding the first block, and write

$$B_{<j} := (B_0, B_1, \dots, B_{j-1}).$$

The endpoint accumulator

$$Y = \text{acc}_{s_m}$$

is a binary string of length at most L . Since its length may vary,

$$|\text{supp}(Y)| \leq \sum_{\ell=0}^L 2^\ell = 2^{L+1} - 1.$$

Hence

$$H(Y) \leq \log_2(2^{L+1} - 1) < L + 1.$$

The chain rule for mutual information gives

$$\begin{aligned} \sum_{j=1}^m I(B_j; Y \mid B_{<j}) &= I(B_1, \dots, B_m; Y \mid B_0) \\ &\leq H(Y \mid B_0) \\ &\leq H(Y) < L + 1. \end{aligned} \tag{31}$$

Consequently, if

$$A \leftarrow [m]$$

is uniform,

$$\mathbb{E}_A [I(B_A; Y \mid B_{<A})] < \frac{L + 1}{m}. \tag{32}$$

We now define the two distributions compared by the KL argument. For fixed (S, j) , let

$$P_{S,j}$$

be the genuine joint law of

$$(B_{<j}, B_j, Y)$$

induced by

$$\text{Exec}_{\Lambda, n}.$$

Define

$$Q_{S,j}$$

on the same space by

$$Q_{S,j}(b_{<j}, b_j, y) := P(b_{<j})P(b_j \mid b_{<j})P(y \mid b_{<j}). \tag{33}$$

Thus $Q_{S,j}$ preserves the genuine marginal of $B_{<j}$ and the genuine conditional marginals of B_j and Y , but makes B_j and Y conditionally independent given $B_{<j}$.

Since mutual information is measured in bits,

$$D_{\text{KL}}(P_{S,j} \parallel Q_{S,j}) = (\ln 2) I(B_j; Y \mid B_{<j}). \tag{34}$$

Now sample (S, A) according to the normalized tuple-packing distribution, independently of the genuine execution. Let P denote the joint law of

$$(S, A, B_{<A}, B_A, Y)$$

in the genuine experiment, and let Q be the law obtained by replacing, conditional on (S, A) ,

$$P_{S,A} \quad \text{with} \quad Q_{S,A}.$$

Because (S, A) has the same marginal under both laws,

$$\begin{aligned} D_{\text{KL}}(P \parallel Q) &= \mathbb{E}_{S,A} [D_{\text{KL}}(P_{S,A} \parallel Q_{S,A})] \\ &\leq \Gamma_m := \frac{(L + 1) \ln 2}{m}. \end{aligned} \tag{35}$$

Thus the cost, in KL divergence, of replacing the genuine selected block by an independent conditional resample is at most Γ_m .

We next show that the resampled law has an efficient operational realization.

Lemma 5.5 (Efficient realization of the resampled law). *Up to the target-collision abort defined below, the distribution Q can be generated by a probabilistic polynomial-time adversary in the soundness experiment.*

Proof. The adversary first samples (S, A) from the normalized tuple-packing distribution. It then generates the canonical genuine execution through time s_m : it samples distinct insertion values according to Definition 2.7, submits them successively to the addition interface, and retains the public accumulator state immediately before time s_A as well as the endpoint accumulator

$$Y = \text{acc}_{s_m}.$$

The adversary next generates a fresh block

$$B'_A$$

independently of the genuine continuation, according to the conditional law

$$P(B_A \mid B_{<A}).$$

Although $B_{<A}$ contains the genuine prefix coins as analysis variables, the adversary need not recover them. Under Definition 2.7, the conditional future-block law is determined operationally by the already used prefix values together with fresh independent addition coins: future insertion values are uniform without replacement from the remaining universe. Hence the law is efficiently samplable from the public prefix values. Uniform sampling without replacement can be implemented by rejection sampling; truncating after polynomially many trials introduces only negligible abort probability because $|\mathcal{X}| \geq 2^{\Omega(\lambda)}$ and $n = \text{poly}(\lambda)$.

Starting from the stored prefix list

$$(x_1, \dots, x_{s_A-1})$$

and accumulator

$$\text{acc}_{s_A-1},$$

the adversary locally applies **Add** to the fresh block B'_A . Conditional on $B_{<A}$, the fresh block and the genuine endpoint accumulator are therefore independent and have conditional laws

$$P(B_A \mid B_{<A}) \quad \text{and} \quad P(Y \mid B_{<A}),$$

respectively. This realizes the factorization (33).

Let

$$x'_{\text{tar}}$$

be the target element occupying position s_A in the fresh block. The adversary aborts if

$$x'_{\text{tar}} \in \{x_1, \dots, x_{s_m}\}.$$

Conditioned on the prefix, x'_{tar} is uniform over all elements not already used before s_A . Since

$$|\mathcal{X}| \geq 2^{\Omega(\lambda)} \quad \text{and} \quad n = \text{poly}(\lambda),$$

the probability that it coincides with one of the at most n genuine inserted elements is negligible. $\square$

The laws P and Q defined above contain only the variables needed for the information-theoretic comparison. We henceforth extend both laws by the same randomized post-processing: replay the selected block from the stored prefix state, form the candidate list, run the public candidate-selection procedure defined below, and perform one final independent invocation of **Ver**. Under P the replay block is B_A ; under Q it is its independent resample B'_A .

By the data-processing inequality,

$$D_{\text{KL}}(P \parallel Q) \leq \Gamma_m \tag{36}$$

continues to hold for the extended experiments. We retain the notation P and Q for these extended laws.

5.3 From Replay to a Soundness Attack

The replay produces at most one target witness per invocation of **Add**, and hence at most n such witnesses. Together with $\perp$, the candidate list has size at most $n + 1$. The adversary does not know which candidate is the retained endpoint witness and need not evaluate **NeedUpd**. Instead it uses the public verification algorithm.

For a fixed verification instance

$$(\text{crs}, \text{acc}, x)$$

and witness w , define

$$p(w) := \Pr[\text{Ver}(\text{crs}, \text{acc}, x, w) = 1],$$

where the probability is over the verifier randomness.

Lemma 5.6 (Public candidate selection). *There exists a PPT procedure **Select** with the following property. Given a public verification instance and at most $n + 1$ candidate witnesses, suppose some candidate w^* satisfies*

$$p(w^*) \geq 1 - \delta.$$

Then, except with probability at most

$$R_{\text{sel}}\delta + 2^{-\lambda-2},$$

where

$$R_{\text{sel}} := \lambda + \lceil \log_2(n + 1) \rceil + 2,$$

*the output of **Select** has true verification probability at least $1/2$.*

Proof. For every candidate w , run

$$R_{\text{sel}}$$

independent invocations of **Ver**, and return the first candidate accepted in every trial. If no candidate passes all trials, return an arbitrary candidate.

The distinguished candidate w^* fails at least one trial with probability at most

$$R_{\text{sel}}\delta.$$

Any candidate w satisfying

$$p(w) < \frac{1}{2}$$

passes all R_{sel} trials with probability less than

$$2^{-R_{\text{sel}}}.$$

Since there are at most $n + 1$ candidates, a union bound gives

$$\Pr[\exists w : p(w) < 1/2 \text{ and } w \text{ passes every trial}] \leq (n + 1)2^{-R_{\text{sel}}} \leq 2^{-\lambda-2}.$$

Outside these two failure events, some candidate passes all trials and every passing candidate has true acceptance probability at least $1/2$. $\square$

We now define the event whose probability will be transferred from the ideal law to the resampled law. Apply **Select** to the replay candidate list against the endpoint accumulator Y , and let

$$\widehat{w}$$

be its output. Then perform one fresh, independent verification.

