Fingerprint Filters Are Optimal

William Kuszmaul* CMU
Jingxun Liang† CMU
Renfei Zhou‡ CMU

Abstract

Dynamic filters are data structures supporting approximate membership queries to a dynamic set S of n keys, allowing a small false-positive error rate ε, under insertions and deletions to the set S. Essentially all known constructions for dynamic filters use a technique known as *fingerprinting*. This technique, which was first introduced by Carter et al. in 1978, inherently requires

$$\log \binom{n\varepsilon^{-1}}{n} = n \log \varepsilon^{-1} + n \log e - o(n)$$

bits of space when $\varepsilon = o(1)$. Whether or not this bound is optimal for *all* dynamic filters (rather than just for fingerprint filters) has remained for decades as one of the central open questions in the area. We resolve this question by proving a sharp lower bound of $n \log \varepsilon^{-1} + n \log e - o(n)$ bits for $\varepsilon = o(1)$, regardless of operation time.

1 Introduction

Filters are space-efficient data structures designed for *approximate set membership queries*, allowing a small false-positive error rate. Specifically, given a key set $S \subset U$ of at most n keys, a filter with a false-positive error rate ε supports a `Query` operation for any $x \in U$ with the following guarantees:

- `Query`(x) returns `true` for each $x \in S$.
- `Query`(x) returns `false` with probability at least $1 - \varepsilon$ for each $x \notin S$.

In other words, queries for elements in S are exact (always returning `true`), while queries for elements outside S are approximate (most likely returning `false`, but sometimes producing false positives).

While similar to exact membership data structures like hash tables in functionality, filters require significantly less space, making them practical for various real-world applications (see, e.g., the surveys [BM04, AK21, TRL12, PNB18, LGM+19, PNK18, BE02, NP19, SGK+20]).

In this paper, we focus on ***dynamic filters***, which are filters that additionally support insertions and deletions to the key set S. In this setting, n serves as the *maximum capacity*, representing an upper bound on the number of elements in S at any given time.

The construction of dynamic filters in the literature has been dominated by a single paradigm: ***fingerprint filters***, introduced by [CFG+78]. In a fingerprint filter, each key is hashed to a random *fingerprint* in the set $\{1, 2, \ldots, n\varepsilon^{-1}\}$.[1] The data structure maintains a succinct hash table for all fingerprints and answers queries based on whether the fingerprint of the queried key is present in the hash table. This framework has been widely used in both theoretical dynamic filter works [CFG+78, PPS05, ANS10, BFK+22, BFG+18, LYY20, BE20] and in practical implementations such as quotient filters [Cle84, PPS05, BFJ+12, PCD+21, PBJP17, GFO18] and cuckoo filters [PR04, FAKM14, CLJW17, LGR+19]. The state of the art on the theory side is due to Bender et al. [BFK+22], who construct dynamic filters achieving $O(1)$-time operations using $n \log \varepsilon^{-1} + n \log e + o(n)$ bits of space[2], as long as $\log \varepsilon^{-1} \in \omega(1) \cap O(\log n / (\log \log \cdots \log n))$ (for any

*Partially supported by NSF grant CNS2504471. `kuszmaul@cmu.edu`.
†`jingxunl@andrew.cmu.edu`.
‡Partially supported by the Jane Street Graduate Research Fellowship and the MongoDB PhD Fellowship. `renfeiz@andrew.cmu.edu`.

[1] Slightly better bounds can be achieved by using a set of size very slightly less than $n\varepsilon^{-1}$ for the fingerprints. However, so long as $\varepsilon = o(1)$, this distinction only ends up affecting the final space usage by an additive $o(n)$ bits.

[2] The operator log denotes the binary logarithm in this paper.

constant number of logarithms). This result comes close to the theoretical limit for fingerprint filters, since fingerprint filters inherently require $\log \binom{n\varepsilon^{-1}}{n} = n \log \varepsilon^{-1} + n \log e - o(n)$ bits of space when $\varepsilon = o(1)$.

What is not known is whether fingerprint filters are *optimal*. Could a dynamic filter—implemented using a different paradigm—hope to break through the $n \log \varepsilon^{-1} + n \log e - o(n)$-bit barrier? This simple question has stood as one of the central open questions in the area for nearly five decades [CFG+78].

It has been known since the late 1970s [CFG+78] that even static filters must use at least $n \log \varepsilon^{-1}$ bits of space. The first separation between static and dynamic filters was given by Lovett and Porat [LP13], who established a lower bound of the form $n \log \varepsilon^{-1} + nf(\varepsilon)$ for some positive function f.[3] For $\varepsilon = o(1)$, Kuszmaul and Walzer [KW24] established a stronger lower bound of

$$n \log \varepsilon^{-1} + \frac{n}{2} \log \frac{4}{\sqrt{17} - 1} - o(n) \approx n \log \varepsilon^{-1} + 0.35n$$

bits. This leaves a gap of approximately $n \log e - n \cdot (\log \frac{4}{\sqrt{17}-1})/2 \approx 1.1n$ bits between the best known lower bound [KW24] and the best bound achievable by fingerprint filters.

In this paper, we prove a sharp lower bound of $n \log \varepsilon^{-1} + n \log e - o(n)$ bits for $\varepsilon = o(1)$. The lower bound is information-theoretic, applying regardless of operation time.

Theorem 1.1. *Suppose U is a set and n, ε are parameters such that $\varepsilon = o(1)$ and $|U| = \omega(n\varepsilon^{-1})$. Any algorithm that implements a dynamic filter with universe U, capacity n, and false-positive error rate ε, and that can support a sequence of $\omega(n)$ insertions and deletions, must use space at least $n \log \varepsilon^{-1} + n \log e - o(n)$ bits.*

We remark that this theorem is not just of theoretical interest. Practical filters often use an error rate ε of $1/256$ [LMSS21], which means $\log \varepsilon^{-1} = 8$. Under these conditions, the $n \log e - o(n) \approx 1.44n$ term in Theorem 1.1 accounts for 15.2% of the overall space usage.[4] Our result demonstrates that this type of overhead is fundamentally unavoidable in both theory and practice.

The proof of Theorem 1.1 is presented in three parts. We begin in Section 3 by proving a special case of Theorem 1.1 under two simplifying assumptions: history independence and monotonicity. This special case serves to illustrate our proof framework, in which we construct a one-way communication protocol that is able to extract $n \log \varepsilon^{-1} + n \log e - o(n)$ bits from a dynamic filter. Then, in Sections 4 and 5, we develop techniques to remove these assumptions, first handling history independence and then monotonicity, respectively. The final results apply to any dynamic filter.

2 Preliminaries

In this section, we formally define the problem that a dynamic filter must solve. Since Theorem 1.1 focuses on space complexity and does not consider running time, we adopt a simple computational model: the data structure uses a fixed-length memory that can be accessed arbitrarily. For randomized data structures, we additionally assume access to an infinite-length read-only tape of independent random bits, which does not contribute to the space usage.[5]

Let $n \in \mathbb{N}$, $\varepsilon \in (0, 1)$, and $U = \{1, 2, \ldots, u\}$ be parameters representing the maximum capacity, the false-positive error rate, and the universe, respectively, for the filter. A dynamic filter with respect to these parameters is defined as follows.

Definition 2.1 (Dynamic filters)**.** A dynamic filter is a randomized data structure $\mathcal{D}$ that maintains approximate membership for a set $S \subset U$ of at most n elements, supporting the following operations:

[3] Notably, their lower bound holds even for *incremental filters* (i.e., filters that only support insertions). It was subsequently shown by [KW24] that this type of lower bound is the best one can hope for in the incremental setting, as there exist incremental filters using $n \log \varepsilon^{-1} + o(n)$ bits when $\varepsilon = o(1)$.

[4] In practice, it is difficult to use a fractional number of bits per key, so real-world implementations tend to incur an overhead of at least $2n$ bits, which accounts for approximately 20% of the overall space.

[5] Some models assume oracle access to a random number generator that outputs independent random bits on demand. This can be simulated using our random tape model by maintaining a pointer to track the last revealed random bit, allowing us to reveal new random bits as needed. The pointer only occupies $o(n)$ bits of space when there are at most $2^{o(n)}$ calls to the random number generator in total, which is negligible compared to the main space usage of the filter.

- `Initialize()`: Creates and returns an initial empty filter state.
- `Query(`x`)`: For any key $x \in S$, returns `true` deterministically. For any key $x \notin S$, returns `false` with probability at least $1 - \varepsilon$, where the probability is taken over the randomness of $\mathcal{D}$ (the key x is not assumed to be random).
- `Insert(`x`)`: Adds x to S (i.e., updates S to $S \cup \{x\}$). This operation assumes $x \notin S$ before insertion.
- `Delete(`x`)`: Removes x from S (i.e., updates S to $S \setminus \{x\}$). This operation assumes $x \in S$ before deletion.

We emphasize that, while insertions and deletions update the set S, the filter must perform these operations using only the information stored in $\mathcal{D}$, the input key x, and the random tape—the filter does not have explicit knowledge of the current set S.

3 Warmup: Lower Bound Under Two Assumptions

In the remainder of this paper, we will prove Theorem 1.1.

Theorem 1.1 (Restated)**.** *Suppose U is a set and n, ε are parameters such that $\varepsilon = o(1)$ and $|U| = \omega(n\varepsilon^{-1})$. Any algorithm that implements a dynamic filter with universe U, capacity n, and false-positive error rate ε, and that can support a sequence of $\omega(n)$ insertions and deletions, must use space at least $n \log \varepsilon^{-1} + n \log e - o(n)$ bits.*

Fix a filter algorithm $\mathcal{D}$ that uses H_{filter} bits of space. All of the filters in the remainder of this paper will be implemented using $\mathcal{D}$ with the same random tape. The goal of the proof is to show that $H_{\text{filter}} \geq n \log \varepsilon^{-1} + n \log e - o(n)$.

In this section, as a warmup, we first prove the statement for special $\mathcal{D}$ with the following two assumptions:

- **History independence:** The memory state of a filter F is uniquely determined by its true key set, along with the random tape used in $\mathcal{D}$. (Therefore, so is the accepted set $\overline{F}$ of the filter F.) Equivalently, any two operational histories that result in the same true key set must also produce the same filter state.
- **Monotonicity (for history-independent filters):** For any two filters F and G with true key sets S and T, respectively, if $T \subseteq S$, then their accepted key sets satisfy $\overline{G} \subseteq \overline{F}$.

Formally, in this section, we prove the following proposition:

Proposition 3.1. *Theorem 1.1 holds for filters that satisfy both history independence and monotonicity.*

3.1 Proof Intuition for Proposition 3.1

To build intuition, we begin by describing a standard argument that one can use to obtain a weaker lower bound, namely that $H_{\text{filter}} \geq n \log \varepsilon^{-1} - o(n)$. Assume to the contrary that there exists a filter algorithm $\mathcal{D}$ with $H_{\text{filter}} < n \log \varepsilon^{-1} - \Omega(n)$. Such a filter would enable an unrealistically efficient one-way communication protocol for Alice to send a sequence of distinct keys $x_1, \ldots, x_n$ (where each $x_i \in U$) to Bob.

On the one hand, from an information-theoretic perspective, Alice must send at least $\log(|U|^{\underline{n}}) \approx n \log |U|$ bits[6], which corresponds to roughly $\log |U|$ bits per key.

On the other hand, if Alice first sends a filter F with a true key set $\{x_1, \ldots, x_n\}$ and then sends the sequence $(x_1, \ldots, x_n)$ as keys from the accepted key set $\overline{F}$ (which is of size roughly $\varepsilon|U|$), she only needs to send approximately $\log(\varepsilon|U|)$ bits per key—saving $\log \varepsilon^{-1}$ bits per key—along with a filter F of at most $H_{\text{filter}} < n \log \varepsilon^{-1} - \Omega(n)$ bits. This means that Alice sends a total of $n \log(\varepsilon|U|) + n \log \varepsilon^{-1} - \Omega(n) = n \log |U| - \Omega(n) = \log |U|^{\underline{n}} - \Omega(n)$ bits to indirectly transmit the sequence $(x_1, \ldots, x_n)$ using the filter algorithm $\mathcal{D}$, which leads to a contradiction, proving the bound.

