Online Mergers and Applications to Registration-Based Encryption and Accumulators

Mohammad Mahmoody
University of Virginia, Charlottesville, VA, USA

Wei Qi
University of Virginia, Charlottesville, VA, USA

Abstract

In this work we study a new information theoretic problem, called *online merging*, that has direct applications for constructing public-state accumulators and registration-based encryption schemes. An online merger receives the sequence of sets $\{1\}, \{2\}, \ldots$ in an online way, and right after receiving $\{i\}$, it can re-partition the elements $1, \ldots, i$ into $T_1, \ldots, T_{m_i}$ by merging some of these sets. The goal of the merger is to balance the trade-off between the maximum number of sets $\mathsf{wid} = \max_{i \in [n]} m_i$ that co-exist at any moment, called the *width* of the scheme, with its *depth* $\mathsf{dep} = \max_{i \in [n]} d_i$, where d_i is the number of times that the sets that contain i get merged. An online merger can be used to maintain a set of Merkle trees that occasionally get merged.

An online merger can be directly used to obtain public-state accumulators (using collision-resistant hashing) and registration-based encryptions (relying on more assumptions). Doing so, the width of an online merger translates into the size of the public-parameter of the constructed scheme, and the depth of the online algorithm corresponds to the number of times that parties need to update their "witness" (for accumulators) or their decryption key (for RBE).

In this work, we construct online mergers with $\mathsf{poly}(\log n)$ width and $O(\log n / \log\log n)$ depth, which can be shown to be optimal for all schemes with $\mathsf{poly}(\log n)$ width. More generally, we show how to achieve optimal depth for a given fixed width and to achieve a 2-approximate optimal width for a given depth d that can possibly grow as a function of n (e.g., $d = 2$ or $d = \log n / \log\log n$). As applications, we obtain accumulators with $O(\log n / \log\log n)$ number of updates for parties' witnesses (which can be shown to be optimal for accumulator digests of length $\mathsf{poly}(\log n)$) as well as registration based encryptions that again have an optimal $O(\log n / \log\log n)$ number of decryption updates, resolving the open question of Mahmoody, Rahimi, Qi [TCC'22] who proved that $\Omega(\log n / \log\log n)$ number of decryption updates are necessary for any RBE (with public parameter of length $\mathsf{poly}(\log n)$). More generally, for any given number of decryption updates $d = d(n)$ (under believable computational assumptions) our online merger implies RBE schemes with public parameters of length that is optimal, up to a constant factor that depends on the security parameter. For example, for any constant number of updates d, we get RBE schemes with public parameters of length $O(n^{1/(d+1)})$.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Registration-based encryption, Accumulators, Merkle Trees

Digital Object Identifier 10.4230/LIPIcs.ITC.2023.15

Funding *Mohammad Mahmoody*: Supported by NSF grants CCF-1910681 and CNS193679.
Wei Qi: Supported by NSF grants CCF-1910681 and CNS193679.

1 Introduction

Registration-based encryption [12] is a primitive that aims to offer what identity-based encryption [26, 5] offers (i.e., a compact public parameter that can be used to encrypt for all identities) but without the key-escrow problem (i.e., that the holder of the master secret key can decrypt all the messages). It was shown [12] that essentially two relaxations will

enable such a primitive. In particular, in RBE the parties generate their own public *and secret* keys, and then they register them to a transparent algorithm called the key curator (KC). However, this comes at the cost of an evolving (compact) public parameter and need for occasional decryption updates. Namely, firstly the public parameter pp_n is now possibly changing after the nth identity registers into the system, and secondly the parties might sometimes need to reach out to the KC for hints/updates so that they can complete their decryption tasks. It was shown in [12] that with a total of $O(\log n)$ number of decryption updates, one can keep the length of the public parameter $\mathsf{poly}(\log n)$ (with a constant that depends on the security parameter). The follow-up works on RBE [13, 16, 9] made progress in various aspects such as assumptions and concrete efficiency, but asymptotically they all required the same $\Theta(\log n)$ number of decryption updates.

How many decryption updates are needed? The above state of affairs left open the possibility that RBE schemes with *sub-logarithmic* $o(\log n)$ number of updates could be constructed. Recently, Qi, Rahimi, and Mahmoody [22] proved that $\Omega(\log n/\log\log n)$ many updates are necessary for any RBE schemes (with a public parameter of size $\mathsf{poly}(\log n)$, as required by the standard definition of RBE) regardless of the computational assumptions used for constructing them, so long as the updates arrived at fixed times. The latter property is known to hold for all constructions of RBE so far. It remained open to close the gap between the upper bound of $O(\log n)$ and the lower bound of $\Omega(\log n/\log\log n)$.

In this work, we further close this gap and show that the lower bound of $\Omega(\log n/\log\log n)$ on the number of decryption updates is optimal (up to a constant factor that depends on the security parameter). We do so by improving the core information theoretic object that is at the center of the original RBE scheme of [12] as well as the accumulators used in such RBE schemes. More specifically, the RBE scheme of [12] relies on a *transparent accumulator* (i.e., one that does not have a secret state) [4, 3, 6, 25] that accumulates all the public-keys tied with their corresponding identities. In such an accumulator, a short digest pp of all the accumulated strings $\{x_1, \ldots, x_n\}$ can be used to efficiently verify membership of, say, x_i in the collection, so long as this verifier is provided with a short witness w_i of the membership. Hence, there are clear similarities between what RBE does with the keys and what an accumulator does with the strings x_i, and so it is not surprising that accumulators are useful for building RBEs. Our main result is to identify a core problem that is also at the heart of transparent accumulators that in turn are used for building RBEs.

The most natural approach for building a transparent accumulator is to use Merkle trees, based on collision resistant hash functions. Namely, one can build a Merkle tree T over the leaves $\mathcal{S} = \{x_1, \ldots, x_n\}$, and publish the label of the root r as the public parameter. Then, to prove membership of x_i in the collection $\mathcal{S}$, one can provide the "Merkle opening" of x_i, which consists of the labels along the path from (the leaf) x_i to the root r as well as the labels of the neighbors of this path. Then the verifier will do the basic sanity checks to pass the verification, and the scheme will be sound so long as the compressing hash function used for building the tree is collision resistant. When we are in the *dynamic* setting and the elements $\{x_1, \ldots, x_n\}$ arrive one by one, we no longer can use a single Merkle tree for hashing them, at least as long as we do not want to change the "opening witnesses" frequently.

The work of Reyzin and Yakoubov [25] showed how to make use of a *collection* of Merkle trees in such a way that the opening of each x_i will needs to be updated only $O(\log n)$ times over the course of the n steps of the system. They called an accumulator with this feature an *asynchronous accumulator*, and their construction was also used in the RBE construction of [12]. The idea behind this accumulator is to keep the collection of trees $\mathcal{T}$ in a way that: (1) any tree $T \in \mathcal{T}$ is always a full binary tree with 2^i leaves, (2) every pair of different trees

$T_1, T_2 \in \mathcal{T}$ have *different* sizes, (3) there is a bijection between the leaves of the trees in $\mathcal{T}$ and the accumulated set of objects. Therefore, the sizes of the trees in $\mathcal{T}$ directly give the *binary representation* of n, where n is the total number of elements in the collection. When a new element x_n arrives, this accumulator first generates a set $\{x_n\}$, and then the accumulator keeps merging the trees of the same size till there is no such pair of trees. It can be seen that the maximum depth of any tree $T \in \mathcal{T}$ will upper bound the number of times that an opening witness of an element $x \in \{x_1, \ldots, x_n\}$ needs to be updated, and this number is $\Theta(\log n)$. This is the core reason that this accumulator and the RBE scheme of [12] require logarithmically many updates.

1.1 Online mergers

In this work, we revisit the way a collection of sets/trees are merged to maintain a collection of "Merkle trees", but we study the problem more abstractly and independently of the direct connection to Merkle trees. We ask how to do the merging, while we try to balance the number of trees versus their maximum depth. More formally, we ask the following question. Suppose the elements $\{x_1, \ldots, x_n\}$ arrive one by one in n rounds, and suppose at the beginning of each round $i+1$ we have a collection of sets $T_1^i, \ldots, T_{m_i}^i$ already that partitions $\{x_1, \ldots, x_i\}$ and the new single-element set $T_{m_i+1}^i = \{x_{i+1}\}$ gets added to the current collection of sets. Then, the job of an *online merger* is to choose how to merge some of the sets and shape the updated collection $T_1^{i+1}, \ldots, T_{m_{i+1}}^{i+1}$ that is a partitioning of $T_1^i, \ldots, T_{m_i}^i, \{x_{i+1}\}$ (and also of $\{x_1, \ldots, x_{i+1}\}$). Since the merger has to decide about these choices in *each* of the n rounds, we call it an *online* merger.[1] The two key parameters of interest for online mergers are the following.

- The **width** of an online merger is the maximum number of sets (i.e., maximum of m_i over all $i \in [n]$) that it ever maintains during the course of its execution in the n rounds. This parameter is important as it captures the size of the digest/public parameter if each of the sets T is a Merkle tree, because it simply counts the number of roots of the trees at its maximum peak.
- The **depth** of an online merger is the maximum of d_i for $i \in [n]$, where d_i is the number of times that the set containing x_i gets merged with other sets. If we choose to represent the sets as trees and merge them as such, the depth of a scheme is simply the maximum depth of the collected trees at the end. If x_i has depth d_i, then (in case the sets shape Merkle trees) the opening witness for the membership of x_i in the collection needs to be updated d_i times.

Key questions about online mergers. On the extreme points, one can achieve width 1 and depth $n-1$ by immediately merging any incoming set, and one can achieve depth 0 and width n by not merging any of the sets. Thus, the interesting question is to find the optimal *trade-off* between the width and depth of online mergers as a function of n. We can ask this question both for online mergers that know the set size n ahead of the time (called *bounded* online mergers) and for those that are "unbounded" and receive the incoming single-element sets without knowing the upper bound n on the final set size (called *unbounded* online mergers). More specifically, we are interested in finding the minimum width needed for mergers that are given an upper bound on their depth, and conversely we would like to find out the minimum depth needed for mergers that are given an upper bound on their

[1] In contrast, an offline merger gets all of $\{x_1, \ldots, x_n\}$ before deciding on how to partition them.

width. The given upper bounds (for width or depth) could be absolute constants or growing functions of n. On the one hand, both width and depth can be $\Theta(\log n)$ simultaneously, due to the accumulator of [25]. On the other hand, as we will show, the tools from [22] can be used to show that the depth of any online merger needs to be at least $\Omega(\log n/\log\log n)$ if the width needs to be $\mathsf{poly}(\log n)$ (which is the standard size of the public parameter for RBE [12]). Prior to our work, it was not known how many updates are necessary for achieving public parameters of length $\mathsf{poly}(\log n)$.

1.2 Our results

Our main result is to find the exact trade-off between the two key parameters (width and depth) of online mergers. As a corollary, we obtain transparent accumulators and RBE schemes that can take as input parameters the number of updates and produce public parameters that are optimal within a constant factor. As a special case, we also obtain accumulators and RBE schemes that have the optimal $O(\log n/\log\log n)$ number of witness/decryption updates, while their public parameter is assumed to be of length $O(\mathsf{poly}(\log n))$. Below, we explain these results in more detail.

Our results about online mergers. To have a reference to judge the optimality of a trade off between depth and width of an online merger, we start by proving a lower bound on this trade off. In particular, using a key combinatorial tool from [22], we first derive a lower bound on the trade-off between the depth d and width w of any online merger, and the lower bound holds even if the set of elements $[n] = \{1,\ldots,n\}$[2] is known to the online merger ahead of the time. In particular, we find lower bound functions $\mathsf{widthLB}(n,d)$ (for the with) and $\mathsf{depthLB}(n,w)$ (for the depth) when we are given the set size n and either of the width w or the depth d as inputs.

