Length Leakage in Oblivious Data Access Mechanisms

Grace Jia  
Yale University

Rachit Agarwal  
Cornell University

Anurag Khandelwal  
Yale University

Abstract

This paper explores the problem of preventing length leakage in oblivious data access mechanisms with passive persistent adversaries. We show that designing mechanisms that prevent both length leakage and access pattern leakage requires navigating a three-way tradeoff between storage footprint, bandwidth footprint, and the information leaked to the adversary. We establish powerful lower bounds on achievable storage and bandwidth footprints for a variety of leakage profiles, and present constructions that perfectly or near-perfectly match the lower bounds.

1 Introduction

Cloud services offer scalable, fault-tolerant, and easy-to-manage systems for storing and querying data. However, using (untrusted) cloud services also leads to significant security concerns: data accesses that used to be contained within an organization’s trusted domain are now visible to potentially untrusted entities on the cloud. A now-long line of work has shown that, even if offloaded data is encrypted, an adversary can exploit data access patterns to learn damaging information about the data [1–4]. To protect against such access pattern attacks, our community has developed a large and active body of research on oblivious data access mechanisms: on different adversarial settings (e.g., active [5] vs. persistent passive [6]), enabling transactional [7] and asynchronous queries [8, 9], enabling distributed proxy deployments [10, 11], achieving performance scalability [12–19], and identifying performance limits [20–25], to name a few.

This paper explores a complementary problem: preventing length leakage in oblivious data access mechanisms. Specifically, both ORAM-based [5] and PANCAKE-based [6] oblivious data access mechanisms assume that all data objects are of the same size; this is far from realistic in practice—recent studies from real-world production systems have established that data object sizes can vary by multiple orders of magnitude [26]. For such realistic scenarios, it becomes important to design systems that not only enable oblivious data access but also protect against length leakage.

Preventing length leakage in oblivious data access mechanisms raises new challenges: as we will show, it requires navigating a complex three-way tradeoff between the storage used by the mechanism (i.e., its storage footprint), the total amount of access traffic to the storage per query (i.e., its bandwidth footprint), and the precise information leaked to the adversary. To provide some intuition, consider the following two extreme design points. One obvious design is “padding”-based [27]: each object is padded with random bits to the largest object size, say, $s_{\text{max}}$ (queries are responded to in an obvious manner); with $n$ objects, this design requires storage footprint equal to $n \cdot s_{\text{max}}$, bandwidth footprint proportional to $s_{\text{max}}$ per query\(^1\), and informally speaking, leaks information about the number of objects ($n$) and the largest object size ($s_{\text{max}}$) to the adversary. At the other extreme are more recent “packing”-based designs [28], in particular size-locked indexes [29]: all data objects are packed into one blob (each data access query is responded to by downloading the entire blob). This requires storage and bandwidth footprint equal to the sum of the sizes of the plaintext objects—$\sum s_i$; informally, such packing-based designs leak information about the sum of the sizes of the plaintext objects to the adversary. Returning to the three-way tradeoff: the first design has higher storage footprint, lower bandwidth footprint, and (informally) less information leakage than the second design. Thus, depending on the storage budget, bandwidth budget, and/or security goals, either of the two design points may be of interest.

This paper explores fundamental limits and achievable design points for preventing length leakage in oblivious data access mechanisms. We focus on passive persistent adversaries, as motivated by the PANCAKE work [6]. We provide details on our system and security model in §2, but briefly, an encryption proxy takes as input a collection of $n$ plaintext objects of sizes $S = \{s_1, s_2, \ldots, s_{\text{max}}\}$, and creates encrypted objects that will be stored in an untrusted storage server. The

\(^{1}\)ORAM and PANCAKE incur $\Omega(\lg(n)) \cdot s_{\text{max}}$ and $3 \cdot s_{\text{max}}$ bandwidth footprint, respectively.
adversary can observe all the (encrypted) queries/responses between the proxy and storage, but it cannot inject queries; instead, queries are sampled from a distribution $\pi$. The encryption mechanism has an estimate of the distribution, and the adversary knows the precise distribution. The adversary “wins” if it can distinguish the sequence of queries from a sequence of uniformly distributed accesses to fixed-sized random bit strings—capturing both access pattern and length leakage. We introduce a formal security framework that captures this setting as real-or-random indistinguishability under chosen distribution and length attack (ROR-CDLA, §7).

### 1.1 Summary of Key Results

We establish the following results (Table 1):

1. First, we establish the intuitive result that padding-based schemes [27] applied to oblivious data access mechanisms [6] yield bandwidth-optimal constructions, while providing strong security: leaking only the number of objects ($n$) and the largest object size ($s_{\text{max}}$) to the adversary. In particular, we establish that any scheme that reveals only $n$ and $s_{\text{max}}$ must incur a minimum bandwidth footprint $\alpha \times s_{\text{max}}$ per query, where $\alpha \geq 1$ is a constant (independent of $n$) for any fixed storage footprint; larger storage footprint may enable a smaller $\alpha$. Furthermore, we establish that padding-based schemes [27] applied to oblivious data access mechanisms [6] achieve an upper bound that precisely matches this lower bound. These results imply that it is possible to design mechanisms that offer strong security at low bandwidth overheads if there is little variance in object sizes (i.e., the average object size is close to the largest) but quite high for use cases where object sizes may vary by orders of magnitude [26]. This motivates the relaxation of security for more storage and bandwidth-efficient schemes.

2. Second, we establish lower and upper bounds for packing-based designs [28, 29]. Specifically, our lower bound shows that when restricted to minimum possible storage—the sum of all the object sizes—any scheme that reveals $n$ and $\sum s_i$ must have a bandwidth overhead of $\sum s_i$ (matching the bandwidth overhead of the size-locked index construction [29]). Our lower bound and the construction whose performance meets the lower bound is more general: given even slightly more storage than $\sum s_i$, it is possible to dramatically reduce bandwidth footprint, albeit with slightly more leakage than size-locked indexes. In contrast to size-locked indexes that pack all objects into one bin, our generalized construction first bin-packs objects across $s_{\text{max}}$-sized bins and then employs a PANCAKE-based oblivious access mechanism over the bins. This construction achieves our generalized lower-bounds, i.e., if a scheme is allowed to leak the plaintext object sizes (instead of the sum of the object sizes as in the size-locked index construction), its bandwidth footprint lower bound decreases to a smaller multiple of $s_{\text{max}}$ (compared to (1)), with the multiple decreasing even faster as the given storage increases. This is particularly useful for datasets with objects of very variable sizes, where bin-packing objects can permit massive reduction of the bandwidth overhead for a given storage footprint.

3. Third, for the case of leaking both the individual object sizes and the access distribution, we present an improvement over the existing state-of-the-art construction, PANCAKE [6]. Our scheme greedily replicates objects with higher access probabilities and is provably bandwidth-optimal. Since the scheme can leak the access distribution, the bandwidth footprint depends on the distribution and can often be much smaller than those for (1). This is particularly useful for real-world key-value stores where accesses follow well-known Zipfian or bimodal distributions [26, 32].

4. Finally, for the case of leaking both the individual object sizes and the access distribution, we demonstrate a fundamental limitation in existing constructions in terms of achieving bandwidth optimality. We demonstrate that existing oblivious data access mechanisms perform uniform load balancing of requests across object replicas; however, non-uniform load balancing is necessary to achieve optimality. We present a Mixed-Integer Linear Program (MILP)-based construction that performs non-uniform load-balancing to achieve bandwidth-optimality, outperforming all PANCAKE-variants introduced above. Depending on the input object sizes and access distribution across them, this construction can achieve a significantly low bandwidth overhead. However, the MILP approach does not scale beyond a few hundred objects, so we approximate its behavior using a polynomial-time scheme (Greedy Stuffed PANCAKE, §6.2).

In §2.3, we discuss how an adversary can use each leakage profile to extract meaningful information about the data.

**Practical implications.** We also empirically characterize the three-way tradeoff space for real-world datasets. Figure 1 shows the bandwidth overhead of each of the constructions in Table 1 as a function of their storage overhead.

Table 1: *Summary of our results* for $n$ objects with sizes $S = \{s_1, \ldots, s_n\}$ and access distribution $\pi$. $\kappa$ is a parameter that bounds storage to $\kappa \cdot n \cdot s_{\text{max}}$, and $N_{\text{OPT}}$ is the minimum number of $s_{\text{max}}$-sized bins required to contain all objects.

<table>
<thead>
<tr>
<th>Leaked Information</th>
<th>Bandwidth Lower-bound</th>
<th>Optimal Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n, s_{\text{max}}$ (§3)</td>
<td>$\left\lceil \frac{</td>
<td>S</td>
</tr>
<tr>
<td>$n, \pi$ (§4)</td>
<td>$\sum_{i=1}^{n} s_i$</td>
<td>Stuffed PANCAKE</td>
</tr>
<tr>
<td>$n, s_{\text{max}}, \pi$ (§5)</td>
<td>No closed form</td>
<td>Greedy PANCAKE</td>
</tr>
<tr>
<td>$n, \pi, x$ (§6)</td>
<td>No closed form</td>
<td>MILP</td>
</tr>
</tbody>
</table>

---

2 A scheme’s bandwidth overhead is its bandwidth footprint relative to an insecure baseline’s footprint. A similar definition holds for storage overhead.

3 Bound only holds for $\delta = s_{\text{max}}$ and $\delta \geq \left\lceil \frac{|S|}{\kappa \cdot N_{\text{OPT}} + 1} \right\rceil \cdot s_{\text{max}}$. 
fixed storage, bandwidth overheads decrease as more information is leaked. Considering dataset-specific characteristics, we find that since Wikipedia has a large range of object sizes, all schemes observe a minimum of $\sim1000 \times$ overhead, since hiding each object’s identity requires each query to fetch the largest object size, which is $1000 \times$ larger than the average (similar observations have been made in prior work [33]). However, schemes that leak the object lengths (as noted in (2), (4)) incur lower bandwidth overheads, confirming our theoretical results. As such, the bandwidth overhead for such schemes is over $10 \times$ lower than schemes that do not leak object sizes. On the other hand, the Twitter dataset with less variability in object sizes can use schemes that do not leak object sizes for stronger security at a lower bandwidth overhead. Specifically, it observes $<10 \times$ bandwidth overhead with 2–3× storage overhead, which reduces to 2–3× bandwidth overhead on leaking object sizes.

For access patterns with varying skew, schemes that do not leak the access distribution have the same bandwidth overhead regardless of skew, as expected; for schemes that leak access distribution, we observe lower bandwidth overheads (as noted in (3), (4)). Bandwidth overheads increase with the access skew since hiding the identity of the most popular object requires injecting more noise for all other objects.

**Takeaways.** For the problem of simultaneously preventing length leakage and access pattern leakage, our results establish a rich tradeoff space between the storage footprint, bandwidth footprint, and information leakage. Our constructions demonstrate that existing oblivious data access mechanisms can resist length leakage attacks with simple modifications in proxy-side encryption mechanisms. Even so, this paper only takes the first step in laying the intellectual foundation for preventing length leakage in oblivious data access mechanisms. Much more work needs to be done in this space; to that end, we close the paper by outlining several avenues of future research.

## 2 Overview

We now describe our system and security models.

### 2.1 System Model

Our setup consists of a key-value (KV) store that supports (single-key) get and put operations on KV pairs $(k, v)$ submitted by one or more clients. Our results can, however, be applied to any data store that supports read and write operations. We focus on a trusted proxy architecture commonly used by encrypted data stores [8, 10–12, 34], which assumes multiple client applications route query requests through a single trusted proxy. The proxy manages the execution of these queries on behalf of the clients, sending queries to the untrusted storage service. We assume all communication channels are encrypted, e.g., using TLS.

### 2.2 Security Model

We now define our adversary and class of considered schemes.

**Threat model.** We use a trusted proxy threat model where the client and proxy servers belong to a trusted domain. We model client queries as independent samples from a distribution $\pi$ over keys, i.e., the probability of access to a key $k$ is $\pi(k)$. This work focuses on settings where the underlying distribution $\pi$ is static. We discuss how our results may translate to the dynamic setting where $\pi$ changes over time in §8.
We assume that the untrusted cloud storage service is controlled by a passive persistent adversary. This is typical of secure cloud-deployments [1, 2, 4, 6, 35], where the adversary (usually the cloud provider) can observe the volume (i.e., the number and size) of client queries, as well as the encrypted data stored at and retrieved from the storage service. It cannot, however, access cryptographic keys or change queries, responses, or stored data. The adversary aims to infer any information about individual KV pairs that are accessed. Similar to prior work [6, 9, 10, 12], we do not target hiding the timing of client queries.

