Near-Optimal Constrained Padding for Object Retrievals with Dependencies

Pranay Jain
Duke University

Andrew C. Reed
United States Military Academy

Michael K. Reiter
Duke University

Abstract

The sizes of objects retrieved over the network are powerful indicators of the objects retrieved and are ingredients in numerous types of traffic analysis, such as webpage fingerprinting. We present an algorithm by which a benevolent object store computes a memoryless padding scheme to pad objects before sending them, in a way that bounds the information gain that the padded sizes provide to the network observer about the objects being retrieved. Moreover, our algorithm innovates over previous works in two critical ways. First, the computed padding scheme satisfies constraints on the padding overhead: no object is padded to more than $c\times$ its original size, for a tunable factor $c > 1$. Second, the privacy guarantees of the padding scheme allow for object retrievals that are not independent, as could be caused by hyperlinking. We show in empirical tests that our padding schemes improve dramatically over previous schemes for padding dependent object retrievals, providing better privacy at substantially lower padding overhead, and over known techniques for padding independent object retrievals subject to padding overhead constraints.

1 Introduction

Extracting information about the contents of encrypted traffic from other features of the traffic, or so-called *traffic analysis*, is a topic with a long history and that remains relevant today in the context of, e.g., webpage fingerprinting (see Sec. 2). Among the most difficult network-visible features to obscure is the size of objects transmitted over the network, since thoroughly doing so incurs substantial cost. For this reason, even many systems designed to support private or anonymous communication do not attempt to address this side channel [16, 32, 45, 52] or assume it away by stipulating that messages be small and fixed-size (e.g., [1, 14]).

The premise of this paper is that the research community can facilitate the adoption of *pretty good* defenses against traffic analysis by benevolent—but perhaps not generous—service providers, by first constraining the costs that will be incurred by the service provider for providing privacy and then enabling the service provider to implement the best privacy protections possible subject to those cost constraints. To that end, in this paper we consider a specific but broadly relevant instance of this problem: an object store that serves objects in response to client requests. We seek to equip the object store with a scheme for padding objects before returning them to clients (encrypted) so as to best hide from a network observer which objects are returned to clients, while adhering to the constraint that the padding can increase each object to at most $c\times$ its original size for a specified constant $c > 1$. Since we allow the network observer to be a client of the object store, as well, we presume the observer knows all of the original object sizes, before padding.

Reed and Reiter [43] recently addressed exactly this problem in the case of *independent* object retrievals, as will be detailed in Sec. 2. However, there are few, if any, practical circumstances in which object retrievals are truly independent. Perhaps the most prominent reason is hyperlinking between objects, as between web objects, which can result in one object being retrieved after another (either automatically or by user action) in a causal fashion. Even when not causally related, retrievals in most applications will exhibit correlations, e.g., based on the topic to which the objects pertain. Unfortunately, achieving an optimal padding scheme (to achieve some well-defined notion of privacy) for *dependent* object retrievals is a challenging problem, as demonstrated by a limited literature rife with heuristic solutions only (again, see Sec. 2).

In this paper we take a significant step forward for the case of dependent object retrievals, by providing and evaluating an efficient algorithm to generate a *padding scheme* for an object store. This padding scheme prescribes how to pad objects before serving them, to hide the object identities from a network observer able to see their padded sizes. The measure of leakage that we use in this work is the Sibson mutual information of order infinity, denoted $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ and defined in Sec. 4, where S is a random variable representing the secret (i.e., the object(s) retrieved) and Y is a random variable

representing what the adversary observes (i.e., the padded object size(s)). Issa, et al. [26] advocate for this measure of leakage because it bounds the information gain $\mathbb{I}(\mathsf{S};\mathsf{Y})$ that Y provides to the adversary (i.e., $\mathbb{I}(\mathsf{S};\mathsf{Y}) \leq \mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$) regardless of the distribution of S and because it has a natural operational interpretation [26]. We reiterate that our efforts to minimize $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ are provided subject to a constraint that no object be padded to more than $c\times$ its original size, for a tunable constant $c > 1$. As such, the object store's bandwidth use—per retrieval, per client, and overall—will grow by at most a factor of c. Moreover, the padding scheme produced by our algorithm is memoryless, and so the object store need not track a client's retrievals to know how to pad objects for it.

We demonstrate the practicality of our algorithm by applying it to three real-world datasets and comparing the padding scheme it produces to other padding schemes in the literature for protecting the identities of objects retrieved from being leaked by their padded sizes [5, 34, 35]. While these other schemes were designed to achieve different privacy measures, we nevertheless show that the scheme produced by our algorithm, measured using *their* measures of privacy, provides competitive results with far less padding overhead. We also show that for our measure of privacy, the schemes in previous works fare poorly compared to ours, even while incurring orders of magnitude greater padding overhead. The datasets on which we demonstrate these improvements were chosen to be representative of ones in which dependent object retrievals are commonplace, namely a dataset of autocomplete lists sent in response to increasingly long sequence of characters (which is typical in search engines), and two datasets of hyperlinked web objects.

To reiterate, our contributions are as follows:

- We devise an efficient algorithm to generate a padding scheme that prescribes how to pad objects before serving them in response to requests. The resulting padding scheme approximately minimizes (specifically, minimizes an upper bound on) $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$, for object retrievals S padded to sizes Y, subject to constraints that it be memoryless and that no object be padded to more than $c\times$ its original size for a specified constant $c \geq 1$.

- We introduce three object datasets with dependencies among object retrievals: a Google Autocomplete dataset that models suggestions for search queries; a Wikipedia dataset that models a dense graph of interconnected pages; and a Linode documentation dataset that models a tree-like browsing pattern, where a user navigates to a particular page on a website by starting at the index. On these datasets, we show empirically that our algorithm produces padding schemes that substantially outperform previous work. Even in comparisons to previous algorithms on different privacy measures *they* were designed to optimize, our algorithm achieves comparable privacy with dramatically lower (and bounded) padding overhead. In comparisons that test an adversary's precision and recall in identifying sequences of object retrievals, our algorithm provides better security in most cases, with the same or lower padding overhead.

2 Related Work

The study of padding schemes that minimize information leakage about the objects being retrieved is a topic that has received considerable attention in the literature. There are two camps of closely related work, one focused on privacy for dependent object retrievals (as we consider here) but without constraints on padding overhead, and another that allows for constraints on padding overhead (as we do here) but assuming independent object retrievals. We introduce these areas of related work below, and then elaborate on several more distantly related areas of research.

2.1 Dependent object retrievals

Padding schemes that attempt to reduce leakage about the objects retrieved when retrievals are dependent have been studied for about the past decade, to our knowledge. Backes, et al. [5] proposed one design that assumes that object retrievals satisfy a Markov assumption, i.e., that the client's next object retrieval is only dependent on the object it most recently retrieved. The measure of leakage that they seek to minimize is $\mathbb{I}(\mathsf{S};\mathsf{Y})$, though they do so only heuristically. Liu, et al. proposed methods to render sequences of retrieved object sizes either k-anonymous [44, 48], meaning that at least k objects are padded to every possible padded size [35], or ℓ-diverse [37], meaning that no object accounts for more than $\frac{1}{\ell}$ of any padded size's probability [34]. None of these algorithms, however, accommodate the specification of padding overhead constraints. Indeed, the Liu, et al. algorithms [34,35] are generally not consistent with a padding constraint c if, e.g., there is an object for which the allowed padding range does not intersect the allowed padding ranges of $k-1$ other objects. We show in our experiments that the overhead they therefore impose is dramatic on the datasets we consider.

2.2 Padding overhead constraints

To our knowledge, object padding schemes subject to padding overhead constraints as we consider here have received attention primarily within the last several years (e.g., [12, 40, 43]). The latest work in this area [43] proposes algorithms to generate padding schemes that optimally hide objects from a network observer in several cases of interest: for an object store that pads its objects once but serves them repeatedly, a padding scheme that is deterministic and that minimizes information gain $\mathbb{I}(\mathsf{S};\mathsf{Y})$ among all deterministic schemes; for an object store that can re-pad objects each time it serves them, a padding scheme that is probabilistic and that minimizes

$\mathbb{I}(\mathsf{S};\mathsf{Y})$; and for an object store that cannot trust (or does not wish to track) the distribution of object retrievals, a padding scheme that minimizes $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$. This last algorithm, called PwoD, provides a close comparison point for our work, since it targets the same leakage measure (i.e., $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$) and adopts the same form of padding overhead constraint. However, the guarantees of these algorithms hinge on object retrievals being independent, which is exactly the assumption we intend to relax in this paper.

2.3 Webpage fingerprinting

More distantly related work includes research in webpage fingerprinting, by which a network adversary attempts to discern the webpage or website that a client is visiting, based on traffic features the attacker can observe. Some works have explored padding of web objects as a defense against such traffic analysis (e.g., [12, 47]) but, to our knowledge, no such work has shown how to maximize privacy (by a precise measure) subject to a bandwidth overhead constraint, as we do here. The vast majority of the research in webpage fingerprinting is specific to the networking context via which the web objects are transferred to the client, leveraging features—for attack or defense—of TCP and/or HTTP (e.g., [36, 47]), HTTPS (e.g., [2, 11, 15, 19, 39]); web proxies (e.g., [24, 47]); SSH proxies (e.g., [6, 33]); or Tor (e.g., [7, 12, 22, 23, 27, 41, 49, 50]). We stress that our research focuses specifically on object sizes, allowing this feature to be discernible to the network attacker, whatever the network context. Our study therefore addresses a central challenge in defending against webpage fingerprinting [17], though by no means the only one [51].

2.4 Untrusted object store

In our threat model (see Sec. 3), the object store is trusted. A substantial body of literature, in contrast, addresses a threat model wherein the object store is itself the adversary from which the secret should be hidden (e.g., [8, 21, 25, 30, 31]). For example, in *private information retrieval* [13], this secret is the object retrieved, whereas in *searchable encryption* [46], the secret is the search term resulting in the retrieval of a set of objects. This threat model is more permissive than ours and so typically requires tools to address it that are different in nature, and that are more expensive, than we consider here, including introducing fake queries (e.g., [20, 38, 42]); breaking retrievals into multiple fixed-size queries of total size larger than the original object and so that themselves are padded (e.g., [28, 29]); or using oblivious RAMs (e.g., [4, 9, 10, 18]). In contrast, our approach does not alter the communication pattern between the client and object store, aside from padding each object to within a factor of c of its original size.