Under P , set

$$x_{\text{tar}} := x_{s_A}.$$

Under Q , let x_{tar} be the element occupying the corresponding target position in the resampled block. Define

$$E := \{\text{Ver}(\text{crs}, Y, x_{\text{tar}}, \hat{w}) = 1\}. \quad (37)$$

We first bound this event under the ideal law.

Lemma 5.7 (Ideal replay success). *Under P ,*

$$P(E) \geq \frac{1}{2} \mathbb{E}_{\text{Exec}_{\Lambda, n}} \left[\left(1 - \frac{\text{Cost}_a(G)}{Z} \right)_+ \right] - \text{negl}(\lambda). \quad (38)$$

Consequently,

$$P(E) \geq \frac{1}{2} \left(1 - \frac{U_a}{Z} \right) - \text{negl}(\lambda). \quad (39)$$

Proof. By Lemma 5.3, on $\neg\text{Bad}$ the candidate list contains

$$w_{s_A, s_m},$$

the witness retained for the target at the genuine endpoint.

Let

$$\delta_{\text{ret}} := 1 - \Pr[\text{Ver}(\text{crs}, Y, x_{s_A}, w_{s_A, s_m}) = 1]$$

on $\neg\text{Bad}$, and set

$$\delta_{\text{ret}} := 0$$

on Bad .

Consider the retained-witness experiment from Definition 2.4 in which the adversary internally samples (S, A) , generates the canonical random execution, stops at time s_m , and designates x_{s_A} as the element to be verified. This is a fixed polynomial-round adversary. Hence NeedUpd-completeness implies

$$\mathbb{E}[\delta_{\text{ret}}] = \text{negl}(\lambda).$$

Since

$$R_{\text{sel}} = \text{poly}(\lambda),$$

also

$$R_{\text{sel}} \mathbb{E}[\delta_{\text{ret}}] = \text{negl}(\lambda).$$

Conditioned on $\neg\text{Bad}$, apply Lemma 5.6 using the retained witness as the distinguished candidate. Except with negligible probability, Select therefore returns a candidate whose true acceptance probability is at least $1/2$. The final independent verification then gives

$$P(E) \geq \frac{1}{2} \Pr[\neg\text{Bad}] - \text{negl}(\lambda). \quad (40)$$

For each fixed realized graph G , Lemma 5.4 gives

$$\Pr[\neg\text{Bad} \mid G] \geq \left(1 - \frac{\text{Cost}_a(G)}{Z} \right)_+.$$

Averaging over

$$\text{Exec}_{\Lambda, n}$$

and substituting into (40) proves (38).

Finally,

$$(1 - x)_+ \geq 1 - x$$

for all real x , and therefore

$$\mathbb{E} \left[\left(1 - \frac{\text{Cost}_a(G)}{Z} \right)_+ \right] \geq 1 - \frac{U_a}{Z}.$$

This gives (39). □

Thus a constant multiplicative gap between the expected weighted update cost and Z produces a constant ideal success probability. It remains to show that the KL divergence between P and Q cannot suppress such an event below inverse-polynomial probability.

5.4 Weighted Transfer for Arbitrary Update Specifications

We use the following elementary rare-event consequence of KL divergence.

Lemma 5.8 (KL rare-event transfer). *Let P, Q be distributions on the same measurable space, and let E be an event. Write*

$$p := P(E) > 0, \quad q := Q(E).$$

If

$$D_{\text{KL}}(P\|Q) \leq \Gamma,$$

then

$$q \geq p \exp\left(-1 - \frac{\Gamma}{p}\right). \quad (41)$$

Proof. By data processing under the indicator of E ,

$$\Gamma \geq p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q}.$$

Moreover,

$$\ln(1-p) \geq -\frac{p}{1-p},$$

so

$$(1-p) \ln(1-p) \geq -p.$$

Since

$$-(1-p) \ln(1-q) \geq 0,$$

the second term in the binary KL divergence is at least $-p$. Therefore

$$p \ln \frac{p}{q} \leq \Gamma + p.$$

Rearranging gives (41). □

In particular, if

$$P(E) \geq \varepsilon$$

for some fixed constant $\varepsilon > 0$ and

$$\Gamma_m = O(\log \lambda),$$

then

$$Q(E) \geq \lambda^{-O(1)}.$$

By [Lemma 5.5](#), the resampled experiment is implementable by a PPT adversary except for a negligible target-collision abort. On the complement of this abort, the resampled target has not been accumulated in the genuine execution. Hence event E is a valid soundness forgery.

We can now state the cryptographic transfer theorem independently of how the tuple packing was obtained.

Theorem 5.9 (General tuple-packing lower bound). *Fix a sound, NeedUpd-complete additive positive accumulator family. Let*

$$n = n(\lambda)$$

be polynomially bounded, let

$$a = a(\lambda) \in \mathbb{Q}_{\geq 0}^n$$

be efficiently computable with polynomial bit complexity, and suppose there is an efficiently samplable feasible m -tuple packing of total mass

$$Z = Z(\lambda) > 0.$$

If

$$\frac{L+1}{m} = O(\log \lambda), \tag{42}$$

then

$$U_a \geq (1 - o(1))Z. \tag{43}$$

Moreover, for every fixed $\delta > 0$,

$$\Pr_{\text{Exec}_{\Lambda, n}} [\text{Cost}_a(G) \leq (1 - \delta)Z] = o(1). \tag{44}$$

Proof. Suppose first that (43) fails. Then for some fixed

$$\delta > 0$$

and an infinite subsequence of security parameters,

$$U_a \leq (1 - \delta)Z.$$

By Lemma 5.7,

$$p := P(E) \geq \frac{\delta}{2} - \text{negl}(\lambda),$$

so p is bounded below by a positive constant along this subsequence.

By (36) and (42),

$$D_{\text{KL}}(P \| Q) \leq \Gamma_m = O(\log \lambda).$$

Applying Lemma 5.8 gives

$$Q(E) \geq \lambda^{-O(1)}.$$

The resampled experiment is generated by the PPT adversary from Lemma 5.5; except with negligible probability its target is absent from the genuine accumulated set. Hence the adversary violates soundness with non-negligible probability, a contradiction.

Since this rules out

$$U_a \leq (1 - \delta)Z$$

for every fixed $\delta > 0$, we obtain

$$U_a \geq (1 - o(1))Z.$$

For the tail statement, suppose instead that (44) fails. Then there exist fixed

$$\delta, \eta > 0$$

and an infinite subsequence such that

$$\Pr_{\text{Exec}_{\Lambda, n}} [\text{Cost}_a(G) \leq (1 - \delta)Z] \geq \eta.$$

On this event,

$$\left(1 - \frac{\text{Cost}_a(G)}{Z}\right)_+ \geq \delta.$$

Therefore [Lemma 5.7](#) gives

$$P(E) \geq \frac{\eta\delta}{2} - \text{negl}(\lambda),$$

which is again bounded below by a positive constant. The same KL-transfer argument yields a non-negligible soundness forgery, a contradiction. $\square$

We now instantiate [Theorem 5.9](#) with the optimal tuple packing constructed in [Section 4](#).