[6]The notation $m^{\underline{n}}$ represents the falling factorial of m of order n, i.e., $m(m-1)\cdots(m-n+1)$.

Next we explain, at a high level, how to extend this approach to obtain our desired lower bound of $H_{\text{filter}} \geq n \log \varepsilon^{-1} + n \log e + o(n)$. The key observation is that, in the previous protocol of sending $(x_1, \ldots, x_n)$ indirectly, Alice can achieve an even greater reduction in communication cost by using a *dynamic* filter. For instance, after Alice sends x_1 using $\log \left|\overline{F}\right| \approx \log(\varepsilon|U|)$ bits, Bob can delete x_1 from F to obtain F'. Alice then only needs to send the remaining keys as elements from $\overline{F'}$. Since F' has a smaller true key set than F and we assume $\mathcal{D}$ is monotonic, the universe of the remaining keys is reduced by replacing $\overline{F}$ by $\overline{F'}$, leading to an additional reduction in communication cost for Alice. Similarly, after Alice sends x_2, Bob can delete x_2 from F', further reducing the universe for the remaining keys. More generally, when Alice sends x_k, Bob has already deleted $x_1, \ldots, x_{k-1}$ from F, resulting in a filter with only $n - k + 1$ keys in its true key set.

Ideally, if we assume the size of the accepted set of a filter scales linearly with the size of the true key set—meaning that a filter with a true key set of size $(n-k+1)$ has roughly $\frac{n-k+1}{n}\varepsilon|U|$ accepted keys—then Alice saves an additionally $\log \frac{n}{n-k+1}$ bits when sending x_k, which is

$$\sum_{k=1}^{n} \log \frac{n}{n-k+1} = \log \frac{n^n}{n!} \approx n \log e$$

bits in total, improving the previous lower bound to the desired $H_{\text{filter}} \geq n \log \varepsilon^{-1} + n \log e + o(n)$.

However, in a general filter, the accepted set size may not scale linearly with the true key set size. In particular, the accepted set size might always be very close to $\varepsilon|U|$, meaning that we barely save any communication cost by deleting keys from F. In this case, Alice can reduce the message size following another strategy: After Alice sends x_1, Bob can build another filter G with true key set $\{x_1\}$. Then, as all the remaining keys are not in the true key set of G, they are unlikely to be in $\overline{G}$, and Alice can send them as elements in $\overline{F} \setminus \overline{G}$ with probability at least $1 - \varepsilon$. This means that in the extreme case where the accepted set size is always almost $\varepsilon|U|$, the universe of all the remaining keys is significantly reduced by replacing $\overline{F}$ with $\overline{F} \setminus \overline{G}$ and so is the communication cost. Similarly, after Alice sends x_2, Bob can insert x_2 to G and gain further universe reduction.

The filter lower bound in this section cleverly combines the previous two ideas of reducing the universe. Initially, Bob has two filters in his hand, one with the true key set $\{x_1, \ldots, x_n\}$, called F, and the other with an empty true key set, called G. During the entire protocol, all the keys that are not sent are in the true key set of F, but not in the true key set of G. Whenever Alice sends a key x, she sends x conditioned on the two filters in Bob's hand (i.e., viewing x as an element of $\overline{F} \setminus \overline{G}$). Then, Bob either deletes x from F, or inserts x to G. For the convenience of notation, we slightly adjust the order in which Alice sends the keys: In each round, she sends either the smallest unsent key (when Bob is going to insert that key to G) or the largest unsent key (when Bob is going to delete that key from F) to guarantee that the true key sets of both F and G are prefixes of $\{x_1, \ldots, x_n\}$.

In fact, as we will see, the final protocol for Alice actually takes the following simple form: Alice first sends the smallest keys for some number of rounds and then switches to sending the largest keys in the remaining rounds. The main technical challenge in the full proof is to show that, no matter how the accepted set size scales with the true key set size, there always exists such a protocol in which Alice can reduce her overall communication cost by an additional $n \log e - o(n)$ bits when compared to the standard protocol.

3.2 Formal Proof of Proposition 3.1

Let $(x_1, \ldots, x_n)$ be a sequence of distinct keys from U which is sampled uniformly at random. Consider a one-way communication game where Alice wants to transmit $(x_1, \ldots, x_n)$ to Bob. Suppose both Alice and Bob have free access to the random tape of $\mathcal{D}$. Note that, no matter how Alice sends her message, she must communicate at least $\log |U|^{\underline{n}}$ bits of entropy.

Claim 3.2. *In any one-way communication protocol where Alice transmits $(x_1, \ldots, x_n)$ to Bob, the entropy of the message sent by Alice is at least $\log |U|^{\underline{n}}$.*

Proof. As Bob can decode $(x_1, \ldots, x_n)$ from the message, the entropy of the message is at least

$$H(x_1, \ldots, x_n \mid \text{random tape of } \mathcal{D}) = H(x_1, \ldots, x_n) = \log |U|^{\underline{n}}. \qquad \square$$

We now construct a protocol where Alice transmits $(x_1, \ldots, x_n)$ indirectly using a filter algorithm. Let $s \in [0, n]$ be an integer parameter to be determined, which will depend only on the filter algorithm $\mathcal{D}$ but not on the keys $(x_1, \ldots, x_n)$. The protocol P_s (which is parameterized by s) is defined in Algorithm 1. We will also explain Algorithm 1 in detail below.

Algorithm 1: Protocol P_s

Subroutine `Send`(x, F, G)**:** // x is in F's true key set, but not in G's.
  $Z \longleftarrow$ indicator for whether $x \in \overline{G}$
  Alice sends Z to Bob
  if $Z = 0$ *(i.e.,* $x \notin \overline{G}$*)* **then**
    Alice sends x as an element in $\overline{F} \setminus \overline{G}$
  else
    Alice sends x as an element in U
$F \longleftarrow$ filter with true key set $\{x_1, \ldots, x_n\}$
$G \longleftarrow$ filter with true key set $\varnothing$
Alice prepares F and sends it to Bob
Bob prepares G by himself
for k *from* 1 *to* s **do**
  Alice sends x_k using `Send`(x_k, F, G)
  Bob inserts x_k to G
for k *from* n *to* $s + 1$ **do**
  Alice sends x_k using `Send`(x_k, F, G)
  Bob deletes x_k from F

Initialization. At the beginning of the protocol P_s, Alice first sends a filter F with the true key set $\{x_1, \ldots, x_n\}$ to Bob, and Bob prepares a filter G with an empty true key set. This step takes H_{filter} bits of communication.

Sending a single key. In the main part of the protocol, Alice sends each key x to Bob using the protocol `Send`(x, F, G), where F and G are two filters held by Bob such that x is in the true key set of F but not in the true key set of G.

In the protocol `Send`(x, F, G), Alice first sends a bit Z to indicate whether x is in the accepted set of G. As x is not in the true key set of G, Z is a highly biased bit, with probability at most ε that Z equals 1. Hence, the entropy of Z is small:

$$H(Z) \le h(\varepsilon) \coloneqq -\varepsilon \log \varepsilon - (1 - \varepsilon) \log(1 - \varepsilon),$$

where h denotes the binary entropy function.

Then, depending on the different values of Z, Alice sends x differently. In the common case where $x \notin \overline{G}$, x must lie in $\overline{F} \setminus \overline{G}$ (since x is in the true key set of F, it is also in the accepted set $\overline{F}$). Alice can then send x as an element of $\overline{F} \setminus \overline{G}$ using only $\log \left|\overline{F} \setminus \overline{G}\right|$ bits. In the rare case that $x \in \overline{G}$, Alice sends x directly as an element in U using $\log |U|$ bits. The message length for this step can be written as

$$Z \log |U| + (1 - Z) \log \left|\overline{F} \setminus \overline{G}\right|,$$

where both Z and $\left|\overline{F} \setminus \overline{G}\right|$ are random variables.

After Alice sends the key x using `Send`(x, F, G), Bob will either insert x to G or delete x from F to reduce the communication cost for future keys.

Order of key sendings. The protocol P_s has two stages. In the first stage, Alice always sends the smallest unsent key in each round (i.e., in the order of $x_1, x_2, \ldots$), and after receiving it, Bob inserts the key into G.

Once Alice has sent $x_1, \ldots, x_s$, they switch to the second stage where Alice instead sends the largest unsent key in each round (i.e., in the order of $x_n, x_{n-1}, \ldots$) and Bob deletes the key from F.

Throughout the protocol, the true key sets of both F and G are prefixes of $\{x_1, \ldots, x_n\}$. Specifically, for any $k \le n$, let F_k denote the filter with the true key set $\{x_1, \ldots, x_k\}$. Since we assume $\mathcal{D}$ is history independent, the true key set uniquely determines the filter. Then, when Alice sends x_k, the two filters F and G held by Bob can be represented as F_{r_k} and F_{ℓ_k} with $\ell_k < k \le r_k$, where

$$\ell_k \coloneqq \begin{cases} k-1 & \text{if } k \le s \\ s & \text{if } s+1 \le k \le n \end{cases} \quad \text{and} \quad r_k \coloneqq \begin{cases} n & \text{if } k \le s \\ k & \text{if } s+1 \le k \le n \end{cases}.$$

Plugging this into the previous calculation, when Alice sends x_k using $\texttt{Send}(x_k, F_{r_k}, F_{\ell_k})$, she first sends an indicator $Z_k \coloneqq \mathbb{1}[x_k \in \overline{F_{\ell_k}}]$, and then sends $Z_k \log|U| + (1-Z_k)\log\left|\overline{F_{r_k}} \setminus \overline{F_{\ell_k}}\right|$ bits. The total entropy of the message for sending x_k is at most

$$h(\varepsilon) + \mathbb{E}\left[Z_k \log|U| + (1-Z_k)\log|\overline{F_{r_k}} \setminus \overline{F_{\ell_k}}|\right],$$

where the expectation is taken over both the randomness of $\mathcal{D}$ and the randomness of the keys $(x_1, \ldots, x_n)$.

Upper bounding the total message entropy. Summing up the message entropy for sending the filter F_n and the message entropy for sending each x_k, the total message entropy of P_s is at most

$$H_{\text{filter}} + nh(\varepsilon) + \sum_{k=1}^{n} \mathbb{E}\left[Z_k \log|U| + (1-Z_k)\log|\overline{F_{r_k}} \setminus \overline{F_{\ell_k}}|\right]. \tag{1}$$

Below, we further simplify (1).

Define a_k for $k = 0, 1, \ldots, n$ as

$$a_k \coloneqq \frac{1}{(1-\varepsilon)n + \varepsilon|U|}\,\mathbb{E}\left[\left|\overline{F_k}\right| - \left|\overline{F_{k-1}}\right|\right],$$

where we adopt the convention that $|\overline{F_{-1}}| = 0$. Since we assume $\mathcal{D}$ is monotonic, the accepted sets satisfy $\overline{F_0} \subseteq \overline{F_1} \subseteq \cdots \subseteq \overline{F_n}$, so each a_k is non-negative. Note that a_k is not a random variable and depends only on the algorithm $\mathcal{D}$ since we already take expectation over all the randomness (including the randomness of $\mathcal{D}$ and the randomness of the keys $(x_1, \ldots, x_n)$) in the definition of a_k. Below, we write (1) in terms of a_k's.