We remark that the mere fact that one can use the tools of [22] to obtain a lower bound for the trade off between depth and width of online mergers is not surprising, as online mergers are *one way* to obtain accumulators and RBEs and the lower bound of [22] applies to any RBE scheme. However, we emphasize that as our starting point in this work we obtain concrete lower bound functions $\mathsf{widthLB}(n,d)$ and $\mathsf{depthLB}(n,w)$ that do *not* hide any unknown constants that can depend, say, on the security parameter, as online mergers are a purely information theoretic object without security parameters. Hence, these lower bound functions allow us to prove *exact* bounds on the trade-off between depth and width of online mergers, which is what we do next. Having the reference lower bounds $\mathsf{widthLB}(n,d)$ and $\mathsf{depthLB}(n,w)$ we show how to achieve positive constructions that (sometimes approximately) match these lower bounds, as stated below.

▶ **Theorem 1** (Optimal bounded online mergers – informally stated)**.** *For any known set size n, there is an efficient construction of online merger MerWid_w of width w that achieves optimal depth $\mathsf{depthLB}(n,w)$, and there is an efficient construction of an online merger MerDep_d of depth d that achieves optimal width $\mathsf{widthLB}(n,d)$.*

See Theorem 13 and Proposition 14 for formalization of the above theorem.

We remark that the appendix in [22] included a graph construction that showed their lower-bound cannot be further improved. However, that graph construction only shows the limitation of the proof *approach* of [22] and does not uniquely determine a positive

[2] Note that even though we will use larger blocks of data in applications of online mergers (i.e., accumulators and RBE schemes) for simplicity we can pretend that the arriving sets are $\{1\},\ldots,\{n\}$.

construction. In fact, any positive construction (e.g., that of our Theorem 1) can be used to obtain such graphs, showing that the approach of [22] cannot lead to better lower bounds, but the reverse is not true.

▶ **Theorem 2** (Approximately optimal unbounded online mergers – informally stated)**.** *If we do* not *know n ahead of the time, and if $d(n)$ is a non-decreasing function of n (e.g., $d(n) = \log n / \log\log n$) that upper bounds the depth of the online merger that we would like to have for a set of n elements, then there is an unbounded online merger that achieves width at most $2 \cdot \mathsf{widthLB}(n, d(n))$.*

See Proposition 19 for a formal statement.

The cost of being unbounded. Theorem 2 only achieves a solution whose width is within 2 multiplicative approximation of the optimal solution. Hence, it brings up the question of whether the approximation factor 2 (or any other factor bigger than 1) is needed here. We prove that this is indeed the case, so long as we aim for *unbounded* online mergers. Namely, in Theorem 22 we show that for unbounded online mergers such overhead is necessary.

Implications to accumulators and RBEs. Theorems 1 and 2 can be directly used to construct accumulators and RBEs whose number of witness/decryption updates are bounded by the *depth* of the corresponding online merger, and whose public parameters are of the size $O_\kappa(w)$, where w is the width of the online merger and the constant in $O_\kappa(w)$ could depend on the security parameter κ of the accumulator/RBE scheme. In fact, Theorem 1 already suffices for obtaining accumulators and RBEs with *optimal* number of updates when we already know the final size of the set of elements/identities that will join the system over time. We then study *unbounded* online mergers who do *not* know the upper bound n on the population size, and obtain an almost tight solution.

The idea is quite straightforward: an online merger can be used to maintain a set of Merkle trees $\mathcal{T} = \{T_1, \ldots, T_m\}$ that would serve as an accumulator for the incoming objects $\{x_1, \ldots, x_n\}$, while the set of the *roots* of the trees $r_1, \ldots, r_m$ would serve as the digest/public parameter. To prove membership of x_i in the set, one has to prove the Merkle opening of x_i with respect to the tree $T \in \mathcal{T}$ that contains x_i as a leaf. Then, so long as the hash function used for constructing the Merkle trees in $\mathcal{T}$ is collision resistant, it would be computationally hard to prove membership of any $x \notin \{x_1, \ldots, x_n\}$ successfully. See Construction 44 for a formal description of this reduction. As formally stated in Proposition 45 the width and depth of the used online merger directly translate (in order) into the number of updates and length of the public parameter of the constructed accumulator. Finally, we observe that this construction of (transparent) accumulators is *tight* in its trade-off between the number of updates vs. the length of public parameter (up to a constant factor that can depend on the security parameter). The reason is that the same proof of the lower bound of [22] for RBEs can be directly adapted to transparent accumulators as well.

Finally, due to the fact that RBE schemes heavily rely on an internal accumulators for compressing the submitted public keys, using our optimal accumulator and extra assumptions (i.e., indistinguishability obfuscation [2, 11, 21] and somewhere-statistically binding hashing [20]) we can adapt the original construction of [12] to obtain an RBE scheme with an optimal $\log n / \log\log n$ number of updates, while all the other efficiency and compactness requirements of the scheme are as defined and required by [12].

▶ **Theorem 3** (RBE with optimal number of decryption updates – informal)**.** *Assuming standard computational hardness assumptions, there is an RBE scheme that has* $\log n / \log \log n$ *number of decryption updates.*

See Construction 28 for a formal adaptation of the construction of [12] to using an arbitrary online merger. The proof of security for this construction is identical to the (rather long) proof of the similar scheme in [12]. In fact, we obtain a more general result: using *any* given number of updates $d(n)$ as input parameter that can depend on n (e.g., $d(n) = \log(n)^{1/2}$) we can use our optimal online merger for that depth function $d(n)$ to obtain RBE schemes with *optimal* length for the public parameter for the given number of updates $d(n)$. An interesting corollary is that even using *just one update* allows us to obtain a scheme with a *sublinear* public parameter of length $O(\sqrt{n})$ and using a larger constant d will lead to significantly smaller public parameters of length $O(n^{1/(d+1)})$.

Other assumptions. We emphasize that our Theorem 3 above is merely to show that one can obtain the right number of updates for RBEs using *some* computational assumptions, and to show that we choose the simplest construction that is based on Indistinguishability obfuscation and Somewhere Statistically Binding Hashing [20]. However, as we explain in the full version of the paper, our main tool of this paper, namely online mergers, are versatile enough to be incorporated into other constructions of RBE based on standard assumptions [14, 16] as well.

1.3 Techniques

We now review some of the ideas used in the proof of our main results about online mergers.

Lower bound. To obtain the lower bound functions $\mathsf{widthLB}(n, d)$ (for the width) and $\mathsf{depthLB}(n, w)$ (for the depth) we take the following steps.

1. We first show how to derive a DAG of out-degree at most d from any merger of depth d, by connecting i to j if the set containing i gets merged in round j (see Definition 8).
2. We then use a result from [22] showing that DAGs with small out-degrees have a substructure, called *skipping sequence* (see Definition 33 and Theorem 34).
3. Finally, we observe that skipping sequences directly imply lower bounds on the width.

Optimal depth for a given constant width. Our starting point is an extremely simple scheme that obtains *optimal depth* (i.e., matching $\mathsf{depthLB}(n, w)$) when we are given any *fixed* width w as input. This scheme works *even if we do not know the set size* n in advance. The scheme can be described in one sentence: when a new set $T = \{n\}$ (as a single-node tree) is added to the collection of trees $\mathcal{T}$, if $\mathcal{T}$ has $w + 1$ trees, merge the newly arrived tree T with all the trees in $\mathcal{T}$ that have *minimum* depth (among the trees already in $\mathcal{T}$). Equivalently, as long as $|\mathcal{T}| = w + 1$, keep merging all the trees of minimum depth in $\mathcal{T}$. See Construction 11 and Theorem 13 for more details. One can interpret this scheme as "lazy merging" approach that does not do any merges when it does not have to, but when the merge is needed it only merges trees of minimal depth (with the incoming tree of depth 0). Interestingly, this simple scheme achieves a depth that is optimal *for every* n, as careful calculations can be used to show that its depth matches the lower bound $\mathsf{depthLB}(n, w)$ exactly.

Optimal width for a given depth and known set sizes. Once we have an online merger MerWid_w that achieves optimal depth for any given set size n and width w, we can turn this scheme around and switch the role of depth and width. Namely, for a given set size n and depth d, the new online merger MerDep_d can find the smallest w such that a construction on set $[n]$ with width w will have a depth at most d, and this would be an optimal choice of the width due to the optimality of the original algorithm MerWid_w. (See Proposition 14 for a formalization.) This finishes the proof of Theorem 1. Note that we lost something in this transformation: although our "width-based" merger MerWid_w did not need to know the set size n (and hence it was unbounded), the new "depth-based" merger MerDep_d needs to know n ahead of the time to find the optimal choice of width w. So, the new online merger is *not* unbounded anymore.

2-approximation for the width of unbounded online mergers. We finally describe how to achieve our online merger of Theorem 2. The key idea is to use our "width-based" merger MerWid_c for specifically chosen values of $n(c)$, by pretending that $n(c)$ is an upper bound on the total size of the streaming set. We will also increase c whenever $d(n)$ jumps by at least one (as it can gradually grow). Our careful choice of $n(c)$ and a "non-black-box" use of the analysis of our scheme MerDep_d allows us to prove that the resulting scheme will never use a width more than $2 \cdot \mathsf{widthLB}(n, d(n))$.

1.4 Related work

In addition to RBE, other works have also pursued paths to eliminate the key escrow problem from IBE. The work of [5] aimed to make the private-key generator decentralized. The works of [17, 18] proposed a notion of "accountablity" for PKG, by proposing how to catch an irresponsible PKG in case of breach. The works of [7, 8, 28] aimed to make it harder for the PKG to find out the receiver's identity. The works of [8, 10] studied interactive key generation that allows hiding user's identities. The work of [1] proposed to mix IBE and public-key encryption by constructing "Certificateless" Public Key Cryptography.

More recently, the work of [27] showed how to make RBE even more transparent by deploying it on blockchain. The two works of [15, 19] showed how to achieve *black-box* constructions of RBE based on assumptions on bilinear maps, while leveraging a polynomially large CRS that can grow with the number of parties. Further more, the work of [15] gave concrete implementations of RBE, and the work of [19] further generalized the notion of RBE to registered *attribute-based* encryption schemes that can handle attributes beyond identity (which is what RBE does).

The concept of cryptographic accumulators was first introduced by Benaloh and de Mare [4]. Using an accumulator, one can represent a set of values $\mathcal{S}$ by a short digest such that (1) there is a witness to prove membership for values in $\mathcal{S}$ and (2) it is infeasible to find such witness for values that are not in $\mathcal{S}$. Later, Barić and Pfitzmann [3] gave a more generalized definition called collision-resistant accumulators and a construction of such accumulators based on the strong RSA assumption. Both of the accumulators are static in the sense that the set of values $\mathcal{S}$ never changes after the digest has been generated. However, for many applications, the set of values can evolve with time. Observing this, Camenish and Lysyanskaya [6] introduced the concept of dynamic accumulators and provided a construction based on the strong RSA assumption. In a dynamic accumulator scheme, the set of values $\mathcal{S}$ can change. Namely, values can be added to or removed from the set. Therefore, both the digest and witnesses might be updated from time to time.

2 Online mergers and partitioners: constructions and lower bounds

In this section, we introduce the key information-theoretic problem of our work. To begin, we need to define merging operation on rooted trees.