3 Problem Statement

We consider a trusted object store that stores n fixed-size objects $\{\mathsf{obj}_s\}_{s\in S}$ where $\#S = n$ (and $\#S$ denotes the cardinality of S), and serves them to clients over encrypted channels. Each client request contains the identifier s of the object to return, which we presume to be of constant size. Objects themselves, however, can be of different sizes; in symbols, if $|\mathsf{obj}_s|$ denotes the size of object obj_s, then for $s \neq s'$, it can be the case that $|\mathsf{obj}_s| \neq |\mathsf{obj}_{s'}|$. Even though objects are returned on encrypted channels, the size of each object (as transmitted, after padding) will be revealed to our adversary, a network observer. This network observer seeks to identify which objects were retrieved by a client, given their sequence of padded sizes. Since we allow the network observer to also be a client of the object store and so to retrieve objects itself, the adversary knows the unpadded size $|\mathsf{obj}_s|$ of each obj_s.

3.1 Padding scheme

Our goal is to provide an algorithm to compute a *padding scheme* $\lceil\cdot\rceil$ for $\{\mathsf{obj}_s\}_{s\in S}$. Formally, $\lceil\cdot\rceil$ takes as input an object identifier s and an object size $|\mathsf{obj}_s|$ and produces a padded object size (a positive integer) with properties defined below. To emphasize that padding will be applied to this object before transmission, we denote this invocation of $\lceil\cdot\rceil$ by simply $\lceil\mathsf{obj}_s\rceil$, and when convenient, we will also use $\lceil\mathsf{obj}_s\rceil$ to denote the padded object or its size. The padding scheme $\lceil\cdot\rceil$ is itself randomized, and so repeated invocations of it with the same object (i.e., same object identifier and size) can return different results. Moreover, since s is an input to $\lceil\cdot\rceil$, $\lceil\mathsf{obj}_s\rceil$ can behave differently from $\lceil\mathsf{obj}_{s'}\rceil$ even if obj_s and $\mathsf{obj}_{s'}$ are of the same size.

The padding scheme $\lceil\cdot\rceil$ that our algorithm will solve for should be *memoryless* in the sense that the object store does not need to track each user's retrieval history in order to implement the padding scheme. We believe that this ensures that our technique is as broadly implementable as possible. First, there is the obvious cost saving for not having to store every user's retrieval history. Second, our technique is "privacy-friendly," i.e., it will work if an object store cannot store the retrieval history of its users, whether that is due to (i) government regulation, (ii) privacy-aware tools employed by the user, or (iii) as a feature of the object store itself.

Let S be a random variable denoting the identifier of the object a client retrieved from the object store, and let Y be a random variable denoting the size to which it was padded when retrieved; i.e., Y takes the value $\lceil\mathsf{obj}_s\rceil$ when $\mathsf{S} = s$. Our first requirement, discussed in Sec. 1, is that padding never grow an object by a factor of more than c, for a padding factor $c \geq 1$.

$$\mathbb{P}\big(\mathsf{Y} > c \times |\mathsf{obj}_s| \;\big|\; \mathsf{S} = s\big) = 0 \tag{1}$$

In addition, we assume that objects cannot be compressed

before their transmission over the network, and so

$$\mathbb{P}\big(\mathsf{Y} < |\mathsf{obj}_s| \;\big|\; \mathsf{S} = s\big) = 0 \tag{2}$$

While numerous networking technologies *do* compress data before transmitting it, we presume that the same network-stack layer that pads the object then encrypts it, rendering the object incompressible by network layers underneath it.

Before we present the privacy goals for the padding scheme $\lceil\cdot\rceil$ in Sec. 3.2, note that due to (1)–(2), it might not be possible for the padding scheme to protect the privacy of all object retrievals. That is, if one object obj_s is of a size such that $[|\mathsf{obj}_s|, c\times|\mathsf{obj}_s|] \cap [|\mathsf{obj}_{s'}|, c\times|\mathsf{obj}_{s'}|] = \emptyset$ for each other object $\mathsf{obj}_{s'}$, then (1)–(2) leave no option but for obj_s to be the only possible object being retrieved when a retrieval of size $\lceil\mathsf{obj}_s\rceil$ is observed on the network. This observation informs the privacy goal we adopt in Sec. 3.2.

In the remainder of this paper, we treat mathematical expressions in which c or $c-1$ is a factor to be integers. In particular, for notational simplicity, we omit the floor function when writing $c\times|\mathsf{obj}_s|$, $(c-1)\times|\mathsf{obj}_s|$, etc., and simply treat them as integers.

3.2 Privacy measure

We would like to produce a padding scheme so that sequential invocations of $\lceil\cdot\rceil$ leak as little about the identities of the requested objects as possible, subject to the constraints on the padding scheme discussed in Sec. 3.1. Issa, et al. [26] argue that the leakage about the value of a discrete random variable S with support S by observing the value of another random variable Y is best captured by

$$\mathbb{I}_\infty(\mathsf{S};\mathsf{Y}) = \log_2 \sum_y \max_{s\in S} \mathbb{P}\big(\mathsf{Y} = y \;\big|\; \mathsf{S} = s\big) \tag{3}$$

Both Issa, et al. [26] and Alvim, et al. [3] advocate for the use of $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ as a measure of leakage as it upper-bounds an adversary's multiplicative gain in correctly guessing any function of S after observing Y, over all distributions of S.

To highlight that we are concerned here with *sequences* of possibly dependent object retrievals, we extend (3) to sequences. Specifically, let $\vec{S} \subseteq S^*$ be a set of sequences of object retrievals, let $\vec{s} \in \vec{S}$ denote a sequence of object retrievals, and let $\vec{\mathsf{S}}$ be a random variable with support $\vec{S}$. (We assume $\mathbb{P}\Big(\vec{\mathsf{S}} = \vec{s}\Big) > 0$ for all $\vec{s} \in \vec{S}$.) We denote the i-th element of a sequence $\vec{s}$ by $\vec{s}_i$, and the length of $\vec{s}$ by $\mathrm{len}(\vec{s})$.

For each $s \in S$, let

$$Y_s \subseteq [|\mathsf{obj}_s|, c\times|\mathsf{obj}_s|] \tag{4}$$

be the padded object sizes that we allow for $\lceil\mathsf{obj}_s\rceil$. While Y_s might be all of $[|\mathsf{obj}_s|, c\times|\mathsf{obj}_s|]$, it need not be, and indeed we will leverage this flexibility in Sec. 4. We define

$$\vec{Y}_{\vec{s}} = Y_{\vec{s}_1} \times \cdots \times Y_{\vec{s}_{\mathrm{len}(\vec{s})}} \tag{5}$$

$$\vec{Y} = \bigcup_{\vec{s}\in\vec{S}} \vec{Y}_{\vec{s}} \tag{6}$$

In words, $\vec{Y}_{\vec{s}}$ is the set of all possible sequences to which $\vec{s}$ could be padded, and $\vec{Y}$ is the set of all possible padded sequences. Finally, let $\vec{\mathsf{Y}}$ be a random variable taking on a sequence of padded sizes. Given this notation, we interpret (3) in our context as

$$\mathbb{I}_\infty\Big(\vec{\mathsf{S}};\vec{\mathsf{Y}}\Big) = \log_2 \sum_{\vec{y}\in\vec{Y}} \max_{\vec{s}\in\vec{S}} \mathbb{P}\big(\vec{\mathsf{Y}} = \vec{y} \;\big|\; \vec{\mathsf{S}} = \vec{s}\big) \tag{7}$$

Our goal, then, is to produce a padding scheme $\lceil\cdot\rceil$ that minimizes (7).

4 Design

In this section we present a linear program that minimizes an upper-bound on $\mathbb{I}_\infty\Big(\vec{\mathsf{S}};\vec{\mathsf{Y}}\Big)$. We then describe a technique that allows us to reduce the number of variables in the linear program, thereby enabling it to run more efficiently on large datasets.

4.1 Linear program

Below, we denote the i-th element of a sequence $\vec{y}$ by $\vec{y}_i$, and the i-th element of a sequence taken on by random variables $\vec{\mathsf{S}}$ and $\vec{\mathsf{Y}}$ by $\vec{\mathsf{S}}_i$ and $\vec{\mathsf{Y}}_i$, respectively. For any $\vec{y}$ and $\vec{s}$ of the same length, and any $i' \in [\mathrm{len}(\vec{y})] = \{1,\ldots,\mathrm{len}(\vec{y})\}$,

$$\prod_{i=1}^{\mathrm{len}(\vec{y})} \mathbb{P}\big(\vec{\mathsf{Y}}_i = \vec{y}_i \;\big|\; \vec{\mathsf{S}}_i = \vec{s}_i\big) \le \mathbb{P}\big(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \;\big|\; \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}\big) \tag{8}$$

since each probability is ≤ 1. By summing (8) over $i' \in [\mathrm{len}(\vec{y})]$, we can conclude

$$\mathrm{len}(\vec{y}) \prod_{i=1}^{\mathrm{len}(\vec{y})} \mathbb{P}\big(\vec{\mathsf{Y}}_i = \vec{y}_i \;\big|\; \vec{\mathsf{S}}_i = \vec{s}_i\big) \le \sum_{i'=1}^{\mathrm{len}(\vec{y})} \mathbb{P}\big(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \;\big|\; \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}\big) \tag{9}$$

As a result, we can upper-bound (7) as

$$\mathbb{I}_\infty\Big(\vec{\mathsf{S}};\vec{\mathsf{Y}}\Big) = \log_2 \sum_{\vec{y}\in\vec{Y}} \max_{\vec{s}\in\vec{S}} \prod_{i=1}^{\mathrm{len}(\vec{y})} \mathbb{P}\big(\vec{\mathsf{Y}}_i = \vec{y}_i \;\big|\; \vec{\mathsf{S}}_i = \vec{s}_i\big) \tag{10}$$

$$\le \log_2 \sum_{\vec{y}\in\vec{Y}} \max_{\vec{s}\in\vec{S}} \frac{1}{\mathrm{len}(\vec{y})} \sum_{i'=1}^{\mathrm{len}(\vec{y})} \mathbb{P}\big(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \;\big|\; \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}\big) \tag{11}$$

where (10) follows from (7) because our padding scheme is memoryless and (11) follows from (10) by substituting (9).

$$\text{minimize} \sum_{\vec{y}\in\vec{Y}} \Pi^{\vec{y}} \quad \text{subject to} \tag{12}$$

$$\sum_{y\in Y_s} \pi_s^y = 1 \qquad \forall s \in S \tag{13}$$

$$\pi_s^y \geq 0 \qquad \forall s \in S, y \in Y_s \tag{14}$$

$$\pi_s^y \leq 1 \qquad \forall s \in S, y \in Y_s \tag{15}$$

$$\Pi^{\vec{y}} \geq \frac{1}{\text{len}(\vec{y})} \sum_{i=1}^{\text{len}(\vec{y})} \pi_{\vec{s}_i}^{\vec{y}_i} \qquad \forall \vec{s} \in \vec{S}, \vec{y} \in \vec{Y}_{\vec{s}} \tag{16}$$

Figure 1: LP to minimize (11).