First, by the definition of a_k, the expected size of the accepted sets can be written as

$$\mathbb{E}\left[\left|\overline{F_k}\right|\right] = ((1-\varepsilon)n + \varepsilon|U|)\sum_{i=0}^{k} a_i = ((1-\varepsilon)n + \varepsilon|U|)\cdot a_{[0,k]}.$$

Here, for clarity of notation, we introduce two abbreviations for partial sums of a_k: For any $\ell \le r$, we use $a_{[\ell,r]}$ to denote $\sum_{i=\ell}^{r} a_i$ and we use $a_{(\ell,r]}$ to denote $\sum_{i=\ell+1}^{r} a_i$. Moreover, since a key outside the true key set of a filter falls into the accepted set with probability at most ε, we obtain $\mathbb{E}\left[\left|\overline{F_n}\right|\right] \le n + \varepsilon(|U| - n) = (1-\varepsilon)n + \varepsilon|U|$, which means $a_{[0,n]} = a_0 + \cdots + a_n \le 1$.

Then, each term in the summation of (1) can be upper bounded in terms of a_k's by the following claim:

Claim 3.3. *For any $k \le n$,*

$$\mathbb{E}\left[Z_k \log|U| + (1-Z_k)\log|\overline{F_{r_k}} \setminus \overline{F_{\ell_k}}|\right] \le \log|U| + (1-\varepsilon)\log a_{(\ell_k, r_k]} + \log\varepsilon + o(1).$$

Proof. Recall that $Z_k \coloneqq \mathbb{1}[x_k \in \overline{F_{\ell_k}}]$ and $p \coloneqq \Pr[Z_k = 1] \le \varepsilon$. By the principle of total probability,

$$\begin{aligned} &\mathbb{E}\left[Z_k \log|U| + (1-Z_k)\log|\overline{F_{r_k}} \setminus \overline{F_{\ell_k}}|\right] \\ &= p\log|U| + (1-p)\,\mathbb{E}\left[\log\left(\left|\overline{F_{r_k}}\right| - \left|\overline{F_{\ell_k}}\right|\right) \mid Z_k = 0\right] \\ &\le p\log|U| + (1-p)\log\left(\mathbb{E}\left[\left|\overline{F_{r_k}}\right| - \left|\overline{F_{\ell_k}}\right| \mid Z_k = 0\right]\right), \end{aligned} \tag{2}$$

where the second inequality is because the function log is concave. Moreover,

$$\begin{aligned}
&\mathbb{E}\big[|\overline{F_{r_k}}| - \big|\overline{F_{\ell_k}}\big| \mid Z_k = 0\big] \\
&= \sum_{i=0}^{\infty} i \cdot \Pr\big[|\overline{F_{r_k}}| - |\overline{F_{\ell_k}}| = i \mid Z_k = 0\big] \\
&\le \sum_{i=0}^{\infty} i \cdot \frac{\Pr\big[|\overline{F_{r_k}}| - |\overline{F_{\ell_k}}| = i\big]}{\Pr[Z_k = 0]} \\
&= \frac{\mathbb{E}\big[|\overline{F_{r_k}}| - |\overline{F_{\ell_k}}|\big]}{\Pr[Z_k = 0]} \le \frac{(1-\varepsilon)n + \varepsilon|U|}{1-\varepsilon} \cdot a_{(\ell_k, r_k]},
\end{aligned} \tag{3}$$

and it is clear that $\mathbb{E}\big[|\overline{F_{r_k}}| - \big|\overline{F_{\ell_k}}\big| \mid Z_k = 0\big] \le |U|$, which implies that (2) is increasing in p. Therefore, by plugging (3) and $p \le \varepsilon$ into (2), we obtain the desired bound

$$\begin{aligned}
&\mathbb{E}\big[Z_k \log|U| + (1-Z_k)\log|\overline{F_{r_k}} \setminus \overline{F_{\ell_k}}|\big] \\
&\le p\log|U| + (1-p)\log\big(\mathbb{E}\big[|\overline{F_{r_k}}| - \big|\overline{F_{\ell_k}}\big| \mid Z_k = 0\big]\big) \\
&\le \varepsilon\log|U| + (1-\varepsilon)\log\left(\frac{(1-\varepsilon)n + \varepsilon|U|}{1-\varepsilon} \cdot a_{(\ell_k, r_k]}\right) \\
&\le \varepsilon\log|U| + (1-\varepsilon)\log\big(\varepsilon|U| \cdot a_{(\ell_k, r_k]}\big) + o(1) \\
&\le \log|U| + (1-\varepsilon)\log a_{(\ell_k, r_k]} + \log\varepsilon + o(1),
\end{aligned}$$

where the third and fourth inequalities are because $\varepsilon = o(1)$ and $|U| = \omega(n\varepsilon^{-1})$, which imply $\log(1-\varepsilon) = o(1)$, $\log\left(\frac{(1-\varepsilon)n+\varepsilon|U|}{\varepsilon|U|}\right) = o(1)$, and $\varepsilon\log\varepsilon = o(1)$. □

Now, plugging Claim 3.3 into (1), the total message entropy of P_s is at most

$$\begin{aligned}
&H_{\text{filter}} + nh(\varepsilon) + \sum_{k=1}^{n}\big(\log|U| + (1-\varepsilon)\log a_{(\ell_k, r_k]} + \log\varepsilon + o(1)\big) \\
&\le H_{\text{filter}} + n\log|U| + n\log\varepsilon + (1-\varepsilon)\sum_{k=1}^{n}\log a_{(\ell_k, r_k]} + o(n) \\
&= H_{\text{filter}} + n\log|U| + n\log\varepsilon + (1-\varepsilon)\left(\sum_{k=1}^{s}\log a_{[k,n]} + \sum_{k=s+1}^{n}\log a_{(s,k]}\right) + o(n),
\end{aligned} \tag{4}$$

[7]

where we use $h(\varepsilon) = o(1)$ in the first inequality.

Choice of s. Now, comparing (4) with the message entropy lower bound in Claim 3.2, we obtain

$$\begin{aligned}
H_{\text{filter}} &\ge \log|U|^{\underline{n}} - n\log|U| - n\log\varepsilon - (1-\varepsilon)\left(\sum_{k=1}^{s}\log a_{[k,n]} + \sum_{k=s+1}^{n}\log a_{(s,k]}\right) - o(n) \\
&\ge n\log\varepsilon^{-1} - (1-\varepsilon)\left(\sum_{k=1}^{s}\log a_{[k,n]} + \sum_{k=s+1}^{n}\log a_{(s,k]}\right) - o(n),
\end{aligned} \tag{5}$$

where we use $\log|U|^{\underline{n}} - n\log|U| \ge n\log\left(1 - \frac{n}{|U|}\right) = -o(n)$ as $|U| = \omega(n)$.

Thus, to establish the desired lower bound for H_{filter}, it suffices to show that there exists a choice of s for which

$$\sum_{k=1}^{s}\log a_{[k,n]} + \sum_{k=s+1}^{n}\log a_{(s,k]} \le -n\log e + o(n),$$

[7] When $s = 0$, the sum $\sum_{k=1}^{s}\log a_{[k,n]}$ is empty and vanishes; when $s = n$, the sum $\sum_{k=s+1}^{n}\log a_{(s,k]}$ is empty and vanishes.

or equivalently that

$$\sum_{k=1}^{s} \log \frac{1}{a_{[k,n]}} + \sum_{k=s+1}^{n} \log \frac{1}{a_{(s,k]}} \geq n \log e - o(n).$$

The next lemma establishes that such an s always exists, and is the main technical lemma of the section.

Lemma 3.4. *Let $a_1, \ldots, a_n \geq 0$ be real numbers such that $a_1 + \cdots + a_n \leq 1$. Then, there exists an s with $0 \leq s \leq n$, such that*

$$\sum_{k=1}^{s} \log a_{[k,n]} + \sum_{k=s+1}^{n} \log a_{(s,k]} \leq -n \log e + o(n). \tag{6}$$

Using Lemma 3.4, we can choose a suitable s such that (5) becomes

$$H_{\text{filter}} \geq n \log \varepsilon^{-1} + (1-\varepsilon) n \log e - o(n) \geq n \log \varepsilon^{-1} + n \log e - o(n).$$

Furthermore, the protocol P_s requires only $O(n)$ insertions and deletions to maintain the filters F and G. Therefore, assuming Lemma 3.4, we have completed the proof of Proposition 3.1.

3.3 Proof of Lemma 3.4

In this subsection, we prove Lemma 3.4. First, by the monotonicity of the logarithm function, we may assume without loss of generality that $a_1 + \cdots + a_n = 1$. Moreover, instead of directly finding an s that satisfies (6), we will inductively establish the existence of an s that meets a more precise constraint:

$$\sum_{k=1}^{s} \log a_{[k,n]} + \sum_{k=s+1}^{n} \log a_{(s,k]} \leq \log \frac{n!}{n^n}. \tag{7}$$

By Stirling's approximation, (7) implies (6), since $\log \frac{n!}{n^n} \leq \log \frac{(n/e)^n}{n^n} + O(\log n) = -n \log e + o(n)$.

Proof of Lemma 3.4. We use induction on n to prove the existence of s satisfying (7).

For the base case $n = 1$, we can choose either $s = 0$ or 1, and (7) simplifies to $\log a_1 \leq 0$, which follows from the condition $a_1 = 1$. For the rest of the proof, consider some $n > 1$, and suppose that the statement holds for all integers smaller than n.

First, we check whether $s = 0$ already satisfies (7). If

$$\sum_{k=1}^{n} \log a_{(0,k]} \leq \log \frac{n!}{n^n}, \tag{8}$$

then we can directly take $s = 0$ to conclude the statement. Below, we assume (8) does not hold.

Now, let j be the smallest index such that

$$\sum_{k=1}^{j} \log a_{(0,k]} > \sum_{k=1}^{j} \log \frac{k}{n}. \tag{9}$$

The existence of such j is ensured by the assumption that (8) does not hold, implying that $j = n$ is always a valid candidate. By the definition of j, we have

$$\sum_{k=1}^{i} \log a_{(0,k]} \leq \sum_{k=1}^{i} \log \frac{k}{n}, \quad \forall i < j. \tag{10}$$

In particular, by comparing (9) and (10) (taking $i = j - 1$), we get that $\log a_{(0,j]} > \log(j/n)$, which implies that

$$a_1 + \cdots + a_j = a_{(0,j]} > j/n \tag{11}$$

We will choose s from the range $j \le s \le n$ by using the inductive hypothesis on a normalized suffix

$$\left(\frac{a_{j+1}}{a_{j+1}+\cdots+a_n}, \frac{a_{j+2}}{a_{j+1}+\cdots+a_n}, \ldots, \frac{a_n}{a_{j+1}+\cdots+a_n}\right).$$

By induction hypothesis, there exists s with $j \le s \le n$, such that

$$\sum_{k=j+1}^{s} \log a_{[k,n]} + \sum_{k=s+1}^{n} \log a_{(s,k]} \le \log \frac{(n-j)!}{(n-j)^{n-j}} + (n-j)\log a_{(j,n]}. \tag{12}$$

Below, we show that such an s also satisfies (7).

Using (12), the left-hand side of (7) can be upper bounded by

$$\begin{aligned}\sum_{k=1}^{s} \log a_{[k,n]} + \sum_{k=s+1}^{n} \log a_{(s,k]} &\le \sum_{k=1}^{j} \log a_{[k,n]} + \log \frac{(n-j)!}{(n-j)^{n-j}} + (n-j)\log a_{(j,n]} \\ &= \sum_{k=2}^{j+1} \log a_{[k,n]} + \log \frac{(n-j)!}{(n-j)^{n-j}} + (n-j-1)\log a_{(j,n]}, \end{aligned} \tag{13}$$

where in the second line we use $\log a_{[1,n]} = 0$ and $\log a_{[j+1,n]} = \log a_{(j,n]}$. The main step in the rest of the proof will be to establish an upper bound on $\sum_{k=2}^{j+1} \log a_{[k,n]}$ using (9) and (10).