▶ **Definition 4** (Merging operation and tree partitioning). *A rooted tree is either a single vertex* rt *(called root), or it has a root* rt *with* $i \geq 1$ *children* $u_1, \ldots, u_i$ *such that each* u_i *is the root of a rooted tree itself. A leaf is any vertex that has no children. (So a tree with a single vertex has a root that is also a leaf.) A merge operation takes a sequence* $(\mathsf{T}_1, \ldots, \mathsf{T}_j)$ *of* $j \geq 1$ *rooted trees and returns a single* T*, where* T *has a root* rt *with* j *children and the* i*th sub-tree of* rt *for* $i \in [j]$ *is* T_i*. For* $k \leq m$*, we say that a set of trees* $\{\mathsf{T}'_1, \ldots, \mathsf{T}'_k\}$ *can be obtained by a* single merge *operation from* $(\mathsf{T}_1, \ldots, \mathsf{T}_m)$*, if there is a subset* $\{\mathsf{T}''_1, \ldots, \mathsf{T}''_\ell\} \subseteq \{\mathsf{T}_1, \ldots, \mathsf{T}_m\}$ *that is merged into* T*, and* $\{\mathsf{T}'_1, \ldots, \mathsf{T}'_k\} = \{\mathsf{T}\} \cup \{\mathsf{T}_1, \ldots, \mathsf{T}_m\} \setminus \{\mathsf{T}''_1, \ldots, \mathsf{T}''_\ell\}$*. We also say that a set of trees* $\{\mathsf{T}'_1, \ldots, \mathsf{T}'_k\}$ *is a* merging *of* $\{\mathsf{T}_1, \ldots, \mathsf{T}_m\}$*, if one can obtain* $\{\mathsf{T}'_1, \ldots, \mathsf{T}'_m\}$ *from* $\{\mathsf{T}_1, \ldots, \mathsf{T}_m\}$ *by a series of (zero or more) single merge operations. A set of rooted trees* $\{\mathsf{T}_1, \ldots, \mathsf{T}_m\}$ *form a* tree partitioning *of* $\mathcal{S}$*, if they have* $|\mathcal{S}|$ *many leaves, and that each* $x \in \mathcal{S}$ *appears as a leaf in exactly one of* $\{\mathsf{T}_1, \ldots, \mathsf{T}_m\}$*.*

It is easy to see that if $\{\mathsf{T}_1, \ldots, \mathsf{T}_m\}$ form a tree partitioning of $\mathcal{S}$, then any merging of $\{\mathsf{T}_1, \ldots, \mathsf{T}_m\}$ will also be a tree partitioning of $\mathcal{S}$.

We now define the main object of our interest, namely an online merger on a set $[n]$.

▶ **Definition 5** (Online merger). *An* online merger M *for* $[n]$ *is a deterministic algorithm that works in* n *rounds as follows. Originally we have an empty set* $\mathcal{T}_0 = \emptyset$ *of trees. In round* i*, we start from* $\mathcal{T}_{i-1} \cup \{i\}$*; namely, a new tree with a single node labeled* i *gets added. Then, the algorithm* M *is allowed to apply any number of merge operations on* $\mathcal{T}_{i-1} \cup \{i\}$ *to reach* $\mathcal{T}_i$ *(at the end of round* i*). We call* $\mathsf{wid}_i = |\mathcal{T}_i|$ *the* width *at (the end of) round* i *and* $\mathsf{wid}_{[n]} = \max_{i \in [n]} |\mathcal{T}_i|$ *simply the width of* M *for* $[n]$*. At round* i *and* $j \in [i]$*, the* depth *of node* j *in round* i*, denoted by* dep_i^j *is the distance of the node* j *from the root in its rooted tree at the end of round* i*.*[3] *The depth of* $\mathcal{T}_i$ *is simply defined as* $\mathsf{dep}_i = \max_{j \in [i]} \mathsf{dep}_i^j$*, and the depth of* M *for* $[n]$ *is* $\mathsf{dep}_{[n]} = \max_{i \in [n]} \mathsf{dep}_i$*. An online merger for* all *the natural numbers* $\mathbb{N}$*, also called an* unbounded *online merger, informally, is one that keeps going forever. More specifically, an unbounded merger gets the inputs* $\{1\}, \{2\}, \ldots$*, and always maintains a tree partitioning over* $[n]$ *for all* $n \in \mathbb{N}$*. For an unbounded merger, we can still define the depths* $d_i, d_{[n]}$ *and widths* $\mathsf{wid}_i, \mathsf{wid}_{[n]}$ *of such mergers for all* $i, n \in \mathbb{N}$*. Any online merger* M *for* $[n]$ *(resp.* $\mathbb{N}$*) defines an online merger for all* $m \in [n]$ *(resp.* $[n], n \in \mathbb{N}$*).*

Extensions and generalizations. The definition above can be extended in multiple ways:

- **Randomness.** We could allow online mergers to be randomized algorithms, in which case the depth and width are defined with respect to a fixed randomness. However, in this work, we construct deterministic mergers, as our focus is on upper bounds, while our lower bounds also directly apply to randomized algorithms as well.
- **Other sets.** For simplicity, we defined mergers on $[n]$, while one can think of mergers who deal with *arbitrary* sets $\mathcal{S}$ of size n.
- **Partitioning.** One can generalize the notion of online mergers to algorithms that arbitrarily change the partitioning of the current set of elements. This class of algorithms are, e.g., useful to model algorithms that maintain a set of Merkle trees, but decide to break down those trees every now and then and reconstruct them from scratch.

[3] This is equal to the number of times that the trees containing j are merged, since $\{j\}$ was added and till the end of round i.

▶ **Definition 6** (Online partitioner). *An* online partitioner Part *for* $[n]$ *and an* unbounded *online partitioner for* $\mathbb{N}$ *are both defined similarly to their corresponding mergers with the difference that they can arbitrarily* re-partition *the current set of vertices rather than merely merging them. The notions of depth and width are defined similarly for partitioners, with the only difference that depth of a node* $j \in [n]$ *will denote the number of times that the set containing* j *has* changed.

Below, we first study lower bounds on depths and widths of bounded mergers. We then give *unbounded* online schemes that (closely) match these lower bounds.

2.1 Lower bounds for bounded partitioners

The goal of this subsection is to present functions $\mathsf{depthLB}(n, w)$ (resp. $\mathsf{widthLB}(n, d)$) that serve as lower bounds on the depth (resp. width) of any online bounded partitioner, assuming that the set size is n and the width is w (resp. depth is d).

We start by defining a DAG for any online partitioner or merger. Since partitioners generalize mergers, we only define these notions for partitioners.

▶ **Definition 7** (Forward DAGs of online partitioners and mergers). *Let* Part *be a deterministic online partitioner for* $[n]$*. Then,* Part *defines the following DAG* G_n *over* $[n]$*:* $(i, j) \in G$ *for* $i \le j$ *if the set* $\mathcal{S}$ *that contains* i *at the* beginning *of round* j *is different from the one that contains* i *at the end of the round* j*. For the special case of mergers,* $(i, j) \in G$ *means that the tree containing* i *at the beginning of round* j *is merged during round* j*.*

We now observe that skipping sequences in the DAGs of online partitioners imply a lower bound on their width.

▶ **Proposition 8** (Lower bound on the width from skipping sequences). *Let* Part *be an online partitioner for* $[n]$*. Let* G_n *be the forward DAG over* $[n]$ *defined by* Part*. Let* $\mathcal{S}$ *be a skipping sequence in* G_n*. Then, the width of* Part *is at least* $|\mathcal{S}|$*.*

Proof. Let $\mathcal{S} = \{s_1 < \cdots < s_\ell\}$. It suffices to show that s_i and s_j belong to different sets at the end of round ℓ for all $1 \le i < j \le \ell$. Assume, on the contrary, that s_i and s_j belong to the same set at the end of round ℓ. This means that at some time k such that $s_j \le k \le s_\ell$ the sets containing s_i, s_j change to include both of them, which implies the existence of the edge $(s_i, k) \in G_n$, where $s_i < s_j \le k \le s_\ell$. Such an edge, however, contradicts the definition of skipping sequences. ◀

If the depth of a partitioner or a merger is at most d, then by definition the out-degree of the vertices in the corresponding forward DAG G_n is at most d. Therefore, using Proposition 8 and Theorem 34 (proved in [22]) we obtain the following lower bound.

▶ **Theorem 9** (Lower bound on the width based on the depth). *Suppose* Part *is an online partitioner for* $[n]$*. Then the following hold for all* $d \ge 0, w \ge 1$*.*

1. *If* $n \ge \binom{w+d}{d+1}$ *and* $\mathsf{dep}_{[n]} \le d$*, then* $\mathsf{wid}_{[n]} \ge w$*.*
2. *If* $n \ge \binom{w+d}{d}$ *and* $\mathsf{wid}_{[n]} \le w$*, then* $\mathsf{dep}_{[n]} \ge d$*.*

Proof. The first part follows directly from Proposition 8 and Theorem 34. To prove the second part, suppose $\mathsf{wid}_{[n]} \le w$ and $\mathsf{dep}_{[n]} \le d-1$. Then, by applying the first part on depth $d-1$ (rather than d), we obtain that the depth should be at least $w+1$, which contradicts $\mathsf{wid}_{[n]} \le w$. ◀

We now present the depthLB and widthLB functions that serve as lower bounds on the depth and width of any online partitioner. Both of them can be expressed through the same function in the following corollary that follows directly from Theorem 9 above.

▶ **Corollary 10** (Lower-bound functions depthLB, widthLB for depth and width). *For natural numbers x, y, let* $\mathsf{minBin}(x,y) = \min\{z \in \mathbb{N}^+ \mid \binom{y+z}{y} > x\}$. *Then, the following holds for any partitioner* Part *over the set* $[n]$.

1. *If* $\mathsf{wid}_{[n]} \leq w$, *then* $\mathsf{dep}_{[n]} \geq \mathsf{depthLB}(n,w)$ *for* $\mathsf{depthLB}(n,w) = \mathsf{minBin}(n,w) - 1$.
2. *If* $\mathsf{dep}_{[n]} \leq d$, *then* $\mathsf{wid}_{[n]} \geq \mathsf{widthLB}(n,d)$ *for* $\mathsf{widthLB}(n,d) = \mathsf{minBin}(n,d+1)$.

2.2 Optimal bounded mergers

In this subsection, we focus on bounded online mergers that know the set-size n in advance and aims to optimally balance the trade-off between the width and the depth. For this setting, we show that a simple construction optimally matches the bounds of Corollary 10. In fact, we show that the following extremely simple construction achieves optimal depth for *all* set sizes, so long as the width is upper bounded by a fixed given amount. In other words, our simple construction is even an *unbounded* online merger for any given bound on the width. As a corollary, when the set-size n and a given *depth* are both known in advance, one can use our simple construction using the width $w = \mathsf{widthLB}(n,d)$ and achieve optimal width for the given depth.

▶ **Construction 11** (Optimal unbounded online merger for fixed width). *The online merger* MerWid_w *is parameterized by a positive integer w. At each round, while the total number of trees equals $w+1$,* MerWid_w *merges the subset of trees consisting of all trees with the minimum depth. (Note that if there is a unique tree of minimum depth, merging it with itself will simply increase its depth by one.)*

By definition the width (i.e., $\mathsf{wid}_i = |\mathcal{T}_i|$, where $\mathcal{T}_i$ is the collection of trees at the end of every round $i \in \mathbb{N}$) in Construction 11 it holds that $\mathsf{wid}_n \leq w$, and hence $\mathsf{wid}_{[n]} \leq w$ for all $n \in \mathbb{N}$. We now analyze the depth.

Alternative construction. Construction 11 sometimes merges a single tree, which simply increases its depth. This artificial merging can be avoided without increasing the depth or width. However, the redundant operations simplify some of the claims below about the analysis of the depth.

▶ **Lemma 12.** *In Construction 11, the following holds.*

1. *At the end of round* $\binom{d+w}{d} - 1$ *there are w trees of depth $d-1$.*
2. *Round* $\binom{d+w}{d}$ *is the first round, at the end of which there is a single tree of depth d.*

Proof. We use induction on d. The base case where $d = 1$ is true by inspection of the first $w+1$ rounds. Assume the claim is true for $d-1$. We show that it is true for d. By assumption we know at round $\binom{d-1+w}{d-1}$, there is exactly one tree of depth d. Reusing the assumption, we know at round $\binom{d+w-1}{d-1} + \binom{d+w-2}{d-1}$ there are 2 trees of depth d and no trees of other depth. Similarly, we know at round $\sum_{i=1}^{w} \binom{d+w-i}{d-1} = \binom{d+w}{d} - 1$ there are w trees of depth d and no trees of other depth. By inspection, we know at round $\binom{d+w}{d}$ there is exactly one tree of depth d and this is the first round where there is a tree of depth d. ◀

Using the lemma above, we can get an upper bound on the depth for Construction 11.