We formulate a linear program (LP) to minimize (11). It uses a variable π_s^y to represent $\mathbb{P}\big(\mathsf{Y} = y \;\big|\; \mathsf{S} = s\big)$, and a variable $\Pi^{\vec{y}}$ to represent $\max_{\vec{s}\in\vec{S}} \frac{1}{\text{len}(\vec{y})} \sum_{i'=1}^{\text{len}(\vec{y})} \mathbb{P}\big(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \;\big|\; \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}\big)$. This LP is shown in Fig. 1. Constraints (13)–(15) ensure that π_s^y forms a probability distribution for each $s \in S$. Constraint (16), together with the minimization objective (12), ensure that a solution to the LP sets $\Pi^{\vec{y}}$ to its intended value. Once solved, the object store can instantiate a padding scheme $\lceil\cdot\rceil$ by padding obj_s to size y (i.e., $\lceil \mathsf{obj}_s \rceil = y$) with probability π_s^y; i.e., $\mathbb{P}\big(\mathsf{Y} = y \;\big|\; \mathsf{S} = s\big) \leftarrow \pi_s^y$.

Note that any two distinct sequences $\vec{s}, \vec{s}' \in \vec{S}$, even if of the same length ($\text{len}(\vec{s}) = \text{len}(\vec{s}')$) and consisting of the same objects ($\{\vec{s}_i\}_{i\in[\text{len}(\vec{s})]} = \{\vec{s}'_i\}_{i\in[\text{len}(\vec{s}')]}$), will result in a constraint of the form (16) for every $\vec{y} \in \vec{Y}_{\vec{s}} \cap \vec{Y}_{\vec{s}'}$. That is, our LP optimizes the probabilities of padded sequences based on both length and the order of object sequences that can pad to them.

4.2 Efficiency

The LP presented in Sec. 4.1 has $\#Y_s$ variables π_s^y for each $s \in S$. The LP also includes one variable $\Pi^{\vec{y}}$ for each $\vec{y} \in \vec{Y}$; recall that $\#\vec{Y}$ also depends on $\#Y_s$ for $s \in S$ (see (5)–(6)). As such, reducing $\#Y_s$ for each s is central to scaling this LP. At the same time, the values in Y_s must be chosen to maximize the intersection with other $Y_{s'}$ sets, subject to each of these sets being small (and subject to (1)–(2)), so that the LP has the opportunity to pad objects to the same size.

To accomplish these contradictory goals, we consider the *anchor sizes* of the object set, defined as follows. Let X represent the set of all unpadded sizes; i.e., $x \in X$ implies that there exists an object for which $|\mathsf{obj}_s| = x$. For any $X' \subseteq X$, define the *anchor sizes* $A(X')$ inductively as follows:

$$A\big(X'\big) = \{a\big(X'\big)\} \cup A\big(X' \setminus B\big(X'\big)\big) \qquad \text{where} \tag{17}$$

$$a\big(X'\big) = \min X' \qquad \text{and} \tag{18}$$

$$B\big(X'\big) = \{x' \in X' : x' \leq c \times a\big(X'\big)\} \tag{19}$$

In words, the anchor sizes of X' include the smallest unpadded object size of X' (see (18)) and the anchor sizes of the set remaining after removing all sizes at most c times this anchor size (see (17), (19)). Note that for any two $a, a' \in A(X)$, it will be the case that $[a, c\times a] \cap [a', c\times a'] = \emptyset$.

For a fixed parameter $k \geq 1$, we select up to k candidate padding sizes $\hat{Y}_a$ from $[a, c\times a]$ of approximately equal distance apart. Specifically, we select

$$\hat{Y}_a = \{a + zq + \min\{z, r\}\}_{z\in[k]} \tag{20}$$

where $(c-1)a = qk + r$ and $0 \leq r < k$. Then, we set

$$Y_s = [|\mathsf{obj}_s|, c \times |\mathsf{obj}_s|] \cap \bigcup_{a\in A} \hat{Y}_a \tag{21}$$

No Y_s is empty, since at least $c \times a \in Y_s$ for the largest anchor size $a \in A(X)$ such that $a \leq |\mathsf{obj}_s|$. Moreover, for any $s \in S$, there are at most two anchor sizes $a, a' \in A(X)$ such that $[|\mathsf{obj}_s|, c\times|\mathsf{obj}_s|] \subseteq [a, c\times a] \cup [a', c\times a']$. So, $\#Y_s \leq 2k$, ensuring that $\#Y_s$ is small. Finally, by limiting each Y_s to elements of $\bigcup_{a\in A} \hat{Y}_a$, we encourage common padding sizes for objects.

For the remainder of this paper we refer to our LP as Padding for Sequences (PFS).

5 Evaluation

In this section we begin by describing the datasets that we created for our assessments and then present three algorithms for padding objects to which we compare. We compare PFS to each of them according to their intended privacy metric and others. We conclude the section with an assessment of the padding overhead that results from each algorithm's padding scheme for each dataset.

5.1 Datasets

For our experiments, we constructed three datasets. We constructed these datasets to be representative of datasets used to evaluate previous padding schemes and to provide significant variations in object sizes and numbers of sequences.

Each dataset consists of (i) a selection of objects from a given object store, which we denote $\{\mathsf{obj}_s\}_{s\in S}$; (ii) the set of possible sequences of object retrievals $\vec{S}$; and (iii) distributions $\mathbb{P}(\mathsf{S} = s)$, $\mathbb{P}\Big(\vec{\mathsf{S}}_{1\ldots i} = \vec{s}_{1\ldots i}\Big)$, and $\mathbb{P}\Big(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s\Big)$, since some algorithms to which we compare require these distributions. Below, we let E denote the set of directed edges in the sequences $\vec{S}$, i.e., $E = \Big\{(s, s') \;\Big|\; \exists \vec{s} \in \vec{S}, i \in [\text{len}(\vec{s}) - 1] : \vec{s}_i = s \wedge \vec{s}_{i+1} = s'\Big\}$.

For each dataset, in order to create $\vec{S}$, we first created a set of maximal length sequences which we denote as $\vec{S}^{\Omega}$. We then created $\vec{S}$ as

$$\vec{S} = \bigcup_{\vec{s}\in\vec{S}^{\Omega}} \bigcup_{i\in[\text{len}(\vec{s})]} \{\vec{s}_{1\ldots i}\} \tag{22}$$

where $\vec{s}_{1\ldots i}$ represents the subsequence of $\vec{s}$ up to, and including, the i-th object. In words, $\vec{S}$ is the prefix closure of $\vec{S}^{\Omega}$. Finally, for each $i \in [\max_{\vec{s}\in\vec{S}^{\Omega}} \text{len}(\vec{s})]$ we calculate $\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i} = \vec{s}_{1\ldots i}\right)$ as

$$\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i} = \vec{s}_{1\ldots i}\right) = \frac{\sum_{\vec{s}'\in\vec{S}^{\Omega}:\vec{s}'_{1\ldots i}=\vec{s}_{1\ldots i}} \mathsf{count}(\vec{s}')}{\sum_{\vec{s}''\in\vec{S}^{\Omega}:\text{len}(\vec{s}'')\geq i} \mathsf{count}(\vec{s}'')} \tag{23}$$

In (23), for each $\vec{s} \in \vec{S}^{\Omega}$ we use $\mathsf{count}(\vec{s})$ to represent a dataset-specific statistic from which we can calculate a distribution.

Google Autocomplete dataset This dataset was created in January 2023 and models the distribution of responses from Google search autocomplete suggestions. For a given search string, Google responds with a list of autocomplete suggestions for each prefix of the string. To obtain these autocomplete responses, we wrote a script that takes a list of words as input and queries the Google autocomplete API for each successive prefix of each word. The dictionary of words was obtained from xkcd Simple Writer[1], a dataset containing the 1,000 most common English words. In this dataset, each $s \in S$ is a prefix of a word, and obj_s is Google's autocomplete response for that prefix. Each $(s,s') \in E$, then, represents two consecutive queries of word prefixes in which s' extends s by one character and in which s' is a prefix of some word in the dataset. We define $\vec{S}^{\Omega}$ to contain each sequence $\vec{s}$ of prefixes of increasing length (i.e., $\vec{s}_i$ is a word prefix of i characters) such that $\vec{s}_{\text{len}(\vec{s})}$ is a word in the dataset. In total, after omitting plurals of words that are formed by simply adding the letter 's', for this dataset $\#\vec{S}^{\Omega} = 899$ words, $\#S = 3{,}870$ word prefixes, and $\#E = 3{,}846$ word prefix extensions. Moreover, $\#\vec{S} = \#S$ since each $s \in S$ is a prefix of a word and since $\vec{S}$ is the prefix closure of $\vec{S}^{\Omega}$.

In addition to S and E, the method of Backes, et al. [5] also requires the transition probabilities between word prefixes, i.e., an actual value for $\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \;\middle|\; \vec{\mathsf{S}}_i = s\right)$ for each $(s,s') \in E$. We model this using the technique they proposed, which uses the number of search results returned by Google for a given word $\vec{s}_{\text{len}(\vec{s})}$ as the value for $\mathsf{count}(\vec{s})$, and then $\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \;\middle|\; \vec{\mathsf{S}}_i = s\right)$ is estimated as

$$\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \;\middle|\; \vec{\mathsf{S}}_i = s\right) = \frac{\sum_{\vec{s}'\in\vec{S}^{\Omega}:\vec{s}'_{i+1}=s'\wedge\vec{s}'_i=s} \mathsf{count}(\vec{s}')}{\sum_{\vec{s}\in\vec{S}^{\Omega}:\vec{s}_i=s} \mathsf{count}(\vec{s})} \tag{24}$$

Wikipedia dataset This dataset was created in June 2023 and models pages retrieved during web surfing. It was created by selecting $s =$ `https://en.wikipedia.org/wiki/Cat` to obtain the HTML of the web page as obj_s. We then selected the first 50 hyperlinks (to articles) on this page and included those in S, retrieving the HTML of the web page for each. We repeated this step once more for these added pages, but increased the number of hyperlinks that we added from those pages to the first 100. We increased from 50 to 100 hyperlinks to help increase the reach of this dataset, as we observed that the articles had many common hyperlinks.

This initial step yielded a set of $\#S = 2{,}804$ articles. To then create the set of sequences $\vec{S}^{\Omega}$, for each $s \in S$, we generated two random walks of length six that both begin at s as the start vertex. This resulted in $\#\vec{S}^{\Omega} = 5{,}606$ unique sequences of webpages that a user might explore while browsing among the articles included in S with an associated $\#E = 10{,}182$ hyperlinks. Taking the prefix closure of these sequences then yielded $\#\vec{S} = 32{,}683$.