Let $f : \mathbb{R}_+ \to \mathbb{R}$ denote the function $f(x) = \log(1 - 2^{-x})$. Then, one can observe that

$$\sum_{k=2}^{j+1} \log a_{[k,n]} = \sum_{k=1}^{j} \log a_{(k,n]} = \sum_{k=1}^{j} \log\big(1 - a_{(0,k]}\big) = \sum_{k=1}^{j} f\big(-\log a_{(0,k]}\big). \tag{14}$$

The right-hand side of (14) can be upper bounded using a variation of Karamata's inequality [Kar32]:

Theorem 3.5 (Karamata's inequality [Kar32])**.** *Let $x_1 \ge x_2 \ge \cdots \ge x_n$ and $y_1 \ge y_2 \ge \cdots \ge y_n$ be two sequences of real numbers, satisfying*

- *$x_1 + \cdots + x_i \ge y_1 + \cdots + y_i$, for all $i < n$, and*
- *$x_1 + \cdots + x_n \le y_1 + \cdots + y_n$.*

Then, for any concave and increasing function $f : \mathbb{R} \to \mathbb{R}$, we have

$$f(x_1) + \cdots + f(x_n) \le f(y_1) + \cdots + f(y_n).$$

For an overview of Karamata's inequality, and other related inequalities, see also [Kad05].

We will use Karamata's inequality over the function f and two sequences $\big(-\log a_{(0,1]}, -\log a_{(0,2]}, \ldots, -\log a_{(0,j]}\big)$ and $(-\log(1/n), -\log(2/n), \ldots, -\log(j/n))$. Clearly, both sequences are non-increasing. Moreover, the premise of Karamata's inequality is satisfied by these two sequences according to (9) and (10). We can also check f is concave and increasing as follows.

Claim 3.6. *The function $f : \mathbb{R}_+ \to \mathbb{R}$ defined as $f(x) = \log(1 - 2^{-x})$ is concave and increasing.*

Proof. $f'(x) = \frac{2^{-x}}{1-2^{-x}} > 0$ for all $x > 0$, and $f''(x) = \frac{-2^x \ln 2}{(2^x-1)^2} < 0$ for all $x > 0$. □

Hence, we can use Karamata's inequality over the function f and the two sequences to obtain

$$\sum_{k=1}^{j} f\big(-\log a_{(0,k]}\big) \le \sum_{k=1}^{j} f\left(-\log \frac{k}{n}\right) = \sum_{k=1}^{j} \log \frac{n-k}{n} = \sum_{k=n-j}^{n-1} \log \frac{k}{n}. \tag{15}$$

Finally, by plugging (14) and (15) into (13), we obtain

$$\begin{aligned}
&\sum_{k=1}^{s} \log a_{[k,n]} + \sum_{k=s+1}^{n} \log a_{(s,k]} \\
&\leq \sum_{k=2}^{j+1} \log a_{[k,n]} + \log \frac{(n-j)!}{(n-j)^{n-j}} + (n-j-1) \log a_{(j,n]} && \text{(by (13))} \\
&\leq \sum_{k=n-j}^{n-1} \log \frac{k}{n} + \log \frac{(n-j)!}{(n-j)^{n-j}} + (n-j-1) \log a_{(j,n]} && \text{(by (14) and (15))} \\
&\leq \sum_{k=n-j}^{n-1} \log \frac{k}{n} + \log \frac{(n-j)!}{(n-j)^{n-j}} + (n-j-1) \log \frac{n-j}{n},
\end{aligned}$$

where in the final step uses that $a_{(j,n]} = 1 - a_{(0,j]} \leq 1 - j/n$ by (11). Continuing, the bound simplifies to

$$\sum_{k=n-j}^{n-1} \log \frac{k}{n} + \sum_{k=1}^{n-j-1} \left(\log \frac{k}{n-j} + \log \frac{n-j}{n} \right) = \sum_{k=1}^{n} \log \frac{k}{n} = \log \frac{n!}{n^n},$$

as desired. This completes the induction step and proves Lemma 3.4. □

4 Beyond History Independence

In this section, we extend the lower bound in Section 3 to remove the assumption of history independence. For a history-dependent filter, the memory state is not purely determined by the true key set and the random seeds, but also depends on the history of operations used to obtain this filter.

Note that we still assume the data structure $\mathcal{D}$ is monotone, with a generalized definition of monotonicity defined as follows (the previous monotonicity in Section 3 is for history-independent filters):

Definition 4.1 (Self-contained operational sequence)**.** We say an operational sequence σ consisting of a series of insertions and deletions is ***self-contained*** if it satisfies the following constraints:

- (No deletions of non-elements.) Each deleted key in σ must have been inserted earlier *in the same sequence* σ and has not been deleted in between.
- (No duplicate insertions.) For any key that appears multiple times in σ, between any two insertions of that key, there must be a deletion of that key.

Equivalently, a self-contained operational sequence is a valid sequence of operations that can be performed on an empty filter.

Clearly, the true key set of a filter is non-decreasing after we apply a self-contained operational sequence over the filter.

Definition 4.2 (Monotonicity for general filters)**.** We say a filter algorithm $\mathcal{D}$ is ***monotone*** if the following is true: For any $0 < j < k$, and for any sequence $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(k))$ of operations starting with an empty filter, and where the suffix $\sigma(j+1), \ldots, \sigma(k)$ is self-contained, the filter G obtained after operation $\sigma(j)$ and the filter F obtained after operation $\sigma(k)$ are guaranteed to satisfy $\overline{G} \subseteq \overline{F}$.

Formally, in this section, we prove the following proposition:

Proposition 4.3. *Theorem 1.1 holds for monotone filters.*

4.1 Proof Intuition for Proposition 4.3

We begin by analyzing the proof of Proposition 3.1 to identify precisely where the assumptions of history independence and monotonicity are used.

In the proof of Proposition 3.1, Bob initially has two filters F and G with true key sets $\{x_1, \ldots, x_n\}$ and $\varnothing$, respectively. Whenever Alice sends a key, Bob either deletes it from F or inserts it into G, keeping both the true key sets of F and G being prefixes of $\{x_1, \ldots, x_n\}$. For any $k \in \{0, 1, \ldots, n\}$, let F_k denote the filter obtained by deleting $x_n, x_{n-1}, \ldots, x_{k+1}$ from the initial F and let G_k denote the filter obtained by inserting $x_1, x_2, \ldots, x_k$ into the initial G. In Proposition 3.1, where we assume $\mathcal{D}$ is history independent, since F_k and G_k share the same true key set $\{x_1, \ldots, x_k\}$, they are actually the same filter states. However, in Proposition 4.3 (without history independence), as F_k (obtained by first inserting $x_1, \ldots, x_n$ to an empty filter and then deleting $x_n, \ldots, x_{k+1}$) and G_k (obtained by directly inserting $x_1, \ldots, x_k$ to an empty filter) are constructed from different operational histories, they are different filter states.

The proof of Proposition 3.1 heavily relies on the fact that F_k and G_k represent the same filter state to upper bound the communication cost. When Alice sends x_k to Bob, the expected communication cost is primarily determined by how many elements are in the sub-universe $\overline{F_{r_k}} \setminus \overline{G_{\ell_k}}$. In particular, the expected communication cost is approximately

$$\mathbb{E}\big[\log|\overline{F_{r_k}} \setminus \overline{G_{\ell_k}}|\big] \le \log \mathbb{E}\big[|\overline{F_{r_k}} \setminus \overline{G_{\ell_k}}|\big] = \log\big(\mathbb{E}\big[|\overline{F_{r_k}}|\big] - \mathbb{E}\big[|\overline{G_{\ell_k}}|\big]\big),$$

where the equality follows from monotonicity. Since F_k and G_k are identical filter states, we can substitute $\mathbb{E}\big[|\overline{G_{\ell_k}}|\big]$ with $\mathbb{E}\big[|\overline{F_{\ell_k}}|\big]$ in the above equation. This substitution enables us to apply Lemma 3.4 to select a suitable parameter s that the protocol can use to achieve a significant reduction in communication cost. However, the same argument does not extend to the case where F_k and G_k are generally different filter states. Specifically, it is possible that for each $k \le n$, the size $\big|\overline{F_k}\big|$ remains close to $\varepsilon|U|$, while $|\overline{G_k}|$ stays close to zero. In this scenario, no choice of s would significantly reduce the communication cost.

To address this issue, the main technical contribution of this section is to modify the definitions of F_k and G_k using a more complicated operational sequence (called the ***obfuscation sequence***). This modification ensures that, although F_k and G_k are still distinct filter states, their distributions under all sources of randomness are close. This closeness guarantees that $\mathbb{E}\big[|\overline{F_k}|\big]$ is close to $\mathbb{E}\big[|\overline{G_k}|\big]$, which in turn allows us to complete the proof.

To illustrate the high-level idea of the obfuscation sequence, we start with the filter states F_k's and G_k's defined previously and demonstrate how to modify them step by step to make their distributions close. Since F_n and G_n are already identical (both obtained by inserting $x_1, \ldots, x_n$), the first nontrivial case to consider is obfuscating F_{n-1} and G_{n-1}. Recall that, compared to the operational sequence $\sigma_{G_{n-1}}$ that produces G_{n-1} (i.e., inserting $x_1, \ldots, x_{n-1}$), the sequence $\sigma_{F_{n-1}}$ leading to F_{n-1} includes an additional insertion and deletion of x_n, making the two filter states different. To obfuscate this difference, we introduce additional operations in $\sigma_{G_{n-1}}$: After inserting $x_1, \ldots, x_{n-1}$, we repeatedly insert and delete a random key, performing this operation a random number of times chosen from $[0, M-1]$ for a large enough parameter M. Then, due to the randomness of x_n, the operational sequence $\sigma_{F_{n-1}}$ can be interpreted as follows: after inserting $x_1, \ldots, x_{n-1}$, we again repeatedly insert and delete a random key, but now for a random number of times chosen from $[1, M]$. Since these distributions are close, this brings the distributions of F_{n-1} and G_{n-1} closer together.

Here, we emphasize that it is crucial to maintain $\sigma_{G_{n-1}}$ as a prefix of $\sigma_{F_{n-1}}$, as this allows us to leverage the monotonicity between F_{n-1} and G_{n-1}. This requirement rules out naive obfuscation strategies, such as keeping $\sigma_{F_{n-1}}$ unchanged while simply appending an additional insertion and deletion of a random key at the end of $\sigma_{G_{n-1}}$. Such an approach would disrupt the structural relationship between the sequences, making it ineffective for our purpose.

Similarly, we can further obfuscate F_{n-2} and G_{n-2} using the same strategy. Suppose the sequence $\sigma_{F_{n-2}}$ leading to F_{n-2} is longer than the sequence $\sigma_{G_{n-2}}$ leading to G_{n-2} by a suffix σ'. To obfuscate this difference, we introduce additional operations at the end of $\sigma_{G_{n-2}}$ as follows: Sample independent sequences $\sigma'_1, \ldots, \sigma'_i$ from the same distribution as σ', where i is a random number drawn from $[0, M-1]$. Then, after inserting $x_1, \ldots, x_{n-2}$, we append $\sigma'_1, \ldots, \sigma'_i$ to the end of $\sigma_{G_{n-2}}$, making its distribution closely match that of $\sigma_{F_{n-2}}$.

This process can be applied recursively to obfuscate F_{n-3} and G_{n-3}, F_{n-4} and G_{n-4}, continuing until we reach F_0 and G_0. To formally define the resulting operational sequences and capture the recursive structure,

we introduce a tree-based framework called the *obfuscating tree*, which will be detailed in the full proof.