Theorem 5.10 (Weighted witness-update lower bound). *Let Λ be a fixed sound, NeedUpd-complete additive positive accumulator family. Let*

$$n = n(\lambda)$$

be polynomially bounded, and let

$$a(\lambda) \in \mathbb{Q}_{\geq 0}^n$$

be efficiently computable with polynomial bit complexity. Fix any constant $c > 1$ and define

$$m_L := \max \left\{ 2, \left\lceil c \frac{L+1}{\log_2 \lambda} \right\rceil \right\}. \quad (45)$$

If

$$m_L \leq n,$$

then

$$U_a \geq (1 - o(1)) \frac{m_L}{m_L - 1} V_{m_L-1}(a). \quad (46)$$

If

$$V_{m_L-1}(a) > 0$$

for all sufficiently large λ , then for every fixed $\delta > 0$,

$$\Pr_{\text{Exec}_{\Lambda,n}} \left[\text{Cost}_a(G) \leq (1 - \delta) \frac{m_L}{m_L - 1} V_{m_L-1}(a) \right] = o(1). \quad (47)$$

Proof. By [Proposition 4.4](#), an optimal dual solution to the fractional blocking LP yields an efficiently samplable m_L -tuple packing. By [Lemma 4.3](#), its total mass satisfies

$$Z \geq \frac{m_L}{m_L - 1} V_{m_L-1}(a). \quad (48)$$

By the definition of m_L ,

$$\frac{L+1}{m_L} = O(\log \lambda),$$

so [Theorem 5.9](#) applies and gives

$$U_a \geq (1 - o(1))Z.$$

Combining this with (48) proves (46).

For the tail statement, assume

$$V_{m_L-1}(a) > 0.$$

Since (48) implies

$$Z \geq \frac{m_L}{m_L - 1} V_{m_L-1}(a),$$

the event

$$\text{Cost}_a(G) \leq (1 - \delta) \frac{m_L}{m_L - 1} V_{m_L-1}(a)$$

is contained in

$$\{\text{Cost}_a(G) \leq (1 - \delta)Z\}.$$

The claim follows from (44). $\square$

For integers $w \geq 1$ and $k \geq 0$, write

$$\Phi_w(k) := \sum_{r \geq 1} \left(k + 1 - \binom{w+r}{r} \right)_+. \quad (49)$$

Corollary 5.11 (Cohort update lower bound). *Under the assumptions of Theorem 5.10, let $C = C(\lambda) \subseteq [n]$ be any predetermined efficiently computable cohort and write $k = |C|$. Then*

$$\mathbb{E} \left[\sum_{i \in C} d_G(i) \right] \geq (1 - o(1)) \frac{m_L}{m_L - 1} \Phi_{m_L-1}(k). \quad (50)$$

If $\Phi_{m_L-1}(k) > 0$ for all sufficiently large λ , then for every fixed $\delta > 0$,

$$\Pr \left[\sum_{i \in C} d_G(i) \leq (1 - \delta) \frac{m_L}{m_L - 1} \Phi_{m_L-1}(k) \right] = o(1). \quad (51)$$

Moreover,

$$\mathbb{E} \left[\sum_{i \in C} d_G(i) \right] \geq (1 - o(1)) \frac{m_L}{m_L - 1} (k - m_L + 1)_+. \quad (52)$$

In particular, $\Phi_{m_L-1}(k) > 0$ whenever $k \geq m_L$.

Proof. Set $a = \mathbf{1}_C$. Fix a feasible ordered tree T with $\max_i \rho_T(i) < w$. A node with $\ell_T(i) < r$ has a root-to-node word containing fewer than r left steps and fewer than w right steps, and there are at most

$$\binom{w+r}{r} - 1$$

such words. Hence any cohort C of size k contains at least

$$\left(k + 1 - \binom{w+r}{r} \right)_+$$

nodes of left-depth at least r . Summing over r gives

$$\sum_{i \in C} \ell_T(i) \geq \Phi_w(k).$$

Taking the minimum over feasible trees yields

$$V_w(\mathbf{1}_C) \geq \Phi_w(k).$$

Now apply Theorem 5.10 with $a = \mathbf{1}_C$. This proves (50). If $\Phi_{m_L-1}(k) > 0$, then also $V_{m_L-1}(\mathbf{1}_C) > 0$, and (51) follows from the lower-tail part of Theorem 5.10. Finally, for $w = m_L - 1$, the $r = 1$ term gives

$$\Phi_{m_L-1}(k) \geq (k - m_L + 1)_+,$$

which proves (52) and is positive whenever $k \geq m_L$. $\square$

The bound (50) depends only on $|C|$ and is uniform over the positions of the cohort. It is a size-only relaxation of $V_w(\mathbf{1}_C)$, which may be larger for a particular labeled cohort. Thus every sufficiently large predetermined cohort has an intrinsic maintenance burden, independent of its location in the insertion order.

When

$$L = \Omega(\log \lambda),$$

the parameter m_L is of order

$$\frac{L}{\log \lambda}.$$

More generally,

$$m_L = \Theta\left(1 + \frac{L}{\log \lambda}\right).$$

Thus the digest length determines the effective right-depth parameter appearing in the ordered-tree optimization. The exact combinatorial quantity

$$V_{m_L-1}(a)$$

from Section 3, transferred through the tuple packing of Section 4, becomes a lower bound on the weighted witness-update cost of an arbitrary sound accumulator.

Remark 5.12 (Fixed-family quantifiers). The $o(1)$ in Theorem 5.10 is interpreted separately for each fixed asymptotically sound accumulator family and each fixed efficiently generated weight family. In particular, the theorem does not take a pointwise infimum over all asymptotically secure accumulator families before taking the asymptotic limit.

5.5 From Linear Probes to Geometric Containment

Theorem 5.10 probes the update geometry one efficiently generated weight family at a time. The main geometric cryptographic statement requires resolving a further quantifier mismatch: a separating direction for a violation of the convex tree region may itself depend on the unknown expected update vector, whereas the weighted theorem requires the direction to be fixed independently of the attacked execution. When the update specification is efficient, we resolve this by a learn-then-replay reduction: learn a violating direction from independent executions and use it only in a fresh soundness attack.

For $w \geq 1$, define the *convex upward tree region*

$$\mathcal{G}_w^{(n)} := \text{conv}(\mathcal{T}_w([n])) + \mathbb{R}_{\geq 0}^n. \quad (53)$$

Thus $\mathcal{G}_w^{(n)}$ is the convex hull of feasible left-depth vectors, closed upward coordinatewise. The tree value V_w is precisely its nonnegative support function from below.

Lemma 5.13 (Support characterization of the tree region). *For every $c > 0$ and every $x \in \mathbb{R}_{\geq 0}^n$,*

$$x \in c\mathcal{G}_w^{(n)} \iff a \cdot x \geq cV_w(a) \text{ for every } a \in \mathbb{R}_{\geq 0}^n. \quad (54)$$

Proof. Suppose first that

$$x \in c\mathcal{G}_w^{(n)}.$$

Then for some

$$z \in \text{conv}(\mathcal{T}_w([n])) \quad \text{and} \quad r \in \mathbb{R}_{\geq 0}^n$$

we have

$$x = cz + r.$$

For every $a \geq 0$,

$$a \cdot x \geq c a \cdot z \geq c \min_{v \in \mathcal{T}_w([n])} a \cdot v = cV_w(a).$$

Conversely, suppose

$$x \notin c\mathcal{G}_w^{(n)}.$$

Since $\mathcal{T}_w([n])$ is finite, $\text{conv}(\mathcal{T}_w([n]))$ is compact; its Minkowski sum with the closed cone $\mathbb{R}_{\geq 0}^n$ is therefore closed. Hence $c\mathcal{G}_w^{(n)}$ is a closed convex upward set, so strong separation gives a vector a such that

$$a \cdot x < \inf_{y \in c\mathcal{G}_w^{(n)}} a \cdot y.$$

Necessarily $a \geq 0$: if some coordinate a_i were negative, upward closure would make the infimum on the right equal to $-\infty$. Therefore

$$\inf_{y \in c\mathcal{G}_w^{(n)}} a \cdot y = c \min_{v \in \mathcal{T}_w([n])} a \cdot v = cV_w(a),$$

which violates the right-hand side of (54). $\square$

The following normalization is what makes a separating direction learnable from polynomially many sampled executions.