For this dataset, we used the Wikipedia Page Views API to instantiate $\mathsf{count}(\vec{s})$ for each $\vec{s} \in \vec{S}^{\Omega}$. To do so, we retrieved the total number of page views for each $s \in S$ for January 2016, which we denote as $\mathsf{pv}(s)$. For each $\vec{s} \in \vec{S}^{\Omega}$ we then set $\mathsf{count}(\vec{s})$ equal to its average page views, i.e.,

$$\mathsf{count}(\vec{s}) = \frac{\sum_{i=1}^{\text{len}(\vec{s})} \mathsf{pv}(\vec{s}_i)}{\text{len}(\vec{s})} \tag{25}$$

Finally, we instantiated $\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \;\middle|\; \vec{\mathsf{S}}_i = s\right)$ for each $(s,s') \in E$ as

$$\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \;\middle|\; \vec{\mathsf{S}}_i = s\right) = \frac{1}{\#\{\hat{s} : (s,\hat{s}) \in E\}} \tag{26}$$

We reiterate that this dataset only captures the size of each article's HTML file. We address this limitation with our next dataset, though we stress that our goal for all of our datasets is to enable a meaningful comparison of candidate padding algorithms (described below) based on the privacy they achieve and padding overhead they induce. Our goal is not to make specific claims about privacy in the context of Wikipedia retrievals, for example.

Linode dataset As with the Wikipedia dataset, this dataset models pages retrieved during web surfing where S represents webpages and E represents hyperlinks between webpages. It was created by crawling the Linode documentation website[2] in April 2020. A difference from the Wikipedia dataset, though, is that in the Linode dataset $|\text{obj}_s|$ represents the total sum of data for the given page's HTML file and all the hyperlinked objects that would be retrieved automatically (images, scripts, etc.).

Another difference between our Wikipedia and Linode datasets is the way in which we generated maximal length sequences. Let $s_{\text{home}} =$ `https://www.linode.com/docs`. Then, for each $s \in S \setminus \{s_{\text{home}}\}$ we include in $\vec{S}^{\Omega}$ all shortest paths from s_{home} to s. In other words, $\vec{s} \in \vec{S}^{\Omega}$ iff: (i) $\vec{s}_1 = s_{\text{home}}$, (ii) $\vec{s}_{\text{len}(\vec{s})} \neq s_{\text{home}}$, and (iii) $\vec{s}$ is a shortest path. This dataset therefore models a user that begins at the Linode documentation homepage and then navigates to a destination page

[1] `https://xkcd.com/simplewriter/words.js`

[2] The crawl included every webpage and linked-to file where the URL began with https://www.linode.com/docs.

Table 1: Dataset statistics.

Statistic	Autocomplete	Wikipedia	Linode
$\min_{s\in S}\lvert\mathsf{obj}_s\rvert$	11B	36,425B	438B
$\text{median}_{s\in S}\lvert\mathsf{obj}_s\rvert$	330B	203,310B	101,325B
$\max_{s\in S}\lvert\mathsf{obj}_s\rvert$	480B	1,745,780B	15,406,625B
$\#S$	3,870	2,804	1,569
$\#\vec{S}^{\Omega}$	899	5,606	2,095
$\#\vec{S}$	3,870	32,683	2,096
$\#E$	3,846	10,182	2,029

by clicking as few links as possible. The dataset contains $\#S = 1{,}569$ webpages, $\#\vec{S}^{\Omega} = 2{,}095$ unique sequences of webpages, and $\#E = 2{,}029$ links between webpages. Since the sequences in this dataset are all shortest paths, taking the prefix closure of $\vec{S}^{\Omega}$ only yields one additional sequence: the sequence of length one that consists of s_{home}.

As this dataset does not have an accompanying Page Views API, for each $\vec{s} \in \vec{S}^{\Omega}$ we set $\mathsf{count}(\vec{s}) = 1$ and we instantiated $\mathbb{P}\big(\vec{\mathsf{S}}_{i+1} = s' \;\big|\; \vec{\mathsf{S}}_i = s\big)$ for each $(s,s') \in E$ as

$$\mathbb{P}\big(\vec{\mathsf{S}}_{i+1} = s' \;\big|\; \vec{\mathsf{S}}_i = s\big) = \frac{1}{\#\{\hat{s} : (s,\hat{s}) \in E\}} \tag{27}$$

Dataset statistics In Table 1 we list various statistics for each of our datasets. The Linode dataset presents the widest variation of object sizes, ranging across five orders of magnitude, despite it having the fewest number of objects ($\#S$) of any of the datasets. The Wikipedia dataset stands out as having both the largest $\#\vec{S}^{\Omega}$ and $\#\vec{S}$ since our strategy for creating this dataset ensured that it consisted of a large number of maximal length sequences which do not share prefixes.

5.2 Comparison algorithms

Here we detail the three algorithms to which we compare. We selected these algorithms as they each target a different privacy metric, and so comparing to all three provides a robust assessment of PFS.

BDK Backes, et al. [5] propose an algorithm to create a deterministic (i.e., per-object) padding scheme $\lceil\cdot\rceil$ such that, for any padded sizes $y, y' \in Y$ and any $s, s' \in S$ such that $\lceil\mathsf{obj}_s\rceil = \lceil\mathsf{obj}_{s'}\rceil = y$,

$$\sum_{\substack{\hat{s}\in S:\\ \lceil\mathsf{obj}_{\hat{s}}\rceil = y'}} \mathbb{P}\big(\vec{\mathsf{S}}_{i+1} = \hat{s} \;\big|\; \vec{\mathsf{S}}_i = s\big) = \sum_{\substack{\hat{s}\in S:\\ \lceil\mathsf{obj}_{\hat{s}}\rceil = y'}} \mathbb{P}\big(\vec{\mathsf{S}}_{i+1} = \hat{s} \;\big|\; \vec{\mathsf{S}}_i = s'\big) \tag{28}$$

In other words, for any two objects obj_s and $\mathsf{obj}_{s'}$ that pad to size y, it must be equally likely for each that its retrieval will be followed by a retrieval padded to size y'. In the remainder of this paper, we refer to the Backes, et al. algorithm as "BDK".

BDK assumes that the generation of object retrieval sequences can be modeled as a Markov chain (i.e., that the distribution over the next object retrieved depends only on the previous). Subject to this assumption, it efficiently calculates entropy $\mathbb{H}\Big(\vec{\mathsf{Y}}\Big)$ for any arbitrary sequence length. It therefore works by randomly producing many candidate per-object padding schemes $\lceil\cdot\rceil$ and then selecting the scheme $\lceil\cdot\rceil$ that produces the lowest $\mathbb{H}\Big(\vec{\mathsf{Y}}\Big)$ for a target average padding overhead. $\mathbb{H}\Big(\vec{\mathsf{Y}}\Big)$, then, serves as an upper-bound for $\mathbb{I}\Big(\vec{\mathsf{S}};\vec{\mathsf{Y}}\Big)$, i.e., the mutual information between $\vec{\mathsf{S}}$ and $\vec{\mathsf{Y}}$.

MVMD and MVMD-D Whereas BDK seeks to control $\mathbb{H}\Big(\vec{\mathsf{Y}}\Big)$—and thereby $\mathbb{I}\Big(\vec{\mathsf{S}};\vec{\mathsf{Y}}\Big)$—Liu, et al. [34, 35] propose techniques that strive to ensure different metrics, namely k-anonymity and ℓ-diversity. In the summary of these algorithms below, recall that we use the subscript $1\ldots i$ to indicate the subsequence of a sequence from the first through the i-th element, inclusive; e.g., $\vec{s}_{1\ldots i}$ represents the subsequence of $\vec{s}$ up to, and including, the i-th object. Similarly, $\vec{S}_{1\ldots i}$ represents the set of all such subsequences, and $\vec{\mathsf{S}}_{1\ldots i}$ represents the random variable over this set. Note that this notation will also be applied to $\vec{y}$, $\vec{Y}$, and $\vec{\mathsf{Y}}$ with the same interpretation.

Now, for the k-anonymity setting, a per-object padding scheme $\lceil\cdot\rceil$ is said to provide k-anonymity for sequences if, $\forall\vec{y} \in \vec{Y}$ such that $\mathbb{P}\Big(\vec{\mathsf{Y}} = \vec{y}\Big) > 0$, and $\forall i \in [\text{len}(\vec{y})]$, it holds that $\#\{s \in S : \mathbb{P}\big(\vec{\mathsf{S}}_i = s \;\big|\; \vec{\mathsf{Y}}_{1\ldots i} = \vec{y}_{1\ldots i}\big) > 0\} \geq k$. That is, if an adversary were to observe the first i padded object sizes, there must be $\geq k$ possible objects that could be $\vec{s}_i$. For the ℓ-diversity setting, a padding scheme $\lceil\cdot\rceil$ is said to provide ℓ-diversity for sequences if, $\forall\vec{y} \in \vec{Y}$ such that $\mathbb{P}\Big(\vec{\mathsf{Y}} = \vec{y}\Big) > 0$, $\forall i \in [\text{len}(\vec{y})]$, and $\forall s \in S$ such that $\mathbb{P}\big(\vec{\mathsf{S}}_i = s \;\big|\; \vec{\mathsf{Y}}_{1\ldots i} = \vec{y}_{1\ldots i}\big) > 0$, it holds that $\mathbb{P}\big(\vec{\mathsf{S}}_i = s \;\big|\; \vec{\mathsf{Y}}_{1\ldots i} = \vec{y}_{1\ldots i}\big) \leq \frac{1}{\ell}$. That is, if an adversary were to observe the first i padded objects sizes, the conditional probability of each object that could be $\vec{s}_i$ must be $\leq \frac{1}{\ell}$.

For both settings, Liu, et al. present greedy algorithms that attempt to create per-object padding schemes $\lceil\cdot\rceil$ that ensure either k-anonymity or ℓ-diversity, and that also attempt to minimize the sum of padding overhead applied to objects. We refer to these algorithms as MVMD and MVMD-D, respectively, and when parameterizing MVMD-D with a target ℓ, we refer to the algorithm as MVMD-ℓ. So, for example, when parameterized with a target $\ell = 3$, we refer to the algorithm as MVMD-3. Roughly, the MVMD and MVMD-D algorithms iterate through each $i \in [\max_{\vec{s}\in\vec{S}}\text{len}(\vec{s})]$ and—for each $\vec{S}' \subseteq \vec{S}$ where for every $\vec{s} \in \vec{S}'$ it is the case that $\vec{s}_{1\ldots i-1}$ is padded to the same $\vec{y}_{1\ldots i-1}$—construct $\lceil\cdot\rceil$ so that $\vec{S}'_i = \bigcup_{\vec{s}\in\vec{S}'}\{\vec{s}_i\}$ is split into two subsets that remain either k-anonymous or ℓ-diverse, and that do so with minimal total overhead. Since the algorithms are greedy, they are not guaranteed to create

Table 2: Inputs required per algorithm.