Finally, even with these obfuscated filter states, several technical challenges remain. The previous construction of operational sequences involves n recursive steps, each increasing the sequence length by a factor of M. As a result, the final sequence grows exponentially, which is impractical.

To mitigate this, Alice divides the n keys $x_1, \ldots, x_n$ into b batches, where b is a relatively small super-constant, with each batch containing n/b keys. Bob updates his filter only after receiving an entire batch, reducing the number of filters he maintains to b. This modification limits the recursion depth to b, yielding an operational sequence of length M^b instead of M^n.

Additionally, for technical reasons, Alice and Bob first agree on a random partition of the universe $U = U_1 \cup U_2 \cup \cdots \cup U_b$, ensuring that keys in the k-th batch are sampled from U_k. While unnecessary in this section, this step will be crucial when addressing the removal of the monotonicity assumption in Section 5.

4.2 Formal Proof of Proposition 4.3

The proof of Proposition 4.3 is still based on a one-way communication game between Alice and Bob. Suppose Alice and Bob have access to the random tape of $\mathcal{D}$ together with another public random tape with free randomness. At the beginning of the game, Alice and Bob first agree on a random partition of the universe that is determined by the free public randomness.

Random partition of the universe. Let $b = \omega(1)$ be a parameter to be determined. Let $\pi = (U_1, \ldots, U_b)$ be a partition of the universe U that is sampled uniformly at random, such that $|U_k| = |U|/b$ for each $k \le b$. Based on this random partition π, the goal of the communication game is as follows.

For each $k \le b$, let $X_k = \left(x_1^{(k)}, \ldots, x_{n/b}^{(k)}\right)$ be a sequence of distinct keys from U_k which is sampled uniformly at random. Alice wants to transmit $(X_1, \ldots, X_b)$ to Bob via the one-way communication protocol. Similar to Claim 3.2, no matter how Alice sends her message, she needs to communicate at least $n \log(|U|/b) - o(n)$ entropy.

Claim 4.4. *The message entropy sent by Alice is at least* $b \log\left((|U|/b)^{\underline{n/b}}\right)$.[8]

Proof. As Bob can decode $(X_1, \ldots, X_b)$ from the message, the entropy of the message is at least

$$H(X_1, \ldots, X_b \mid \text{random tapes}) = H(X_1, \ldots, X_b \mid \pi) = \sum_{k=1}^{b} H(X_k \mid U_k) = b \log\left((|U|/b)^{\underline{n/b}}\right). \qquad \square$$

Obfuscation sequences. Next, we construct the protocols Q_s that allow Alice to send $(X_1, \ldots, X_b)$ indirectly to Bob. The protocol is similar to the protocols P_s, but we will perform additional obfuscation operations on the filters F and G to make their distributions similar, as described in Section 4.1. Below, we construct the sequence of operations that we will use for constructing F and G—we call this the ***obfuscation sequence***.

We first construct a random depth-b tree T, referred to as the ***obfuscating tree***. Let M be a large integer parameter to be determined. The tree T is built recursively: We begin with the root node v_0, designated as the *level-0 node*. Using the public random tape, we sample d_{v_0} as a uniformly random integer from $[1, M]$ and create d_{v_0} children for v_0, forming the *level-1 nodes*. Similarly, for each level-1 node v, we independently sample $d_v \in [1, M]$ and create d_v children. All the children of level-1 nodes collectively form the *level-2 nodes*. This process continues iteratively until we reach level b, at which point we obtain a depth-b tree T. Finally, for each $k \le b$, we refer to the rightmost level-k node as v_k, and then the rightmost path of T can be represented as $(v_0, \ldots, v_b)$. See Fig. 1.

Next, we label each edge of T with a sequence of keys. For each $k \le b$, the rightmost edge $\{v_{k-1}, v_k\}$ is labeled with Alice's keys X_k. For any other edge e that connects a level-$(k-1)$ node to a level-k node—referred to as a *level-k* edge—we independently sample a sequence $Y_k^{(e)} = (y_1, \ldots, y_{n/b})$ of distinct keys from U_k and assign $Y_k^{(e)}$ to e. As a result, each level-k edge is labeled with a sequence of n/b distinct keys from U_k.

[8] As in earlier sections, we use the notation $m^{\underline{n}}$ to represent the falling factorial of m of order n, i.e., $m(m-1)\cdots(m-n+1)$.

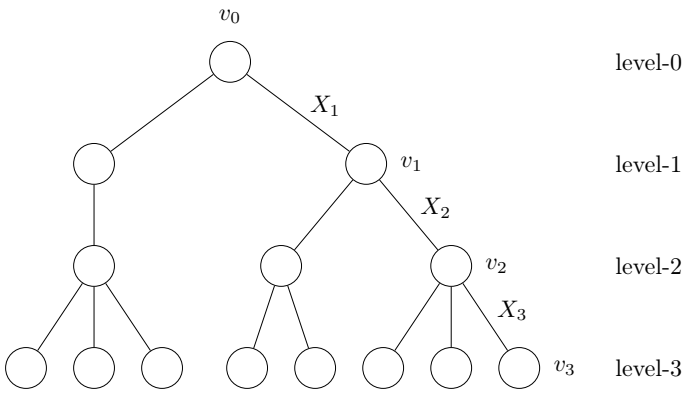


Figure 1: Obfuscating tree of depth $b = 3$. Each node has a randomly chosen number of children from the range $[1, 3]$. The rightmost path, represented as (v_0, v_1, v_2, v_3), has edge labels X_1, X_2, X_3.

The edge-labeled tree T naturally induces an operational sequence σ, referred to as the ***obfuscation sequence***, as follows. The sequence σ is defined according to a depth-first search (DFS) traversal of T. Whenever the traversal moves down along an edge, we insert the keys that appear in the edge label, and whenever it moves up along an edge, we delete the keys that appear in the edge label. In particular, when the traversal moves to a level-k node v of the tree, the corresponding prefix of σ will result in a filter with its true key set being the union of edge labels along the path from v to the root v_0 (i.e., consisting of n/b keys from each of $U_1, U_2, \ldots, U_k$).

Throughout the protocol Q_s, the filters F and G will always have their operational histories as prefixes of σ. Specifically, for each $k \le b$, define σ_{G_k} as the prefix of σ that terminates just before inserting X_{k+1} (corresponding to the traversal reaching the point where all children of v_k, except for v_{k+1}, have been visited, and the traversal is about to move to v_{k+1}). As a special case, σ_{G_b} terminates just after inserting X_b. Similarly, for each $k \le b$, define σ_{F_k} as the prefix of σ that terminates just before deleting X_k (corresponding to the traversal just before moving up from v_k to v_{k-1}). See Fig. 2.

Let F_k and G_k denote the filter states obtained after applying the operational sequence σ_{F_k} and σ_{G_k} on the initial empty filter, respectively. Then, both F_k and G_k will have their true key sets being the union of $X_1, \ldots, X_k$. Moreover, for any $\ell \le r$, it is easy to check that σ_{G_ℓ} is a prefix of σ_{F_r} and their difference is a self-contained operational sequence, which implies $\overline{G_\ell} \subseteq \overline{F_r}$ by the monotonicity assumption. Similarly, it is easy to check that the accepted set $\overline{G_k}$ is non-decreasing in k.[9]

Based on these obfuscated filter states, the protocol Q_s is defined as follows in Algorithm 2.

Note that the obfuscating tree T, along with all its edge labels (except for the rightmost edges), is sampled using only public randomness. Hence, Bob can construct G_0 by executing σ_{G_0} on an initially empty filter without any input from Alice. More generally, as long as Bob knows X_k, he can update the filter G_{k-1} to G_k by applying the difference between σ_{G_k} and $\sigma_{G_{k-1}}$; likewise, as long as Bob knows X_k, he can also update the filter F_k to F_{k-1} by applying the difference between $\sigma_{F_{k-1}}$ and σ_{F_k}.

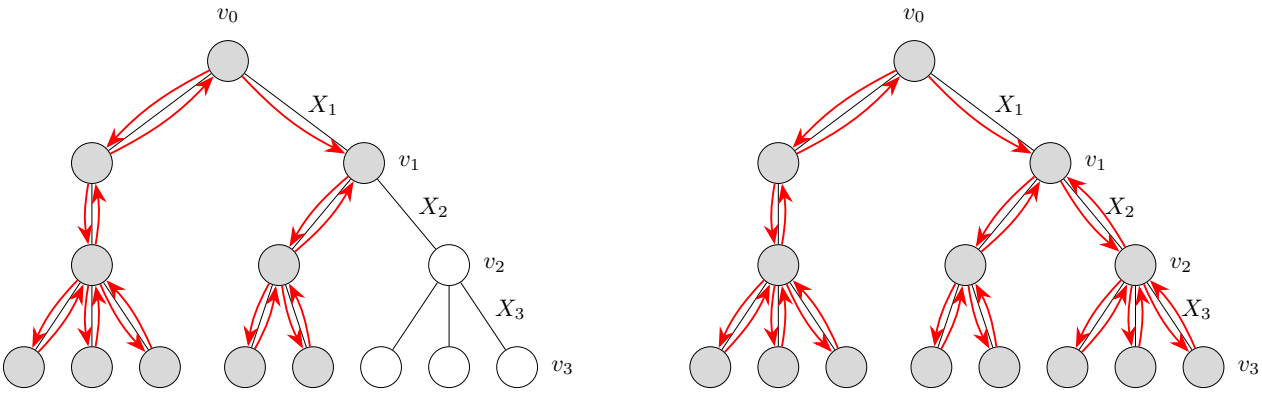
Below, we analyze the communication cost of the protocol Q_s and determine a suitable choice of $s \in [0, b]$ to establish the desired lower bound for the filter.

Cost of sending a single key. In the main part of the protocol Q_s, Alice transmits each key to Bob using $\texttt{Send}(X_k, F_r, G_\ell)$, where $\ell < k \le r$ (meaning that X_k is part of F_r's true key set but is disjoint from G_ℓ's). We begin by upper bounding the message entropy incurred by the subroutine $\texttt{Send}(X_k, F_r, G_\ell)$.

Similar to the proof of Proposition 3.1, we define parameters $a_0, \ldots, a_b$ such that

$$a_k \coloneqq \frac{1}{(1-\varepsilon)n + \varepsilon|U|} \cdot \mathbb{E}\big[\big|\overline{G_k}\big| - \big|\overline{G_{k-1}}\big|\big],$$

[9]Note that, unlike $\overline{G_k}$, we cannot conclude that $\overline{F_k}$ is non-decreasing in k. This is because σ_{F_k} is not necessarily a prefix of $\sigma_{F_{k+1}}$—in fact, the latter is shorter than the former. Fortunately, the monotonicity of G_k will suffice for our purposes.



(a) Operational sequence σ_{G_1}.

(b) Operational sequence σ_{F_1}.

Figure 2: Operational sequences σ_{G_1} and σ_{F_1}. The sequence σ_{G_1} follows a DFS traversal of T and stops before traversing (v_1, v_2), while the sequence σ_{F_1} terminates just after traversing (v_2, v_1).