▶ **Theorem 13** (Construction 11 achieves optimal depth for all set sizes and widths). *In Construction 11, the depth* $\mathsf{dep}_{[n]}$ *of* MerWid_w *for set size n is* $\mathsf{depthLB}(n,w) = \mathsf{minBin}(n,w) - 1$, *which is optimal.*

Proof. Using Lemma 12, we know for $n \in [\binom{d+w}{d}, \binom{d+w+1}{d+1} - 1]$, the depth of M is exactly d. For such choice of n, d, it holds that $\binom{w+d+1}{w} > n$ and $\binom{w+d}{w} \leq n$. Therefore, by definition, this means that $d+1 = \mathsf{minBin}(n, w)$, and hence $d = \mathsf{minBin}(n, w) - 1$. ◀

▶ **Proposition 14** (Achieving optimal width, given set size and depth)**.** *Let n, d be two positive integers. Using MerWid_w in Construction 11 with width $w = \mathsf{widthLB}(n, d) = \mathsf{minBin}(n, d+1)$ will have depth at most d. Consequently, the scheme will use optimal width.*

Proof. Using Lemma 12, we know round $\binom{d+1+w}{d+1}$ is the first round where there is a tree of depth $d+1$. However, by construction we know $\binom{d+1+w}{d+1} > n$. Thus, the depth will be at most d. ◀

2.3 Unbounded online mergers

In this section, we study unbounded online mergers that do *not* know the set size $[n]$ in advance. We first design unbounded mergers for a given constant depth and then will generalize our construction to the setting in which d is a growing function of n (e.g., $d = \log n$).

2.3.1 Unbounded mergers for a given fixed depth

The key idea of our extension to unbounded mergers is as follows. Even though unbounded mergers play in a game with an infinite number of rounds, one can divide the rounds into stages where each stage only has a finite number of consecutive rounds. We can then treat each stage as an independent known-size merging game, which allows us to use Construction 11.

▶ **Construction 15** (Unbounded online mergers for given fixed depth d)**.** *We construct a merger MerDep_d which uses MerWid_w in Construction 11 as a subroutine. We first partition the set of rounds into* stages *$\mathcal{S}_i$ that consist of consecutive rounds and that $|\mathcal{S}_i| = \binom{d+i}{d}$. In stage $\mathcal{S}_i$ (i.e., for round $k, k \in [\sum_{j=1}^{i-1} |\mathcal{S}_j| + 1, \sum_{j=1}^{i} |\mathcal{S}_j|]$) we use MerWid_w using width $w = i$ and treat the incoming sets $\{\ell\}, \ell \in \mathcal{I}$ as if it is $\{\ell - \sum_{j=1}^{i-1}\}$. While doing so, we ignore the trees that are constructed in the previous $i-1$ stages (i.e., keep them as part of the set of trees, without merging them).*

▶ **Proposition 16.** *In Construction 15, we have $\mathsf{wid}_{[n]} \leq 2 \cdot \mathsf{widthLB}(n, d) - 1$ for all $d \geq 0$.*

Proof. If $d = 0$, Construction 15 is trivially optimal in width. Below, assume $d > 0$. Uing Lemma 12, we know each completed stage ends up with exactly one tree of depth d.

Let k be the smallest positive integer such that $\binom{d+1}{d} + \binom{d+2}{d} + \cdots + \binom{d+k}{d} = \binom{d+1+k}{d+1} - 1 \geq n$. There are at least $k-1$ completed stage. If the last stage is also completed, there are in total k trees. Otherwise, since the width of the last stage is bounded by k, the total width is bounded by $2k-1$. Note that $\binom{d+1+k}{d+1} > n$ but $\binom{d+k}{d+1} \leq n$, which means $k = \mathsf{minBin}(n, d+1) = \mathsf{widthLB}(n, d)$. ◀

2.3.2 Unbounded online mergers for growing depths

Let $d(n)$ be a non-decreasing function from $\mathbb{N}$ to $\mathbb{N}$ modeling our desired upper bound on $\mathsf{dep}_{[n]}$. For example, we might want to have a scheme with depth $\log \log n$ or $\log n / \log \log n$ (rounded to integers). In this subsection, we show how to achieve online mergers that respect the upper bound $\mathsf{dep}_{[n]} \leq d(n)$, while achieving a width that is within 2 multiplicative factor of the optimal width, as it will hold that $\mathsf{wid}_{[n]} \leq 2\mathsf{widthLB}(n, d(n))$.

Before defining the construction, here we define *jumping points* of the function $d(n)$.

▶ **Definition 17** (Jumping points). *Suppose $d(n)\colon \mathbb{N} \to \mathbb{N}$ is non-decreasing. Define the set of* jumping points *of $d(n)$, $\{a_1, a_2, a_3, \dots\}$, as follows.*

1. $a_1 = 1$;
2. *For $i > 1$, a_i is the smallest positive integer satisfying $d(a_i) > d(a_{i-1})$.*

▶ **Construction 18** (Approximately optimal merger for a growing depth). *Suppose $\{a_1, a_2, a_3, \dots\}$ are the jumping points of the non-decreasing depth function $d(n)$. Our unbounded merger $\mathsf{MerDep}_{d(n)}$ uses an unbounded merger MerDep_c for* fixed *depth c as a subroutine. Originally, there is no trees, and the algorithm $\mathsf{MerDep}_{d(n)}$ works as follows at round n.*

- *If $n = a_i$ for $i \in \{1, 2, \dots\}$, then $\mathsf{MerDep}_{d(n)}$ will merge all the current trees (including the single-node tree $\{a_i\}$ that has just arrived) into one tree.*
- *For $a_i < n < a_{i+1}$, $\mathsf{MerDep}_{d(n)}$ ignores the single merged tree that was shaped in round a_i and will treat the arriving sets $\{\ell\}$ as $\{\ell - a_i\}$ and runs MerDep_c for $c = d(a_i) = d(n)$.*

The fact that the depth of the trees of Construction 18 satisfy $\mathsf{dep}_{[n]} \le d(n)$ is immediate by its definition and the fact that $d(n)$ is non-decreasing. Specifically, for $n = a_i$, assuming the depth was previously at most $d(a_i - 1)$, by doing the single merge, the single merged tree will have depth at most $1 + d(a_i - 1) \le d(a_i) = d(n)$. Hence, in the following we focus on analyzing its width.

▶ **Proposition 19.** *Suppose $\mathsf{wid}^c_{[n]}$ is the width of MerDep_c that is used inside Construction 18. Then the width of the final merger in Construction 18 is at most $\mathsf{wid}^{d(n)}_{[n]} + 1$.*

Proof. At every round a_i for $i \in \{1, 2, \dots\}$, there will be exactly one tree. At round n where $a_i < n < a_{i+1}$, the width of $\mathsf{MerDep}_{d(a_i)}$ is upper bounded by $\mathsf{wid}^{d(a_i)}_{[n-a_i]}$. Since $\mathsf{wid}^c_{[n]}$ is a non-decreasing function of n for any fixed c and that $d(a_i) = d(n)$, we have $\mathsf{wid}^{d(a_i)}_{[n-a_i]} \le \mathsf{wid}^{d(a_i)}_{[n]} = \mathsf{wid}^{d(n)}_{[n]}$. Also note that the tree built at a_i is not touched during the rounds n such that $a_i < n < a_{i+1}$. Therefore, the width $\mathsf{wid}_{[n]}$ of the merger in Construction 18 is upper bounded by $\mathsf{wid}^{d(n)}_{[n]} + 1$. ◀

The following corollary follows immediately from Propositions 16 and 19.

▶ **Corollary 20** (2-approximating the width for a given growing depth). *If $d(n)$ is a non-decreasing depth function of n, and if one uses Construction 15 as the subroutine in Construction 18, then the resulting construction will have width bounded as $\mathsf{wid}_{[n]} \le 2 \cdot \mathsf{widthLB}(n, d(n))$.*

How about unbounded online mergers for a growing width? The results above leave one case uncovered: what if we want to have an unbounded merger that satisfies width $\mathsf{wid}_{[n]} \le w(n)$ for a given non-decreasing function $w(n)$? For the constant $w(n) = w$, we already have an optimal solution (see Theorem 13). But, what if $w(n)$ is an increasing function of n? Here we argue that it is in fact impossible to find an unbounded merger that approximates the optimal depth within any constant factor, the way Corollary 20 does this for the width. The reason is that the width function $w(n)$ can remain small for too long (when n grows) and then suddenly jump significantly. For example, suppose $w(i) = 1$ for all $i < n$, and $w(n) = n$. Then, the depth is forced to grow linearly $d_{n'} = n' - 1$ for all $n' < n$, while after reaching round n, the width is suddenly allowed to be n, for which we do not need any depth more than 1. However, we have already paid the cost of having depth $d_n > n - 1$.

Examples of choices of depth functions. Here we demonstrate the bounds on the width that follow from choosing the depth in various ways in the construction of Corollary 20.

▶ **Corollary 21.** *In Construction 18, and for non-decreasing depth function $d(n)$ it holds that* $\mathsf{wid}_{[n]} = O(d \cdot n^{\frac{1}{d+1}})$*, where* $d = d(n)$*. In particular, if* $d(n) = c \cdot \log n / \log\log n$*, then* $\mathsf{wid}_{[n]} < \mathsf{poly}(\log n)$*.*

Proof. Let $w = \mathsf{widthLB}(n, d) = \mathsf{minBin}(n, d+1)$. Using the bound $\binom{m}{k} \geq (\frac{m}{k})^k$, we have $(\frac{w+d}{d+1})^{d+1} \leq \binom{w+d}{d+1} \leq n$. We can then bound $w \leq (d+1) \cdot n^{\frac{1}{d+1}} - d$. By Corollary 20, we know $\mathsf{wid}_{[n]} \leq 2 \cdot (d+1) \cdot n^{\frac{1}{d+1}} - 2d$.

Now, we prove the second part. By the first part, we know $\mathsf{wid}_{[n]} = O(d \cdot n^{\frac{1}{d+1}})$, where $d = d(n) = c \cdot \log n / \log\log n$. In addition, we have $n^{\frac{1}{d+1}} = n^{\frac{1}{c \cdot \log n / \log\log n + 1}} < (2^{\log n})^{\frac{\log\log n}{c \log n}} = \log^{1/c} n$. Therefore, $\mathsf{wid}_{[n]} = O(d \cdot n^{\frac{1}{d+1}}) = O(\log^{1+1/c} n / \log\log n)$. ◀

2.3.3 Stronger lower bounds for unbounded online mergers

In this section, we show that there is a real cost to pay when we aim for *unbounded* online mergers. Namely, we show that when the merger is *not* aware of the set size n, it *cannot* match the lower bound $\mathsf{widthLB}(n, d)$, even though we *could* match this bound knowing n ahead of the time. In particular, we prove the following theorem.

▶ **Theorem 22** (Stronger lower bound for depth-one unbounded online mergers)**.** *Let* M *be an unbounded online merger (for* $\mathbb{N}$*) whose depth is bounded by 1 (i.e., only one merge is allowed for each element's set). Then,* $\mathsf{wid}_{[n]} \not\leq (2\sqrt{2} - \Omega(1))\sqrt{n}$*.*

To appreciate the bound of Theorem 22 we observe that when we know n ahead of the time, we can *beat* this lower bound. As shown in Proposition 23, there is a merger satisfying $\mathsf{wid}_{[n]} \leq \sqrt{2n}$. So, the lower bound of Theorem 22 is strictly larger than the optimal bound for the setting of knowing set sizes.