**Considered schemes.** We consider a class of schemes $\mathcal{M}$, where each scheme $\Pi \in \mathcal{M}$ is defined as a pair of algorithms $\Pi = (\text{Init}, \text{Query})$. While we defer the formal security definitions to §7, we focus on an intuitive description here.

Init is an initialization algorithm that takes as input:
- a set $KV$ of $n$ key-value pairs $\{(k_1, v_1), \ldots, (k_n, v_n)\}$,
- a set $S$ of value sizes $\{s_1, \ldots, s_n\}$, where $s_i$ is the size of $v_i$,
- a distribution $\pi$ over $KV$, where $\pi(k)$ corresponds to the probability of accessing key $k$,
- a set $\sigma$, which determines the total storage capacity available to the scheme.

Init outputs the following:
- a new set $EKV$ of $n$ KV pairs $\{(\hat{k}_1, \hat{v}_1), \ldots, (\hat{k}_n, \hat{v}_n)\}$,
- a mapping $P$ that maps each key $k_i$ in $KV$ to a set of keys in $EKV$ whose values “contain” $v_i$ in a contiguous sequence of $s_i$ bytes. If $\text{Keys}(KV)$ and $\text{Values}(KV)$ denote the set of keys and values in $KV$ respectively, then:
  $$\forall k \in \text{Keys}(KV), P(k) = \{ \hat{k} \in \text{Keys}(EKV) | \hat{v} \text{ contains } v \}$$
- a set $\pi_r$ of functions indexed by key $k$, where $\pi_r[k]$ takes as input $\hat{k} \in \text{Keys}(EKV)$ and outputs the access probability of the copy of $k$’s value contained in $\hat{k}$’s encrypted value $^4$,
- a distribution $\pi_f$ over $EKV$, where $\pi_f(\hat{k})$ denotes the probability of accessing key $\hat{k}$ via “fake” queries or noise,
- the proportion of real queries $\delta$ to $EKV$, and,
- a constant $B \geq 1$ that will be used by Query and determine bandwidth footprint.

The mapping $P$ is stored at the proxy and is hidden from the adversary; so are $\pi_r$ and $\pi_f$. Note that since values of KV are contained in a contiguous sequence of $s_i$ bytes in values of EKV (identified by $P$), $\mathcal{M}$ implicitly rules out compression or other encoding schemes that increase or decrease the size of data contained in values $v$ in $\text{Values}(KV)$. Each key-value pair $(\hat{k}, \hat{v})$ in EKV is ultimately encrypted by applying a secretly keyed pseudorandom function (PRF) to the key (e.g., HMAC) to generate a label, denoted $F(\hat{k})$, and symmetrically encrypting the value using authenticated encryption, denoted $E(v')$. These labels and encrypted values are ultimately stored in the storage service, while the secret keys needed for $F$ and $E$ are stored at the proxy and, therefore, hidden from an adversary. Because $F$ is deterministic, the proxy can perform operations for key $\hat{k}$ by instead requesting $F(\hat{k})$. We have omitted the two required cryptographic secret keys in our notation for simplicity. We cryptographically bind labels and value ciphertexts by using the label as associated data with $E$.

EKV additionally satisfies the following three properties:

**Property 1.** Each value $\hat{v}$ in $\text{Values}(EKV)$ has the same size $\hat{s} \geq \max(s_i)$.

**Property 2.** The total storage used by values $\text{Values}(EKV)$ must be less than $\sigma$, i.e., $\hat{s} \times \hat{n} \leq \sigma$.

**Property 3.** Each $\hat{v}$ in $\text{Values}(EKV)$ is “contained” in at least one $v_j$ in $\text{Values}(KV)$, i.e., $\forall k_i \in \text{Keys}(KV), |P(k_i)| \geq 1$.

Intuitively, Property 1 ensures protection against length-leakage, while Property 2 bounds the scheme’s storage footprint. Finally, Property 3 ensures that all the data in KV is still present in EKV, i.e., the scheme is lossless.

Query is a query algorithm that transforms a query on plaintext keys in $\text{Keys}(KV)$ to one or more queries on encrypted keys in $\text{Keys}(EKV)$. Query takes as input:
- Init’s outputs $P$, $\pi_r$, $\pi_f$, $\delta$, $B$ and,
- a sequence of queries $Q = \{q_1, \ldots, q_n\}$ on plaintext keys in $\text{Keys}(KV)$, where each query $q_j$ is drawn from the distribution $\pi$; i.e., $\Pr[q_j = k_i] = \pi(k_i)$.

Query generates as output:
- a sequence of queries $\hat{Q} = \{\hat{q}_1, \ldots, \hat{q}_n\}$ over encrypted keys in $\text{Keys}(EKV)$, where $\hat{q} = q \cdot B$.

The generated sequence of queries satisfies two properties:

**Property 4.** All keys in EKV are accessed with equal probability, i.e., $\forall i, j$, $\Pr[\hat{q}_j = F(\hat{k}_i)] = 1/\hat{n}$.

**Property 5.** The expected number of accesses to any key $k \in \text{Keys}(KV)$ that maps to in $P$:
  $$\forall k \in \text{Keys}(KV), E(k \in Q) \leq \sum_{\hat{k} \in P(k)} E(\hat{k} \in \hat{Q})$$

Property 4 ensures security by requiring protection against access pattern leakage. Property 5 ensures correctness by requiring that all queries in $Q$ are answered in expectation over the randomness in Query’s execution. We use “in expectation”

---

^4Note that the function $\pi_r[k]$ is not a distribution over $P(k)$, since $\sum \pi_r[k](\hat{k}) = \pi(k)$, i.e., the original access probability of key $k$ and not 1.

^5There is a negligible probability for two distinct keys $\hat{k}_i, \hat{k}_j$ to have $F(\hat{k}_i) = F(\hat{k}_j)$; we ignore this case for simplicity.
rather than “with high probability” to strengthen our results: since our lower bounds (§3, §4) hold for the more relaxed expectation-based correctness, they must also hold for more stringent probability-based correctness. Property 5 thus excludes from \( \mathcal{M} \) any mechanism that achieves security (i.e., Properties 1 and 4) “incorrectly”, e.g., by ignoring a large fraction of client queries or responding with incorrect values.

We base our analysis on two key performance metrics:

- The storage footprint of \( \Pi \) is defined as \( \sigma \), the total storage capacity available to the scheme.
- The bandwidth footprint of \( \Pi \) is defined as \( \beta := B \cdot \delta \), the number of bytes fetched from EKV per query in \( Q \).

### 2.3 Considered Leakage Profiles

How much information can be leaked about the underlying data store differs by application; we capture these standards in the leakage profile \( \mathcal{L} \) atop our considered class of schemes \( \mathcal{M} \). Leakage profiles describe an upper bound on what an adversary learns from the execution of some scheme \( \Pi \in \mathcal{M} \); any information other than the output of \( \mathcal{L} \) must be hidden.

Again, we only provide an intuitive description of the security goal with leakage here and defer a formal simulator-based ROR-CDLA security definition to §7. Given any input store KV with access distribution \( \pi \), value sizes \( S \), storage footprint \( \sigma \), and a sequence \( Q \) of \( q \) queries sampled from \( \pi \), we consider the “real world” where \( \Pi \) produces a sequence of (encrypted) queries \( \hat{Q} \). In contrast, the “ideal world” uses a sequence of \( q \cdot B \) accesses generated from a uniform distribution over a set of \( \hat{n} \) random bit strings of size \( \delta \). For some leakage profile \( \mathcal{L} \), a scheme \( \Pi \in \mathcal{M} \) is \( \mathcal{L} \)-secure (or, \( \Pi \in \mathcal{M}_\mathcal{L} \)) if the outputs of the real and ideal worlds are indistinguishable. In the ideal world, \( \hat{n} \), \( \delta \) and \( B \) are generated by an algorithm \( \delta \) that only sees \( \mathcal{L}(KV,S,\pi,\sigma) \). In other words, any adversary’s view of \( \Pi \)’s execution must be simulatable given only \( \mathcal{L} \)’s output and no other information about \( \Pi \)’s inputs or internal state. In §7, we will show that the real and ideal worlds correspond to ROR-CDLA\(^0\) and ROR-CDLA\(^1\), respectively.

We emphasize that these leakage profiles only relax what information is revealed during the initialization of the encrypted store EKV. Any \( \Pi \in \mathcal{M} \) must still meet Properties 1-5, ensuring that for every query, the identity of the accessed key and its associated value are hidden. Thus, all schemes in our considered class \( \mathcal{M} \) resist existing access pattern attacks [1, 36, 37] and length leakage attacks [2, 38, 39], since they rely on leakage due to accesses to the KV store via client queries, i.e., information revealed after initialization.

We focus on four leakage profiles that determine whether or not the scheme reveals KV’s value sizes or access distribution:

- \( \mathcal{L}_m \) (§3): \( \mathcal{M}_m \) is the subset of schemes in \( \mathcal{M} \) with leakage profile \( \mathcal{L}_m \), where \( \mathcal{L}_m(KV,S,\pi,\sigma) = (n,s_{\text{max}},B) \). Intuitively, an adversary only learns the number of keys and the maximum value size in KV, along with the number of encrypted queries \( \Pi \) makes, \( q \cdot B \), ensuring protection against both access distribution and length leakage-abuse attacks. Since our considered leakage profiles have the same input and outputs that always contain \( n, B \), we will omit them in our notation.

- \( \mathcal{L}_\pi \) (§4): \( \mathcal{M}_\pi \) denotes the class of schemes in \( \mathcal{M} \) with leakage profile \( \mathcal{L}_\pi \), where \( \mathcal{L}_\pi \) outputs \( (s_{\text{max}},\pi) \). For any scheme \( \Pi \in \mathcal{M}_\pi \), an adversary can observe the sizes of all values in KV \( S \) and the output of \( \mathcal{L}_m \). This permits, for instance, compression-based attacks in the vein of CRIME [38] and BREACH [39], which leverage the sizes of encrypted values to learn about the plaintext values. As a concrete example, consider two sensitive database tables with noticeably different value size distributions, e.g., flu records in one table (with smaller, less detailed reports) and cancer records in another (with longer, more detailed reports), in a medical database. In this setting, the attacker can distinguish between the tables and identify when a request is for a flu or cancer record. This is analogous to attacks in traffic analysis literature over websites, which distinguish between different websites by observing the sum of packet sizes across its visited webpages [40].

- \( \mathcal{L}_S \) (§5): \( \mathcal{M}_S \) denotes the class of schemes in \( \mathcal{M} \) with leakage profile \( \mathcal{L}_S \), where \( \mathcal{L}_S \) outputs \( (S,\pi) \). For any scheme \( \Pi \in \mathcal{M}_S \), an adversary can identify the access distribution \( \pi \) over KV and the output of \( \mathcal{L}_m \). Like in the previous setting, an attacker can distinguish between two database tables by their access distributions, e.g., a table with flu records and uniform access pattern from a table with cancer records and skewed access pattern. Note that \( \mathcal{L}_S \) still excludes traditional access pattern attacks [1, 2, 36, 37] that exploit the mapping between KV pairs and their access frequencies.

- \( \mathcal{L}_{\text{Str}} \) (§6): \( \mathcal{M}_{\text{Str}} \) denotes the class of schemes in \( \mathcal{M} \) with leakage profile \( \mathcal{L}_{\text{Str}} \), where \( \mathcal{L}_{\text{Str}} \) outputs \( (S,\pi) \). For any scheme \( \Pi \in \mathcal{M}_{\text{Str}} \), an adversary can identify the value lengths \( S \) and access distribution \( \pi \). In this setting, attacks on schemes can exploit the leakage of size distribution (i.e., attacks in \( \mathcal{L}_S \)), access distribution (i.e., attacks in \( \mathcal{L}_\pi \), or a combination of the two (e.g., a volume and access pattern leakage-abuse attack [41]). Continuing our medical database example, a third table of arthritis records with uniform access patterns and longer detailed reports) and cancer records in another (with longer, more detailed reports), in a medical database. In this setting, the attacker can distinguish between the tables and identify when a request is for a flu or cancer record. This is analogous to attacks in traffic analysis literature over websites, which distinguish between different websites by observing the sum of packet sizes across its visited webpages [40].

3 \( \mathcal{L}_m \): Revealing the largest value size

Under our strictest leakage profile, the adversary may only learn the largest value size \( (s_{\text{max}}) \) and the number of keys \( (n) \). We do not explore stricter profiles that hide even \( s_{\text{max}} \) since they would yield impractically high bandwidth footprint overheads (even \( \mathcal{L}_m \) yields impractical overheads as noted in §1.1). To understand what constructions are possible under this leakage profile, consider a KV with three key-value pairs, with sizes and access probabilities, as shown in Figure 2a.
First, since no value size other than \( s_{\text{max}} = 5 \) can be leaked, all values in the output EKV must be at least 5 bytes in size.