Algorithm	Sets			Distributions		
	S	$\vec{S}$	E	$\mathbb{P}(\mathsf{S}=s)$	$\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i}=\vec{s}_{1\ldots i}\right)$	$\mathbb{P}\left(\vec{\mathsf{S}}_{i+1}=s' \middle\vert \vec{\mathsf{S}}_i=s\right)$
BDK [5]	✓		✓	✓		✓[3]
MVMD-D [34]	✓	✓			✓	
PwoD [43]	✓					
PFS	✓	✓				

the padding scheme $\lceil\cdot\rceil$ that minimizes the padding overhead for a given k or ℓ. Furthermore, there are instances where k-anonymity or ℓ-diversity cannot be achieved, either due to their algorithms making a greedy choice at i that prevents upholding k or ℓ at $i' > i$, or simply because the distribution $\vec{\mathsf{S}}_{1\ldots i}$ is not distributed in a way that supports the chosen metric. In such cases, these algorithms will construct $\lceil\cdot\rceil$ so that all $s \in \vec{S}'_i$ (which cannot be split further) will be padded to the same y.

Note that ℓ-diversity subsumes k-anonymity, in that an ℓ-diverse padding scheme is also ℓ-anonymous. In our security evaluations we compare only to the MVMD-D algorithm, as the more challenging (i.e., more secure) competitor.

Finally, for both MVMD and MVMD-D, $\lceil\cdot\rceil$ is a function of both $\mathsf{obj}_{\vec{s}_i}$ and $\{\mathsf{obj}_{s'}\}_{s'\in\vec{s}_{1\ldots i-1}}$; i.e., the resulting padding schemes are not memoryless, as are the other algorithms we consider (including our LP). Thus, an object store that uses either of these algorithms must be capable of (i) tracking a client's previous object retrievals and (ii) storing a value for $\lceil\mathsf{obj}_s\rceil$ for each $\vec{s}_{1\ldots i-1} \in \vec{S}_{1\ldots i-1}$ such that $\mathbb{P}\left(\vec{\mathsf{S}}_i = s \;\middle|\; \vec{\mathsf{S}}_{1\ldots i-1} = \vec{s}_{1\ldots i-1}\right) > 0$.

PwoD Reed and Reiter [43] present an algorithm, called Padding without a Distribution (PwoD), that produces a per-object padding scheme $\lceil\cdot\rceil$ minimizing $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ for the case of independent object retrievals. Also, unlike BDK and MVMD-D, PwoD enforces constraint (1). We therefore include PwoD as an algorithm to which we compare.

Summary of algorithm inputs To summarize, Table 2 shows the inputs taken by each of the algorithms to which we compare, and the inputs taken by our own. BDK and MVMD-D require more detailed information than PFS, in that they require probability distributions. Still, below we will show that PFS outperforms them, in terms of both privacy and padding overhead. PwoD requires less information be input than PFS, but as we will show, PFS puts its knowledge of the sequences $\vec{S}$ to good use.

[3] Backes, et al. assume $\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \;\middle|\; \vec{\mathsf{S}}_i = s\right)$ is the same for all i.

5.3 BDK and MVMD-D comparisons

Since both BDK and MVMD-D were designed to address a different metric, in this section we compare PFS against these two algorithms in "head-to-head" tests using the competitor's metric.[4] For each test, we calculated the competitor's metric over increasing sequence lengths for each of our three datasets. Specifically, for each $i \in [\max_{\vec{s}\in\vec{S}^\Omega} \text{len}(\vec{s})]$, the metric was calculated over all $\vec{s} \in \vec{S}^\Omega$ such that $\text{len}(\vec{s}) \geq i$. Additionally, at each i, those sequences that are longer than i were truncated to i and then the metrics were calculated over these truncated sequences. This method is consistent with our calculation of $\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i} = \vec{s}_{1\ldots i}\right)$ in (23).

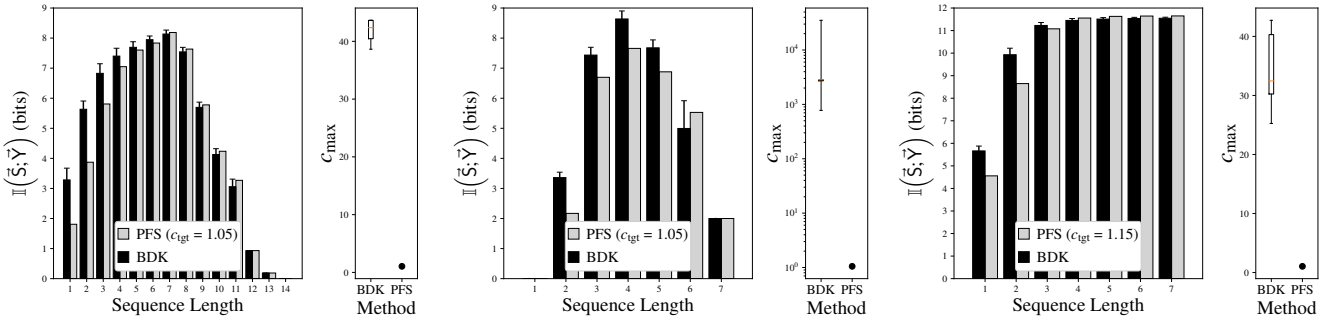
We calculated the various metrics in this way in order to model the adversary's knowledge *as the adversary observed sequential object retrievals*. For example, if the adversary observed i retrievals, then the adversary knew that $\vec{s}$ could not be any sequence where $\text{len}(\vec{s}) < i$, and so those sequences were no longer included when calculating the given metric. However, it could be the case that $\vec{s}$ was longer than i, and so those sequences were included when calculating the metric.

As mentioned in Sec. 1, the premise of our work is to enable an object store to provide privacy for its clients while simultaneously constraining the amount of padding overhead applied to each object that it serves. Therefore, in addition to comparing PFS against each algorithm using the algorithm's intended privacy metric, we also calculated the maximum pad factor c resulting from the competitor's padding scheme $\lceil\cdot\rceil$. We will leverage $c_{\max}$ when referring to this *resultant* c produced by a padding scheme $\lceil\cdot\rceil$ and c_{tgt} when referring to the *target* c that is provided as an input parameter to PFS. We do this to enable a comparison between the cost (in terms of overhead) of PFS to that of each competitor when "competing" on each of their privacy metrics.

Finally, note that for all tests in Sec. 5, we set our efficiency parameter $k = 2$. We describe the effect of varying k in Sec. 6.

BDK For our comparison to BDK [5], we measured against $\mathbb{I}\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$, as this is the metric it is designed to reduce. As mentioned in Sec. 5.2, BDK is designed to be run multiple times to produce candidate solutions for $\lceil\cdot\rceil$, from which an object store would then select the most suitable $\lceil\cdot\rceil$ in terms of privacy and padding overhead. To model this intended usage, for each dataset we ran BDK 1,000 times and, for each $i \in [\max_{\vec{s}\in\vec{S}^\Omega} \text{len}(\vec{s})]$, we plot the minimum and maximum values (the bar and whisker extending above the bar, respectively) for $\mathbb{I}\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ across all 1,000 candidate padding schemes $\lceil\cdot\rceil$. For each padding scheme we also computed its $c_{\max}$ and plot these values as a box plot, to the right of the $\mathbb{I}\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ plot.

[4] We save the comparison to PwoD for Sec. 5.4. We do this because our chosen metric $\mathbb{I}_\infty\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ is the extension of $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$—the privacy metric that PwoD minimizes—to our setting.



(a) Autocomplete (b) Linode (c) Wikipedia

Figure 2: Comparing PFS to BDK using $\mathbb{I}\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

As shown in Fig. 2, BDK was outperformed by PFS with $c_{\text{tgt}} = 1.05$ for both the Autocomplete and Linode dataset, and for $c_{\text{tgt}} = 1.15$ for Wikipedia. This is despite BDK yielding ranges for $c_{\max}$ much higher than PFS. We attribute BDK's under-performance to the fact that the algorithm will not allow two objects to be padded to the same size if doing so would violate the strict equality in (28).

We draw attention to two trends in these plots. The first is that for both the Autocomplete and Linode datasets, the values for $\mathbb{I}\left(\vec{S};\vec{Y}\right)$ rise and then fall as the sequence length increases. This is attributable to the fact that the number of sequences of a given length begins to decrease at length 7 for Autocomplete and at length 4 for Linode. The second trend is that the results for $\mathbb{I}\left(\vec{S};\vec{Y}\right)$ for Wikipedia are non-decreasing. This is attributable to the fact that, since all sequences are of length 7, the information leaked about $\vec{S}$ cannot decrease as the sequence is extended.

MVMD-D For our comparison to MVMD-D [34], we assessed each algorithm's ability to ensure that ℓ-diversity is achieved. In this section, we use ℓ_{tgt} when referring to the *target* ℓ that was provided as an input parameter to MVMD-D. For these tests, we compared MVMD-D with $\ell_{\text{tgt}} = 3$ (i.e., MVMD-3) against PFS with $c_{\text{tgt}} = 2.0$. Additionally, in Fig. 3 we depict the results of our tests using (i) $\ell_{\min}$, which we define for a given i as

$$\min_{\vec{y}_{1\ldots i} \in \vec{Y}_{1\ldots i}} \frac{1}{\max_{s \in S} \mathbb{P}\left(\vec{S}_i = s \;\middle|\; \vec{Y}_{1\ldots i} = \vec{y}_{1\ldots i}\right)} \tag{29}$$

and which is equivalent to Liu, et al.'s [34] definition of ℓ (see Sec. 5.2), and (ii) ℓ_{avg}, which we define for a given i as

$$\frac{\sum_{\vec{y}_{1\ldots i} \in \vec{Y}_{1\ldots i}} \frac{1}{\max_{s \in S} \mathbb{P}\left(\vec{S}_i = s \;\middle|\; \vec{Y}_{1\ldots i} = \vec{y}_{1\ldots i}\right)}}{\#\vec{Y}_{1\ldots i}} \tag{30}$$

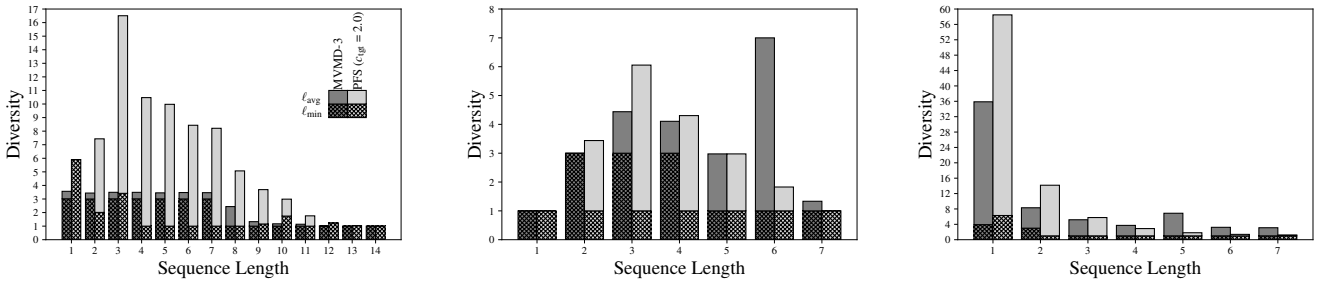
We define ℓ_{avg} in this way as it enables us to describe how privacy-preserving an algorithm's resultant padding scheme $\lceil\cdot\rceil$ was across all $\vec{y}_{1\ldots i} \in \vec{Y}_{1\ldots i}$, even if the algorithm failed to achieve ℓ-diversity in the strict sense. This is helpful even when assessing MVMD-D, as it is not guaranteed to yield a padding scheme that respects its ℓ_{tgt}.