Algorithm 2: Protocol Q_s

Subroutine `Send`(X_k, F_r, G_ℓ): // X_k is contained in F_r's true key set, but disjoint with G_ℓ's.
  for *each* x *in* X_k **do**
    $Z_x \longleftarrow$ indicator for whether $x \in \overline{G_\ell}$
    Alice sends Z_x to Bob
    if $Z_x = 0$ *(i.e.,* $x \notin \overline{G_\ell}$*)* **then**
      Alice sends x as an element in $\left(\overline{F_r} \setminus \overline{G_\ell}\right) \cap U_k$
    else
      Alice sends x as an element in U_k
Alice prepares F_b and sends it to Bob
Bob prepares G_0 by himself
for k *from* 1 *to* s **do**
  Alice sends X_k using `Send`(X_k, F_b, G_{k-1})
  Bob computes G_k using X_k and G_{k-1}
for k *from* b *to* $s+1$ **do**
  Alice sends X_k using `Send`(X_k, F_k, G_s)
  Bob computes F_{k-1} using X_k and F_k

where the expectation is taken over all sources of randomness, including the random tapes and the randomly sampled keys $X_1, \ldots, X_b$. Since $\overline{G_k}$ is increasing in k, we get $a_k \geq 0$ for each k. Moreover, we have

$$\mathbb{E}\big[\big|\overline{G_k}\big|\big] = ((1-\varepsilon)n + \varepsilon|U|) \cdot a_{[0,k]}$$

for each k, and $a_{[0,b]} = \mathbb{E}\big[\big|\overline{G_b}\big|\big]/((1-\varepsilon)n + \varepsilon|U|) \leq 1$. Below, we establish an upper bound for message entropy of $\texttt{Send}(X_k, F_r, G_\ell)$ in terms of a_k's.

Lemma 4.5. *Suppose parameters b, M satisfy $M = 4^b$, $b = \omega(1)$, and $9^{b^2} = o(\varepsilon|U|/n)$.*[10] *Then, the message entropy incurred by $\texttt{Send}(X_k, F_r, G_\ell)$ is at most*

$$\frac{n}{b} \cdot \log(|U|/b) + \frac{n}{b} \log \varepsilon + (1-\varepsilon)\frac{n}{b} \cdot \log\left(a_{(\ell,r]} + \frac{2}{4^b}\right) + o\Big(\frac{n}{b}\Big).$$

Proof. Since the protocol $\texttt{Send}(X_k, F_r, G_\ell)$ sends each key x in X_k separately, we only need to upper bound the cost of sending each single key x.

Similar to Proposition 3.1, to send a key x, Alice must first send a biased bit $Z_x := \mathbb{1}[x \in \overline{G_\ell}]$, which has entropy at most $h(\varepsilon)$. Then, depending on the value of Z_x, she either sends $\log\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big|$ bits (in the common case where $Z_x = 0$) or $\log|U_k|$ bits (in the rare case where $Z_x = 1$). Thus, the total entropy is at most

$$\begin{aligned}
&h(\varepsilon) + \mathbb{E}\big[Z_x \log|U_k| + (1-Z_x)\log\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big|\big] \\
&\leq h(\varepsilon) + \varepsilon \log|U_k| + (1-\varepsilon)\,\mathbb{E}\big[\log\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big| \mid x \notin \overline{G_\ell}\big] \\
&\leq h(\varepsilon) + \varepsilon \log(|U|/b) + (1-\varepsilon)\log \mathbb{E}\big[\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big| \mid x \notin \overline{G_\ell}\big] \\
&\leq h(\varepsilon) + \varepsilon \log(|U|/b) + (1-\varepsilon)\log \frac{\mathbb{E}\big[\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big|\big]}{\Pr\big[x \notin \overline{G_\ell}\big]},
\end{aligned} \tag{16}$$

where the second inequality is due to the concavity of the logarithm, and the third inequality uses a similar technique as Claim 3.3: For any non-negative integer random variable X and event $\mathcal{E}$, we have

$$\mathbb{E}[X \mid \mathcal{E}] = \sum_{k=0}^{\infty} k \cdot \Pr[X = k \mid \mathcal{E}] \leq \sum_{k=0}^{\infty} k \cdot \frac{\Pr[X=k]}{\Pr[\mathcal{E}]} = \frac{\mathbb{E}[X]}{\Pr[\mathcal{E}]}.$$

Below, we further simplify $\mathbb{E}\big[\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big|\big]$ in terms of a_k's. This is achieved by the following two claims. At a high level, the first claim says that, when evaluating $\mathbb{E}\big[\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big|\big]$, we can, to a first approximation, treat U_k as independent of $\overline{F_r} \setminus \overline{G_\ell}$. The second claim then says that, thanks to the obfuscation performed within the obfuscation tree, the expectation of $|\overline{F_r}|$ is guaranteed to be almost the same as that of $|\overline{G_\ell}|$. Once we have these claims in place, we will be able to complete the proof using essentially the same framework as in Proposition 3.1.

Claim 4.6. *For any ℓ, r, k with $0 \leq \ell < k \leq r \leq b$, we have*

$$\mathbb{E}\big[\big|\big(\overline{F_r} \setminus \overline{G_\ell}\big) \cap U_k\big|\big] \leq \frac{|U|}{b|U| - nM^{b+1}}\,\mathbb{E}\big[\big|\overline{F_r} \setminus \overline{G_\ell}\big|\big] + \frac{nM^{b+1}}{b}.$$

Proof. We analyze the expectation by conditioning on the randomness of the obfuscation sequence σ and the random tape $\mathrm{tape}(\mathcal{D})$ used by the algorithm $\mathcal{D}$.

Fix any possible choices σ^* and $\mathrm{tape}(\mathcal{D})^*$. For the rest of the proof of the claim, we condition on $\sigma = \sigma^*$ and $\mathrm{tape}(\mathcal{D}) = \mathrm{tape}(\mathcal{D})^*$. This means that the filter states F_r and G_ℓ are completely fixed, and so is the set $\overline{F_r} \setminus \overline{G_\ell}$. Let A^* denote the set $\overline{F_r} \setminus \overline{G_\ell}$.

On the other hand, the distribution of the set U_k under this condition can be described as follows. First, conditioning on $\mathrm{tape}(\mathcal{D})^*$ does not affect the distribution of U_k, because U_k and σ are determined by another public random tape, which is independent of $\mathrm{tape}(\mathcal{D})$. Furthermore, conditioning on σ^* slightly alters the

[10] Recall that $|U| = \omega(n\varepsilon^{-1})$. Hence, $\varepsilon|U|/n = \omega(1)$ and we can choose a superconstant b such that $9^{b^2} = o(\varepsilon|U|/n)$.

distribution of U_k in the following way. Each key that appears in σ^* (i.e., in one of the edge labels of the obfuscating tree) has its membership in U_k determined by σ^*, based on the depth of the corresponding edge in the obfuscating tree. Suppose there are L such keys in total, of which L_k belong to U_k. Then, outside these L keys, U_k behaves as a uniformly random subset of size $|U|/b - L_k$. In particular, for each key that is not present in σ^*, it belongs to U_k with probability exactly $\frac{|U|/b - L_k}{|U| - L}$.

Based on these observations, the expectation of $|(\overline{F_r} \setminus \overline{G_\ell}) \cap U_k|$ can be upper bounded as follows:

$$\begin{aligned}
&\mathbb{E}\big[|(\overline{F_r} \setminus \overline{G_\ell}) \cap U_k| \mid \sigma^*, \mathrm{tape}(\mathcal{D})^*\big] = \mathbb{E}[|A^* \cap U_k| \mid \sigma^*, \mathrm{tape}(\mathcal{D})^*] \\
&= \sum_{y \in A^*} \Pr[y \in U_k \mid \sigma^*, \mathrm{tape}(\mathcal{D})^*] \le L_k + \frac{|U|/b - L_k}{|U| - L}|A^*|. \qquad (17)
\end{aligned}$$

Moreover, as the obfuscating tree has at most $M^b + M^{b-1} + \cdots + 1 < M^{b+1}/2$ edges, we have $L_k \le L < nM^{b+1}/b$. Plugging this into (17), we further obtain

$$\mathbb{E}\big[|(\overline{F_r} \setminus \overline{G_\ell}) \cap U_k| \mid \sigma^*, \mathrm{tape}(\mathcal{D})^*\big] \le \frac{nM^{b+1}}{b} + \frac{|U|/b}{|U| - nM^{b+1}/b}\,\mathbb{E}\big[|\overline{F_r} \setminus \overline{G_\ell}| \mid \sigma^*, \mathrm{tape}(\mathcal{D})^*\big].$$

Then, by taking expectation over σ^* and $\mathrm{tape}(\mathcal{D})^*$ on both sides, we obtain the desired inequality. □

Claim 4.7. *For any $k \le b$, we have*

$$\mathbb{E}\big[|\overline{F_k}|\big] \le \mathbb{E}\big[|\overline{G_k}|\big] + \frac{(1-\varepsilon)n + \varepsilon|U|}{M}.$$

Proof. Let C_k be the number of children of the node v_k in the obfuscating tree. The key point of the proof is to show that, for each integer i with $1 \le i \le M-1$, we have

$$\mathbb{E}\big[|\overline{F_k}| \mid C_k = i\big] = \mathbb{E}\big[|\overline{G_k}| \mid C_k = i+1\big]. \qquad (18)$$

Eq. (18) follows from the key observation that the distribution of $(\sigma_{F_k} \mid C_k = i)$ matches that of $(\sigma_{G_k} \mid C_k = i+1)$.

To establish this, recall that σ_{F_k} is derived from the DFS traversal of the obfuscating tree T, which terminates after fully exploring the subtree rooted at v_k. Meanwhile, σ_{G_k} is defined similarly, but its traversal halts after exploring all the subtrees rooted at v_k's children except for the last one, which is rooted at v_{k+1}. Consequently, both $(\sigma_{F_k} \mid C_k = i)$ and $(\sigma_{G_k} \mid C_k = i+1)$ explore the first i children of v_k, ensuring their corresponding traversals follow the same distribution.

Moreover, note that the edge labels in T are sampled independently, and every edge at the same level j is drawn from the same distribution—whether it is the rightmost edge (labeled by X_j) or a non-rightmost edge (labeled by an independent sequence of n/b distinct keys from U_j). It follows that $(\sigma_{F_k} \mid C_k = i)$ and $(\sigma_{G_k} \mid C_k = i+1)$ are identically distributed.

Based on (18), we can conclude the desired inequality by

$$\begin{aligned}
\mathbb{E}\big[|\overline{F_k}|\big] &= \sum_{i=1}^{M} \frac{1}{M}\,\mathbb{E}\big[|\overline{F_k}| \mid C_k = i\big] \\
&= \sum_{i=1}^{M-1} \frac{1}{M}\,\mathbb{E}\big[|\overline{G_k}| \mid C_k = i+1\big] + \frac{1}{M}\,\mathbb{E}\big[|\overline{F_k}| \mid C_k = M\big] \\
&\le \sum_{i=1}^{M} \frac{1}{M}\,\mathbb{E}\big[|\overline{G_k}| \mid C_k = i\big] + \frac{1}{M}\,\mathbb{E}\big[|\overline{F_k}| \mid C_k = M\big] \\
&\le \mathbb{E}\big[|\overline{G_k}|\big] + \frac{(1-\varepsilon)n + \varepsilon|U|}{M},
\end{aligned}$$

where the last inequality uses that $\mathbb{E}_{\mathrm{tape}(\mathcal{D})}\big[|\overline{F_k}|\big] \le (1-\varepsilon)n + \varepsilon|U|$ since the filter algorithm $\mathcal{D}$ only allows a positive error rate ε. □

Combining Claim 4.6 and Claim 4.7, $\mathbb{E}\big[|(\overline{F_r} \setminus \overline{G_\ell}) \cap U_k|\big]$ can be further simplified as

$$\begin{aligned}
&\mathbb{E}\big[|(\overline{F_r} \setminus \overline{G_\ell}) \cap U_k|\big] \\
&\le \frac{|U|}{b|U| - nM^{b+1}} \mathbb{E}\big[|\overline{F_r} \setminus \overline{G_\ell}|\big] + \frac{nM^{b+1}}{b} && \text{(by Claim 4.6)} \\
&= \frac{|U|}{b|U| - nM^{b+1}} \big(\mathbb{E}\big[|\overline{F_r}|\big] - \mathbb{E}\big[|\overline{G_\ell}|\big]\big) + \frac{nM^{b+1}}{b} && \text{(by monotonicity)} \\
&\le \frac{|U|}{b|U| - nM^{b+1}} \left(\mathbb{E}\big[|\overline{G_r}|\big] - \mathbb{E}\big[|\overline{G_\ell}|\big] + \frac{(1-\varepsilon)n + \varepsilon|U|}{M}\right) + \frac{nM^{b+1}}{b} && \text{(by Claim 4.7)} \\
&= \frac{|U|}{b|U| - nM^{b+1}} ((1-\varepsilon)n + \varepsilon|U|)\left(a_{(\ell,r]} + \frac{1}{M}\right) + \frac{nM^{b+1}}{b} \\
&= \frac{\varepsilon|U|}{b} \cdot (1 + o(1)) \cdot \left(a_{(\ell,r]} + \frac{1}{M}\right) + \frac{nM^{b+1}}{b}, && (19)
\end{aligned}$$

where the last line uses the following two approximations:

- Since $|U| = \omega(n\varepsilon^{-1})$, we have $(1-\varepsilon)n + \varepsilon|U| = \varepsilon|U| \cdot (1 + o(1))$.
- Since $\varepsilon|U|/n = \omega\left(9^{b^2}\right)$ and $M = 4^b$, we have $|U|/n = \omega\left(9^{b^2}\right) = \omega\big(M^{b+1}/b\big)$, which means $b|U| - nM^{b+1} = b|U| \cdot (1 - o(1))$.