It may be tempting to pack the values of keys \( k_0 \) and \( k_1 \) into a single 5-byte value for space efficiency (Figure 2b); however, this scheme is \( \notin \mathcal{ML}_m \). By observing that EKV consumes 10 bytes of storage, the adversary learns a bound on the sum of the value lengths and that at least one value is smaller than \( s_{\text{max}} \). As such, for any scheme in \( \mathcal{ML}_m \), its storage footprint must be independent of the value sizes \( S \). We, therefore, focus on schemes that pad the values for \( k_0 \) and \( k_1 \) to size \( s_{\text{max}} \).

The second security requirement is a uniform access distribution across keys in EKV. Consider a scheme that injects “fake” accesses to each padded value to equalize the access probabilities of the keys. However, the additional volume of access traffic to EKV leaks information about the original access distribution over the input KV, e.g., an access distribution skewed towards \( k_0 \) in Figure 2a requires more fake accesses to \( k_1 \) and \( k_2 \), while one where all key access probabilities are equal does not require any additional fake accesses. An adversary can distinguish between these two cases by observing the traffic volume to EKV. Thus, the bandwidth footprint of any scheme in \( \mathcal{ML}_m \) must be independent of the access distribution \( \pi \) over KV.

Other schemes that naively replicate the key-value pairs or combine replication and fake accesses can still be insecure under \( \mathcal{ML}_m \). Consider a scheme that replicates the padded value of \( k_0 \) and divides its accesses equally between the copy and the original. This results in four padded values (Figure 2c), and the remaining non-uniformity in access distribution can be smoothed using fake accesses. This scheme can still reveal information about the original access distribution over the input KV: if replication is used to smooth access frequencies based purely on the skew in the input distribution, then the total number of replicas (which is output by \text{Init}) leaks this degree of skew. In particular, if \( \pi \) were uniform, no replicas would be needed, and the output EKV would only contain three key-value pairs instead of four replicas in our example. As such, the storage footprint of a scheme in \( \mathcal{ML}_m \) must also be independent of the access distribution \( \pi \).

We formalize these intuitive observations, i.e., a scheme \( \Pi \in \mathcal{ML}_m \) must have output storage and bandwidth footprints independent of the value sizes and access distributions over KV, into performance lower bounds for any scheme \( \Pi \in \mathcal{ML}_m \) in §3.1. We then show in §3.2 that an adaptation of the PANCAKE algorithm [6] — dubbed Padded PANCAKE or PPC — is a bandwidth-optimal \( \mathcal{ML}_m \) scheme.

### 3.1 Performance Bounds

Any scheme \( \Pi \in \mathcal{ML}_m \) must output an EKV that has the same storage footprint \((\hat{n} \cdot \hat{s})\) across all input stores KV of n key-value pairs — otherwise it reveals information about the value sizes \( S \), or at the very least, some information about \( \sum k_i \). For correctness, all data encoded in KV must also be in EKV (Property 3). An input where all values have size \( s_{\text{max}} \) requires EKV to store at least \( n \cdot s_{\text{max}} \) bytes for no information to be lost from KV. Then for any \( n \)-key input to \( \Pi \), the output EKV must always have a storage footprint of at least \( n \cdot s_{\text{max}} \). Moreover, we find that in minimizing bandwidth footprint, schemes in \( \mathcal{ML}_m \) do not benefit from using EKV value size \( \hat{s} \) larger than \( s_{\text{max}} \). This is because larger EKV values increase the number of bytes transferred per access without reducing the proportion of fake queries required. We generalize these observations to all schemes in \( \mathcal{M} \) that do not leak \( S \) in the following two lemmas, deferring their proofs to Appendix B.

**Lemma 1.** Any \( \Pi \in \mathcal{M} \) with leakage \( \mathcal{L} \) and \( S \notin \mathcal{L} \) requires storage footprint \( \sigma \geq n \cdot s_{\text{max}} \).

**Lemma 2.** Given storage footprint \( \sigma = \kappa \cdot n \cdot s_{\text{max}} \), a scheme \( \Pi \in \mathcal{M} \) with leakage \( \mathcal{L} \) and \( S \notin \mathcal{L} \) that incurs the minimum possible bandwidth footprint must use \( \hat{s} = s_{\text{max}} \).

In Theorem 3.1, we leverage the above lemmas to show that the minimum bandwidth footprint for any scheme in \( \mathcal{ML}_m \) with storage footprint \( \sigma = \kappa \cdot n \cdot s_{\text{max}} \) \((\kappa \geq 1)\) is \( \frac{|\kappa|}{|\kappa| - n - 1} \cdot s_{\text{max}} \). Since \( \mathcal{ML}_m \) schemes do not leak \( \pi \), their bandwidth footprint must be the same for any \( \pi \). Intuitively, our proof identifies inputs \( S \) and \( \pi \) that maximize any scheme’s bandwidth footprint — this happens when all values are as large as possible and access distribution is as skewed as possible. An \( \mathcal{ML}_m \) scheme must incure the same bandwidth footprint for any other \( S \) and \( \pi \), so the minimum bandwidth footprint achievable in this worst case is also the lower-bound for any \( \mathcal{ML}_m \) scheme.

**Theorem 3.1.** Any \( \Pi \in \mathcal{ML}_m \) with storage footprint \( \sigma = \kappa \cdot n \cdot s_{\text{max}} \) \((\kappa \geq 1)\) must have bandwidth footprint \( \beta \geq \frac{|\kappa|}{|\kappa| - n - 1} \cdot s_{\text{max}} \).

**Proof.** Suppose for contradiction that \( \Pi \) incurs storage footprint \( \sigma = \kappa \cdot n \cdot s_{\text{max}} \) and minimum possible bandwidth footprint \( \beta = (1 - \delta) \cdot \frac{|\kappa|}{|\kappa| - n - 1} \cdot s_{\text{max}} \) for some \( 0 < \delta < 1 \). We will show that the scheme cannot be \( \mathcal{ML}_m \)-secure.

We define an \( \mathcal{L}_m \)-adversary \( A \) as in Figure 3. The intuition behind \( A \)'s attack — and our proof — is that given a heavily skewed access distribution as input, \( \Pi \) does not have enough
To this end, we show that \( A \) to distinguish between \( \text{ROR-CDLA}^0 \) and \( \text{ROR-CDLA}^1 \) by examining the sequence of output queries \( \hat{Q} \) and performing a simple statistical test: it simply guesses that it is interacting with the real \( \Pi \) if the maximum number of queries to any key in \( \text{EKV} \) is above some threshold. The threshold must be carefully chosen such that the probability of exceeding it is low for a uniform distribution of queries but high for the non-uniform distribution of queries that \( \Pi \) would generate with insufficient bandwidth. This forms the basis of Step 2, where we show that the adversary has a non-trivial probability of distinguishing between \( \text{ROR-CDLA}^0 \) and \( \text{ROR-CDLA}^1 \) for \( \delta > 0 \), completing the proof.

**Step 1:** Given \( A \)'s choice of inputs, \( \Pi \)'s output \( \text{EKV} \) has a non-uniform access distribution. To this end, we show that \( h \in \text{Keys(\text{EKV})} \) with access probability \( \Pr[h] > \frac{1}{\hat{n}} \).

Recall that in the \( \text{KV} \) generated by \( A_1 \), all \( n \) values are of size \( s_{\text{max}} \), the most heavily accessed key (which we shall call \( h \)) has access probability \( 1 - \epsilon \), and the remaining \( n - 1 \) keys have access probability \( \frac{\epsilon}{n-1} \). Lemma 2 shows that for \( \Pi \) to incur the minimum possible bandwidth it must use \( \delta = s_{\text{max}} \). Then the number of values in \( \text{EKV} \) is \( \hat{n} = \left\lfloor \frac{q}{\epsilon} \right\rfloor = \lfloor \kappa n \rfloor \), and each value can contain exactly one value from \( \text{KV} \). Now for \( \Pi \) to fulfill Property 5, the sum of accesses to each \( h \in \text{Pk}(h) \) in \( \hat{Q} \) must be at least the number of accesses to \( h \) in \( Q \) in expectation. This gives the inequality below:

\[
(1 - \epsilon) \cdot q \leq \left( \sum_{h \in \text{Pk}(h)} \Pr[h \in \hat{Q}] \right) \cdot q \cdot B
\]

or,

\[
B \geq \frac{1 - \epsilon}{\sum_{h \in \text{Pk}(h)} \Pr[h \in \hat{Q}]}
\]

Since bandwidth footprint is proportional to \( B \), in the best case \( \Pi \) can minimize the bandwidth footprint by splitting the access probability of \( h \) uniformly across \( P(h) \), i.e.,
$A_2$’s test does not leverage $\hat{h}$, $\delta$ and $B$ generated by the algorithm $S$. $A$ works against all possible $S$.

**Step 2(i):** We upper-bound $\Pr[\text{ROR-CDLA}^0_{\Pi,\ell_m,A,q}(\lambda) = 1]$, i.e., the probability that $A$ guesses incorrectly that the queries $\hat{Q}$ it is given are sampled from a uniform distribution rather than generated by $\Pi$.

To do so, we focus on the key $\hat{h} \in P(h)$ with access probability $p$. $A_2$ outputs 1 if, for all keys in EKV, the number of queries they receive is less than or equal to $A'$’s threshold. We can, therefore, use the probability that $\hat{h}$ receives $\leq \left\lfloor (c + 3\sqrt{c})\ln\hat{h}\right\rfloor$ queries as an upper bound for $\Pr[\text{ROR-CDLA}^0_{\Pi,\ell_m,A,q}(\lambda) = 1]$, and compute the same using the one-sided Chebyshev inequality. While we defer its full derivation to Appendix B.1, we state the upper bound below:

$$\Pr[\text{ROR-CDLA}^0_{\Pi,\ell_m,A,q}(\lambda) = 1] \leq \frac{(1 - \varepsilon)(\hat{h}(1 - \delta) - (1 - \varepsilon))}{(1 - \varepsilon)(\hat{h}(1 - \delta) - (1 - \varepsilon)) + c\ln\hat{h} \left[ \frac{(1 - \varepsilon - \frac{1}{\hat{h}} - \frac{1}{\hat{h}})}{\hat{h}(1 - \delta)^2} \right]}. \quad (2)$$

Note that $A$ must choose a small enough $\varepsilon$ such that $a > 0$, specifically $\varepsilon < 1 - (1 + 3\sqrt{c})(1 - \delta)$. Since we also need $\varepsilon > 0$ for the $\pi$ chosen by $A$ to be a valid distribution, the attack can only work with $\delta > 3 + \sqrt{c}$.

**Step 2(ii):** We lower-bound $\Pr[\text{ROR-CDLA}^1_{\Pi,\ell_m,A,S,q}(\lambda) = 1]$, i.e., the probability that $A$ guesses correctly that the queries in $\hat{Q}$ are sampled from a uniform distribution.

Once again, $A$ outputs 1 if all keys receive more than $(c + 3\sqrt{c})\ln\hat{h}$ queries, so we represent the number of queries to a key in EKV with a binomial random variable. We then use the Chernoff bound to compute the upper bound of the probability that all keys receive $\geq (c + 3\sqrt{c})\ln\hat{h}$ queries, which we subtract from 1 to obtain the desired lower bound on $\Pr[\text{ROR-CDLA}^1_{\Pi,\ell_m,A,S,q}(\lambda) = 1]$. Again, while its derivation is deferred to Appendix B.1, we have the lower bound:

$$\Pr[\text{ROR-CDLA}^1_{\Pi,\ell_m,A,S,q}(\lambda) = 1] \geq 1 - \hat{h}^{-2}. \quad (3)$$

Finally, we compute $A$’s advantage using (2) and (3):

$$\text{Adv}^{\text{ROR-cdla}}_{\Pi,\ell_m,A,S}(\lambda) = |\Pr[\text{ROR-CDLA}^0_{\Pi,\ell_m,A,q}(\lambda) = 1] - \Pr[\text{ROR-CDLA}^1_{\Pi,\ell_m,A,S,q}(\lambda) = 1]|$$

$$\geq \left| 1 - \frac{1}{\hat{h}^2} - \frac{(1 - \varepsilon)(\hat{h}(1 - \delta) - (1 - \varepsilon))}{(1 - \varepsilon)(\hat{h}(1 - \delta) - (1 - \varepsilon)) + c\ln\hat{h} \left[ \frac{(1 - \varepsilon - \frac{1}{\hat{h}} - \frac{1}{\hat{h}})}{\hat{h}(1 - \delta)^2} \right]} \right|$$

$$= f(\hat{h}, \delta, \varepsilon)$$

giving us our lower-bound function $f$ on the advantage. We let $A$ set $\varepsilon \to 0$ and $c = \hat{h}^2$, giving us:

$$f(\hat{h}, \delta) = \left| 1 - \hat{h}^{-2} - \frac{\hat{h}(1 - \delta) - 1}{\hat{h}(1 - \delta) - 1 + \frac{\hat{h}^2\ln\hat{h} \left[ \frac{(1 - \varepsilon - \frac{1}{\hat{h}} - \frac{1}{\hat{h}})}{\hat{h}(1 - \delta)^2} \right]}{\hat{h}(1 - \delta)^2}} \right|$$

Figure 5 shows that for large $\hat{h}$, the bound on $\delta > \frac{3 + \sqrt{c}}{\hat{h}^2}$, for which $A$’s attack works becomes vanishingly small, and $\hat{h}(\varepsilon, \delta)$ is a nonzero value that rapidly approaches 1. Since $f$ represents a lower bound on $\text{Adv}^{\text{ROR-cdla}}_{\Pi,\ell_m,A,S}(\lambda)$, our adversary $A$ has a non-zero advantage, completing the proof. \hfill \Box

Observe that as $n$ grows large, the lower-bound converges to $\frac{\kappa_{\text{max}}}{\kappa}$, which depends solely on the storage footprint since $\kappa = \frac{\sigma}{\kappa_{\text{max}}}$. This highlights a fundamental trade-off between storage and bandwidth footprints: an optimal $\mathcal{ML}_m$ scheme can leverage more storage to lower the bandwidth footprint.