In Fig. 3, we see that, for $c_{\text{tgt}} = 2$, PFS was generally unable to achieve $\ell_{\min} > 1$. When we broaden our analysis to gauge how well PFS achieved ℓ_{avg}, we see that PFS provided roughly the same, and in some cases much better, ℓ_{avg} than MVMD-3 for the objects in our datasets. Moreover, PFS offered this protection for $c_{\max}$ that was far less than MVMD-3's. Indeed, the $c_{\max}$ for each of MVMD-3's padding schemes $\lceil\cdot\rceil$ was similar to that of BDK across each of the three datasets. Finally, we see that even MVMD-3 yielded $\ell_{\min} = 1$ for each of the three datasets, starting at lengths 8, 5, and 3 for Autocomplete, Linode, and Wikipedia, respectively.

5.4 $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ comparisons

In this section we compare each algorithm against our chosen metric $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$. Since both PwoD and PFS take c as input, for this test we ran both algorithms with $c_{\text{tgt}} \in \{1.05, 1.25, 1.5, 2.0\}$ for each of our three datasets. For BDK we compare to a single run of the algorithm, and for MVMD-D we compare to MVMD-3. The results are shown in Fig. 4.

In both Fig. 4a and Fig. 4b we observe that, as c_{tgt} increased, there was a trend that PFS performed increasingly better than PwoD until $c_{\text{tgt}} = 2.0$, at which point PFS and PwoD performed similarly. Indeed, for $c_{\text{tgt}} = 1.5$, in Fig. 4b, PFS yielded $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ that is 7.4% lower than PwoD for length 4, and in Fig. 4a, PFS yielded $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ that is 24.2% lower than PwoD for length 8. Also, we see that BDK was not competitive in either Fig. 4a or Fig. 4b, and that MVMD-3 was not competitive in Fig. 4a but was competitive in Fig. 4b for only the longest sequences. This is despite both BDK and MVMD-3 yielding values for $c_{\max}$ that were much larger than



(a) Autocomplete (MVMD-3 $c_{\max} = 31.73$) (b) Linode (MVMD-3 $c_{\max} = 5707.33$) (c) Wikipedia (MVMD-3 $c_{\max} = 43.07$)

Figure 3: Comparing PFS to MVMD-3 using ℓ-diversity as the privacy metric.

PFS (see Fig. 2 and Fig. 3).

In Fig. 4c we observe that PwoD performed slightly better than PFS for short sequences. However, by length 5, the difference is only noticeable for $c_{\text{tgt}} = 2.0$, and by length 7, PFS yielded lower values for $\mathbb{I}_\infty\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ for $c_{\text{tgt}} = \{1.05, 1.25, 2.0\}$ (though in all cases PFS and PwoD differed by ≤ 0.01bits). We also see that both BDK and MVMD-3 were competitive for longer sequence lengths, though we reiterate that their solutions for $\lceil\cdot\rceil$ yielded large values for $c_{\max}$.

Although PwoD performed well on our three datasets—and indeed comparable to PFS on the Wikipedia dataset—it is possible for PwoD to perform arbitrarily worse than PFS as the length of sequences increase. To demonstrate this point, we created a synthetic dataset as follows. We first created 256 sequences, each of $\text{len}(\vec{s}) = 8$, i.e., each sequence represents eight successive object retrievals. For these first 256 sequences, the sizes of the objects were assigned one of $\{1\text{B}, 2\text{B}\}$ so that the resultant sequences yield all of the binary combinations from $[0, 255]$. In addition to these 256 sequences, we added a single sequence where each object in the sequence is 3B. We then created another 256 sequences as before, but whose objects were assigned one of $\{20\text{B}, 21\text{B}\}$. Finally, we added a single sequence where each object in the sequence is 10B. Thus, for our synthetic dataset $\#\vec{S}^\Omega = 514$.

Given this synthetic dataset, we set $c_{\text{tgt}} = 2.0$. In this case, $\mathbb{I}_\infty\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ is minimized when $\lceil\cdot\rceil$ pads all objects of size $\{1\text{B}, 2\text{B}\}$ to 2B and all objects of size $\{20\text{B}, 21\text{B}\}$ to 21B. This causes the smallest 256 sequences to be indistinguishable from each other, as well as the largest 256 sequences, leaving only the 3B-sequence and 10B-sequence isolated. Indeed, PFS yielded this solution for $\lceil\cdot\rceil$. The $\lceil\cdot\rceil$ given by PwoD, however, padded all objects of size $\{2\text{B}, 3\text{B}\}$ to 3B, thereby leaving the smallest 256 sequences identifiable. As shown in Fig. 5, this caused $\mathbb{I}_\infty\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ for PwoD to increase linearly as the sequence length increased, whereas the $\mathbb{I}_\infty\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ for PFS remained constant at 2 bits.

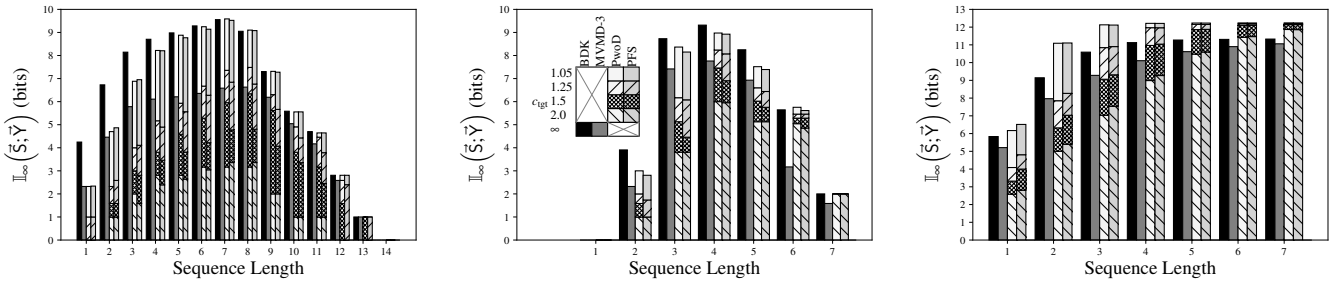
PwoD struggled against this dataset since, by default, it iterates through object sizes from largest-to-smallest, thereby assigning objects that are 20B to be padded to 21B, and objects that are 2B to be padded to 3B. However, simply running PwoD from smallest-to-largest would yield a similarly suboptimal $\lceil\cdot\rceil$, as then PwoD would pad all objects of size $\{10\text{B}, 20\text{B}\}$ to 20B, thus leaving the *largest* 256 sequences identifiable.

5.5 Attacker's recall and precision

One challenge of minimizing $\mathbb{I}_\infty\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$ in our algorithm is that this measure might be less familiar and understandable than others. To put the benefits of our algorithm into a form that might be more understandable, in this section we report the results of a study that demonstrates the privacy that our algorithm and others' achieve, in terms of the *precision* and *recall* with which a network adversary can identify sequences of interest when retrieved.

For this study, we assumed that the adversary had a set of *target* sequences $\vec{S}^\odot \subset \vec{S}^\Omega$ that it aimed to identify by observing the padded (and encrypted) traffic. The adversary observed a sequence of padded objects up to a given length m—to include the m-length prefixes of those sequences in $\vec{S}$ that are longer than m—and afterwards determined whether this sequence corresponded to a sequence from its target set $\vec{S}^\odot$. More specifically, upon observing a sequence $\vec{s}$ of object retrievals padded to sizes $\vec{y}$, the attacker can calculate $\mathbb{P}\left(\vec{\mathsf{S}} \in \vec{S}^\odot \;\middle|\; \vec{\mathsf{Y}} = \vec{y}\right)$ for any $\vec{S}^\odot$ of interest. For any threshold τ, the adversary returned *true* iff $\mathbb{P}\left(\vec{\mathsf{S}} \in \vec{S}^\odot \;\middle|\; \vec{\mathsf{Y}} = \vec{y}\right) \geq \tau$. For a given τ, the adversary's *recall* is the fraction of sequences from $\vec{S}^\odot$ for which the adversary returned *true*, and the adversary's *precision* is the fraction of its *true* detections for which the sequence was in $\vec{S}^\odot$.

We conducted the precision-recall tests as follows. For a given dataset and sequence length m, an individual trial consisted of randomly selecting 5% of the elements of $\vec{S}^\Omega$ of length m to constitute $\vec{S}^\odot$. We ran 100 such trials and we present the precision-recall curves as the average over these 100 trials. In the context of this study, lower values of the adversary's precision and recall indicate a more secure



(a) Autocomplete (b) Linode (c) Wikipedia

Figure 4: Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

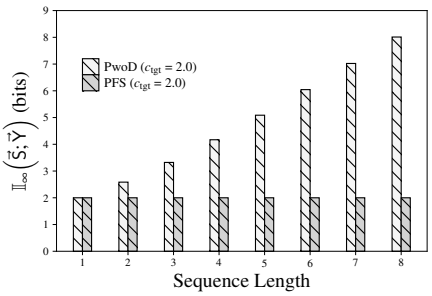


Figure 5: Comparing PFS to PwoD on the synthetic dataset.

padding algorithm.

Results for Autocomplete are depicted in Fig. 6. For this dataset, we tested at sequence lengths of 7, 8, and 9. For each of these lengths, the dataset consisted of a sufficiently large number of words to model an adversary's target set. As shown in Fig. 6, at all three lengths, PFS yielded the lowest precision and recall values. The only exception is with sequences of length 7, where MVMD-3 achieved better precision and recall values for two values of τ. Furthermore, it is evident from the plots that as the sequences grow longer, the performance of PFS increased relative to that of the other algorithms. Indeed, for sequences of length 8 and 9, PFS performed significantly better than the competitors. Moreover, an apparent trend in this dataset is that, whereas the adversary's performance against PFS was fairly consistent, the adversary improved against the other algorithms as the sequence length increased.