▶ **Proposition 23.** *Let n be a given positive integer. There is an online merger for $[n]$ whose depth is bounded by 1 and width is bounded by $\sqrt{2n}$.*

Proof. From Proposition 14, we know MerWid_w in Construction 11 with width $w = \mathsf{minBin}(n, 2)$ is an online merger for $[n]$ whose depth is bounded by 1. By definition, we know $\binom{w+1}{2} \leq n$. We then have $w \leq \frac{\sqrt{8n+1}-1}{2} \leq \sqrt{2n}$. ◀

Moreover, as shown in Proposition 24, Construction 15 matches this bound up to an additive constant gap.

▶ **Proposition 24.** *The unbounded merger of Construction 15 has* $\mathsf{wid}_{[n]} \leq 2\sqrt{2n}$ *for* $d = 1$*.*

Proof. From Proposition 16, we know $\mathsf{wid}_{[n]} \leq 2 \cdot \mathsf{widthLB}(n, 1) = 2 \cdot \mathsf{minBin}(n, 2) \leq 2\sqrt{2n}$. ◀

The key tool we use is a special kind of depth one merger called 1-regular merger. First recall that a *stage* is a sequence of consecutive rounds. Intuitively, a merger is 1-regular if rounds can be divided into stages such that at the last round of every stage all trees added during this stage are merged into one tree. Note that since the merger has depth one, no trees are merged at times other than the last round of a stage.

▶ **Definition 25** (1-regular merger). *Let* M *be an unbounded online merger (for* $\mathbb{N}$*) whose depth is bounded by* 1*. Let* $\mathcal{T} = \{t_1 < t_2 < t_3 < \cdots\}$ *be the set of all rounds where the merge operation is performed. Let* $t_0 = 0$*. We say* M *is* 1-regular *if at every* $t_i \in \mathcal{T}$ *the set of single-node trees* $\{t_{i-1}+1, t_{i-1}+2, \cdots, t_i\}$ *are all merged into one set (i.e., tree of depth 1).*

We first show that given any M whose depth is bounded by 1, there is a 1-regular M' that is at least as good as M with respect to width.

▶ **Lemma 26.** *Let* M *be a merger for* $\mathbb{N}$ *whose depth is bounded by* 1*. Then, there exists a 1-regular* M' *for* $\mathbb{N}$ *such that the width of* M' *is at most the width of* M *at every round* n*.*

Proof. If M is already 1-regular, then we are done. Otherwise, let $\mathcal{T} = \{t_1 < t_2 < t_3 < \cdots\}$ be the set of all rounds where the merge operation is performed by M. Let $t_0 = 0$. M' simply merges $\{t_{i-1}+1, t_{i-1}+2, \cdots, t_i\}$ at t_i. At round $t_i \in \mathcal{T}$, M' has only i trees while M has at least i trees since it has performed at least i merge operations, and due to the depth being 1, these merge operations are on separate nodes. At other times, since no merge operation is performed, M has at least as many trees as M' does. ◀

The following lemma is also useful for the proof of Theorem 22.

▶ **Lemma 27.** *Let* $j > 2$ *be an integer, and* $d < 2$ *and* c *be two positive reals. Define the function* $f(i) := \frac{i \cdots j}{(i-d)\cdots(j-d)} \cdot c$ *for arbitrary positive integer* $i > j$*. Then,* $f(i) = o(i^2)$*.*

Proof. We first define the function $g(i) := \frac{i \cdots j}{(i-2)\cdots(j-2)} \cdot c$ for arbitrary positive integer $i > j$. Note that we have $g(i) = \frac{i \cdot (i-1)}{(j-1)\cdot(j-2)} \cdot c$. Therefore, we have $g(i) = \theta(i^2)$. Then, it suffices to prove that $f(i) = o(g(i))$.

Let γ be an arbitrary positive real number. We show that $g(i) > \gamma \cdot f(i)$ for sufficiently large i. Equivalently, we prove that $\ln g(i) - \ln f(i) > \ln \gamma$ for sufficiently large i. We have

$$\ln g(i) - \ln f(i) = \ln \frac{i-d}{i-2} + \cdots + \ln \frac{j-d}{j-2} = \sum_{k=j-2}^{i-2} \ln\left(1 + \frac{2-d}{k}\right).$$

It suffices to show that the series $\sum_{k=j-2}^{\infty} \ln(1 + \frac{2-d}{k})$ diverges. To this end, we use integral test to show that it diverges. Note that

$$\int \ln\left(1 + \frac{2-d}{x}\right) dx = (2-d) \cdot \ln|x + 2 - d| + x \cdot \ln\left(1 + \frac{2-d}{x}\right) + C.$$

Therefore, we have $\int_{j-2}^{\infty} \ln(1 + \frac{2-d}{x}) dx = \infty$, which implies that $\sum_{k=j-2}^{i-2} \ln(1 + \frac{2-d}{k}) > \ln \gamma$ for sufficiently large i. ◀

Proof of Theorem 22. By Lemma 26, it suffices to prove the theorem for 1-regular mergers. Therefore, we assume M is 1-regular. Let $\mathcal{T} = \{t_1 < t_2 < t_3 < \cdots\}$ be the set of all rounds where M performs a merge operation. Let $t_0 = 0$ and $s_i = t_i - t_{i-1}$ be the size of stage i.

We first consider the case where $\mathcal{T}$ is a finite set. Then, starting at some round, no merge operation is ever performed, which means the width will be $\Theta(n)$.

Let $d < 2$ be an arbitrary positive real number. We now consider the case where $\mathcal{T}$ is an infinite set and there are only finitely many i satisfying $s_i \geq d \cdot \frac{t_i}{i}$. Then, there is a j such that $s_i < d \cdot \frac{t_i}{i}$ for $i \geq j$. Note that $s_i = t_i - t_{i-1} < d \cdot \frac{t_i}{i}$, which implies $t_i < \frac{i}{i-d} \cdot t_{i-1}$. We then have $t_i < \frac{i \cdots j}{(i-d)\cdots(j-d)} \cdot t_{j-1}$. From Lemma 27, we know $t_i = o(i^2)$, which means $i > \omega(\sqrt{t_i})$. In this case, we know $\mathsf{wid}_{[t_i]} \geq \omega(\sqrt{t_i})$.

Finally, we consider the case where $\mathcal{T}$ is an infinite set and there are infinitely many i satisfying $s_i \geq d \cdot \frac{t_i}{i}$. Let's consider one such i. At round $t_i - 1$, the width is at least $i - 1 + s_i - 1 > i + d \cdot \frac{t_i - 1}{i} - 2 \geq 2 \cdot \sqrt{d \cdot (t_i - 1)} - 2$. Note that this holds for arbitrary $d < 2$, which means that $\mathsf{wid}_{[n]} \not\leq (2\sqrt{2} - \Omega(1))\sqrt{n}$. Otherwise, there would be a constant $d' < 2$ such that $\mathsf{wid}_{[n]} \leq 2\sqrt{d' \cdot n}$ for sufficiently large n, contradicting that $\mathsf{wid}_{[m]} \geq 2 \cdot \sqrt{d \cdot m} - 2$ for a constant $d > d'$ and $m = t_i - 1$, as proved above. ◀

3 RBE with optimal number of decryption updates

In this section, we observe that the same approach of using online merger for transparent additive accumulators in Section B extends to registration based encryption schemes (RBEs), while the length of the digests and number of witness updates would correspond to the length of the public parameter and the number of decryption updates. Hence, we can obtain RBEs with *optimal* number of decryption updates matching the lower bound of [22]. The basic definitions of RBE and the primitives used in our construction can be found in Section A. Roughly speaking, the construction is the same as the IO-based construction of [12], while we use our own updates-optimal accumulator instead of the one by [25].

▶ **Construction 28.** *We will use an IO scheme* $(\mathsf{Obf}, \mathsf{Eval})$*, and a SSB hash function system* $(\mathsf{Hash}, \mathsf{HGen})$ *and a PKE scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$*. Using them together with a merger* M*, we show how to implement the subroutines of RBE according to Definition 37.*

- $\mathsf{Stp}(1^\kappa) \to (\mathsf{pp}_0, \mathsf{aux}_0)$*: This algorithm outputs* $\mathsf{pp}_0 = \mathsf{hk}_2 \leftarrow \mathsf{HGen}(1^\kappa, 2, 0)$ *and* $\mathsf{aux} = \emptyset$ *is empty. Then, initialize* HMer *(see Definition 43) using* M *and* H*. H is defined such that* $H(\mathsf{Val}(\mathsf{x}_1), \ldots, \mathsf{Val}(\mathsf{x}_i)) = \mathsf{Hash}(\mathsf{hk}_i, (\mathsf{Val}(\mathsf{x}_1), \ldots, \mathsf{Val}(\mathsf{x}_i)))$*. Note that here we have only sampled* hk_2*. If another key* hk_i *is needed in* Reg *because* HMer *needs to merge more trees,* Reg *will generate the key on the fly by running* HGen*.*
- $\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{pp}_n, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}_{n+1}$*: First add a new tree whose root has value* $\mathsf{Hash}(\mathsf{hk}_2, (\mathsf{id}, \mathsf{pk}))$ *with* id *and* pk *as the children nodes. We then let* HMer *handles the merging of trees. If another key* hk_i *is needed, run* $\mathsf{HGen}(1^\kappa, i, 0)$ *to get the key. Then, output the list of pairs of root and depth of all trees* $((\mathsf{rt}_1, \mathsf{d}_1), \ldots, (\mathsf{rt}_\eta, \mathsf{d}_\eta))$ *together with all keys* hk_i *as* pp_{n+1}*.*
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{id}, \mathsf{m}) \to \mathsf{ct}$*: First parse* pp *to get a list of pairs of root and depth of all trees* $((\mathsf{rt}_1, \mathsf{d}_1), \ldots, (\mathsf{rt}_\eta, \mathsf{d}_\eta))$*. Generate programs* $\mathsf{P}_1, \ldots, \mathsf{P}_\eta$ *where* P_i *works as follows:*
  Hardwired values: rt_i*,* d_i*,* $(\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$*,* m*,* id*,* r *(the randomness)*
  Input: pth
  1. *Parse* $\mathsf{pth} = ((\ell_{\mathsf{d}_i}, r_{\mathsf{d}_i}), \ldots, (\ell_1, r_1))$*, and if not possible, output* $\perp$*.*
  2. *If* $\mathsf{id} \neq \ell_{\mathsf{d}_i}$*, then output* $\perp$*.*
  3. *Compute* $\mathsf{tmp}_{j-1} = \mathsf{Hash}(\mathsf{hk}_{\mathsf{len}_j}, (\ell_j, \mathsf{tmp}_j, r_j))$ *for* $j = \mathsf{d}_i, \ldots, 1$ *where* $\mathsf{tmp}_{\mathsf{d}_i}$ *is the empty string and* len_j *is the length of* $(\ell_j, \mathsf{tmp}_j, r_j)$*. (Note that* ℓ_j *and* r_j *are tuples.) If* $\mathsf{tmp}_0 = \mathsf{rt}_i$*, then output* $\mathsf{E}(\mathsf{r}_{\mathsf{d}_i}, \mathsf{m}; \mathsf{r})$ *by using* $\mathsf{r}_{\mathsf{d}_i}$ *as the public key and* r *as the randomness, otherwise output* $\perp$*.*

  Then, output $\mathsf{ct} := (\mathsf{pp}, \mathsf{Obf}(\mathsf{P}_1), \ldots, \mathsf{Obf}(\mathsf{P}_\eta))$ *where* Obf *is IO obfuscation.*
- $\mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \to \mathsf{u}$*: First locate the tree T having* id *as one of the leaves. If no such tree exists, halt. Otherwise, let d be the depth of T and* $(p_d, \ldots, p_0)$ *be the path from* p_d*, which is* id*, to the root* p_0*. Let* ℓ_i *(resp.* r_i*) be the tuple of values of left (resp. right) siblings of* p_i *for* $i \in [d-1]$*. Note that the sibling of* id *is* pk*. Let* $\ell_d = \mathsf{id}$ *and* $r_d = \mathsf{pk}$*. Output* $\mathsf{w} = ((\ell_d, r_d), \ldots, (\ell_1, r_1))$*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct}) \to \mathsf{m}$*: Parse* $\mathsf{ct} = (\mathsf{pp}, \mathsf{Obf}(\bar{\mathsf{P}}_1), \ldots, \mathsf{Obf}(\bar{\mathsf{P}}_\eta))$*. Form* $\mathsf{m}_i = \mathsf{D}_{\mathsf{sk}}(\bar{\mathsf{P}}_i(\mathsf{u}))$ *for each program* $\bar{\mathsf{P}}_i$*. Output the first* $\mathsf{m}_i \neq \perp$*.*

Completeness of Construction 28 is straightforward.