### 3.2 Padded PANCAKE: Optimal $\mathcal{ML}_m$ scheme

We now present a $\mathcal{ML}_m$ scheme, Padded PANCAKE (Figure 6), defined as $\text{PPC} := (\text{Init}_{\text{PPC}}, \text{Query}_{\text{PPC}})$ and adapted from PANCAKE [6] to the variable-length setting. We begin with a summary of PANCAKE’s approach and then describe Padded PANCAKE’s key differences.

PANCAKE leverages knowledge of the access distribution $\pi$ to ensure an adversary observes a uniform access distribution over the encrypted KV pairs. First, it selectively replicates KV pairs with high access probability while ensuring the total number of keys in the encrypted store is exactly $2 \cdot n$ (adding fake replicas if necessary). This smooths out the access distribution over the resulting encrypted KV pairs to some extent; the remaining non-uniformity is removed by injecting fake queries. To this end, it computes a fake access distribution $\pi_f$ over the replicated KV pairs required to ensure uniform distribution, assuming the proportion of real queries is $\delta$ (set to 1/2 by default). All encrypted queries are sent in batches of size $B = 3$ to ensure that the adversary cannot distinguish fake queries from real ones. PANCAKE thus secures the KV store against access pattern attacks with a $3 \times$ bandwidth overhead and $2 \times$ storage overhead but assumes that the store contains only uniform-size values.

Padded PANCAKE generalizes PANCAKE to support values of arbitrary sizes. At a high level, it pads every key to the largest value size, $s_{\text{max}}$, and then simply applies PANCAKE to the set of padded KV pairs. More specifically, given a total storage footprint $\sigma = \kappa \cdot n \cdot s_{\text{max}}$, its initialization function, $\text{Init}_{\text{PPC}}$, creates $\hat{h} = \frac{\sigma}{s_{\text{max}}} = \lfloor \kappa n \rfloor$ values in the output EKV. Of these values, $n$ contain one of each value $v \in$ KV padded to length $s_{\text{max}}$ and encrypted. Of the remaining $\hat{n} - n$ values, some hold padded and encrypted replicas of values in KV that have access probability $> \frac{1}{n + 1}$, while the rest are dummy replicas to ensure that the number of values in EKV is exactly $\hat{n}$. Query$_{\text{PPC}}$ generates a sequence of $q \cdot B$ real and
fake queries to ensure a uniform access distribution across $k \in \text{Keys}(\text{EKV})$.

\begin{verbatim}
Init_{PCP}(\text{KV}, S, n, \pi, \sigma):
   s_{max} \leftarrow \max_{i \in \text{KV}} s_i
   n \leftarrow |\text{KV}||\pi|/s_{max}
   EKV \leftarrow 0, r \leftarrow 0
   For (k, v) \in \text{KV}:
      R(k) \leftarrow \{\pi(k)/(\delta - n + 1)\}
   For j \in [1, \ldots, R(k)]:
      k_j \leftarrow k
      \sigma(k_j) \leftarrow \text{pad}(\pi(k_j), s_{max})
      \hat{P}(k_j) \leftarrow \{k_j\}
      \sigma_i(k_j) = \frac{\sigma_i(k_j)}{|\text{KV}|}
      EKV \leftarrow EKV \cup \{\text{EKV} \leftarrow \hat{P}(k_j), \sigma(k_j), \hat{\pi}(k_j)\}
   r \leftarrow r + |R(k)|
   For j \in [1, \ldots, r - r):
      k_{i+1,j} \leftarrow \sigma_i^R(k_{i,j})
      \sigma_i(k_{i+1,j}) = 1/\hat{n}
      EKV \leftarrow EKV \cup \{\text{EKV} \leftarrow \hat{P}(k_{i+1,j}), \sigma(k_{i+1,j})\}
   \delta \leftarrow (\delta - n + 1)/\hat{n}, B \leftarrow 1/\delta
   Return EKV, P, \pi, \sigma, B.

Query_{PCP}(Q, P, \pi, \sigma, \hat{\pi}, S, R):
   \hat{Q} \leftarrow Q, \hat{\sigma} \leftarrow \hat{\pi}(Q)
   For k \in \text{Keys}(\text{EKV}):
      \hat{\pi}(k) = \frac{\sigma_i(k)}{|\text{KV}|}
   \hat{\pi}(k) \leftarrow \hat{\pi}(k)
   AddToQueue(k)
   For i = 1 to \hat{Q}:
      \text{QueueNotEmpty} \leftarrow \{0, 1\}
      \text{QueueNotEmpty} \leftarrow 1
      \hat{k} \leftarrow \text{Dequeue}()
      If \text{QueueNotEmpty} = 0:
         \hat{k} \leftarrow \hat{k}
      \hat{k} \leftarrow \hat{k} \cup \text{EKV} \leftarrow \hat{P}(k)
   Return \hat{Q}
\end{verbatim}

Figure 6: Padded PANCAKE’s Init and Query algorithms. The function pad(x, s) returns string x padded to size s.

Next, we show that PPC is bandwidth-optimal, i.e., it achieves the bandwidth lower bound given by Theorem 3.1.

**Theorem 3.2.** For any storage footprint $\sigma = \kappa \cdot n \cdot s_{max}$ ($\kappa \geq 1$), PPC is a bandwidth-optimal $\mathcal{M}_m$ scheme.

**Proof.** By construction in Figure 6, $\text{Init}_{PCP}$ sets $B = \frac{n}{\kappa \cdot n}$, while ensuring $\pi$, $\sigma$ and $\delta$ are always valid for any $\pi$. As such, PPC has $\beta = B \cdot s_{max} = \frac{|\text{KV}| \cdot s_{max}}{|\text{KV}| - N_{OPT} + 1}$, equal to the lower bound given by Theorem 3.1. \hfill $\square$

PPC’s bandwidth optimality demonstrates the viability of PANCAKE-based solutions in $\mathcal{M}$ — simply padding values in PANCAKE enables an optimal scheme in $\mathcal{M}_m$ that runs in polynomial time. However, its bandwidth overheads may not be practical for many storage scenarios (e.g., for the Wikipedia dataset with large variance in data sizes, §1.1). We explore relaxed leakage profiles for lower bandwidth footprints next.

4 $\mathcal{L}_S$: Revealing the value sizes

Now we consider the leakage profile $\mathcal{L}_S = (S)$, i.e., the adversary learns the sizes of values in KV. Since schemes in $\mathcal{M}_S$ do not leak the access distribution ($\pi$), their bandwidth and storage footprint must be independent of $\pi$. However, leaking $S$ allows us to consider new constructions not in $\mathcal{M}_m$ — those whose bandwidth and/or storage footprint may depend on $S$, e.g., the packing example in Figure 2b. We note that it may be possible to leak more information than $s_{max}$ and less than $S$, e.g., the sum of value sizes ($\sum s_i$), as explored in recent work [33]. However, we restrict our focus on leaking $S$ since there are inputs for which $\sum s_i$ directly reveals $S$, e.g., if there are $n - 1$ values of size $s_{max}$ and one value of size $s_{max} - 1$, knowing $\sum s_i = n \cdot s_{max} - 1$ immediately reveals $S$. Moreover, our bandwidth lower bounds for leaking $S$ hold for leaking $\sum s_i$. Indeed, our findings complement the recent work, which investigates size-based leakage in greater depth.

4.1 Performance Bounds

The following theorem presents the minimum bandwidth footprint for any scheme $\Pi \in \mathcal{M}_S$ with storage footprint $\sigma = \kappa \cdot n \cdot s_{max}$. Our proof is similar to that of Theorem 3.1: since a scheme in $\mathcal{M}_S$ must have the same bandwidth footprint independent of $\pi$, we construct a worst-case $\pi$ to maximize the bandwidth footprint, giving us a lower-bound. However, the relaxed leakage profile allows this lower-bound to be lower than that of Theorem 3.1, depending on how well values in KV can be packed into $s_{max}$-sized bins.

**Theorem 4.1.** Let $N_{OPT} \leq n$ be the minimum number of $s_{max}$-sized bins needed to bin-pack all values in KV. Any $\Pi \in \mathcal{M}_S$ with storage footprint $\sigma = \kappa \cdot n \cdot s_{max}$ ($\kappa > 0$) and $\delta = s_{max}$ or $\delta \geq |\text{KV}| \cdot s_{max}$ must have bandwidth footprint $\beta \geq |\text{KV}| - N_{OPT} + 1 \cdot s_{max}$.

**Proof.** Suppose for contradiction that $\Pi$ has storage footprint $\sigma = \kappa \cdot n \cdot s_{max}$ and bandwidth footprint $\beta = (1 - \delta) \cdot \frac{|\text{KV}| - N_{OPT} + 1 \cdot s_{max}}{\|\text{KV}\| - N_{OPT} + 1}$. Let $\mathcal{L}_S$-secure. Since our proof approach is similar to the proof of Theorem 3.1, we omit common aspects while outlining the salient differences.

Let $\mathcal{L}_S$-adversary $A$ be defined as in Figure 3 with a single modification: instead of $S \leftarrow \{s_{max}, \ldots, s_{max}\}$, $A_1$ sets $S \leftarrow \{s_1, \ldots, s_n\}$, which is a list of arbitrary sizes sorted in descending order. The order ensures that the key with the largest value size has access probability $1 - \epsilon$, which is crucial to $A$’s approach. We will show that given the large, heavily accessed key and limited range for $\delta$, $\Pi$ cannot maintain a uniform access distribution over EKV with the assumed bandwidth footprint in Step 1. $A_2$ can then use the same threshold test from Figure 3 to distinguish between $\mathcal{ROR}_{\mathcal{M}_S, A, \delta}^0(\lambda)$ and $\mathcal{ROR}_{\mathcal{M}_S, A, \delta}^1(\lambda)$ in Step 2.

**Step 1:** Given $A$’s choice of inputs, $\Pi$’s output EKV has a non-uniform access distribution. $A_1$ generates KV of variable-sized values, where the largest value is also most heavily accessed. The key for this value (say, $h$) has access probability $1 - \epsilon$, and the remaining $n - 1$ keys have access probability $\frac{\epsilon}{n}$. We will show that one of the keys in EKV that maps to $h$ must have access probability $> 1/\hat{n}$.

Since $\Pi \in \mathcal{M}$, by Property 1, each value $\hat{v} \in \text{Values}(\text{EKV})$ has size $\delta$ and by Property 2, $|\text{EKV}| = \hat{n} = \kappa s_{max}/\delta$. To allow $\Pi$ to use as much storage as possible, we set $\hat{n} = \lfloor \kappa s_{max}/\delta \rfloor$. For Property 3, $\Pi$ maps each $k \in \text{Keys}(\text{KV})$ to a nonempty
Again, minimizing bandwidth footprint (which is proportional to $B$) requires minimizing $|P(h)|$, and we consider the following two cases for it:

**Case 1:** $\delta \geq \frac{|\kappa|}{|\kappa| - N_{OPT} + 1}$. Then $\hat{\beta} = B \cdot \delta = \frac{(1 - \delta) \cdot |\kappa| \cdot s_{max}}{|\kappa| - N_{OPT} + 1}$ so we have:

$$B \geq \frac{1 - \delta}{\delta} \cdot \frac{|\kappa| \cdot s_{max}}{|\kappa| - N_{OPT} + 1} \leq (1 - \delta) < 1$$

violating the constraint $B \geq 1$, making this case impossible.