Fig. 7 shows the results of the precision-recall study on the Linode dataset. For this experiment, we chose sequences of length 3, as a majority of the sequences in the dataset were of length 3. In this experiment, we see that PFS achieved the lowest values of adversarial precision and recall, again beating out the competitors, though PwoD produced results that were competitive with PFS.

Fig. 8 shows the results of the precision-recall study on the Wikipedia dataset. This dataset was constructed such that all sequences have length 7. For this experiment, we found that higher values for c_{tgt} were needed to significantly reduce the adversary's performance, and so we set $c_{\text{tgt}} = 2.0$ for both PFS and PwoD. Still, despite this large value for c_{tgt}, we show in Sec. 5.6 that even this choice of c_{tgt} resulted in much lower padding overhead than BDK and MVMD-3. Overall, as shown in Fig. 8, PFS and PwoD yielded similar results, both offering better security than MVMD-3 and BDK.

5.6 Padding overhead

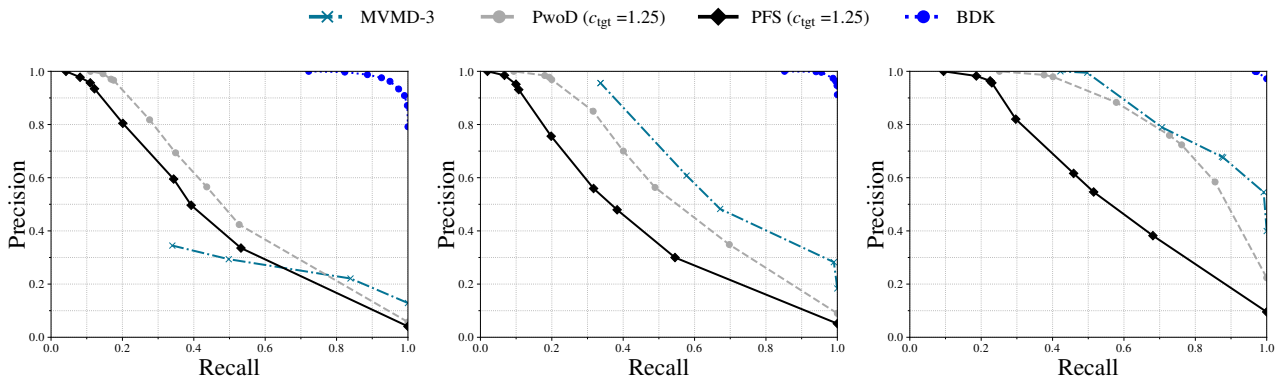
Fig. 9 depicts the padding factors produced by each algorithm, where the x-axis shows $c_{\max}$, the maximum padding factor across all objects in the dataset, and the y-axis shows c_{avg}, the mean value of the padding factors across the dataset. We calculated c_{avg} as

$$\frac{\sum_{s\in S}\sum_{y\in Y}\mathbb{P}\left(\lceil \mathsf{obj}_s \rceil = y \;\middle|\; \mathsf{S}=s\right)\times\frac{y}{|\mathsf{obj}_s|}}{\#S} \tag{31}$$

Unsurprisingly, both BDK and MVMD-D produced significantly higher values for $c_{\max}$ across all three datasets, as neither place any constraints on the padding size. On the other hand, $c_{\max}$ for both PFS and PwoD are constrained by c_{tgt}, and thus were close to the chosen value of this parameter. Regarding c_{avg}, on the Autocomplete dataset, BDK and MVMD-D yielded comparable results to PFS and PwoD since the distribution of object sizes in this dataset varies less than the others; most of the objects are in the range of 200B to 480B. For the Linode and Wikipedia datasets, though, using either BDK or MVMD-D would cause the object store's objects to grow substantially in size, and would likely be impractical to implement.

6 Execution Cost

In this section, we evaluate the execution cost of PFS on the datasets described in Sec. 5. We begin with a description of our experiments and their results in Sec. 6.1 and then describe a much faster, and nearly as good, alternative in Sec. 6.2.



(a) Sequence length = 7 (b) Sequence length = 8 (c) Sequence length = 9

Figure 6: Adversary's recall and precision for detecting words from the Autocomplete dataset.

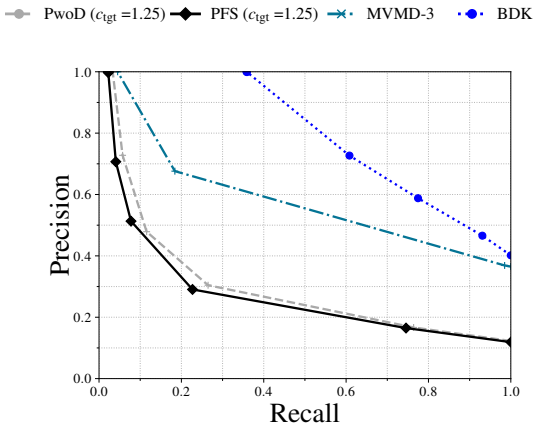


Figure 7: Adversary's recall and precision for detecting sequences of length 3 from the Linode dataset.

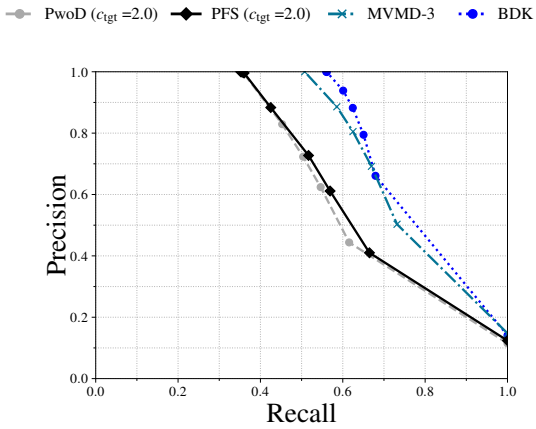


Figure 8: Adversary's recall and precision for detecting sequences of length 7 from the Wikipedia dataset.

6.1 Runtime of PFS

We implemented PFS with the Gurobi Optimizer[5] and ran tests on a server with a 24-core CPU and 128GB of memory. For each dataset, for $c_{tgt} \in \{1.01, 1.05, 1.1, 1.25, 1.5, 2.0\}$, and for $k \in \{2, 3\}$, we ran PFS and measured the time it took for PFS to (i) create the model, (ii) run the optimization, and (iii) return $\lceil \cdot \rceil$. For $k = 2$ we ran each test 10 times and report the average runtimes; for $k = 3$ we ran each test a single time. Our results are shown in Fig. 10.

Our runtime results demonstrate that on some datasets PFS (with $k = 2$) was able to quickly produce $\lceil \cdot \rceil$, e.g., under 2 seconds for Linode (Fig. 10b), and so could be run as objects are added to the object store or their sizes are modified. On those datasets where PFS takes longer to produce $\lceil \cdot \rceil$, PFS could be run on a scheduled basis and applied as batch updates to the object store.

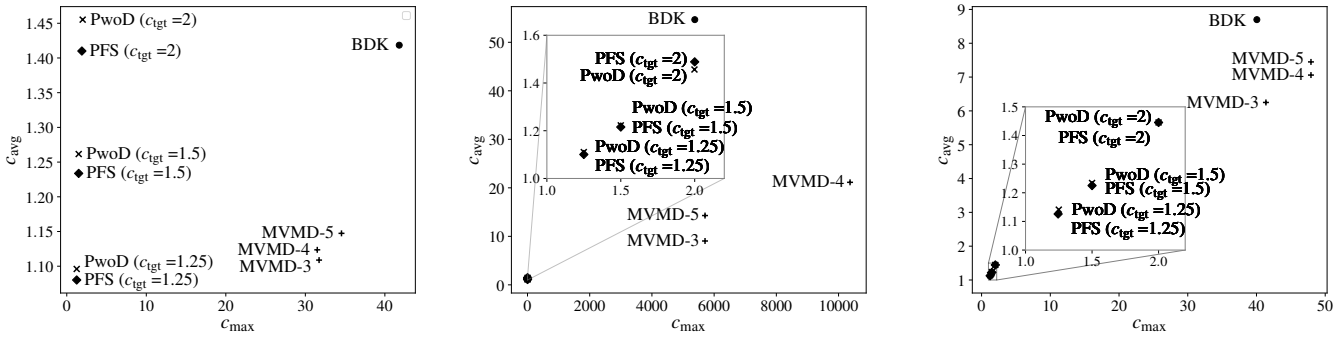
Our results also depict the challenge of increasing k. For instance, in Fig. 10a and Fig. 10c, $k = 3$ resulted in runtimes that were orders of magnitude longer than $k = 2$. Additionally, for Autocomplete (Fig. 10a), the results for PFS with $c_{tgt} = 2.0$ are not plotted as the memory requirements exceeded the capacity of our test setup.

6.2 Alternative Approach: PFG

As just described, PFS is costly for large datasets. Moreover, not all object stores may be able to produce $\vec{S}$, e.g., if it is not possible to enumerate all $\vec{s} \in \vec{S}$ or if $\#\vec{S}$ is too large to work with. For those object stores, so long as they can produce both S and E,[6] an alternative approach is to create the set $S \cup E$, i.e., treat each $s \in S$ as a sequence of length 1 and each $(s, s') \in E$ as a sequence of length 2, and then give this set in place of $\vec{S}$ to the LP from Fig. 1 without any further modifications. The rationale for this approach is that S and E together form a directed graph, from which $\vec{S}$ can be seen as a subset of the walks possible in this graph. Thus, this approach targets the

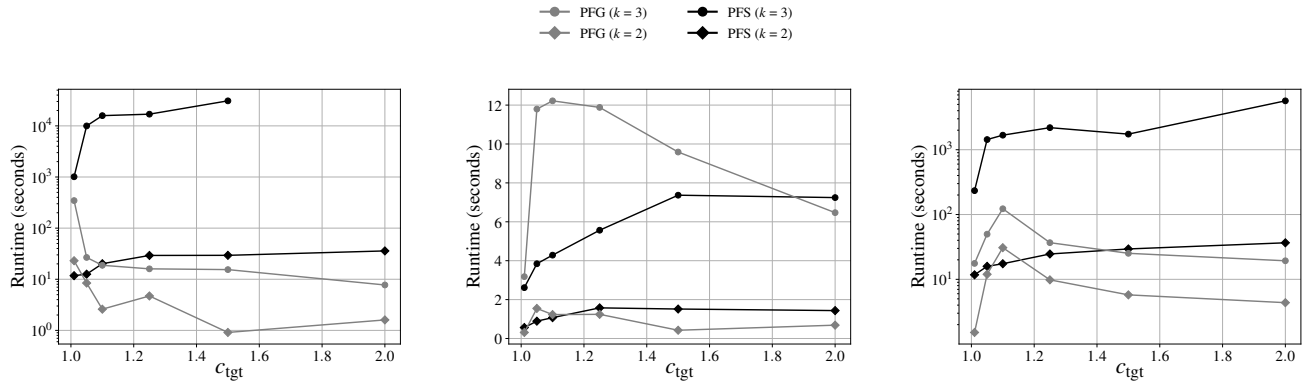
[5] https://www.gurobi.com

[6] An example of such an object store might be a web server that can easily enumerate S, the pages that it serves, and E, the hyperlinks between pages.