Lemma 5.14 (Clipping a normalized separator). *Let $a \in \mathbb{R}_{\geq 0}^n$. If*

$$V_w(a) \geq 1,$$

and

$$b_i := \min\{a_i, 1\},$$

then

$$V_w(b) \geq 1.$$

Consequently, whenever $x \in \mathbb{R}_{\geq 0}^n$ and

$$x \notin c\mathcal{G}_w^{(n)},$$

there exists $a \in [0, 1]^n$ satisfying

$$V_w(a) = 1 \quad \text{and} \quad a \cdot x < c.$$

Proof. Fix any feasible tree T . If there is an index i with

$$a_i > 1 \quad \text{and} \quad \ell_T(i) \geq 1,$$

then

$$\sum_j b_j \ell_T(j) \geq b_i \ell_T(i) \geq 1.$$

Otherwise every coordinate changed by clipping has left-depth zero, and hence

$$\sum_j b_j \ell_T(j) = \sum_j a_j \ell_T(j) \geq 1.$$

Taking the minimum over feasible trees proves $V_w(b) \geq 1$.

For the consequence, Lemma 5.13 gives some $a \geq 0$ with

$$a \cdot x < cV_w(a).$$

Since $x \geq 0$, the strict inequality implies $V_w(a) > 0$. Scale a so that $V_w(a) = 1$, clip it coordinatewise to obtain b , and let

$$\gamma := V_w(b) \geq 1.$$

Then

$$\hat{a} := \frac{b}{\gamma}$$

satisfies

$$0 \leq \hat{a}_i \leq 1, \quad V_w(\hat{a}) = 1,$$

and, because $x \geq 0$,

$$\hat{a} \cdot x \leq b \cdot x \leq a \cdot x < c.$$

□

For $x \in \mathbb{R}_{\geq 0}^n$, define

$$\beta_w(x) := \min \{a \cdot x : 0 \leq a \leq \mathbf{1}, V_w(a) \geq 1\}. \quad (55)$$

Proposition 5.15 (Efficient bounded separation). *Let $1 \leq w < n$, and let $x \in \mathbb{Q}_{\geq 0}^n$ have polynomial bit complexity. Then $\beta_w(x)$ and a minimizing rational vector a of polynomial bit complexity can be computed in polynomial time. Moreover, for all $c > 0$,*

$$x \notin c\mathcal{G}_w^{(n)} \implies \beta_w(x) < c. \quad (56)$$

Proof. Because $w < n$, every feasible ordered tree has at least one left edge: a tree with all left-depths zero would be the right chain and would have maximum right-depth $n - 1 \geq w$. Hence

$$V_w(\mathbf{1}) \geq 1,$$

so the feasible region in (55) is nonempty.

Write the constraint $V_w(a) \geq 1$ as

$$a \cdot v \geq 1 \quad \text{for every } v \in \mathcal{T}_w([n]).$$

Together with $0 \leq a \leq \mathbf{1}$, this is a bounded rational polytope. Given a candidate a , the dynamic program of Proposition 3.7 computes $V_w(a)$ and, by backtracking, a minimizing tree. If $V_w(a) < 1$, its left-depth vector is a violated constraint; otherwise all tree constraints are satisfied. Thus we have a polynomial-time separation oracle. Every returned tree vector has integer coordinates in $[0, n - 1]$, so the separating inequalities have polynomial encoding length; on polynomial-bit rational input this is a polynomial-bit strong separation oracle. Constructive optimization–separation equivalence for rational polyhedra [6] therefore computes the exact optimum and a rational minimizer of polynomial bit complexity in polynomial time.

Finally, if $x \notin c\mathcal{G}_w^{(n)}$, the second part of Lemma 5.14 supplies a feasible $a \in [0, 1]^n$ with $V_w(a) = 1$ and $a \cdot x < c$. Hence $\beta_w(x) < c$. □

We now prove the main geometric cryptographic statement. The support characterization alone is not enough: a separating direction for a violation of the tree region may depend on the unknown expected update vector and hence cannot simply be hard-wired into the one-probe theorem. Our *learn-then-replay* reduction first learns such a direction from independent canonical executions, freezes it, and only then runs the packing-and-replay attack on a fresh soundness execution. Conditional on the training transcript, the learned direction is therefore fixed throughout the cryptographic reduction.

Theorem 5.16 (Expected-vector geometric containment). *Let Λ be a fixed sound, NeedUpd-complete additive positive accumulator family whose update specification is efficient in the sense of Definition 2.3. Let*

$$n = n(\lambda)$$

be polynomially bounded, fix a constant $c > 1$, and define m_L as in (45). Suppose

$$m_L \leq n,$$

and set

$$w_L := m_L - 1, \quad \alpha_L := \frac{m_L}{m_L - 1}, \quad \mu_\lambda := \mathbb{E}_{\text{Exec}_{\Lambda, n}}[d_G].$$

Then for every fixed $\delta \in (0, 1)$, for all sufficiently large λ ,

$$\mu_\lambda \in (1 - \delta)\alpha_L \mathcal{G}_{w_L}^{(n)}. \quad (57)$$

Equivalently, for every such δ and all sufficiently large λ , there is a distribution $\Pi_{\lambda, \delta}$ over ordered trees T on $[n]$ with

$$\max_i \rho_T(i) < w_L$$

such that simultaneously for every $i \in [n]$,

$$\mathbb{E}[d_G(i)] \geq (1 - \delta)\alpha_L \mathbb{E}_{T \leftarrow \Pi_{\lambda, \delta}}[\ell_T(i)]. \quad (58)$$

Proof. Fix $\delta \in (0, 1)$ and suppose, toward a contradiction, that (57) fails for infinitely many security parameters. Write

$$\alpha := \alpha_L, \quad w := w_L, \quad \mu := \mu_\lambda,$$

and set

$$\varepsilon := \frac{\delta}{8}.$$

We first describe a polynomial-time training phase that is independent of the soundness execution. Because the update specification is efficient, an algorithm can sample an independent canonical execution

$$\omega \leftarrow \text{Exec}_{\Lambda, n}$$

and compute its entire vector d_G exactly by evaluating $\text{NeedUpd}(i, t)$ for all $i < t$. Since

$$0 \leq d_G(i) \leq n,$$

polynomially many independent training executions suffice, by Hoeffding's inequality and a union bound, to produce an empirical mean $\hat{\mu}$ such that

$$\|\hat{\mu} - \mu\|_\infty \leq \frac{\varepsilon}{n} \quad (59)$$

with probability $1 - \text{negl}(\lambda)$. For example, $O(n^4(\lambda + \log n)/\varepsilon^2)$ samples suffice. On the event (59), every $a \in [0, 1]^n$ satisfies

$$|a \cdot \hat{\mu} - a \cdot \mu| \leq \varepsilon. \quad (60)$$

The failure of (57) and Lemma 5.14 imply

$$\beta_w(\mu) < (1 - \delta)\alpha.$$

Using the same proposition, compute from $\hat{\mu}$ an exact minimizer

$$\hat{a} \in \arg \min \{a \cdot \hat{\mu} : 0 \leq a \leq \mathbf{1}, V_w(a) \geq 1\}.$$

On the event (59), choose a minimizer a^* for $\beta_w(\mu)$. Then

$$\begin{aligned} \hat{a} \cdot \mu &\leq \hat{a} \cdot \hat{\mu} + \varepsilon \\ &\leq a^* \cdot \hat{\mu} + \varepsilon \\ &\leq a^* \cdot \mu + 2\varepsilon \\ &< (1 - \delta)\alpha + \frac{\delta}{4} \\ &\leq \left(1 - \frac{\delta}{2}\right)\alpha, \end{aligned} \quad (61)$$

where the last inequality uses $\alpha \geq 1$.