**Case 2:** $\hat{\delta} = s_{max}$. Then $\hat{n} = |\kappa|$. Since $h$ has size $s_{max}$, the values of each $h \in P(h)$ may only contain the value corresponding to $h$, and nothing else. Then to maximize $|P(h)|$, $\Pi$ must minimize the number of values that contain values of the remaining keys $\text{Keys}(KV) \setminus \{h\}$. This minimum number is equal to the optimal number of $s_{max}$-sized bins needed to bin-pack the values of $\text{Keys}(KV) \setminus \{h\}$, which is $N_{OPT} - 1$. Therefore, $|P(h)| \leq \hat{n} - (N_{OPT} - 1) = |\kappa| - N_{OPT} + 1$. Substituting this into (4), we get:

$$p \geq \frac{1 - \delta}{B(|\kappa| - N_{OPT} + 1)}$$

Rearranging (5) gives us $\frac{1}{|\kappa| - 1 - \delta} = \frac{1}{p|\kappa| - 1 - N_{OPT} + 1}$ for $\hat{\delta} = s_{max}$, which simplifies the above lower bound on $p$ to:

$$p \geq \frac{1}{|\kappa| \cdot \frac{1 - \delta}{1 - \delta} - \hat{n}} = \frac{1}{|\kappa|} \frac{1 - \epsilon}{\hat{n}}$$

EKV then contains a key with access probability $p = \frac{1}{|\kappa|} \frac{1 - \epsilon}{\hat{n}}$, which is the same result as in Theorem 3.1. While the value of $\hat{n}$ may differ, $A$ can still choose an $\epsilon$ such that $p > \frac{1}{|\kappa|}$, giving a non-uniform distribution. Once again, a computationally unbounded adversary can distinguish between $\Pi$'s non-uniform access distribution and a uniform access distribution with a sufficiently-long $\hat{Q}$. However, ROR-CDLA security requires $A_2$ (Figure 3) to distinguish between the distributions in polynomial time. Specifically, we must show that $A$ has non-negligible advantage $Adv_{\Pi \in \mathcal{M}_{S}, \kappa, A, \hat{S}}(\hat{\lambda})$ against $\Pi$ for all simulator algorithms $\hat{S}$. Fortunately, the approach for showing this is identical to Step 2 in the proof of Theorem 3.1, since $A_2$ uses the exact same threshold-based test; we avoid repeating it for brevity.

**Theorem 4.1** imposes a bandwidth lower bound on $\mathcal{M}_{S}$ schemes, but only for $\hat{\delta} = s_{max}$ and $\hat{\delta} \geq \frac{|\kappa| \cdot s_{max}}{|\kappa| - N_{OPT} + 1}$. The bound does not hold between these two sizes, as values in EKV are large enough to benefit from a more strategic placement of KV values across EKV values. We confirm this intuition with a simple counterexample.

**Lemma 3.** There exists a scheme $\Pi \in \mathcal{M}_{S}$ with $s_{max} < \hat{\delta} < \frac{|\kappa|}{|\kappa| - N_{OPT} + 1}$ and input $(KV, S, \pi, \sigma)$ for which $\Pi$ has bandwidth footprint $\hat{\beta} < \frac{|\kappa|}{|\kappa| - N_{OPT} + 1} \cdot s_{max}$.

**Proof.** Let $\text{Keys}(KV) := \{k_1, k_2\}$ with value sizes $S := \{10, 1\}$ and $\sigma = 22$, i.e., $\kappa = \sigma / s_{max} = 2.2$. The minimum number of $s_{max}$-sized bins needed to bin-pack the values of $KV$ is $N_{OPT} = 2$, so Theorem 4.1 gives the bandwidth lower bound $\hat{\beta} = \frac{|\kappa|}{|\kappa| - N_{OPT} + 1} \cdot s_{max} = 20$. Now we give a scheme $\Pi$ with $10 < \hat{\delta} < 20$ and bandwidth footprint $\hat{\beta} < 20$ for all $\pi$. Let $\Pi$ have $\hat{\delta} = 11$, making $\hat{n} = 2$. Let each value in EKV contain a replica of both values in KV. Then $\Pi$ can equally divide access probabilities across EKV, i.e., $\forall k \in \text{Keys}(KV), \forall k \in \text{Keys}(EKV), \pi_r[k] = \pi(k) / 2$, which means both keys in EKV have access probability $1/2$. No additional queries are needed to smooth the access distribution, so $\Pi$ can set $\hat{B} = 1$, making $\hat{\beta} = B \cdot \hat{\delta} = 11 < 20$. $\square$

In this example, a small increase in $\hat{\delta}$ above $s_{max}$ vastly improves space utilization. The possibility of this occurrence depends on $\hat{n}, \hat{\delta}$, and the corresponding bin-packing. For large $n$, however, such cases should be rare, and Theorem 4.1’s bound should hold for most $\hat{\delta}$.

Next, we introduce a polynomial-time $\mathcal{M}_{S}$ scheme that achieves the bound outlined in Theorem 4.1 for $\hat{\delta} = s_{max}$.

**4.2 Stuffed PANCAKE: Optimal $\mathcal{M}_{S}$ scheme**

Our proposed scheme in $\mathcal{M}_{S}$ exploits revealing $S$ by bin-packing the values in KV into equal-sized bins, and then using PANCAKE to equalize access probabilities across bins.

![Pancake pseudocode](image)
then leverages \( \text{Init}_{\text{SPC}} \) with storage footprint \( \sigma = [\kappa n] \cdot s_{\max} \) to generate the final EKV, \( \pi, \delta, B \), and post-processes outputs \( P \) and \( \pi_e \) to ensure correct mappings from keys in EKV. Query_{\text{SPC}} simply uses Query_{\text{PPC}} to generate queries to EKV. Note that while we use \( \hat{s} = s_{\max} \) in our description of SPC, it can easily use \( \hat{s} > s_{\max} \). We now show that SPC achieves Theorem 4.1’s lower bound.

**Theorem 4.2.** For any storage footprint \( \sigma = \kappa \cdot n \cdot s_{\max} (\kappa \geq 1) \), SPC is a bandwidth-optimal \( \mathcal{ML}_\pi \) scheme.

**Proof.** In \( \text{Init}_{\text{SPC}} \), OptBinPack generates KV’ containing \( N_{\text{OPT}} \) values of size \( s_{\max} \) each. This KV’ is then input to \( \text{Init}_{\text{PPC}} \) along with storage \( [\kappa n] \cdot s_{\max} \); since Query_{\text{SPC}} uses the unmodified Query_{\text{PPC}}, SPC’s bandwidth footprint is the same as PPC’s, except with \( n := N_{\text{OPT}} \) and \( \sigma := [\kappa n] \cdot s_{\max} \). Substituting these values into Theorem 3.1, we have that SPC has \( \hat{\beta} = \frac{\hat{s}}{\hat{s} - N_{\text{OPT}} - 1} \cdot s_{\max} = \frac{[\kappa n]}{[\kappa n] - N_{\text{OPT}} + 1} \cdot s_{\max} \), equal to the bound in Theorem 4.1. \( \square \)

Theorem 4.2 shows that SPC’s freedom to leak size distribution permits it to achieve a lower bandwidth footprint compared to PPC for the same storage footprint.

**Approximating** OptBinPack. Optimal bin-packing is an NP-hard problem; however, we can use near-optimal approximate algorithms that run in polynomial time, e.g., First-Fit Decreasing (FFD) which uses \( \frac{1}{1 + n^{1/2}} \) bins [42].

5 \( \mathcal{L}_\pi \): Revealing the access distribution

Our next leakage profile is \( \mathcal{L}_\pi = (s_{\max}, \pi) \). Like \( \mathcal{L}_S, \mathcal{L}_\pi \) is a relaxation of \( \mathcal{L}_m \) but allows new constructions that are distinct from those under both \( \mathcal{L}_m \) and \( \mathcal{L}_S \). Since schemes in \( \mathcal{ML}_\pi \) do not leak value sizes \( S \), their storage and bandwidth footprints must be independent of \( S \); however, they can depend on the access distribution \( \pi \). For instance, the packing approach of Figure 2b is not in \( \mathcal{ML}_\pi \) for the same reason that it is not in \( \mathcal{ML}_m \); it reveals information about the value sizes. However, the scheme in Figure 2c is in \( \mathcal{ML}_\pi \) because the access distribution can be revealed.

Unlike prior sections, we do not have a closed-form lower-bound on bandwidth footprint for \( \mathcal{ML}_\pi \); we instead directly provide a bandwidth-optimal \( \mathcal{ML}_\pi \) scheme.

5.1 Greedy PANCAKE: Optimal \( \mathcal{ML}_\pi \) scheme

We now present a scheme in \( \mathcal{ML}_\pi \) that greedily replicates values with the highest access probabilities, then smooths the access distribution across the replicas with fake queries.

Let Greedy PANCAKE be \( \text{GPC} := (\text{Init}_{\text{GPC}}, \text{Batch}_{\text{GPC}}) \). Given \( \sigma = \kappa \cdot n \cdot s_{\max} \) storage, \( \text{Init}_{\text{GPC}} \) starts with an array \( R \) of \( n \) elements, where \( R[i] = |P(k_i)| \) is the total number of keys \( k_i \in \text{Keys}(EKV) \) that key \( k_i \in \text{KV} \) is mapped to. The access probability of \( k_i \) is split equally across the keys in \( P \), i.e., for all \( k_i \in P, \pi_{\text{Init}}(k_i)(k_i) = \pi(k_i) / |R[i]| \). Initially \( R[i] = 1 \) for all \( k_i \); for each of the remaining \( n - k \) keys in EKV, the scheme assigns to it the key \( k_i \) with the highest split access probability \( \pi(k_i) / |R[i]| \) at that point. Like in previous PANCAKE-based schemes, \( \text{Init}_{\text{GPC}} \) also computes \( \pi, \delta, B \) to be used in the query algorithm \( \text{Query}_{\text{GPC}} := \text{Query}_{\text{PPC}} \).

**Figure 8:** Greedy PANCAKE’s Init & Query algorithms. We omit Query_{\text{GPC}} arguments since they are the same as Query_{\text{PPC}}’s.

Theorem 5.1 shows that Greedy PANCAKE is bandwidth-optimal if the entire storage footprint \( \sigma = \kappa \cdot n \cdot s_{\max} \) must be used. We defer the proof to Appendix B but summarize our approach here in two steps. First, since the bandwidth footprint of all schemes in \( \mathcal{ML}_\pi \) must be independent of \( S \), by Lemma 2, increasing the size \( \hat{s} \) of values in EKV beyond \( s_{\max} \) is not beneficial to reducing bandwidth footprint for schemes in \( \mathcal{ML}_\pi \). Second, we show that Greedy PANCAKE using \( \hat{s} = s_{\max} \) is bandwidth-optimal using the “greedy stays ahead” argument: for each of the \( n - k \) keys in EKV to which GPC assigns a key from KV, the resulting maximal split access probability \( \hat{\beta}_{\max} = \max_{k_i \in \text{Keys}(KV)} \pi(k_i) / |R[i]| \) is at most that of an optimal algorithm at the same step.

**Theorem 5.1.** For any storage footprint \( \sigma = \kappa \cdot n \cdot s_{\max} (\kappa \geq 1) \), GPC achieves the lowest bandwidth footprint \( \hat{\beta} \) for any \( \mathcal{ML}_\pi \) scheme that uses exactly \( \sigma \) storage.

Intuitively, since GPC can reveal the access distribution, it allocates storage to values proportional to their access probability, maximally utilizing the given storage footprint to minimize bandwidth footprint. A caveat for Greedy PANCAKE, however, is that it is only bandwidth-optimal among schemes that must use all of the storage given, i.e. *exactly* \( \kappa \cdot n \cdot s_{\max} \) bytes. There are cases when using storage less than \( \kappa \cdot n \cdot s_{\max} \) results in a lower bandwidth footprint. Consider \( \pi = \left( \frac{1}{2}, \frac{1}{2}, \frac{1}{2} \right) \) and \( \sigma = \theta s_{\max} \); GPC creates six replicas, i.e., \( \hat{\beta} \geq \sigma / \hat{\beta}_{\max} = \frac{2}{3} s_{\max} \). However, one can get a uniform access distribution over EKV with just five replicas: assign 2 replicas each to the keys with...
We formulate the problem of assigning values \( \beta = s_{\max} \). With this intuition, we can modify GPC to achieve the minimum \( \beta \) for any \( \sigma \leq \kappa \cdot n \cdot s_{\max} \).