▶ **Proposition 29** (Compactness and Efficiency of Construction 28)**.** *Construction 28 satisfies the compactness requirements of Definition 38.*

Proof. The length of public parameter is the sum of the length of keys, which is bounded by $\kappa \cdot \mathsf{wid}_{[n]}$, and roots, which is again bounded by $\kappa \cdot \mathsf{wid}_{[n]}$, and depth, which is bounded by $\mathsf{wid}_{[n]} \cdot \log \mathsf{dep}_{[n]}$. In total, it is bounded by $(2 \cdot \kappa + \log \mathsf{dep}_{[n]}) \cdot \mathsf{wid}_{[n]}$. The number of updates is bounded by $\mathsf{dep}_{[n]}$. The size of update for an accumulated value is the number of trees that are merged with the tree it belongs times the length of the roots. Since the depth is bounded by $\mathsf{dep}_{[n]}$, we know at most $\mathsf{dep}_{[n]}$ merges can happen. Since the width is bounded by $\mathsf{wid}_{[n]}$, we know at most $\mathsf{wid}_{[n]} \cdot \mathsf{dep}_{[n]}$ trees are merged. Therefore, the size of update is bounded by $\kappa \cdot \mathsf{wid}_{[n]} \cdot \mathsf{dep}_{[n]}$. For efficiency, note that the total number of merges during the addition of a value is bounded by $\mathsf{wid}_{[n]} \cdot \mathsf{dep}_{[n]}$. Also note that one can use an appropriate data structure that efficiently finds the tree an identity belongs. ◀

▶ Remark 30. By Corollary 21, we know that if we use the fully online merger in Construction 18 where $d(n) = \frac{\log n}{\log \log n}$, the number of updates is bounded by $\frac{\log n}{\log \log n}$ and the length of public parameter is $\mathsf{poly}(\kappa, \log(n))$, resolving the open question of [22].

▶ **Proposition 31** (Security of Construction 28)**.** *Construction 28 satisfies the soundness requirements of Definition 39.*

Proof (Sketch). The proof is almost identical to the proof in [12]. The key idea is that the index-hiding and somewhere statistically binding property forces all PPT algorithms to behave as if the public key is statistically binding to the root of the tree. Then, obfuscation and the semantic security of public key encryption guarantees that encryptions of different messages are indistinguishable. We refer readers to [12] for details. ◀

References

1 Sattam S Al-Riyami and Kenneth G Paterson. Certificateless public key cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 452–473. Springer, 2003.

2 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-44647-8_1.

3 Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, pages 480–494, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

4 Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT '93*, pages 274–285, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

5 Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-44647-8_13.

6 Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, pages 61–76, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

7 Zhaohui Cheng, Richard Comley, and Luminita Vasiu. Remove key escrow from the identity-based encryption system. In *Exploring New Frontiers of Theoretical Informatics*, pages 37–50. Springer, 2004.

8 Sherman SM Chow. Removing escrow from identity-based encryption. In *International Workshop on Public Key Cryptography*, pages 256–276. Springer, 2009.

9 Kelong Cong, Karim Eldefrawy, and Nigel P Smart. Optimizing registration based encryption. In *IMA International Conference on Cryptography and Coding*, pages 129–157. Springer, 2021.

10 Keita Emura, Shuichi Katsumata, and Yohei Watanabe. Identity-based encryption with security against the KGC: a formal model and its instantiation from lattices. In *European symposium on research in computer security*, pages 113–133. Springer, 2019.

11 Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29 2013. IEEE Computer Society Press. doi:10.1109/FOCS.2013.13.

12 Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from ibe. In *Theory of Cryptography Conference*, pages 689–718. Springer, 2018.

13 Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 63–93, Beijing, China, April 14–17 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17259-6_3.

14 Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography – PKC 2019*, pages 63–93, Cham, 2019. Springer International Publishing.

15 Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. Cryptology ePrint Archive, Paper 2022/1505, 2022. URL: https://eprint.iacr.org/2022/1505.

16 Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 621–651, Santa Barbara, CA, USA, August 17–21 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-56784-2_21.

17 Vipul Goyal. Reducing trust in the PKG in identity based cryptosystems. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 430–447, Santa Barbara, CA, USA, August 19–23 2007. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-74143-5_24.

18 Vipul Goyal, Steve Lu, Amit Sahai, and Brent Waters. Black-box accountable authority identity-based encryption. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 427–436. ACM, 2008.

19 Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. Cryptology ePrint Archive, Paper 2022/1500, 2022. URL: https://eprint.iacr.org/2022/1500.

20 Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13 2015. Association for Computing Machinery. doi:10.1145/2688073.2688105.

21 Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

22 Mohammad Mahmoody, Wei Qi, and Ahmadreza Rahimi. Lower bounds for the number of decryption updates in registration-based encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography*, pages 559–587, Cham, 2022. Springer Nature Switzerland.

23 Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 121–145, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

24 Ilker Ozcelik, Sai Medury, Justin Broaddus, and Anthony Skjellum. An overview of cryptographic accumulators. In *Proceedings of the 7th International Conference on Information Systems Security and Privacy*. SCITEPRESS – Science and Technology Publications, 2021. doi:10.5220/0010337806610669.

25 Leonid Reyzin and Sophia Yakoubov. Efficient asynchronous accumulators for distributed pki. In *Security and Cryptography for Networks: 10th International Conference, SCN 2016, Amalfi, Italy, August 31–September 2, 2016, Proceedings 10*, pages 292–309. Springer, 2016.

26 Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53, Santa Barbara, CA, USA, August 19–23 1984. Springer, Heidelberg, Germany.

27 Qin Wang, Rujia Li, David Galindo, Qi Wang, Shiping Chen, and Yang Xiang. Transparent registration-based encryption through blockchain. *Distributed Ledger Technologies: Research and Practice*, 2022.

28 Quanyun Wei, Fang Qi, and Zhe Tang. Remove key escrow from the BF and Gentry identity-based encryption with non-interactive key generation. *Telecommunication Systems*, pages 1–10, 2018.

A Preliminaries

▶ **Definition 32** (Forward DAGs). *Let $G = (\mathcal{V}_G, \mathcal{E}_G)$ be a directed acyclic graph (DAG) with vertices $\mathcal{V}_G = [n]$ (in case of being finite) or $\mathcal{V}_G = \mathbb{N}$ (in case of being infinite). We write $i \in G$ if $i \in \mathcal{V}_G$, ad we write $(i,j) \in G$ if $(i,j) \in \mathcal{E}_G$ (i.e., there is an edge from i to j in G). We call G a forward DAG, if for all $(i,j) \in G$, we have $i \leq j$. For any vertex u, by $\deg^+(u) = |\{v \mid (u,v) \in G\}|$ we denote the* out-degree *of u, and we let $\deg^+(G) = \max_{u \in G} \deg^+(u)$.*

▶ **Definition 33** (Skipping sequences [22]). *Let G be a forward DAG (see Definition 32). We call $\mathcal{S} = \{u_1 < u_2 < \cdots < u_k\} \subseteq \mathcal{V}_G$ a* skipping sequence *if for every $i \leq k-1$ and every edge $(u_i, v) \in G$, it holds that: either $v < u_{i+1}$ or $v > u_k$ (i.e., $v \notin \{u_{i+1}, u_{i+1}+1, \ldots, u_k\}$).*

▶ **Theorem 34** (Skipping sequences in low-degree DAGs [22]). *Let G be a forward DAG over vertices $[n]$, where $n \geq \binom{w+d}{d+1}$ for $w, d \in \mathbb{N}$, and that that $\deg^+(G) \leq d$. Then, there exists a skipping sequence in G of size at least w.*

▶ **Definition 35** (Indistinguishability obfuscation). *A uniform PPT algorithm* Obf *is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\kappa\}_{\kappa \in \mathbb{N}}$ (where each C_κ is a set indexed by a security parameter κ) if the following holds:*

- ***Completeness.*** *For all security parameters $\kappa \in \mathbb{N}$ and all circuits $C \in \mathcal{C}_\kappa$, we obtain an obfuscation with the same function:*

$$\Pr_{\mathsf{Obf}}[C' \equiv C : C' = \mathsf{Obf}(1^\kappa, C)] = 1.$$

- ***Security.*** *For any PPT distinguisher D, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\kappa \in \mathbb{N}$*, for all pairs of functionally equivalent circuits* $C_1 \equiv C_2$ *from the same family* $C_1, C_2 \in \mathcal{C}_\kappa$*,*

$$|\Pr_{\mathsf{Obf}}[D(1^\kappa, \mathsf{Obf}(1^\kappa, C_1)) = 1] - \Pr_{\mathsf{Obf}}[D(1^\kappa, \mathsf{Obf}(1^\kappa, C_2)) = 2]| = 1.$$

▶ **Definition 36** (SSB hash functions [23])**.** *A somewhere statistically binding hash system consists of two polynomial time algorithms* $\mathsf{HGen}, \mathsf{Hash}$*.*

- $\mathsf{HGen}(1^\kappa, 1^s, L, i) \to \mathsf{hk}$ *: This algorithm takes as input the security parameter* 1^κ*, a block size* s*, an input length* $L \leq 2^\kappa$ *and an index* $i \in \{0, \ldots, L-1\}$ *and outputs a hashing key* hk*. Without loss of generality, we assume that* $s = \kappa$*. Therefore, we will not write* s *explicitly.*
- $\mathsf{Hash}(\mathsf{hk}, \mathsf{x}) \to \mathsf{y}$ *: This deterministic algorithm takes as input a hashing key* hk *and a value* $\mathsf{x} \in \{0,1\}^{s \cdot L}$ *and outputs a hash* $\mathsf{y} \in \{0,1\}^\kappa$*.*

We require the following properties:

- ***Index hiding.*** *We consider the following game between an attacker* $\mathcal{A}$ *and a challenger:*
  - *The attacker* $\mathcal{A}(1^\kappa)$ *chooses parameters* 1^s*,* L *and two indices* $i_0, i_1 \in \{0, \ldots, L-1\}$*.*
  - *The challenger chooses a bit* $b \leftarrow \{0,1\}$ *and sets* $\mathsf{hk} \leftarrow \mathsf{HGen}(1^\kappa, 1^s, L, i_b)$*.*
  - *The attacker* $\mathcal{A}$ *gets* hk *and outputs a bit* b'*.*