Moreover, we have $\varepsilon|U|/n = \omega\left(9^{b^2}\right) = \omega\big(4^b \cdot M^{b+1}\big)$, which means $nM^{b+1} < \varepsilon|U|/4^b$. Hence, (19) can be further simplified as

$$\frac{\varepsilon|U|}{b} \cdot (1 + o(1)) \cdot \left(a_{(\ell,r]} + \frac{1}{M}\right) + \frac{nM^{b+1}}{b} \le \frac{\varepsilon|U|}{b} \cdot (1 + o(1)) \cdot \left(a_{(\ell,r]} + \frac{2}{4^b}\right). \tag{20}$$

Plugging (20) into (16), the message entropy of sending a single key is further upper bounded by

$$\begin{aligned}
&h(\varepsilon) + \varepsilon \log(|U|/b) + (1-\varepsilon) \log \frac{\mathbb{E}\big[|(\overline{F_r} \setminus \overline{G_\ell}) \cap U_k|\big]}{\Pr\big[x \notin \overline{G_\ell}\big]} \\
&\le o(1) + \varepsilon \log(|U|/b) + (1-\varepsilon)\left(\log\left(\frac{\varepsilon|U|}{b} \cdot (1 + o(1)) \cdot \left(a_{(\ell,r]} + \frac{2}{4^b}\right)\right) - \log(1-\varepsilon)\right) \\
&= \log(|U|/b) + (1-\varepsilon)\log\varepsilon + (1-\varepsilon)\log\left(a_{(\ell,r]} + \frac{2}{4^b}\right) + o(1) \\
&= \log(|U|/b) + \log\varepsilon + (1-\varepsilon)\log\left(a_{(\ell,r]} + \frac{2}{4^b}\right) + o(1).
\end{aligned}$$

Hence, by summing up the communication cost for each key x in X_k, we obtain the desired upper bound on the total message entropy incurred by $\texttt{Send}(X_k, F_r, G_\ell)$:

$$\frac{n}{b} \cdot \log(|U|/b) + \frac{n}{b}\log\varepsilon + (1-\varepsilon)\frac{n}{b} \cdot \log\left(a_{(\ell,r]} + \frac{2}{4^b}\right) + o\left(\frac{n}{b}\right). \qquad \square$$

Putting the pieces together. Now, to establish a lower bound for H_{filter}, we analyze the total communication cost incurred by the protocol Q_s. Similar to Proposition 3.1, the communication cost of Q_s consists of the following two parts:

- First, Alice transmits the filter state F_b, which requires H_{filter} bits.
- Then, for each $k \le b$, Alice sends X_k using $\texttt{Send}(X_k, F_{r_k}, G_{\ell_k})$, where the parameters ℓ_k and r_k are defined as follows:

$$\ell_k \coloneqq \begin{cases} k-1 & \text{for } k : k \le s \\ s & \text{for } k : s+1 \le k \le b \end{cases} \quad \text{and} \quad r_k \coloneqq \begin{cases} b & \text{for } k : k \le s \\ k & \text{for } k : s+1 \le k \le b \end{cases}.$$

We now choose parameters M and b such that $M = 4^b$, $b = \omega(1)$, and $9^{b^2} = o(\varepsilon|U|/n)$. With this choice, the communication cost incurred by each $\texttt{Send}(X_k, F_{r_k}, G_{\ell_k})$ can be upper bounded using Lemma 4.5. Summing over all terms, the total message entropy of Q_s is at most

$$H_{\text{filter}} + n\log(|U|/b) + n\log\varepsilon + (1-\varepsilon)\frac{n}{b}\cdot\left(\sum_{k=1}^{b}\log\left(a_{(\ell_k,r_k]} + \frac{2}{4^b}\right)\right) + o(n).$$

Comparing to Claim 4.4, we obtain

$$H_{\text{filter}} + n\log(|U|/b) + n\log\varepsilon + (1-\varepsilon)\frac{n}{b}\cdot\left(\sum_{k=1}^{b}\log\left(a_{(\ell_k,r_k]} + \frac{2}{4^b}\right)\right) + o(n) \geq b\log\left((|U|/b)^{\underline{n/b}}\right). \tag{21}$$

Moreover, as $|U| = \omega(n)$, we have

$$\left(\frac{|U|}{b}\right)^{\underline{n/b}} \geq \left(\frac{|U|}{b} - \frac{n}{b}\right)^{n/b} = \left(\frac{|U|}{b}\right)^{n/b}\left(1 - \frac{n}{|U|}\right)^{n/b} \geq \left(\frac{|U|}{b}\right)^{n/b}\left(1 - o\left(\frac{n}{b}\right)\right).$$

Therefore, the right-hand side of (21) can be further simplified as

$$b\log\left((|U|/b)^{\underline{n/b}}\right) \geq b\left(\log\left((|U|/b)^{n/b}(1-o(n/b))\right)\right) \geq n\log(|U|/b) - o(n),$$

and we obtain

$$H_{\text{filter}} \geq n\log\varepsilon^{-1} - (1-\varepsilon)\frac{n}{b}\cdot\left(\sum_{k=1}^{b}\log\left(a_{(\ell_k,r_k]} + \frac{2}{4^b}\right)\right) - o(n). \tag{22}$$

Finally, we select an appropriate s, following a similar approach as in Proposition 3.1, to derive the desired filter lower bound from (22). By Lemma 3.4, there exists an s with $0 \leq s \leq b$ such that

$$\sum_{k=1}^{b}\log a_{(\ell_k,r_k]} \leq -b\log e + o(b). \tag{23}$$

Below, we show that such s suffices to imply the desired filter lower bound.

Lemma 4.8. *For any real numbers $z_1, \ldots, z_b \in (0,1]$, suppose that*

$$\sum_{k=1}^{b}\log z_k \leq -b\log e + o(b). \tag{24}$$

Then, we have

$$\sum_{k=1}^{b}\log\left(z_k + \frac{2}{4^b}\right) \leq -b\log e + o(b). \tag{25}$$

Proof. Depending on how large each z_k is, there are two cases.

Case 1. Suppose there exists a $k^* \leq b$, such that $z_{k^*} \leq 3^{-b}$. Then,

$$\log\left(z_{k^*} + \frac{2}{4^b}\right) < \log\frac{3}{3^b} < \log\frac{1}{e^b} = -b\log e.$$

For any other $k \neq k^*$, we have

$$\log\left(z_k + \frac{2}{4^b}\right) \leq \log\left(1 + \frac{2}{4^b}\right) = o(1).$$

Summing them together, we obtain (25), as desired.

Case 2. Suppose for any $k \le b$, we have $z_k > 3^{-b}$. Then,

$$\log\left(z_k + \frac{2}{4^b}\right) = \log z_k + \log\left(1 + \frac{2}{4^b \cdot z_k}\right) < \log z_k + \log\left(1 + \frac{2}{(4/3)^b}\right) = \log z_k + o(1).$$

Summing over all k's, we obtain

$$\sum_{k=1}^{b} \log\left(z_k + \frac{2}{4^b}\right) \le \sum_{k=1}^{b} (\log z_k + o(1)) \le -b \log e + o(b),$$

where the last inequality follows from (24). □

Combining Lemma 4.8 and (23), we get

$$\sum_{k=1}^{b} \log\left(a_{(\ell_k, r_k]} + \frac{2}{4^b}\right) \le -b \log e + o(b).$$

Plugging this inequality into (22), we obtain

$$\begin{aligned} H_{\text{filter}} &\ge n \log \varepsilon^{-1} - (1-\varepsilon) \cdot \frac{n}{b} \cdot (-b \log e + o(b)) - o(n) \\ &= n \log \varepsilon^{-1} + n \log e - o(n). \end{aligned}$$

Finally, we verify that the lower-bound construction can be implemented using $f(n)$ operations, for any $f(n) = \omega(n)$. Since the obfuscation sequence σ contains at most nM^{b+1}/b operations, the protocol Q_s requires at most nM^{b+1}/b insertions and deletions to maintain the filters F and G. By choosing $b = \omega(1)$ to grow sufficiently slowly, we can ensure that the number of operations is at most $f(n)$. This completes the proof of Proposition 4.3.

5 Beyond Monotonicity

In the previous sections, we proved Theorem 1.1 under the monotonicity assumption. In this section, we extend the proof to work for general filters without assuming the accepted set is monotone.

The high-level idea of the proof is to define a modified version of the accepted set (called the ***reconstructible set***) for a general filter—one that remains monotone while preserving all the necessary properties of an accepted set. Specifically, let $\widetilde{F}$ denote the reconstructible set of a filter F. Then, $\widetilde{F}$ needs to satisfy the following properties:

- Every key in the true key set of F must also be in the reconstructible set $\widetilde{F}$.
- Any key not in the true key set of F is included in $\widetilde{F}$ with probability at most ε, taken over the randomness of $\mathcal{D}$.
- The reconstructible set is monotone in the sense that, for any $\ell \le r$, it satisfies $\widetilde{G_\ell} \subseteq \widetilde{G_r}$ and $\widetilde{G_\ell} \subseteq \widetilde{F_r}$, where F_k's and G_k's are the filters defined in Section 4.

One can verify that the proof of Proposition 4.3 remains valid when all instances of the accepted set are replaced with the reconstructible set, as long as the listed properties hold. Thus, it suffices to construct a reconstructible set that satisfies these properties.

The definition of the reconstructible set depends on the partition π of the universe. Fix a partition π. Throughout this section, we consider only operational sequences that *conform* to π, defined as follows.

Definition 5.1. We say an operational sequence σ ***conforms*** to a partition π if it is self-contained and satisfies the following property. If we execute σ on an initially empty filter, then:

- Whenever the size of the true key set is in the range $[(k-1)n/b, kn/b)$, σ inserts only keys from U_k.

- Whenever the size of the true key set is in the range $((k-1)n/b, kn/b]$, σ deletes only keys from U_k.

Clearly, the obfuscation sequence defined in Section 4 conforms to the partition π. Below, we define the reconstructible set for any filter state F that arises from executing an operational sequence conforming to π on the initial empty filter.

Definition 5.2 (Reconstructible set)**.** The ***reconstructible set*** $\widetilde{F}$ of a filter F is defined as the set of all keys x satisfying the following property: There exists a filter state F' that arises from executing an operational sequence σ' conforming to π on the initial empty filter, such that

- F' has the same memory representation as F.
- F' has the same true key set size as F.
- x belongs to the true key set of F'.

For a given key x, any operational sequence σ' with the above properties is said to be a ***reconstruction sequence***. The keys in the reconstructible set are precisely those for which at least one reconstruction sequence exists.