**GPC**. We define \( \text{GPC}^* = (\text{Init}_{\text{GPC}^*}, \text{Query}_{\text{GPC}^*}) \), where \( \text{Init}_{\text{GPC}^*} \) iterates through storage footprints \( \sigma \) from \( n \cdot s_{\max} \) to \( \kappa \cdot n \cdot s_{\max} \). For each \( \sigma \), \( \text{GPC}^* \) runs \( (\text{EKV}, P, \pi, \pi_f, \delta, B) \leftarrow \text{Init}_{\text{GPC}^*}(KV, S, \pi, \sigma) \) and tracks a running minimum on bandwidth footprint across the outputs from \( \text{Init}_{\text{GPC}^*} \), and returns the \((\text{EKV}, P, \pi, \pi_f, \delta, B)\) for the global minimum.

**Theorem 5.2.** For any storage footprint \( \sigma = \kappa \cdot n \cdot s_{\max} (\kappa \geq 1) \), \( \text{GPC}^* \) is a bandwidth-optimal \( M \mathcal{L}_{\pi} \) scheme.

**Proof.** This follows from Theorem 5.1 and \( \text{GPC}^* \)'s use of GPC in its exhaustive search for the optimal bandwidth footprint over storage footprints from \( n \cdot s_{\max} \) to \( \kappa \cdot n \cdot s_{\max} \).

### 6 \( \mathcal{L}_{S\pi} \): Revealing value sizes & distribution

Our last considered leakage profile is \( \mathcal{L}_{S\pi} = (S, \pi) \), where the adversary may know both the value sizes \( S \) and access distribution \( \pi \) over KV. As the most relaxed leakage profile considered, \( \mathcal{L}_{S\pi} \) allows for a much larger solution space than the previous leakage profiles. For instance, both packing (Figure 2b) and replication (Figure 2c) can be used by schemes in \( M \mathcal{L}_{S\pi} \). Unfortunately, the search space for a performance-optimal scheme in \( M \mathcal{L}_{S\pi} \) is too large, and we currently do not have a polynomial time algorithm that achieves the performance lower-bound. However, we provide a MILP formulation that yields a bandwidth-optimal construction (§6.1), along with Greedy Stuffed PANCAKE (§6.2), a sub-optimal poly-time algorithm in \( M \mathcal{L}_{S\pi} \) that is efficient in practice.

#### 6.1 MILP-based optimal \( M \mathcal{L}_{S\pi} \) scheme

We formulate the problem of assigning values \( v \in \text{Values}(KV) \) and access probabilities to values \( \hat{v} \in \text{Values}(	ext{EKV}) \) to minimize bandwidth footprint as an MILP.

For a key \( k \in \text{Keys}(KV) \), let \( \hat{p}_{k,k} \) be the portion of its real access probability \( \pi(k) \) assigned to \( \hat{k} \in \text{Keys}(	ext{EKV}) \); note that if \( \hat{k} \notin P(k) \) then \( \hat{p}_{k,k} = 0 \), and \( \sum_{k \in \text{EKV}} \hat{p}_{k,k} = \pi(k) \).

Let \( \hat{p}_k = \sum_{k \in \text{EKV}} \hat{p}_{k,k} \), i.e., the summed access probabilities of the replicas of keys \( k \in \text{Keys}(KV) \) assigned to a key \( \hat{k} \in \text{Keys}(	ext{EKV}) \). Let \( \hat{p}_{\max} = \max_{k \in \text{EKV}} \hat{p}_k \), and \( \kappa^* \) be the corresponding key in \( \text{Keys}(	ext{EKV}) \). The objective function for our MILP is given below. We prove that it lower-bounds the bandwidth footprint in Lemma 5 (Appendix B).

\[
\text{minimize } \hat{n} \cdot \hat{s} \cdot \hat{p}_{\max} \tag{6}
\]

Our goal is subject to the following constraints:

(a) The real access probabilities must be between 0 and 1.
(b) The real access probabilities across the replicas of a key \( k \in KV \) must sum to \( k^* \)'s real access probability.

(c) A key \( k \in KV \) assigned to \( \hat{k} \in EKV \) via \( P \) should have non-zero access probability only if \( k \in P(k) \).

(d) Sum of value sizes \( s_i \in S \) corresponding to keys \( k_i \in KV \) assigned to \( \hat{k} \in EKV \) via \( P \) must not exceed \( \hat{s} \).

Let \( y_{k,\hat{k}} \in \{0,1\} \) be a decision variable that denotes whether or not key \( k \in KV \) mapped to key \( \hat{k} \in EKV \) via \( P \). Then the above constraints can be encoded in the MILP as follows:

\[
\begin{align*}
\forall k \in KV, \hat{k} \in EKV, & \quad 0 \leq \hat{p}_{k,\hat{k}} \leq 1 \tag{a} \\
\forall \hat{k} \in EKV, & \quad \sum_{k \in \text{EKV}} \hat{p}_{k,\hat{k}} = \pi(k) \tag{b} \\
\forall k \in KV, \hat{k} \in EKV, & \quad \hat{p}_{k,\hat{k}} \leq y_{k,\hat{k}} \cdot \pi(k) \tag{c} \\
\forall \hat{k} \in EKV, & \quad \sum_{k \in \text{EKV}} y_{k,\hat{k}} \cdot s_i \leq \hat{s} \tag{d}
\end{align*}
\]

The above formulation is not linear: since \( \hat{p}_{\max} \) is the maximum of \( \hat{n} \) linear functions of decision variables, the goal (6) contains a min-max term. However, this term can be linearized through standard techniques [43], making it solvable. Also, since the formulation does not ensure Property 4, we adapt our MILP scheme below to fulfill this requirement.

For each storage capacity \( \kappa \cdot n \cdot s_{\max} \) and feasible pack size \( \hat{s} \) ranging from \( s_{\max} \) to \( s_{\max} = \sum_i s_i \), \( \text{Init}_{\text{MILP}} \) runs a program solver \( \text{Solve}(\cdot) \) on the problem formulation. After choosing the bandwidth-optimal pack size, \( \text{Init}_{\text{MILP}} \) computes \( (\text{EKV}, P, \pi_f) \) according to the solver output, and \((\pi_f, \delta, B)\) based on \( \text{Init}_{\text{GPC}^*} \). Like GPC (§5.1), MILP uses exactly the storage footprint \( \sigma \) given. As with GPC, this has a simple fix: run \( \text{Init}_{\text{MILP}} \) for each feasible storage footprint from \( s_{\max} \) to \( \sigma \) for a bandwidth-optimal scheme for all \( \sigma \leq \kappa \cdot n \cdot s_{\max} \). The algorithm definitions are deferred to the full version [44].

**Theorem 6.1.** For any storage footprint \( \sigma = \kappa \cdot n \cdot s_{\max} (\kappa > 0) \), MILP is a bandwidth-optimal \( M \mathcal{L}_{S\pi} \) scheme.

**Proof.** Optimality follows from the objective function of MILP, while feasibility follows from its constraints.

While the MILP enables an optimal \( M \mathcal{L}_{S\pi} \) solution, it is an NP-hard problem. We show a sub-optimal but polynomial-time construction in \( M \mathcal{L}_{S\pi} \) next.

#### 6.2 PANCAKE-based poly-time \( M \mathcal{L}_{S\pi} \) scheme

Greedy Stuffed PANCAKE (GSPC) is a combination of SPC and GPC**: \( \text{Init}_{\text{GSPC}} \) first packs the values of KV into \( s_{\max} \)-sized bins, then passes the result to \( \text{Init}_{\text{GPC}^*} \), while \( \text{Query}_{\text{GSPC}} \) is the same as \( \text{Query}_{\text{GPC}^*} \) (see the full version [44] for pseudocode). While GSPC runs in polynomial time, it is sub-optimal:

**Lemma 4.** There exists a scheme \( \Pi \in M \mathcal{L}_{S\pi} \) and input \((KV, S, \pi, \sigma)\) for which \( \Pi \) has bandwidth footprint \( \beta \) lower than GSPC’s.
We present our new security model ROR.

The counterexample highlights the key shortcomings of sampling \( q \) and \( L \) instead of the inverse of any polynomial.

A scheme is secure if the advantage \( \pi \) on input \( \sigma \) is delineated by \( \{(4,0,1), \{1,0,2,3,0,1\}, \{(0,1,5), (2,0,15)\}, \{(1,0,15), (2,0,15)\}\). Since \( \rho_{\max} = 0.3 \), \( \Pi \) can set \( B = \hat{n} \times 0.3 = 1.2 \) to get \( \hat{\beta} = 4.8 < 6.4 \).

The counterexample highlights the key shortcomings of GSPC: it cannot (i) replicate individual values in KV, or (ii) perform non-uniform load balancing across replicas. Nonetheless, as shown in §1.1 GSPC is bandwidth-efficient for large \( n \).

7 Formal Security Analysis using ROR-CDLA

We present our new security model ROR-CDLA, which builds on prior work [6] to capture schemes in \( M \) as defined in §2.2. It extends prior work’s ROR-CDA model [6], which captures security against access pattern attacks by a passive persistent adversary without accounting for length leakage attacks.

Technical preliminaries. All algorithms including adversaries are assumed to run in time polynomial in the security parameter, denoted \( \lambda \). A scheme is secure if the advantage gained by any adversary is negligible, i.e., tends to zero faster than the inverse of any polynomial.

Definition 1 (ROR-CDLA with simulator).

Let \( \Pi = (\text{Init, Query}) \) be an encrypted storage scheme as defined in §2 and let \( L \) be a leakage function. For algorithms \( A \) and \( S \), we define the two games:

\[
\text{ROR-CDLA}^0_{\Pi, L, A, S}(\lambda): (KEV, S, \pi, \sigma) \rightarrow A(1^\lambda) \text{ picks inputs } (KV, S, \pi, \sigma).
\]

The game then runs \( (KEV, P, \pi, \pi, A, S) \rightarrow \Pi(KEV, S, \pi, \sigma) \), samples \( q \) queries \( (Q) \) from \( \pi \), and runs \( Q \rightarrow \text{Init}(KV, S, \pi, \sigma) \). The game stops when it outputs an answer to the query.

Figure 9: ROR-CDLA security game for key-value store schemes.

Proof. Let the input have sizes \( S = \{4, 3, 2, 1\} \), access distribution \((0.1, 0.1, 0.3, 0.5)\), and storage footprint \( \sigma = 16 \).

First, GSPC sets \( s = 4 \) and bin-packs the values into \( N_{OPT} = 3 \) bins \( \{(4), \{3, 2\}, \{1\}\} \) with access probabilities \((0.1, 0.1, 0.3, 0.5)\). GSPC then runs \( \text{Init}_{\text{GSPC}} \), which replicates the last bin, giving \( \{(4), \{3, 2\}, \{1\}\} \) with access probabilities \((0.1, 0.1, 0.3, 0.5)\). Then GSPC sets \( B = \hat{n} \times 0.4 = 1.6 \) to get \( \hat{B} = B \cdot \hat{n} = 6.4 \).

Now consider an alternate scheme \( \Pi \) that makes the following assignment, where each tuple \((s, p)\) denotes a value of size \( s \) with access probability \( p \), and separate values in EKV are delineated by \( \{\}: \{(4,0,1), (4,0,2), (3,0,1)\}, \{(0,1,5), (2,0,15)\}, \{(1,0,15), (2,0,15)\}\). Since \( \rho_{\max} = 0.3 \), \( \Pi \) can set \( B = \hat{n} \times 0.3 = 1.2 \) to get \( \hat{B} = 4.8 < 6.4 \).

Theorem 1. ROR-CDLA is secure if the advantage \( \pi \) on input \( \sigma \) is negligible, i.e., tends to zero faster than the inverse of any polynomial.

Figure 10: ROR-CDA vs ROR-CDLA. The second column shows for each leakage profile \( L \) whether or not any ROR-CDA-secure scheme is also in \( M \). The third column shows whether any scheme in \( M \) is also ROR-CDA-secure.

Comparison to ROR-CDA. While ROR-CDLA builds on ROR-CDA, it captures a distinct security goal. As shown in Figure 10, no ROR-CDA scheme is ROR-CDA-secure under any leakage profile: unlike ROR-CDA, ROR-CDLA security requires hiding individual value sizes during access.