The training algorithm next applies [Proposition 4.4](#) to the learned weight vector $\hat{a}$ and $m = m_L$, obtaining an efficiently samplable m_L -tuple packing of total mass $\hat{Z}$. Since

$$V_w(\hat{a}) \geq 1,$$

[Lemma 4.3](#) gives

$$\hat{Z} \geq \alpha V_w(\hat{a}) \geq \alpha. \quad (62)$$

Combining (61) and (62), on the good training event we have

$$U_{\hat{a}} = \hat{a} \cdot \mu \leq \left(1 - \frac{\delta}{2}\right) \hat{Z}, \quad (63)$$

where $U_{\hat{a}}$ denotes expectation over a fresh execution, conditional on the learned weight vector.

We now run the replay attack of [Section 5](#) on an independent genuine soundness execution, using the learned packing. Include the entire training transcript among the random variables under both the ideal law P and the resampled law Q . The transcript has the same marginal under both laws and is independent of the genuine soundness execution. Conditional on any training transcript, the packing distribution is therefore fixed and independent of that execution, exactly as required in [Lemma 5.4](#).

Let **Good** denote the event (59). By (63), conditioned on every transcript in **Good**,

$$\Pr[\text{Bad} \mid \text{training transcript}] \leq 1 - \frac{\delta}{2}.$$

Thus

$$\Pr[\text{Good} \wedge \neg \text{Bad}] \geq \frac{\delta}{2} \Pr[\text{Good}].$$

The proof of [Lemma 5.7](#) applies to this augmented experiment. Indeed, the completeness adversary can perform the same independent training internally before sampling the tuple and target, so the accumulated completeness and candidate-selection errors remain negligible. Therefore

$$P(E) \geq \frac{1}{2} \Pr[\text{Good} \wedge \neg \text{Bad}] - \text{negl}(\lambda) \geq \frac{\delta}{4} - \text{negl}(\lambda). \quad (64)$$

The KL argument is unchanged by the training phase. Conditional on every training transcript, the argument leading to (36) gives

$$D_{\text{KL}}(P \parallel Q) \leq \Gamma_{m_L} = \frac{(L+1) \ln 2}{m_L} = O(\log \lambda),$$

and averaging over the common training transcript preserves the same bound. By [Lemma 5.8](#) and (64), the resampled experiment therefore succeeds with inverse-polynomial probability along the infinite subsequence on which containment fails. The full procedure—the training phase, bounded separation, tuple-packing computation, and resampled replay—runs in probabilistic polynomial time. Except for the negligible target-collision abort of [Lemma 5.5](#), event E is a valid soundness forgery. This contradicts soundness and proves (57).

Finally, membership in

$$(1 - \delta)\alpha_L \mathcal{G}_{w_L}^{(n)}$$

means that for some

$$z \in \text{conv}(\mathcal{T}_{w_L}([n]))$$

we have

$$\mu_\lambda \geq (1 - \delta)\alpha_L z$$

coordinatewise. Representing z as the expectation of a distribution $\Pi_{\lambda, \delta}$ over feasible tree vectors gives (58). $\square$

The containment in [Theorem 5.16](#) is a statement about the expected update vector μ_λ . It does not assert that each realized update vector d_G lies near the tree region.

6 Lower Bounds from the Equal-Weight Geometry

We now specialize the tree geometry to the uniform weight vector

$$a = \mathbf{1}.$$

The function Φ_w , introduced in [Corollary 5.11](#) as a universal size-only cohort benchmark, becomes the exact equal-weight optimum when the cohort is the entire population: [Corollary 6.2](#) will show that $V_w(\mathbf{1}) = \Phi_w(n)$. This section illustrates what becomes available once the full update geometry is known. Scalar bounded-depth tree resource tradeoffs have classical antecedents [\[2, 3, 9\]](#). Here we refine the scalar optimum to the full left-depth distribution. The master result is an exact formula for every left-depth tail of an extremal bounded-right-depth tree, attained simultaneously by one tree. Summing these tails yields

$$V_w(\mathbf{1}) = \min_{T: \max_i \rho_T(i) < w} \sum_i \ell_T(i),$$

while other readouts recover the previously studied maximum- and total-update bounds and give exact lower bounds for a broader family of symmetric update costs. Thus one geometric theorem can be reused across many objectives, rather than requiring a separate extremal argument for each one. All of them come from the same equal-weight geometry characterized in [Section 3](#).

6.1 Exact Left-Depth Tail Bounds

Consider the infinite binary prefix tree $\{L, R\}^*$. A word containing exactly x symbols L and y symbols R occurs with multiplicity

$$\binom{x+y}{x}.$$

Hence the number of binary words containing fewer than r left steps and fewer than w right steps is

$$\sum_{x=0}^{r-1} \sum_{y=0}^{w-1} \binom{x+y}{x} = \binom{w+r}{r} - 1. \quad (65)$$

This gives an exact lower bound on every left-depth tail.

Theorem 6.1 (Left-depth tail bound). *Let T be any ordered binary tree on n nodes satisfying*

$$\max_i \rho_T(i) < w.$$

Then for every $r \geq 1$,

$$|\{i : \ell_T(i) \geq r\}| \geq \left(n + 1 - \binom{w+r}{r} \right)_+. \quad (66)$$

Moreover, there exists an ordered binary tree attaining equality simultaneously for every $r \geq 1$.

Proof. Every node of T is identified by its root-to-node word in $\{L, R\}^*$. Since $\rho_T(i) < w$ for every node, [\(65\)](#) shows that at most

$$\binom{w+r}{r} - 1$$

nodes can have left-depth below r . Therefore

$$|\{i : \ell_T(i) \geq r\}| \geq \left(n + 1 - \binom{w+r}{r} \right)_+.$$

It remains to show simultaneous tightness. Let

$$q := \max \left\{ r \geq 0 : \binom{w+r}{r} \leq n \right\}.$$

Take every binary word containing fewer than q symbols L and fewer than w symbols R . By (65), this gives

$$\binom{w+q}{q} - 1$$

nodes. We must add

$$k := n - \left(\binom{w+q}{q} - 1 \right)$$

further nodes. The boundary consisting of words with exactly q symbols L and fewer than w symbols R has size

$$\sum_{y=0}^{w-1} \binom{q+y}{q} = \binom{w+q}{q+1}.$$

By maximality of q ,

$$n < \binom{w+q+1}{q+1},$$

and therefore

$$k \leq \binom{w+q}{q+1}.$$

Choose an initial segment of size k of a linear extension of the prefix order on this boundary. Together with the interior words, the resulting set is prefix-closed and hence defines a binary tree. Label its nodes by inorder order to obtain an ordered binary tree on $[n]$.

For every $r \leq q$, this tree contains every word having fewer than r left steps and fewer than w right steps, while none of the added boundary words has left-depth below r . Hence exactly

$$\binom{w+r}{r} - 1$$

nodes have left-depth below r , and equality holds in (66).