To conclude the proof of Theorem 1.1, we check this reconstructible set has the desired properties in the following lemma.

Lemma 5.3. *For any filter states F and G that arise from executing operational sequences σ_F and σ_G conforming to π on the initial empty filter, respectively, the following holds:*

- *The true key set of F is contained in $\widetilde{F}$, and $\widetilde{F}$ is contained in $\overline{F}$.*
- *Suppose σ_G is a prefix of σ_F and their difference is self-contained. If the true key set size of G is a multiple of n/b, say $\ell n/b$, then we have $\widetilde{G} \subseteq \widetilde{F}$.*

Proof. First, it is easy to verify that the true key set of F is contained in $\widetilde{F}$. This is because the sequence σ_F serves as the reconstruction sequence for each key in the true key set of F, ensuring that all such keys are included in $\widetilde{F}$.

Second, for any key $x \in \widetilde{F}$, we show that x is also in $\overline{F}$. By the definition of the reconstructible set, there exists a sequence σ' conforming to π, such that the filter F' resulting from the operational history σ' has the same memory representation as F, and x belongs to the true key set of F'. Since the true key set of F' is contained in the accepted set $\overline{F'}$, we have $x \in \overline{F'}$. Moreover, since F' shares the same memory representation with F, the query algorithm over F' behaves exactly the same as over F, implying $\overline{F'} = \overline{F}$. Therefore, combining these two results, we conclude $x \in \overline{F}$.

Finally, we show the reconstructible set is monotone. For any $x \in \widetilde{G}$, we show that x is also in $\widetilde{F}$. Let σ'_G be the reconstruction sequence of x in G. Then, we claim that the operational sequence σ'_F, defined by concatenating σ'_G with the difference $\sigma_F - \sigma_G$, is a valid reconstruction sequence of x in F.

We first verify that σ'_F is self-contained. Let G' be the filter state resulting from the operational history σ'_G. Since both σ'_G and $\sigma_F - \sigma_G$ are self-contained, the only way σ'_F could fail to be self-contained is if there exists a key x that is in the true set for G', but is then inserted again in $\sigma_F - \sigma_G$ without first being deleted. Let us assume that this happens, and derive a contradiction.

By the properties of the reconstruction sequence, the true key set size of G' matches that of G, which is $\ell n/b$. Since σ'_G conforms to π, all keys in the true key set of G' must belong to $\bigcup_{k=1}^{\ell} U_k$, meaning $x \in \bigcup_{k=1}^{\ell} U_k$.

On the other hand, because $\sigma_F - \sigma_G$ is self-contained, all insertions in $\sigma_F - \sigma_G$ occur when the true key set size is at least $\ell n/b$. By the definition of conformity to π, these insertions must be from $\bigcup_{k=\ell+1}^{b} U_k$. This contradicts the assumption that x is inserted again in $\sigma_F - \sigma_G$, and completes the proof that σ'_F is self-contained.

Now, since both σ'_G and $\sigma_F - \sigma_G$ conform to π, it is easy to further check that σ'_F also conforms to π. Therefore, σ'_F is a valid reconstruction sequence of x in F, meaning that $x \in \widetilde{F}$, as desired. □

6 Open Questions

We conclude with two directions for future work.

Tight upper and lower bounds for $\varepsilon^{-1} = \Theta(1)$. The lower bounds in this paper establish that, for $\varepsilon = o(1)$, fingerprint filters achieve redundancy within $o(n)$ bits of optimal for any dynamic filter.

When $\varepsilon^{-1} = \Theta(1)$, we conjecture that fingerprint filters should continue to be optimal, but with the following caveat. When $\varepsilon^{-1} = \Theta(1)$, the fingerprint filter can no longer ignore the fact that the fingerprints it is storing form a *multiset* rather than a set. Any space-optimal implementation would have to encode this multiset in an information-theoretically optimal way, based on the distribution that the multiset comes from. Even from an upper-bound perspective, constructing such a fingerprint filter—while supporting time-efficient operations—remains open.

Equally interesting is to prove strong lower bounds for the case of $\varepsilon^{-1} = \Theta(1)$. This appears to be considerably more difficult than the case of $\varepsilon^{-1} = \omega(1)$. It does not appear, for example, that the current proofs can, with more careful bookkeeping, be extended to get tight bounds in this regime.

Tight lower bounds for value-dynamic retrieval. Closely related to the problem of dynamic filters is the problem of *value-dynamic retrieval* [KW24]: Here, the task is to encode a function $f : S \to [2^v]$ for a set $S \subseteq U$ of n keys, while supporting queries of the form "return $f(s)$" and updates of the form "update $f(s) := x$", where in both cases the user is responsible for guaranteeing that $s \in S$.

The classic way to solve value-dynamic retrieval is with a minimal perfect hash function [FK84, HT01], which necessarily incurs $n \log e - o(n)$ bits of redundancy. Kuszmaul and Walzer [KW24] prove that $\Omega(n)$ bits of redundancy are necessary. They also show that, when $v = O(1)$, it is possible to construct an upper bound that achieves redundancy $n \log e - \Omega(n)$ bits. However, for the common case of $v = \omega(1)$, they conjecture that minimal perfect hashing should be optimal—that is, that any solution must incur $n \log e - o(n)$ bits of redundancy.

In [KW24], the high-level approach that they take in their lower bounds is essentially the same for dynamic filters and for value-dynamic retrieval. We remark, however, that this connection appears to disintegrate as one tries to prove sharp bounds for the redundancy. The techniques in the current paper do *not* appear to meaningfully extend to the value-dynamic retrieval problem. This is in part due to the following challenges:

- **No "empty states":** In our lower bound for filters, we rely implicitly on the ability to send *two* filters from Alice to Bob, where one of them is the empty filter, which can therefore be sent for free. For value-dynamic retrieval, there is no analogue to an empty filter—if Alice starts at data structure A and performs updates to get to data structure B, then Bob cannot recover either A or B for free (unless Bob happens to already know S).
- **Avoiding inoperable states:** In value-dynamic retrieval, if the user ever performs an illegal update (updates $f(u)$ for some $u \notin S$), then the resulting data structure need not offer any guarantees of any sort (for either queries or updates). This significantly limits the types of operation sequences that Alice can use in constructing her communication protocol, and seems to prevent Alice from being able to rely on key-set changes in order to communicate information.

Based on these difficulties, we suspect that an entirely different approach will be needed if one is to prove Kuszmaul and Walzer's conjecture [KW24]. Such a lower bound would be of great interest.

References

[AK21] Anes Abdennebi and Kamer Kaya. A Bloom filter survey: Variants for different domain applications. Preprint arXiv:2106.12189, June 2021.

[ANS10] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *Proc. 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 787–796, 2010.

[BE02] James Blustein and Amal El-Maazawi. Bloom filters—a tutorial, analysis, and survey. Technical Report CS-2002-10, Dalhousie University, Halifax, NS, Canada, December 2002.

[BE20] Ioana O. Bercea and Guy Even. A dynamic space-efficient filter with constant time operations. In *Proc. 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 11:1–11:17, 2020.

[BFG+18] Michael A. Bender, Martín Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. In *Proc. 59th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 182–193, 2018.

[BFJ+12] Michael A. Bender, Martín Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. *Proceedings of the VLDB Endowment*, 5(11):1627–1637, July 2012.

[BFK+22] Michael A. Bender, Martín Farach-Colton, John Kuszmaul, William Kuszmaul, and Mingmou Liu. On the optimal time/space tradeoff for hash tables. In *Proc. 54th ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1284–1297, 2022.

[BM04] Andrei Broder and Michael Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4), January 2004.

[CFG+78] Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *Proc. 10th ACM Symposium on Theory of Computing (STOC)*, pages 59–65, 1978.

[Cle84] John G. Clerry. Compact hash tables using bidirectional linear probing. *IEEE Transactions on Computers*, 33(9):828–834, September 1984.

[CLJW17] Hanhua Chen, Liangyi Liao, Hai Jin, and Jie Wu. The dynamic cuckoo filter. In *Proc. 25th IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, 2017.

[FAKM14] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo filter: Practically better than Bloom. In *Proc. 10th ACM International on Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 75–88, 2014.

[FK84] Michael L. Fredman and János Komlós. On the size of separating systems and families of perfect hash functions. *SIAM Journal on Algebraic Discrete Methods*, 5(1):61–68, March 1984.

[GFO18] Afton Geil, Martín Farach-Colton, and John D. Owens. Quotient filters: Approximate membership queries on the GPU. In *Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 451–462, 2018.

[HT01] Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Proc. 18th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 317–326, 2001.

[Kad05] Zoran Kadelburg. Inequalities of Karamata, Schur and Muirhead, and some applications. *The Teaching of Mathematics*, 8(1):31–45, 2005.

[Kar32] Jovan Karamata. Sur une inégalité relative aux fonctions convexes. *Publications de l'Institut Mathématique*, 1(1):145–147, 1932.

[KW24] William Kuszmaul and Stefan Walzer. Space lower bounds for dynamic filters and value-dynamic retrieval. In *Proc. 56th ACM Symposium on Theory of Computing (STOC)*, pages 1153–1164, 2024.

[LGM+19] Lailong Luo, Deke Guo, Richard T. B. Ma, Ori Rottenstreich, and Xueshan Luo. Optimizing Bloom filter: Challenges, solutions, and comparisons. *IEEE Communications Surveys & Tutorials*, 21(2):1912–1949, 2019.

[LGR+19] Lailong Luo, Deke Guo, Ori Rottenstreich, Richard T. B. Ma, Xueshan Luo, and Bangbang Ren. The consistent cuckoo filter. In *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pages 712–720, 2019.

[LMSS21] David J. Lee, Samuel McCauley, Shikha Singh, and Maximilian Stein. Telescoping filter: A practical adaptive filter. In *Embedded Systems and Applications*, 2021.

[LP13] Shachar Lovett and Ely Porat. A space lower bound for dynamic approximate membership data structures. *SIAM Journal on Computing*, 42(6):2182–2196, 2013.

[LYY20] Mingmou Liu, Yitong Yin, and Huacheng Yu. Succinct filters for sets of unknown sizes. In *Proc. 47th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 79:1–79:19, 2020.

[NP19] Sabuzima Nayak and Ripon Patgiri. A review on role of Bloom filter on DNA assembly. *IEEE Access*, 7:66939–66954, 2019.

[PBJP17] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A general-purpose counting filter: Making every bit count. In *Proc. ACM International Conference on Management of Data (SIGMOD)*, pages 775–787, 2017.

[PCD+21] Prashant Pandey, Alex Conway, Joe Durie, Michael A. Bender, Martín Farach-Colton, and Rob Johnson. Vector quotient filters: Overcoming the time/space trade-off in filter design. In *Proc. International Conference on Management of Data (SIGMOD)*, pages 1386–1399, New York, NY, USA, 2021.

[PNB18] Ripon Patgiri, Sabuzima Nayak, and Samir Kumar Borgohain. Role of Bloom filter in big data research: A survey. *International Journal of Advanced Computer Science and Applications*, 9(11), 2018.

[PNK18] Ripon Patgiri, Sabuzima Nayak, and Samir Kumar Borgohain. Preventing DDoS using Bloom filter: A survey. *EAI Endorsed Transactions on Scalable Information Systems*, 5(19):e3, November 2018.

[PPS05] Anna Pagh, Rasmus Pagh, and Srinivasa Rao Satti. An optimal Bloom filter replacement. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 823–829, 2005.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, May 2004.

[SGK+20] Amritpal Singh, Sahil Garg, Ravneet Kaur, Shalini Batra, Neeraj Kumar, and Albert Y. Zomaya. Probabilistic data structures for big data analytics: A comprehensive review. *Knowledge-Based Systems*, 188(C), January 2020.

[TRL12] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of Bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2012.