For \( L_m \) and \( L_S \), all ROR-CDLA schemes are ROR-CDA-secure as they hide the access distribution \( \pi \), revealing only value size related information (\( s_{\max} \) and \( S \) respectively) during initialization. The similarity of the definitions means that \( L_m \) and \( L_S \)-secure schemes can be derived from an ROR-CDA scheme such as PANCake with minor transformations (i.e., to hide value sizes). The opposite holds for \( \pi \)-leaking profiles \( L_{\pi} \) and \( L_{\pi \delta} \), as ROR-CDA requires hiding access distribution during both access and initialization. While our secure constructions borrow selective replication and fake querying techniques from PANCake, they do not directly use PANCake.

We prove the security of our constructions across §3.2–§6.2 in the full version of our paper [44].

8 Discussion & Future Work

Our work opens up several directions for future research:
**Dynamic distributions.** Our analysis focused on data stores with static access patterns and value size distributions. While our lower bounds still hold for dynamic distributions, finding upper bounds is an important next step. For example, an adaptive alternative to MILP would be valuable, as MILP currently requires complete re-initialization should the KV store’s access or size distribution change. PANCAKE’s [6] adaptations to dynamic access distributions provide a starting point, but additional study is required for dynamic value sizes.

**Performance bounds for efficient schemes.** In §6.2 we introduced Greedy Stuffed PANCAKE scheme as a polynomial-time approximation of MILP that always finds a feasible, but possibly suboptimal, solution. An analysis of whether its performance is within a bounded factor of the optimal would be interesting for future research.

**Completeness of bounds.** As Lemma 3 has shown, the lower bound on bandwidth footprint for $\mathcal{ML}_S$ does not hold for $s_{max} < \delta < \left\lceil \frac{\log N_{OPT}}{\log \left( \frac{s_{MAX}}{s_{MAX}} \right)} \right\rceil$. This leaves a gap for which the lower bound on bandwidth footprint, as well as an optimal $\mathcal{ML}_S$ scheme, is unknown. We believe multi-dimensional bin-packing approaches, which take both access probability and value sizes into account, could provide answers for both.

**Active adversaries.** Our constructions allow us to characterize the storage, bandwidth, and security tradeoff in $\mathcal{M}$. However, $\mathcal{M}$ excludes active adversaries from our threat model. While our lower bounds for passive persistent adversaries also hold for active adversaries, our proposed constructions would require additional protections against injected queries. Applying length leakage analysis to oblivious access schemes designed for this stronger threat model, such as ORAMs, would be a promising direction.

**Compression.** $\mathcal{M}$ rules out schemes that reduce the value sizes in EKV via compression. However, such schemes may improve throughput at lower storage footprints and even achieve lower bandwidth than our bounds. Extending our analysis to compression schemes will further expand the discussion on the storage-bandwidth-security tradeoff.

9 Conclusion

We explored the problem of hiding length leakage in oblivious data access mechanisms with passive persistent adversaries. We developed novel security models with different leakage profiles and strong lower bounds on achievable storage and bandwidth footprint under them. We also presented provably secure schemes for each profile that are bandwidth-optimal.

**Acknowledgments**

We thank the anonymous reviewers and the shepherd for their valuable feedback on our paper. We also thank Thomas Ris-tenpart for many insightful discussions during this work. Our research is supported in part by NSF Awards CNS-2054957, CNS-2047220, CNS-2147946 and CNS-2112562.

**References**


A Additional Experimental Results

In addition to the Twitter and Wikipedia datasets, we also apply our key results from §1.1 to two workloads from Facebook’s Memcached deployments [26]: (i) the USR workload for user account status information, and (ii) the ETC workload for large-scale, general-purpose KV storage.

Figure 11 remains consistent with our theoretical results: all schemes observe a drop in bandwidth overhead as more storage is provided, and the strongest leakage profile incurs the highest bandwidth overhead.

Facebook-USR has a bimodal distribution of small value sizes that differ by only 5 bytes. This allows all leakage profiles to observe < 10× bandwidth overheads at a mere 1.3× storage overhead. In fact, the size distribution of this dataset is so extreme that revealing it does not bring performance benefits for data with near-uniform sizes; there is no arrangement of values that better utilizes storage compared to simply padding each value to \( s_{\text{max}} \). \( \mathcal{L}_m \) and \( \mathcal{L}_S \) thus have near-identical bandwidth overheads, as do \( \mathcal{L}_P \) and \( \mathcal{L}_{SP} \).

Meanwhile, Facebook-ETC has a long-tailed size distribution like Wikipedia’s, and exhibits similar bandwidth overhead trends — the minimum storage overhead for \( \mathcal{L}_m \) and \( \mathcal{L}_P \) leakage settings is over 1200×, and the bandwidth overhead for \( \mathcal{L}_m \) is more than 10^9. Bandwidth overheads under \( \mathcal{L}_S \) and \( \mathcal{L}_{SP} \) are much lower in comparison but still over 1000×, since hiding each value’s identity requires fetching for each query \( s_{\text{max}} \) bytes, which in Facebook-ETC is > 1000× larger than the average value size. This again confirms our intuition that long-tailed size distributions are more suited to \( \mathcal{L}_S \) or \( \mathcal{L}_{SP} \) leakage settings that avoid the costs of hiding value lengths.

As for access distribution skewness, the results match Figure 1: schemes that do not leak the access distribution (PPC and SPC) observe the same bandwidth overhead regardless of access skew across key-value pairs. For schemes that do leak access distribution (GPC and GSIPC), bandwidth overheads increase with the access skew due to noise injection to hide the identity of the most popular key.

B Additional Lemmas & Definitions

We present a bound on \( B \) for any scheme in \( \Pi \in \mathcal{M} \):

**Lemma 5.** For any scheme \( \Pi \in \mathcal{M} \), let the sum of the real access probabilities of keys \( k \in \text{Keys}(\Pi) \) contained in key \( \hat{k} \in \text{Keys}(\Pi_{EKV}) \) be \( \hat{\rho}_k = \sum_{s \in \mathcal{S}} \pi_s[k](\hat{k}) \). Let \( \hat{\rho}_{\text{max}} = \max_{s \in \mathcal{S}} \hat{k} \in \Pi_{EKV}(\hat{k}) \); then \( B \geq \hat{n} \cdot \hat{\rho}_{\text{max}} \).

**Proof.** Since \( \delta \) is the proportion of real accesses in \( \hat{Q} \) and \( B \) is the ratio of the number of queries in \( \hat{Q} \) compared to \( Q \), we use \( \delta \) to give a lower bound on \( B \). For Property 4, \( \Pi \) must choose \( \delta \) and \( \pi_s[k] \) such that the probability of accessing any \( k \in \Pi_{EKV} \) is \( 1/\hat{n} \), i.e.,

\[
\delta \cdot \hat{\rho}_k + (1 - \delta) \cdot \pi_f(\hat{k}) = \frac{1}{\hat{n}}
\]

or,

\[
\delta \cdot \sum_{k \in \Pi_{EKV}} \pi_s[k](\hat{k}) + (1 - \delta) \cdot \pi_f(\hat{k}) = \frac{1}{\hat{n}}
\]

For the probability \( \pi_f(\hat{k}) \) to have a valid non-negative value in the above equation for any \( \hat{k} \), \( \Pi \) must have \( \delta \leq \frac{1}{\hat{n} \sum_{s \in \mathcal{S}} \pi_s[k](\hat{k})} \leq \frac{1}{\hat{n} \cdot \hat{\rho}_{\text{max}}} \).

Summing Property 5’s inequality over all \( k \in \text{Keys}(\Pi) \):

\[
E(\text{real queries in } \hat{Q}) \leq E(\text{real queries in } Q)
\]

Since all queries in \( Q \) are real, and the expected number of real accesses in \( \hat{Q} \) is \( \delta \cdot |Q| \leq \delta \cdot Q \cdot B \), we have \( q \leq \delta \cdot Q \cdot B \), i.e., \( B \geq \frac{1}{\delta} \geq \hat{n} \cdot \hat{\rho}_{\text{max}} \).

We provide the proofs of Lemmas 1 and 2 (§3.1).

**Lemma 1.** Any \( \Pi \in \mathcal{M} \) with leakage \( \mathcal{L} \) and \( S \notin \mathcal{L} \) requires storage footprint \( \sigma \geq n \cdot s_{\text{max}} \).

**Proof.** Let us consider a scheme \( \Pi \) that has storage footprint \( \sigma < n \cdot s_{\text{max}} \) when possible, i.e., the sum of its input sizes \( \sum_{s \in \mathcal{S}} s_i < n \cdot s_{\text{max}} \). Against this scheme, we give an \( O(n) \)-time adversary \( A \) that, for all algorithms \( \delta \), has advantage \( \text{Adv}_{\mathcal{L} \subseteq \mathcal{M}, \mathcal{L}, \delta, \mathcal{A}, \mathcal{G}}(\lambda) = 1/2 \).

Now we construct \( A \). \( A_1 \) chooses value sizes from one of two distributions: \( S_0 = \{ s_{\text{max}}, \ldots, s_{\text{max}} \} \), i.e., all values have size \( s_{\text{max}} \); or \( S_1 = \{ s_{\text{max}}, 1, \ldots, 1 \} \), i.e., only one value has size \( s_{\text{max}} \), while the rest are size 1. The two size distributions are chosen such that for \( S_0 \) and \( S_1 \), \( \Pi \) will output EKV0 and EKV1 respectively — while EKV0 must have storage footprint \( \sigma \geq n \cdot s_{\text{max}} \) to fulfill Property 3, EKV1 has a lower storage footprint.

\( A_1 \) flips a coin \( b \sim \{0, 1\} \) and assigns value sizes corresponding to \( S_b \). It then outputs a KV with the chosen size distribution and uniform \( \pi \). \( A_2 \) takes as input EKV and \( \hat{Q} \). \( A_2 \) ignores \( \hat{Q} \), but checks if the storage footprint of EKV is less than \( n \cdot s_{\text{max}} \) bytes (i.e., \( b = 1 \)) or at least \( n \cdot s_{\text{max}} \) bytes (i.e., \( b = 0 \)). If the storage footprint matches the coin-flip \( b \), \( A_2 \) outputs 1. Otherwise, it outputs 0.

Suppose \( A \) is in \( \text{ROR-CDLA}^{0}_{\Pi, \mathcal{L}, \mathcal{A}, \mathcal{G}}(\lambda) \) interacting with the real \( \Pi \). If \( A \) chooses \( S_0 \), \( \Pi \) always outputs an EKV using \( n \cdot s_{\text{max}} \) bytes of storage. Similarly if \( A \) chooses \( S_1 \), \( \Pi \) always outputs an EKV using less storage, meaning \( \Pr[\text{ROR-CDLA}^{0}_{\Pi, \mathcal{L}, \mathcal{A}, \mathcal{G}}(\lambda) = 1] = 1 \).
As such, we consider the worst-case with probability \( A \) with the storage footprint that with a simulator \( S \) only knows the output of \( \Sigma \) on the KV given by \( A \), which is \( n \) and \( s_{\text{max}} \). Without access to \( b \), \( \Sigma \) can only guess whether \( A \) chose \( S_0 \) or \( S_1 \), and output pack sizes with the storage footprint that \( A \) expects. \( \Sigma \) guesses correctly with probability \( \frac{1}{2} \), so \( \Pr[\text{ROR-CDLA}_1^{\Pi, L, A, S, q} (\lambda) = 1] = \frac{1}{2} \).

Thus \( A \)'s advantage \( \text{Adv}_{\text{ror-cdla}} (\Pi, L, A, S, \gamma) = |1 - \frac{1}{2}| = \frac{1}{2} \). \( \square \)

**Lemma 2.** Given storage footprint \( \sigma = \kappa \cdot n \cdot s_{\text{max}} \), a scheme \( \Pi \in \mathcal{M} \) with leakage \( L \) and \( S \notin L \) that incurs the minimum possible bandwidth footprint must use \( \hat{s} = s_{\text{max}} \).

**Proof.** Let \( \Pi \) be given inputs \( KV, S, \pi, \sigma \). Since \( \Pi \) does not leak \( S \), it must incur the same bandwidth footprint for all \( S \). As such, we consider the worst-case \( S = \{s_{\text{max}}, \ldots, s_{\text{max}}\} \).

By Property 1, each value \( \hat{v} \in \text{Values}(EKV) \) has the same size, say \( \hat{s} = \gamma \cdot s_{\text{max}} \) for some \( \gamma \geq 1 \). Since all the values \( v \in \text{Values}(KV) \) are size \( s_{\text{max}} \), we can view each value \( \hat{v} \in \text{Values}(EKV) \) as a set of \( \gamma \) “slots”, where each slot can hold a copy of an \( s_{\text{max}} \)-sized input value \( v \in \text{Values}(KV) \).