For $r > q$, maximality of q gives

$$\binom{w+r}{r} > n,$$

so the right-hand side of (66) vanishes. Every node of the constructed tree has left-depth at most q , so the left-hand side also vanishes. $\square$

Summing the exact tail sequence gives the corresponding minimum total left-depth value in our post-insertion convention.

Corollary 6.2 (Exact equal-weight value). *Let*

$$q = q(n, w) := \max \left\{ r \geq 0 : \binom{w+r}{r} \leq n \right\}.$$

Then

$$V_w(\mathbf{1}) = \sum_{r \geq 1} \left(n + 1 - \binom{w+r}{r} \right)_+ \tag{67}$$

$$= q(n+1) - \binom{w+q+1}{q} + 1. \tag{68}$$

Equivalently, with Φ_w defined in (49),

$$V_w(\mathbf{1}) = \Phi_w(n).$$

Proof. For every ordered tree T ,

$$\sum_i \ell_T(i) = \sum_{r \geq 1} |\{i : \ell_T(i) \geq r\}|.$$

The lower bound therefore follows from [Theorem 6.1](#), and the tree constructed there attains equality simultaneously for every r . The summand in (67) is positive exactly for $1 \leq r \leq q$. Hence

$$\Phi_w(n) = \sum_{r=1}^q \left(n + 1 - \binom{w+r}{r} \right).$$

Using the hockey-stick identity

$$\sum_{r=0}^q \binom{w+r}{r} = \binom{w+q+1}{q}$$

gives (68). □

6.2 Recovering Maximum and Total Update Bounds

The tail formula above recovers the two scalar update measures that have been studied most directly in the accumulator literature.

Maximum lifetime frequency. By [Theorem 6.1](#), every ordered tree satisfying $\max_i \rho_T(i) < w$ also satisfies

$$\max_i \ell_T(i) \geq q(n, w),$$

and the construction in its proof attains equality. Combining this with the update-vector characterization of [Section 3](#) gives

$$\min_{G: G \text{ blocks every } (w+1)\text{-tuple}} \max_i d_G(i) = \min_{T: \max_i \rho_T(i) < w} \max_i \ell_T(i) = q(n, w), \quad (69)$$

where

$$q(n, w) = \max \left\{ r : \binom{w+r}{r} \leq n \right\}.$$

This is the same combinatorial threshold underlying the maximum witness-update tradeoffs in [\[7, 10\]](#). Thus the maximum-frequency bound is one scalar readout of the same ordered-tree geometry.

Lifetime total. For equal weights,

$$\text{Cost}_1(G) = |E(G)| = \sum_i d_G(i).$$

On the integral blocking side, the update-vector characterization of [Section 3](#) together with [Corollary 6.2](#) gives

$$\tau_{w+1}(\mathbf{1}) = V_w(\mathbf{1}) = \Phi_w(n).$$

On the cryptographic side, applying [Theorem 5.10](#) gives

$$\mathbb{E}_{\text{Exec}_{\Lambda, n}} [|E(G)|] \geq (1 - o(1)) \frac{m_L}{m_L - 1} \Phi_{m_L-1}(n). \quad (70)$$

Moreover, for every fixed $\delta > 0$,

$$\Pr_{\text{Exec}_{\Lambda, n}} \left[|E(G)| \leq (1 - \delta) \frac{m_L}{m_L - 1} \Phi_{m_L-1}(n) \right] = o(1). \quad (71)$$

Indeed, $m_L \leq n$ implies $\Phi_{m_L-1}(n) > 0$, so the tail statement of [Theorem 5.10](#) applies.

Thus maximum lifetime frequency and lifetime total are two different scalar functionals of the same update-vector geometry: the former reads the largest left-depth, while the latter sums all left-depths.

Fixed right-depth parameter. For every fixed $w \geq 1$,

$$\Phi_w(n) \sim \frac{w}{w+1} (w!)^{1/w} n^{1+1/w}. \quad (72)$$

Indeed, if $q = q(n, w)$, then

$$\binom{w+q}{q} = \frac{q^w}{w!} + O(q^{w-1}),$$

so

$$q = (w!n)^{1/w} (1 + o(1)).$$

Furthermore,

$$\binom{w+q+1}{q} = \frac{q^{w+1}}{(w+1)!} (1 + o(1)),$$

and substitution into (68) gives (72). For example,

$$\Phi_1(n) = \binom{n}{2}.$$

Polylogarithmic right-depth parameter. For every fixed $K > 0$, if

$$w \leq (\log n)^K,$$

then

$$\Phi_w(n) = \Omega\left(\frac{n \log n}{\log \log n}\right). \quad (73)$$

To see this, choose

$$r = \left\lfloor c \frac{\log n}{\log \log n} \right\rfloor$$

for a sufficiently small constant $c = c(K) > 0$. Using

$$\binom{w+r}{r} \leq \left(\frac{e(w+r)}{r}\right)^r$$

and $w \leq (\log n)^K$, we have

$$\log \binom{w+r}{r} = O(r \log \log n).$$

For sufficiently small c ,

$$\binom{w+r}{r} \leq \frac{n}{2}$$

for all sufficiently large n . The same bound holds for every $1 \leq s \leq r$, and therefore (67) gives

$$\Phi_w(n) \geq \sum_{s=1}^r \left(n+1 - \binom{w+s}{s} \right) \geq \frac{nr}{2} = \Omega\left(\frac{n \log n}{\log \log n}\right).$$

Consequently, whenever

$$m_L \leq n \quad \text{and} \quad m_L - 1 \leq (\log n)^K$$

for some fixed K , (70) yields

$$\mathbb{E}_{\text{Exec}_{\Lambda, n}} [|E(G)|] = \Omega\left(\frac{n \log n}{\log \log n}\right). \quad (74)$$

BCCK obtain the same total-update scale in a succinct regime by a different argument, together with strong concentration guarantees [4]. Here the same scale follows from the explicit equal-weight tree value and its transfer through Theorem 5.10.

6.3 Further Symmetric Lower Bounds

The previous subsection recovers two familiar scalar lower bounds from the equal-weight tree geometry. The underlying combinatorial statement is stronger. The extremal tree of [Theorem 6.1](#) simultaneously minimizes every left-depth tail, so many other cost functions can be read off from the same object without solving a new extremal problem.

Let $T^\star$ be an ordered binary tree attaining equality in (66) for every $r \geq 1$, and write

$$\ell_T^\downarrow(1) \geq \dots \geq \ell_T^\downarrow(n)$$

for the left-depths of T arranged in nonincreasing order.

Proposition 6.3 (Universal optimizer for symmetric monotone costs). *For every ordered binary tree T on n nodes satisfying*

$$\max_i \rho_T(i) < w,$$

we have

$$\ell_T^\downarrow(j) \geq \ell_{T^\star}^\downarrow(j) \quad \text{for every } j \in [n]. \quad (75)$$

Consequently, for every symmetric coordinatewise nondecreasing function

$$F : \mathbb{N}^n \rightarrow \mathbb{R},$$

$$\min_{T: \max_i \rho_T(i) < w} F((\ell_T(i))_{i=1}^n) = F((\ell_{T^\star}(i))_{i=1}^n). \quad (76)$$

Proof. Fix $j \in [n]$ and let

$$r := \ell_{T^\star}^\downarrow(j).$$

If $r = 0$, then (75) is immediate because left-depths are nonnegative. Suppose therefore that $r \geq 1$.