  We require that for any PPT attacker $\mathcal{A}$ *we have* $|\Pr[b = b'] - \frac{1}{2}| = \mathsf{negl}(\kappa)$ *in the above game.*
- ***Somewhere statistically binding.*** *Let* $\mathsf{x} \in \{0,1\}^{s \cdot L}$ *and* $i \in \{0, \ldots, L-1\}$*. By* $\mathsf{x}[i]$ *we denote the sub-string of* x *starting at* $s \cdot i + 1$ *and ending at* $s \cdot (i+1)$*. We say that* hk *is statistically binding for an index* $i \in \{0, \ldots, L-1\}$ *if there do not exist any values* $\mathsf{x}, \mathsf{x}' \in \{0,1\}^{s \cdot L}$ *with* $\mathsf{x}[i] \neq \mathsf{x}'[i]$ *such that* $\mathsf{Hash}(\mathsf{hk}, \mathsf{x}) = \mathsf{Hash}(\mathsf{hk}, \mathsf{x}')$*. We require that for any parameters* s*,* L *and any integer* $i \in \{0, \ldots, L-1\}$ *we have:*

$$\Pr_{\mathsf{HGen}}[\mathsf{hk} \text{ is statistically binding for index } i : \mathsf{hk} \leftarrow \mathsf{HGen}(1^\kappa, 1^s, L, i)] \geq 1 - \mathsf{negl}(\kappa).$$

A.1 Registration-Based Encryption

▶ **Definition 37** (Syntax of RBE)**.** *PPT algorithms* $(\mathsf{Gen}, \mathsf{Reg}, \mathsf{Enc}, \mathsf{Upd}, \mathsf{Dec})$ *form a* registration-based encryption *(RBE for short) if they work together as follows.*

- ***Generating CRS.*** *A common random string* crs *of length* $\mathsf{poly}(\kappa)$ *is publicly sampled at the beginning, for the security parameter* κ*.*
- ***Key Generation.*** $\mathsf{Gen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$*: The randomized algorithm* Gen *outputs a pair of public and secret keys* $(\mathsf{pk}, \mathsf{sk})$*. The key generation algorithm is run by any honest party locally who wants to register itself into the system.*
- ***Registration.*** $\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}'$*: The deterministic algorithm* Reg *takes as input the CRS* crs*, current public parameter* pp*, a registering identity* id *and a public key* pk *(supposedly for the identity* id*), and it outputs* pp' *as the updated public parameters. The* Reg *algorithm uses* read *and* write *access to* auxiliary information aux *which will be updated into* aux' *during the process of registration and helps with the efficiency of the registration and updates (below). The system is initialized with* $\mathsf{pp}, \mathsf{aux} = \bot$*.*
- ***Encryption.*** $\mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{m}) \to \mathsf{ct}$*: The randomized algorithm* Enc *takes as input the CRS* crs*, a public parameter* pp*, a recipient identity* id*, and a plaintext message* m*, and it outputs a ciphertext* ct*.*

- ***Update.*** $\mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \mathsf{u}$*: The deterministic algorithm* Upd *takes as input the current public parameter* pp*, an identity* id*, and a public key* pk*. It has* read only *oracle access to* aux *and generates an update information* u *that can help* id *to decrypt its messages.*
- ***Decryption.*** $\mathsf{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct}) \to \mathsf{m}$*: The deterministic decryption algorithm* Dec *takes as input a secret key* sk*, an update information* u*, and a ciphertext* ct*, and it outputs a message* $\mathsf{m} \in \{0,1\}^*$ *or in* $\{\perp, \mathsf{GetUpd}\}$*. The symbol* $\perp$ *indicates a syntax error while* GetUpd *indicates that more recent update information (than* u*) might be needed for decryption.*

The Reg *and* Upd *algorithms are performed by the party called* key curator*, which we call KC for short, and* aux *can be seen as the state held by the KC.*

▶ **Definition 38** (Completeness, compactness, and efficiency of RBE)**.** *Consider the following game* $\mathsf{Comp}_{\mathcal{A}}(\kappa)$ *between a challenger* $\mathcal{C}$ *and an interactive* computationally unbounded *adversary* $\mathcal{A}$ *who is yet limited to* $\mathsf{poly}(\kappa)$ *rounds of interaction.*

1. ***Initialization.*** *$\mathcal{C}$ sets* $\mathsf{pp} = \perp$, $\mathsf{aux} = \perp$, $\mathsf{u} = \perp$, $\mathcal{D} = \emptyset$, $\mathsf{id}^* = \perp$, $t = 0$, *and* $\mathsf{crs} \leftarrow U_{\mathsf{poly}(\kappa)}$*, and sends the sampled* crs *to $\mathcal{A}$.*
2. *Till $\mathcal{A}$ continues (which is at most* $\mathsf{poly}(\kappa)$ *steps), proceed as follows. At every iteration, $\mathcal{A}$ chooses exactly one of the actions below to perform.*
   a. ***Registering a corrupted (non-target) identity.*** *$\mathcal{A}$ sends some* $\mathsf{id} \notin \mathcal{D}$ *and* pk *to $\mathcal{C}$. $\mathcal{C}$ registers* $(\mathsf{id}, \mathsf{pk})$ *by letting* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ *and* $\mathcal{D} := \mathcal{D} \cup \{\mathsf{id}\}$.
   b. ***Registering the (uncorrupted) target identity.*** *This step is allowed only if* $\mathsf{id}^* = \perp$*. In that case, $\mathcal{A}$ sends some* $\mathsf{id}^* \notin \mathcal{D}$ *to $\mathcal{C}$. $\mathcal{C}$ then samples* $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(1^\kappa)$*, updates* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$ *and* $\mathcal{D} := \mathcal{D} \cup \{\mathsf{id}^*\}$*, and sends* pk^* *to $\mathcal{A}$.*
   c. ***Encrypting for the target identity.*** *This step is allowed only if* $\mathsf{id}^* \neq \perp$*. In that case, $\mathcal{C}$ sets* $t = t + 1$*. $\mathcal{A}$ sends* $\mathsf{m}_t \in \{0,1\}^*$ *to $\mathcal{C}$ who then sets* $\mathsf{m}'_t := \mathsf{m}_t$ *and sends back a corresponding ciphertext* $\mathsf{ct}_t \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{m}_t)$ *to $\mathcal{A}$.*
   d. ***Decryption for the target identity.*** *$\mathcal{A}$ sends a* $j \in [t]$ *to $\mathcal{C}$. $\mathcal{C}$ then lets* $\mathsf{m}'_j = \mathsf{Dec}(\mathsf{sk}^*, \mathsf{u}, \mathsf{ct}_j)$*. If* $\mathsf{m}'_j = \mathsf{GetUpd}$*, $\mathcal{C}$ gets* $\mathsf{u} = \mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}^*)$ *and then* $\mathsf{m}'_j = \mathsf{Dec}(\mathsf{sk}^*, \mathsf{u}, \mathsf{ct}_j)$.

Let $n = |\mathcal{D}|$ *be the number of identities registered till a specific moment. We require the following properties to hold for all such $\mathcal{A}$ (as specified above) and for all the moments during the game* $\mathsf{Comp}_{\mathcal{A}}(\kappa)$.

- ***Completeness.*** *The adversary $\mathcal{A}$ wins, if there is some* $j \in [t]$ *for which* $\mathsf{m}'_j \neq \mathsf{m}_j$*. We require that* $\Pr[\mathcal{A} \text{ wins } \mathsf{Comp}_{\mathcal{A}}(\kappa)] = \mathsf{negl}(\kappa)$.
- ***Parameterized compactness and efficiency.*** [4]
  - ***Size of public parameter.*** $|\mathsf{pp}| = \mathsf{w}(\kappa, n)$.
  - ***Number of updates.*** *The total number of invocations of* Upd *for identity* id^* *in Step 2(d) of the game* $\mathsf{Comp}_{\mathcal{A}}(\kappa)$ *is at most* $\mathsf{d}(\kappa, n)$.
  - ***Size of update.*** $|\mathsf{u}| \leq \mathsf{poly}(\kappa, \mathsf{w}, \mathsf{d})$.
  - ***Runtime of registration and update.*** *The running time of each invocation of* Reg *and* Upd *is at most* $\mathsf{poly}(\kappa, \mathsf{w}, \mathsf{d})$.

▶ **Definition 39** (Security of RBE)**.** *For any interactive PPT adversary $\mathcal{A}$, consider the following game* $\mathsf{Sec}_{\mathcal{A}}(\kappa)$ *between $\mathcal{A}$ and a challenger $\mathcal{C}$.*

[4] If both $\mathsf{w}(\kappa, n)$ and $\mathsf{d}(\kappa, n)$ are $\mathsf{poly}(\kappa, \log n)$, then this becomes the standard definition.

1. ***Initialization.*** *$\mathcal{C}$ sets* $\mathsf{pp} = \bot$, $\mathsf{aux} = \bot$, $\mathcal{D} = \emptyset$, $\mathsf{id}^* = \bot$, $\mathsf{crs} \leftarrow U_{\mathsf{poly}(\kappa)}$ *and sends the sampled* crs *to* $\mathcal{A}$.
2. *Till $\mathcal{A}$ continues (which is at most* $\mathsf{poly}(\kappa)$ *steps), proceed as follows. At every iteration, $\mathcal{A}$ chooses exactly one of the actions below to perform.*
   a. ***Registering non-target identity.*** *$\mathcal{A}$ sends some* $\mathsf{id} \notin \mathcal{D}$ *and* pk *to $\mathcal{C}$. $\mathcal{C}$ registers* $(\mathsf{id}, \mathsf{pk})$ *by* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ *and* $\mathcal{D} := \mathcal{D} \cup \{\mathsf{id}\}$.
   b. ***Registering the target identity.*** *This step can be run only if* $\mathsf{id}^* = \bot$. *$\mathcal{A}$ sends some* $\mathsf{id}^* \notin \mathcal{D}$ *to $\mathcal{C}$. $\mathcal{C}$ then samples* $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(1^\kappa)$, *updates* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$, $\mathcal{D} := \mathcal{D} \cup \{\mathsf{id}^*\}$, *and sends* pk^* *to $\mathcal{A}$.*
3. ***Encrypting for the target identity.*** *If* $\mathsf{id}^* = \bot$, *then $\mathcal{A}$ first sends some* $\mathsf{id}^* \notin \mathcal{D}$ *to $\mathcal{C}$ (this is for modeling encryptions for non-registered target identities.) Next $\mathcal{A}$ sends two messages m_0, m_1 of the same length to C. Next, $\mathcal{C}$ generates* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, m_b)$, *where* $b \leftarrow \{0,1\}$ *is a random bit, and sends* ct *to* $\mathcal{A}$.
4. *The adversary $\mathcal{A}$ outputs a bit b' and wins the game if $b = b'$.*

An RBE scheme is secure if for all PPT $\mathcal{A}$, $\Pr[\mathcal{A} \text{ wins } \mathsf{Sec}_{\mathcal{A}}(\kappa)] < \frac{1}{2} + \mathsf{negl}(\kappa)$.

B Accumulators with optimal number of witness updates

An accumulator is a primitive in which a sequence of inputs $x_1, \ldots, x_n$ are added gradually to a pool, while (1) there is a compact digest/public parameter that is the representative of the added inputs $\{x_1, \ldots, x_n\}$, and (2) using the right witness, and the digest, one can prove the membership of x_i to the set. Such accumulators are called *additive* accumulators [24]. In this section, we use our results about online mergers to study the relation between the length of the digest and the number of updates for accumulators. Also, the accumulators that we study are transparent. Namely, there is no secret state hold by the accumulator.

In the remainder of this section, we first give the formal definition of transparent additive accumulators. Then, we give a general construction of such accumulators from collision resistant hash functions and online mergers. Finally, by applying our results on online merging, we will study the relation between the length of the digest and the number of times that the witnesses of membership need to be updated in such accumulators.