By Property 3, each value in KV maps to at least one \( s_{\text{max}} \)-sized slot in EKV’s values. The goal now is to map each key in KV to one or more slots in a manner that incurs at most \( \hat{B} \) bandwidth footprint. Note that since every \( v \in \text{Values}(KV) \) is exactly size \( s_{\text{max}} \), non-integer values of \( \gamma \) would only result in wasted space in each value \( \hat{v} \) and, therefore, wasted bandwidth per access; as such, we only focus on integral \( \gamma \) values. Since EKV uses total storage \( \kappa \cdot n \cdot s_{\text{max}} \), the number of values in EKV is \( \hat{n} = \frac{n}{\gamma} \).

Let \( \hat{p}_k \) be the sum of the access probabilities of the keys \( k \in \text{Keys}(KV) \) assigned to key \( \hat{k} \in \text{Keys}(EKV) \), i.e., \( \hat{p}_k = \sum_{k \in \text{KV}} \pi_k [k] (\hat{k}) \). Let \( \hat{p}_{\text{max}} = \max_{\hat{k} \in \text{EKV}} (\hat{p}_k) \). Lemma 5 gives us \( B \geq \hat{n} \cdot \hat{p}_{\text{max}}, \) so we have bandwidth footprint:

\[
\beta = B \cdot \gamma \cdot s_{\text{max}} \geq \hat{n} \cdot \hat{p}_{\text{max}} \cdot \gamma \cdot s_{\text{max}} = \kappa n \cdot \frac{\hat{p}_{\text{max}} \cdot \gamma \cdot s_{\text{max}}}{\gamma} = \sigma \cdot \hat{p}_{\text{max}}
\]

As such, minimizing \( \hat{p}_{\text{max}} \) minimizes \( \beta \). Note \( \hat{p}_{\text{max}} \) is monotonically non-decreasing in \( \gamma \); larger \( \gamma \) would allow more keys \( k \in \text{Keys}(KV) \) to be mapped to keys \( \hat{k} \in \text{Keys}(EKV) \) and increase \( \hat{p}_{\text{max}} \). Thus, the smallest possible value \( \gamma = 1 \) minimizes bandwidth footprint \( \beta \). \( \square \)

**B.1 Bandwidth Bound of \( M_L^m \)**

We present the full version of Step 2 of Theorem 3.1’s proof, including the derivation of the lower bound on the advantage \( \text{Adv}_{\text{ror-cdla}} (\Pi, L, A, S, q) (\lambda) \) of adversary \( A \) as defined in Figure 3.

**Step 2: \( A \) has a non-negligible advantage.** To show this, we lower-bound \( \text{Adv}_{\text{ror-cdla}} (\Pi, L, A, S, q) (\lambda) \) by a function \( f(n, \epsilon, \delta) \), and show that with \( \epsilon \to 0 \) and sufficiently large values of \( n \), \( f(\hat{n}, \epsilon, \delta) \) approaches \( 1 \) for \( 0 < \delta < 1 \) and \( \epsilon \to 0 \). Since

\[
\text{Adv}_{\text{ror-cdla}} (\Pi, L, A, S, q) (\lambda) = |\Pr[\text{ROR-CDLA}_1^{\Pi, L, A, S, q} (\lambda) = 1] - \Pr[\text{ROR-CDLA}_1^{\Pi, L, A, S, q} (\lambda) = 1]|,
\]

we obtain this function by finding (i) an upper bound on \( \Pr[\text{ROR-CDLA}_1^{\Pi, L, A, S, q} (\lambda) = 1] \), and (ii) a lower bound on \( \Pr[\text{ROR-CDLA}_1^{\Pi, L, A, S, q} (\lambda) = 1] \). We also note that since \( A_2 \)'s test does not leverage \( \hat{n}, \hat{s} \) and \( B \) generated by the algorithm \( \delta, A \) works against all possible \( \delta \).

**Step 2(i):** We upper-bound \( \Pr[\text{ROR-CDLA}_1^{\Pi, L, A, S, q} (\lambda) = 1] \), i.e., the probability that \( A \) guesses incorrectly that the queries \( \hat{Q} \) it is given are sampled from a uniform distribution rather than generated by \( \Pi \).
To do so, we focus on the key \( \hat{h} \in P(h) \) with access probability \( p \). \( A_2 \) outputs 1 if, for all keys in EKV, the number of queries they receive is less than or equal to \( A \)'s threshold. We can, therefore, use the probability that \( \hat{h} \) receives \( \leq (c + 3\sqrt{\epsilon})\ln \hat{n} \) queries as an upper bound for \( \Pr[\text{ROR-CDLA}_{\lambda}^0(\lambda) = 1] \), and compute the same using the one-sided Chebyshev inequality, as we show next.

Let \( X_\hat{h} \) be the random variable that counts the number of queries to \( \hat{h} \) in \( \hat{Q} \). \( X_\hat{h} \) follows a binomial distribution \( B(\hat{q}, p) \) with mean \( \mu = \hat{q} \times \frac{1 + \epsilon}{n} \) and variance \( \sigma^2 = \hat{q}(1 - p) = \bigg( \frac{1 - \frac{1 + \epsilon}{n} - \frac{1 + \epsilon}{n} \bigg) \bigg( 1 - \frac{1}{\hat{q}} + \frac{1 + \epsilon}{n} \bigg) \). Since \( \hat{q} = \epsilon \ln \hat{n} \), we have \( \mu = \epsilon \ln \hat{n} \left( 1 - \frac{1}{\hat{n}} - 1 - \frac{3}{\sqrt{\epsilon}} \right) \) and \( \sigma^2 = \frac{\epsilon \ln \hat{n} (1 + \epsilon) (1 - \hat{q})(1 + \epsilon) (1 - \hat{q})}{\hat{n}^2} \).

To compute \( \Pr[X_\hat{h} \leq (c + 3\sqrt{\epsilon})\ln \hat{n}] \), we apply the one-sided Chebyshev inequality [45]: \( \Pr[X_\hat{h} \leq \mu] = \frac{\sigma^2}{\epsilon \ln \hat{n} (1 + \epsilon) (1 - \hat{q})(1 + \epsilon) (1 - \hat{q})} \). We set:

\[
\frac{\sigma^2}{\epsilon \ln \hat{n} (1 + \epsilon) (1 - \hat{q})(1 + \epsilon) (1 - \hat{q})} = \frac{c}{1 - \hat{q} - 1 - \frac{3}{\sqrt{\epsilon}}}
\]

Then, substituting into Chebyshev’s inequality, we have:

\[
\Pr[X_\hat{h} \leq (c + 3\sqrt{\epsilon})\ln \hat{n}] \leq \frac{c}{1 - \hat{q} - 1 - \frac{3}{\sqrt{\epsilon}}}
\]

This gives us our upper-bound:

\[
\Pr[\text{ROR-CDLA}_{\lambda}^0(\lambda) = 1] \leq \frac{c}{1 - \hat{q} - 1 - \frac{3}{\sqrt{\epsilon}}}
\]

Note that \( A \) must choose a small enough \( \epsilon \) such that \( a > 0 \), specifically \( \epsilon < 1 - (1 + 3\sqrt{\epsilon})(1 - \hat{q}) \). Since we also need \( \epsilon > 0 \) for the \( \pi \) chosen by \( A \) to be a valid distribution, the attack can only work with \( \hat{q} > \frac{2}{3\sqrt{\epsilon}} \).

**Step 2(ii):** We lower-bound \( \Pr[\text{ROR-CDLA}_{\lambda}^1(\lambda) = 1] \), i.e., the probability that \( A \) guesses correctly that the queries in \( \hat{Q} \) are sampled from a uniform distribution.

Once again, \( A \) outputs 1 if all keys receive more than \( (c + 3\sqrt{\epsilon})\ln \hat{n} \) queries, so we represent the number of queries to a key in EKV with a binomial random variable. We then use the Chernoff bound to compute the upper bound of the probability that all keys receive \( (c + 3\sqrt{\epsilon})\ln \hat{n} \) queries, which we subtract from 1 to obtain the desired lower bound on \( \Pr[\text{ROR-CDLA}_{\lambda}^1(\lambda) = 1] \), as we show next.

Let \( X_k \) be the random variable that counts the number of queries to \( k \in \text{Keys} \). Each \( X_k \) has a binomial distribution \( B(\hat{q}, 1/\hat{n}) \). To find \( \Pr[X_k \leq (c + 3\sqrt{\epsilon})\ln \hat{n}] \), we apply the Chernoff bound [46]:

\[
\Pr[X_k \geq (1 + a)\mu] \leq \exp \left( \frac{-a^2\mu}{3} \right) \text{ for any } 0 < a < 1
\]

Substituting in \( a = 3/\sqrt{\epsilon} \):

\[
\Pr[X_k \geq (c + 3\sqrt{\epsilon})\ln \hat{n}] \leq \exp \left( \frac{-3}{\sqrt{\epsilon}} \frac{\epsilon \ln \hat{n}}{3} \right) = \hat{n}^{-3}
\]

By the union bound, the probability that any key receives more than \( (c + 3\sqrt{\epsilon})\ln \hat{n} \) queries is \( \leq \hat{n}^{-3} = \hat{n}^{-2} \). Thus,

\[
\Pr[\text{ROR-CDLA}_{\lambda}^1(\lambda) = 1] \geq 1 - \hat{n}^{-2}.
\] (3)

**B.2 Bandwidth-optimality of GPC**

Here we prove that Greedy Pancake is bandwidth-optimal.

**Theorem 5.1.** For any storage footprint \( \sigma = \kappa \cdot n \cdot s_{\max}(\kappa \geq 1) \), GPC achieves the lowest bandwidth footprint \( \hat{\beta} \) for any \( \mathcal{ML}_\pi \) scheme that uses exactly \( \sigma \) storage.

**Proof.** Any \( \mathcal{ML}_\pi \) scheme must have the same bandwidth footprint for any input sizes \( S \); as such, Lemma 2 holds for schemes in \( \mathcal{ML}_\pi \); GPC already uses \( \hat{S} = s_{\max} \) satisfying this requirement. We show that GPC with \( \kappa \cdot n \cdot s_{\max} \) storage achieves the optimal bandwidth footprint for \( S := \{s_{\max}, \ldots, s_{\max}\} \), and, by extension, all other \( S \).

Let \( R[\hat{i}] = |P(k_j)| \) be the number of keys \( k_j \in \text{Keys}(\text{EKV}) \) mapped to key \( k_j \in \text{Keys}(\text{KV}) \) by GPC, and let \( R'[\hat{i}] \) be the number of \( k_j \in \text{Keys}(\text{EKV}) \) mapped to key \( k_i \in \text{Keys}(\text{KV}) \) by an optimal algorithm. Let \( f(j, R) := \max \{ k_j \in \text{Keys}(\text{KV}) | \pi(k_j) / R[\hat{i}] \} \); after \( j \) out of \( \hat{n} \) keys in EKV have been assigned a key in KV. By induction on \( j \), we show that \( f(j, R) \leq f(j, R') \) for \( n \leq j \leq \hat{n} \).

**Base case:** \( j = n \). Since each key \( k \in \text{Keys}(\text{KV}) \) is initially assigned exactly one key \( k \in \text{Keys}(\text{EKV}) \), we have:

\[
f(j, R) = f(j, R') = \max \{ k_j \in \text{Keys}(\text{KV}) | \pi(k_j) \}
\]

**Induction.** Let \( f(j, R) \leq f(j, R') \) hold for some \( j > n \). Consider the \( (j + 1)^{th} \) key \( k_{j+1} \in \text{Keys}(\text{EKV}) \) to be assigned a key in \( \text{Keys}(\text{KV}) \). GPC assigns to it \( h = \arg \max_{k_j \in \text{Keys}(\text{KV})} \pi(k_j) / R'[\hat{i}] \), which means before this step, \( \forall \hat{k} \in P(h), \pi[\hat{k}](h) = f(j, R') \). Since we know \( f(j, R) \leq f(j, R') \), the best choice that the optimal algorithm has in minimizing \( f(j + 1, R') \) is to assign its \( \hat{k}_{j+1} \) to the key \( h^* \) for which \( \forall \hat{k} \in P(h), \pi[\hat{k}](h) = f(j, R') \). Since \( f(j, R') \geq f(j, R) \), we still have that \( f(j + 1, R) \leq f(j + 1, R') \).

We now show that the optimal \( f(j + 1, R) \) allows GPC to achieve the minimum bandwidth footprint. Assume for the sake of contradiction that there is an optimal \( R' \) that has a lower bandwidth footprint than \( R \). Bandwidth footprint is \( \hat{\beta} = B \cdot \hat{s} \), and by Lemma 5, \( B \geq \hat{n} \cdot f(j, R') \). Then \( R' \) using less bandwidth than \( R \) can only mean that \( f(j, R') < f(j, R) \) for \( j = \hat{n} \) — a contradiction. 

\[\Box\]