At least j nodes of $T^\star$ have left-depth at least r . Since $T^\star$ attains equality in (66),

$$|\{i : \ell_{T^\star}(i) \geq r\}| = \left(n + 1 - \binom{w+r}{r} \right)_+.$$

By [Theorem 6.1](#), every feasible tree T has at least as many nodes of left-depth at least r . Hence

$$\ell_T^\downarrow(j) \geq r = \ell_{T^\star}^\downarrow(j),$$

proving (75). Since F is symmetric, it depends only on the sorted coordinates. Since it is coordinatewise nondecreasing, (75) implies

$$F((\ell_T(i))_{i=1}^n) \geq F((\ell_{T^\star}(i))_{i=1}^n).$$

□

Combining the proposition with the update-vector characterization of [Section 3](#) gives, for every symmetric coordinatewise nondecreasing F ,

$$\min_{G: G \text{ blocks every } (w+1)\text{-tuple}} F(d_G) = F((\ell_{T^\star}(i))_{i=1}^n). \quad (77)$$

Thus a single extremal tree simultaneously determines the exact lower bound for every such cost function in the integral blocking problem. The following examples illustrate different ways of reading the same extremal geometry.

Threshold costs. For $r \geq 1$, define

$$H_r(d) := |\{i : d_i \geq r\}|.$$

This quantity measures how many coordinates incur at least r updates. By [Theorem 6.1](#),

$$\min_{T: \max_i \rho_T(i) < w} H_r((\ell_T(i))_{i=1}^n) = \left(n + 1 - \binom{w+r}{r} \right)_+. \quad (78)$$

Thus [Theorem 6.1](#) determines the entire sequence of update-depth thresholds, rather than only its maximum or its sum.

Top- k costs. For $d \in \mathbb{N}^n$, let

$$d_1^\downarrow \geq \dots \geq d_n^\downarrow$$

be its coordinates in nonincreasing order, and define

$$K_k(d) := \sum_{j=1}^k d_j^\downarrow.$$

This family interpolates between the two classical scalar objectives:

$$K_1(d) = \|d\|_\infty, \quad K_n(d) = \|d\|_1.$$

For every nonnegative integer vector d ,

$$K_k(d) = \sum_{r \geq 1} \min \{k, |\{i : d_i \geq r\}|\}.$$

Therefore

$$\min_{T: \max_i \rho_T(i) < w} K_k((\ell_T(i))_{i=1}^n) = \sum_{r \geq 1} \min \left\{ k, \left(n + 1 - \binom{w+r}{r} \right)_+ \right\}. \quad (79)$$

Hence maximum lifetime frequency and lifetime total are the two endpoints of an exact family of intermediate objectives determined by the same tail sequence.

Separable costs. More generally, let

$$f : \mathbb{N} \rightarrow \mathbb{R}$$

be nondecreasing, and write

$$\Delta f(r) := f(r) - f(r-1) \geq 0.$$

For every $d \in \mathbb{N}^n$,

$$\sum_{i=1}^n f(d_i) = n f(0) + \sum_{r \geq 1} \Delta f(r) |\{i : d_i \geq r\}|.$$

It follows that

$$\min_{T: \max_i \rho_T(i) < w} \sum_{i=1}^n f(\ell_T(i)) = n f(0) + \sum_{r \geq 1} \Delta f(r) \left(n + 1 - \binom{w+r}{r} \right)_+. \quad (80)$$

Thus every nondecreasing separable cost is obtained by applying a different set of nonnegative weights to the same left-depth tails.

For example, taking

$$f(x) = x^p, \quad p \geq 1,$$

gives

$$\min_{T: \max_i \rho_T(i) < w} \sum_{i=1}^n \ell_T(i)^p = \sum_{r \geq 1} (r^p - (r-1)^p) \left(n+1 - \binom{w+r}{r} \right)_+ . \quad (81)$$

Equivalently,

$$\min_{T: \max_i \rho_T(i) < w} \|(\ell_T(i))_{i=1}^n\|_p = \left[\sum_{r \geq 1} (r^p - (r-1)^p) \left(n+1 - \binom{w+r}{r} \right)_+ \right]^{1/p} .$$

These examples are different readouts of the same extremal update-vector geometry. The ordered-tree characterization therefore provides a common source of exact lower bounds for a broad class of update-cost measures. Sections 4 and 5 transfer source-linear objectives to arbitrary cryptographic executions. We next turn to the positive side and show that the same tree value has an exact operational interpretation through online mergers.

7 Exact Online-Merger Realizations

The same tree frontier also describes the constructive side of the tradeoff. We now show that this labeled geometry is operationally realizable. At every fixed merger width w , the minimum weighted post-insertion merge cost is exactly $V_w(a)$, and every feasible tree left-depth vector is realized as the per-item merge vector of a corresponding schedule. Through the standard Merkle-forest transformation, this gives positive accumulators realizing the same vector frontier.

Tree-based formulations of merge schedules have classical antecedents in data-structure dynamization [3] and are especially direct in Bigtable merge compaction [9]; we use the online-merger model of [7]. The point here is that the temporal blocking problem of Section 3 independently reaches exactly the same frontier, so the lower-bound and constructive sides meet at one common width-level geometry.

7.1 From Ordered Trees to Online Mergers

We use the online-merger model of [7]. At round t , a new singleton component $\{t\}$ is added to the current collection of live components, after which the current partition may be coarsened by merging live components. Components only merge and never split. The width of a merger is the maximum number of live components at the end of a round. For an item i , let $M(i)$ denote the number of rounds $t > i$ in which the live component containing i participates in such a merge.

Let T be an ordered binary tree on $[n]$. Process its nodes in inorder. At round t , let v_t be the node labeled t , and let $I_L(v_t)$ be the set of labels in its left subtree. After the singleton component $\{t\}$ arrives, merge all currently live components whose leaves lie in $I_L(v_t)$ together with $\{t\}$ into one component. If the left subtree is empty, no merge is performed.

Proposition 7.1 (Tree-to-merger realization). *For every ordered binary tree T :*

1. *the number of live components at the end of every round is at most*

$$1 + \max_i \rho_T(i);$$

2. *for every item i , the component containing i participates in exactly*

$$\ell_T(i)$$

post-insertion merge rounds.

Consequently, if

$$\max_i \rho_T(i) < w,$$

the merger has width at most w and weighted post-insertion merge cost

$$\sum_{i=1}^n a_i \ell_T(i).$$

In particular, an optimal tree for $V_w(a)$ gives a width- w online merger of weighted cost exactly

$$V_w(a).$$

Proof. Let $W(T)$ be the maximum end-of-round number of live components while processing the inorder interval of T , starting with no external live components. If T_L and T_R are the left and right subtrees of the root, then at the root round all components produced by T_L , together with the new root singleton, are merged into one component. That component remains live while T_R is processed. Hence

$$W(T) \leq \max\{W(T_L), 1 + W(T_R)\}.$$

With $W(\emptyset) = 0$, induction gives

$$W(T) \leq 1 + \max_i \rho_T(i).$$

Now fix an item i . After its insertion round, its component is merged at the round corresponding to an ancestor v exactly when i lies in the left subtree of v . These ancestors are in bijection with the left edges on the root-to- i path. Thus the component containing i participates in exactly

$$\ell_T(i)$$

post-insertion merge rounds. □

There is also a converse connection from bounded-width mergers back to the blocking problem.

Lemma 7.2 (Width implies blocking). *Let M be an online merger on $[n]$ of width at most w . Define a forward graph G_M by placing an edge*

$$(i, t), \quad i < t,$$

whenever the live component containing item i participates in a merge at round t . Then

$$d_{G_M}(i) = M(i)$$

for every i , where $M(i)$ is the number of post-insertion merge rounds involving item i , and G_M blocks every increasing $(w+1)$ -tuple.