▶ **Definition 40** (Accumulators). *An accumulator scheme consists of four algorithms.*

- ***Key Generation.*** $\mathsf{Gen}(1^\kappa) \to \mathsf{k}$: *The randomized algorithm* Gen *takes as input the security parameter* 1^κ *and outputs an accumulator key* k.
- ***Addition.*** $\mathsf{Add}^{[\mathsf{aux}]}(\mathsf{k}, \mathsf{pp}, \mathsf{v}) \to \mathsf{pp}'$: *The deterministic algorithm* Add *takes as input an accumulator key* k, *a digest* pp *and a value* v, *has* read *and* write *access to the auxiliary information* aux, *and outputs a new digest* pp'.
- ***Witness update.*** $\mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{v}) \to \mathsf{w}$: *The deterministic algorithm* Upd *takes as input a digest* pp, *a value* v, *has* read-only *access to* aux *and outputs a witness* w.
- ***Verification.*** $\mathsf{Ver}(\mathsf{k}, \mathsf{pp}, \mathsf{v}, \mathsf{w}) \to \mathsf{b}$: *The deterministic algorithm* Ver *takes as input an accumulator key* k, *a digest* pp, *a value* v, *a witness* w *and outputs a bit* $\mathsf{b} \in \{0,1\}$.

▶ **Definition 41** (Completeness of accumulators). *For any interactive computationally unbounded adversary $\mathcal{A}$ that still has a limited* $\mathsf{poly}(\kappa)$ *round complexity, consider the following game* $\mathsf{Comp}_{\mathcal{A}}(\kappa)$ *between $\mathcal{A}$ and a challenger $\mathcal{C}$.*

1. ***Initialization.*** *The challenger $\mathcal{C}$ sets* $\mathsf{pp} = \bot$, $\mathsf{aux} = \bot$, $\mathsf{t} = \bot$, $\mathcal{V} = \emptyset$, *generates* $\mathsf{k} \leftarrow \mathsf{Gen}(1^\kappa)$ *and sends* k *to the adversary $\mathcal{A}$.*
2. *Till $\mathcal{A}$ continues (which is at most* $\mathsf{poly}(\kappa)$ *steps), $\mathcal{A}$ chooses one of the following operations to perform.*

a. ***Addition.*** *$\mathcal{A}$ sends a value $\mathsf{v} \notin \mathcal{V}$ to $\mathcal{C}$ and $\mathcal{C}$ sets $\mathsf{pp} \leftarrow \mathsf{Add}^{[\mathsf{aux}]}(\mathsf{k}, \mathsf{pp}, \mathsf{v})$ and adds v to $\mathcal{V}$.*

b. ***Set target.*** *$\mathcal{A}$ sends a value $\mathsf{v} \in \mathcal{V}$ to $\mathcal{C}$. If $\mathsf{t} \neq \perp$, $\mathcal{C}$ does nothing. Otherwise, $\mathcal{C}$ sets $\mathsf{t} \leftarrow \mathsf{v}$.*

c. ***Witness generation.*** *If $\mathsf{t} = \perp$, $\mathcal{C}$ does nothing. Otherwise, $\mathcal{C}$ generates $\mathsf{w} \leftarrow \mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{t})$.*

The adversary $\mathcal{A}$ wins the game if $\mathsf{Ver}(\mathsf{k}, \mathsf{pp}, \mathsf{t}, \mathsf{w}) = 0$ for at least once in Step 2c. We call an accumulator scheme complete if $\Pr[\mathcal{A}$ wins in $\mathsf{Comp}_{\mathcal{A}}(\kappa)] = \mathsf{negl}(\kappa)$ for any $\mathcal{A}$.

Let $n = |\mathcal{V}|$ be the number of values added till a specific moment during the game $\mathsf{Comp}_{\mathcal{A}}(\kappa)$. We define the number of updates of an accumulator system at time n to be the number of all possible different witnesses generated for value t in Step 2c.

Security requires that no PPT adversary can find a witness for values that are not added.

▶ **Definition 42** (Security of accumulators). *For any interactive PPT adversary $\mathcal{A}$, consider the following game $\mathsf{Sec}_{\mathcal{A}}(\kappa)$ between $\mathcal{A}$ and a challenger $\mathcal{C}$.*

1. ***Initialization.*** *The challenger $\mathcal{C}$ sets $\mathsf{pp} = \perp$, $\mathsf{aux} = \perp$, $\mathcal{V} = \emptyset$, generates $\mathsf{k} \leftarrow \mathsf{Gen}(1^\kappa)$ and sends k to the adversary $\mathcal{A}$.*
2. ***Addition.*** *Till $\mathcal{A}$ continues (which is at most $\mathsf{poly}(\kappa)$ steps), $\mathcal{A}$ sends a value $\mathsf{v} \notin \mathcal{V}$ to $\mathcal{C}$ and $\mathcal{C}$ sets $\mathsf{pp} \leftarrow \mathsf{Add}^{[\mathsf{aux}]}(\mathsf{k}, \mathsf{pp}, \mathsf{v})$ and adds v to $\mathcal{V}$.*
3. *The adversary $\mathcal{A}$ outputs $\mathsf{v} \notin \mathcal{V}$ and a witness w and wins the game if $\mathsf{Ver}(\mathsf{k}, \mathsf{pp}, \mathsf{v}, \mathsf{w}) = 1$.*

We call an accumulator scheme secure if $\Pr[\mathcal{A}$ wins in $\mathsf{Sec}_{\mathcal{A}}(\kappa)] = \mathsf{negl}(\kappa)$ for any PPT $\mathcal{A}$.

To construct accumulators, we first construct a special kind of merger called *hash tree merger* that has a hash function H as subroutine. The difference is that now the nodes in the trees also have a string as its value. Let x be a node. For simplicity, we assume that every node x is uniquely identified. We use $\mathsf{Val}(x)$ to denote its value. When merging a list of trees whose roots are $(\mathsf{rt}_1, \ldots, \mathsf{rt}_k)$, the root of the new tree has value $H(\mathsf{Val}(\mathsf{rt}_1), \ldots, \mathsf{Val}(\mathsf{rt}_k))$.

▶ **Definition 43** (Hash tree merger). *Let M be an online merger and H be a hash function. At every round, the* hash tree merger $\mathsf{HMer}(\mathsf{M}, H)$ *first uses M to merge trees. When a set of trees whose roots are $(\mathsf{rt}_1, \ldots, \mathsf{rt}_k)$ are being merged, they will first be ordered according to their unique identifiers. Without loss of generality, we can assume that $\mathsf{rt}_1 < \cdots < \mathsf{rt}_k$. Then, $\mathsf{HMer}(\mathsf{M}, H)$ assigns the root of the new tree the value $H(\mathsf{Val}(\mathsf{rt}_1), \ldots, \mathsf{Val}(\mathsf{rt}_k))$.*

Now, we give the construction of accumulators from collision resistant hash functions, where an online merger M is given as a subroutine. Looking ahead, the auxiliary information aux will be a list of hash trees where the accumulated values are stored in the leaves of the trees. The digest pp is simply the values of all the roots.

▶ **Construction 44.** *Let M be an online merger and $(\mathsf{Gen}, \mathsf{Hash})$ be a collision resistant hash function. Using them, we implement an accumulator according to Definition 40 as follows.*

- $\mathsf{Gen}(1^\kappa) \to \mathsf{k}$: *Sample and output a key $\mathsf{k} \leftarrow \mathsf{Gen}(1^\kappa)$ for the function $(\mathsf{Gen}, \mathsf{Hash})$. Initialize $\mathsf{HMer}(\mathsf{M}, H)$ such that $H(\mathsf{Val}(\mathsf{x}_1), \ldots, \mathsf{Val}(\mathsf{x}_t)) = 1||\mathsf{Hash}(\mathsf{k}, (\mathsf{Val}(\mathsf{x}_1), \ldots, \mathsf{Val}(\mathsf{x}_t)))$ for arbitrary $t \in \mathbb{N}$.*
- $\mathsf{Add}^{[\mathsf{aux}]}(\mathsf{k}, \mathsf{pp}, \mathsf{v}) \to \mathsf{pp}'$: *Parse aux as a list of trees. Add a new tree with a single node whose value is $0||\mathsf{v}$. Let HMer handle the merging. Output the values of the roots of the trees as pp'.*
- $\mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{v}) \to \mathsf{w}$: *First parse aux as a list of hash trees and locate one tree containing a leaf of value $0||v$. If no such tree exists, halt. Otherwise, let d be the depth of the tree and $(p_d, \ldots, p_0)$ be the path from p_d, which has value $0||\mathsf{v}$, to the root p_0. Let ℓ_i (resp. r_i) be*

the tuple of values of left (resp. right) siblings of p_i for $i \in [d]$. Note that when computing hashes, we first order the roots according to their identifiers. Thus, it is legitimate to talk about tuples of left siblings and right siblings. Output $\mathsf{w} = ((\ell_d, r_d), \ldots, (\ell_1, r_1))$.

- $\mathsf{Ver}(\mathsf{k}, \mathsf{pp}, \mathsf{v}, \mathsf{w}) \to \mathsf{b}$: *First parse* $\mathsf{pp} = \{\mathsf{rt}_1, \ldots, \mathsf{rt}_k\}$ *and* $\mathsf{w} = ((\ell_d, r_d), \ldots, (\ell_1, r_1))$. *Then, compute* $\mathsf{tmp}_{i-1} = 1||\mathsf{Hash}(\mathsf{k}, (\ell_i, \mathsf{tmp}_i, r_i))$ *for* $i = d, \ldots, 1$, *where* $\mathsf{tmp}_d = 0||\mathsf{v}$. *Output 1 if* $\mathsf{tmp}_0 \in \mathsf{pp}$ *and 0 otherwise.*

Completeness of Construction 44 is straightforward. We now bound the length of digest and number of updates.

▶ **Proposition 45** (Length of digest and number of updates of Construction 44)**.** *In Construction 44, after adding n inputs, the length of digest is bounded by $(\kappa+1) \cdot w_{[n]}$ and the number of update is bounded by* $\mathsf{dep}_{[n]}$.

Proof. The length of digest equals the number of trees times the length of the root, which is bounded by $(\kappa + 1) \cdot w_{[n]}$. An update for an accumulated value is required only when the tree it belongs get merged. Therefore, the number of update is bounded by the depth of the merger which is $\mathsf{dep}_{[n]}$. ◀

We now prove that Construction 44 satisfies the security requirement of Definition 42.

▶ **Proposition 46** (Security of Construction 44)**.** *Construction 44 is a secure accumulator according to Definition 42.*

Proof. It suffices to show that when $\mathcal{A}$ wins $\mathsf{Sec}_{\mathcal{A}}(\kappa)$, a collision for the underlying hash function is found. Let v be the value outputted by $\mathcal{A}$. Note that v can not be one of the trees which has only a single node. Otherwise, it means $\mathcal{A}$ has already added v. Therefore, $\mathcal{A}$ must have also outputted a witness $\mathsf{w} = ((\ell_{d'}, r_{d'}), \ldots, (\ell_1, r_1))$. Let $\mathsf{tmp}_{i-1} = 1||\mathsf{Hash}(\mathsf{k}, (\ell_i, \mathsf{tmp}_i, r_i))$ for $i = d', \ldots, 1$ where $\mathsf{tmp}_{d'} = 0||\mathsf{v}$. Since v is accepted, we know there must exist a tree $\mathcal{T}$ whose root $rt = \mathsf{tmp}_0$. Let d be the depth of $\mathcal{T}$. Let $\{\mathsf{tmp}_k, \ldots, \mathsf{tmp}_0\}$ be the longest sub-path of $\{\mathsf{tmp}_{d'}, \ldots, \mathsf{tmp}_0\}$ that exist in $\mathcal{T}$.

1. Suppose $k = d$ and $k = d'$. This contradicts the assumption that $\mathcal{A}$ wins the game as v must be one of the leaves.
2. Suppose $k = d$ and $k < d'$. Note that tmp_k begins with 1 while the values of all nodes of depth k in $\mathcal{T}$ begins with 0, which means we have found a collision.
3. Suppose $k < d$ and $k = d'$. Note that tmp_k begins with 0 while the values of all nodes of depth k in $\mathcal{T}$ begins with 1, which means we have found a collision.
4. Suppose $k < d$ and $k < d'$. In this case we have also found a collision because $\mathsf{tmp}_{k+1} \neq \mathsf{tmp}'_{k+1}$. ◀