Proof. The degree identity follows directly from the definition of G_M .

Suppose that

$$S = (s_1 < \dots < s_{w+1})$$

is not blocked by G_M . We claim that the items

$$s_1, \dots, s_{w+1}$$

must lie in distinct live components at the end of round s_{w+1} .

Otherwise, suppose s_j and s_k , with $j < k$, belong to the same component by that time. Since components never split, at some round

$$s_k \leq t \leq s_{w+1}$$

the component containing s_j must participate in a merge that joins it with the component containing s_k . Hence

$$(s_j, t) \in E(G_M).$$

Since

$$s_{j+1} \leq s_k \leq t \leq s_{w+1},$$

this edge blocks S at position j , a contradiction.

Thus at least $w + 1$ live components are present at the end of round s_{w+1} , contradicting the width bound. $\square$

Putting the two directions together with [Corollary 3.6](#) gives the following exact correspondence.

Corollary 7.3 (Weighted online-merger optimum). *For every $w \geq 1$ and every $a \in \mathbb{R}_{\geq 0}^n$,*

$$V_w(a) = \min_{G: G \text{ blocks every } (w+1)\text{-tuple}} \text{Cost}_a(G) = \min_{\substack{M: \text{online merger} \\ \text{width}(M) \leq w}} \sum_{i=1}^n a_i M(i).$$

Proof. The first equality is [Corollary 3.6](#). By [Proposition 7.1](#), an optimal tree for $V_w(a)$ induces a width- w merger of weighted cost $V_w(a)$, so

$$\min_{\substack{M: \text{online merger} \\ \text{width}(M) \leq w}} \sum_i a_i M(i) \leq V_w(a).$$

Conversely, let M be any width- w merger. By [Lemma 7.2](#), its induced graph G_M blocks every increasing $(w + 1)$ -tuple and satisfies

$$d_{G_M}(i) = M(i)$$

for every i . Therefore

$$\sum_i a_i M(i) = \text{Cost}_a(G_M) \geq V_w(a).$$

Taking the minimum over M gives the reverse inequality. $\square$

Thus the same ordered-tree geometry governs both sides: right-depth controls merger width, while left-depth records the full vector of per-item post-insertion merge counts.

7.2 Merkle-Forest Realization

The online merger can in turn be implemented as a Merkle-forest accumulator using the transformation of [\[7\]](#). Maintain one Merkle tree for each live component and store its root in the public accumulator state. Whenever several components are merged, replace them by a Merkle tree for their union. A membership witness changes exactly when the component containing its element participates in such a merge.

Corollary 7.4 (Weighted Merkle realization). *Assume collision-resistant hashing with σ -bit roots. Let $n = n(\lambda)$ be polynomially bounded, let*

$$a = a(\lambda) \in \mathbb{Q}_{\geq 0}^n$$

be efficiently computable with polynomial bit complexity, and let $w = w(\lambda) \geq 1$ be efficiently computable. Then there is an additive positive accumulator whose update vector on every length- n execution is the left-depth vector of an optimal tree for $V_w(a)$. In particular,

$$\text{Cost}_a(G) = V_w(a)$$

on every such execution.

The construction maintains at most w live Merkle roots, has public state

$$O(\sigma w + \log n)$$

bits, has perfect **NeedUpd-completeness**, and is sound under collision resistance.

Proof. Replacing w by $\min\{w, n\}$ if necessary, we may assume $w \leq n$. Compute an optimal tree for $V_w(a)$ using the dynamic program of [Proposition 3.7](#), and realize it by [Proposition 7.1](#). Maintain one Merkle root for each live component.

Set $\text{NeedUpd}(i, t) = 1$ exactly when the component containing item i participates in a post-insertion merge at round t . By [Proposition 7.1](#),

$$d_G(i) = \ell_T(i)$$

for every i , and hence

$$\text{Cost}_a(G) = \sum_i a_i d_G(i) = \sum_i a_i \ell_T(i) = V_w(a).$$

Because the merger schedule is fixed by the precomputed tree, the current round determines the live component intervals; the public state therefore consists of at most w Merkle roots and a round counter.

A retained witness remains valid whenever its component is unchanged, so the resulting update specification has perfect **NeedUpd-completeness**. Soundness follows from the standard collision-resistance argument for the Merkle-forest construction. $\square$

Exactness at the width level and the remaining bit-level gap. At the combinatorial and online-merger width level, the lower-bound and constructive sides coincide exactly through the same value $V_w(a)$. Translating width into cryptographic public-state bits introduces the representation cost of live cryptographic roots. With σ -bit Merkle roots and public-state budget L , the construction supports width on the order of

$$\frac{L}{\sigma},$$

whereas the lower bound of [Section 5](#) uses an effective right-depth parameter on the order of

$$\frac{L}{\log \lambda}.$$

Thus the geometric width–update tradeoff is exact; the remaining gap in public-state bits is the gap between these two width representations.

References

- [1] Abusalah, H., Anthoine, G., Avitabile, G., Giunta, E.: Lower bounding update frequency in short accumulators and vector commitments. In: Advances in Cryptology – EUROCRYPT 2026. Lecture Notes in Computer Science, vol. 16545, pp. 184–213. Springer (2026)
- [2] Bentley, J.L., Brown, D.J.: A general class of resource tradeoffs. *Journal of Computer and System Sciences* **25**(2), 214–238 (1982). [https://doi.org/10.1016/0022-0000\(82\)90005-8](https://doi.org/10.1016/0022-0000(82)90005-8)
- [3] Bentley, J.L., Saxe, J.B.: Decomposable searching problems i: Static-to-dynamic transformation. *Journal of Algorithms* **1**(4), 301–358 (1980). [https://doi.org/10.1016/0196-6774\(80\)90015-2](https://doi.org/10.1016/0196-6774(80)90015-2)

- [4] Bonneau, J., Chen, J., Christ, M., Karantaidou, I.: Merkle mountain ranges are optimal: On the witness update frequency of cryptographic accumulators. In: *Advances in Cryptology – CRYPTO 2025, Part II*. pp. 170–202. Springer (2025). https://doi.org/10.1007/978-3-032-01878-6_6
- [5] Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: *Advances in Cryptology – CRYPTO 2002*. pp. 61–76. Springer (2002)
- [6] Grötschel, M., Lovász, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization*. Springer, second corrected edn. (1993)
- [7] Mahmoodi, M., Qi, W.: Online mergers and applications to registration-based encryption and accumulators. In: *4th Conference on Information-Theoretic Cryptography (ITC 2023)*. *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 267 (2023), article 15
- [8] Mahmoodi, M., Qi, W., Rahimi, A.: Lower bounds for the number of decryption updates in registration-based encryption. In: *Theory of Cryptography Conference (TCC 2022)*, Part I. *Lecture Notes in Computer Science*, vol. 13747, pp. 559–587. Springer (2022)
- [9] Mathieu, C., Staelin, C., Young, N.E., Yousefi, A.: Bigtable merge compaction. *arXiv:1407.3008* (2015)
- [10] Qi, W.: Tight lower bound on witness update frequency in additive positive accumulators. *IACR Communications in Cryptology* **2**(4) (2026). <https://doi.org/10.62056/av7t7ta5v>
- [11] Soóki-Tóth, B., Glasz, B., Kavousi, A., Seres, I.A.: Private delegation of (non-)membership proof updates in cryptographic accumulators. *Cryptology ePrint Archive, Report 2026/832* (2026)