(a) Autocomplete (b) Linode (c) Wikipedia

Figure 9: Padding overhead factors for each padding algorithm.



(a) Autocomplete (b) Linode (c) Wikipedia

Figure 10: Model runtime for PFS and PFG as a function of c_{tgt}.

graph of objects and their dependencies in an effort to reduce $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$. We refer to this alternative method as Padding for Graphs (PFG).

We additionally executed PFG and show its performance results in Fig. 10, alongside those for PFS. Our results show that PFG is quite efficient overall. For all three datasets, PFG (with $k = 2$) yielded fast runtimes, with most tests finishing in under 10 seconds. Furthermore, our results show that, for the same k, PFG tended to beat PFS by a wide margin in our runtime evaluations, with the only exception being for Linode (Fig. 10b) with $k = 3$.

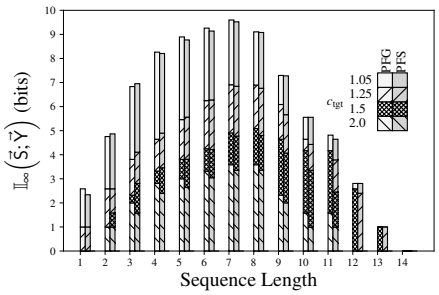
We also evaluated PFG to compare its privacy results to those of PFS, i.e., using the same tests as in Fig. 4. We present the results of this test in Fig. 11. (We do not present the results for the Linode dataset, as PFG yielded values for $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ that were almost indistinguishable from PFS.)

In Fig. 11, we see that for both Autocomplete and Wikipedia, for $c_{\text{tgt}} \in \{1.25, 1.5\}$, PFG outperformed PFS for shorter sequences (up to length 5 for Autocomplete and length 4 for Wikipedia). This is particularly noteworthy for the Autocomplete dataset, given that PFS outperformed PwoD by quite a large margin for these same values of c_{tgt}. We attribute these results to the fact that PFG optimizes for short sequences (specifically, single-object and two-object sequences).

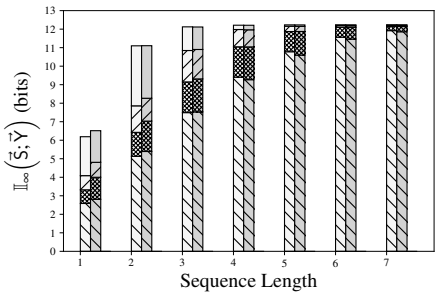
As sequence lengths increased, though, Fig. 11 shows that PFS began to outperform PFG. This is most apparent on the Autocomplete dataset for lengths ≥ 9. Still, these results demonstrate that if an object store is unable to use PFS (due to an inability to produce $\vec{S}$ or due to runtime concerns), it can still use PFG and leverage its knowledge of S and E to effectively reduce $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$.

7 Conclusion

In this paper, we provided an algorithm, PFS, that is able to efficiently produce a padding scheme $\lceil\cdot\rceil$ that an object store can use to conceal from a network observer the objects it serves to clients, despite the network observer leveraging object dependencies in its inferences. We compared PFS's security against prior algorithms, including on the security metrics that those algorithms themselves aim to optimize. We showed



(a) Autocomplete



(b) Wikipedia

Figure 11: Comparing PFG to PFS with $k = 2$.

that PFS outperformed prior work while maintaining reasonable constraints on the padding overhead. Compared to BDK, PFS achieved lower values for $\mathbb{I}\left(\vec{S};\vec{Y}\right)$ with little padding overhead; against PwoD, PFS achieved either comparable or lower values of $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$; and compared to MVMD-D, PFS achieved better diversity across all objects in each dataset, for less padding overhead. We also compared each algorithm by assessing an adversary's precision and recall as it predicted sequences of objects by observing their padded sizes. Our results showed that PFS was highly effective in such a setting, as it was consistently either among the top performers or beat the other algorithms by a wide margin. Most importantly, in all cases, PFS was constrained by the tunable padding factor c. By providing a configurable limit on the padding overhead, PFS is able to provide an object store with a means to pad sequences of objects that is both effective against a network observer and that is practical to implement. Finally, we provided a competitive alternative to PFS named PFG that is based on the same underlying LP, and which is suitable in settings where an object store is unable to generate $\vec{S}$ or doing so is too costly.

Availability

Our datasets and implementations are available at https://github.com/pranay-jain/constrained-padding-sequences.

Acknowledgments

This work was supported in part by NSF grant 2207214. The views expressed in this paper are those of the authors and do not reflect the official policy or position of the U.S. Military Academy, the Department of the Army, the Department of Defense, the National Science Foundation, or the U.S. Government.

References

[1] I. Abraham, B. Pinkas, and A. Yanai. Blinder – scalable, robust anonymous committed broadcast. In *27th ACM Conference on Computer and Communications Security*, pages 1233–1252, October 2020.

[2] H. F. Alan and J. Kaur. Client diversity factor in HTTPS webpage fingerprinting. In *9th ACM Conference on Data and Application Security and Privacy*, March 2019.

[3] M. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *25th IEEE Computer Security Foundations Symposium*, June 2012.

[4] G. Asharov, T.-H. Hubert Chan, K. Nayak, R. Pass, L. Ren, and E. Shi. Locality-preserving oblivious RAM. In *Advances in Cryptology – EUROCRYPT 2019*, volume 11477 of *Lecture Notes in Computer Science*, pages 214–243, May 2019.

[5] M. Backes, G. Doychev, and B. Kopf. Preventing side-channel leaks in web traffic: A formal approach. In *20th ISOC Network and Distributed System Security Symposium*, February 2013.

[6] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy vulnerabilities in encrypted HTTP streams. In *5th Privacy Enhancing Technologies Symposium*, volume 3856 of *Lecture Notes in Computer Science*, pages 1–11, May 2005.

[7] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *19th ACM Conference on Computer and Communications Security*, pages 605–616, October 2012.

[8] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *22nd ACM Conference on Computer and Communications Security*, pages 668–679, October 2015.

[9] A. Chakraborti, A. Aviv, S. G. Choi, T. Mayberry, D. Roche, and R. Sion. rORAM: Efficient range ORAM with $o(\log_2 n)$ locality. In *26th ISOC Network and Distributed System Security Symposium*, February 2019.

[10] Z. Chang, D. Xie, and F. Li. Oblivious RAM: A dissection and experimental evaluation. *Proceedings of the VLDB Endowment*, 9(12), August 2016.

[11] H. Cheng and R. Avnur. Traffic analysis of SSL encrypted web browsing. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.1201, 1998.

[12] G. Cherubin, J. Hayes, and M. Juarez. Website fingerprinting defenses at the application layer. *Proceedings on Privacy Enhancing Technologies*, 2017(2):186–203, 2017.

[13] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6), 1998.

[14] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *36th IEEE Symposium on Security and Privacy*, May 2015.

[15] G. Danezis. Traffic analysis of the HTTP protocol over TLS. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.92.3893, 2003.

[16] G. Danezis. The traffic analysis of continuous-time mixes. In *4th Privacy Enhancing Technologies Symposium*, volume 3424 of *Lecture Notes in Computer Science*, May 2004.

[17] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *33rd IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[18] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3), May 1996.

[19] R. Gonzalez, C. Soriente, and N. Laoutaris. User profiling in the time of HTTPS. In *16th Internet Measurement Conference*, pages 373–379, November 2016.

[20] P. Grubbs, A. Khandelwal, M.-S. Lacharité, L. Brown, L. Li, R. Agarawal, and T. Ristenpart. PANCAKE: Frequency smoothing for encrypted data stores. In *29th USENIX Security Symposium*, August 2020.

[21] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *40th IEEE Symposium on Security and Privacy*, May 2019.

[22] J. Hayes and G. Danezis. *k*-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium*, pages 1187–1203, August 2016.

[23] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-Bayes classifier. In *1st ACM Workshop on Cloud Computing Security*, pages 31–42, November 2009.

[24] A. Hintz. Fingerprinting websites using traffic analysis. In *2nd Privacy Enhancing Technologies Symposium*, volume 2482 of *Lecture Notes in Computer Science*, pages 171–178, April 2002.

[25] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th ISOC Network and Distributed System Security Symposium*, February 2012.

[26] I. Issa, A. B. Wagner, and S. Kamath. An operational approach to information leakage. *IEEE Transactions on Information Theory*, 66(3), March 2020.

[27] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A critical evaluation of website fingerprinting attacks. In *21st ACM Conference on Computer and Communications Security*, pages 263–274, November 2014.

[28] S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In *Advances in Cryptology – EUROCRYPT 2019*, volume 11477 of *Lecture Notes in Computer Science*, pages 183–213, May 2019.

[29] S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakage suppression. In *Advances in Cryptology – CRYPTO 2018*, volume 10991 of *Lecture Notes in Computer Science*, pages 339–370, August 2018.

[30] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In *23rd ACM Conference on Computer and Communications Security*, October 2016.

[31] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. Data recovery on encrypted databases with *k*-nearest neighbor query leakage. In *40th IEEE Symposium on Security and Privacy*, May 2019.

[32] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. Timing attacks in low-latency mix systems. In *8th International Conference on Financial Cryptography and Data Security*, volume 3110 of *Lecture Notes in Computer Science*, pages 251–265, September 2004.

[33] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *13th ACM Conference on Computer and Communications Security*, pages 255–263, October 2006.

[34] W. M. Liu, L. Wang, P. Cheng, K. Ren, S. Zhu, and M. Debbabi. PPTP: Privacy-preserving traffic padding

in web-based applications. *IEEE Transactions on Dependable and Secure Computing*, 11(6):538–552, November-December 2014.

[35] W. M. Liu, L. Wang, K. Ren, P. Cheng, and M. Debbabi. k-indistinguishable traffic padding in web applications. In *12th Privacy Enhancing Technologies Symposium*, volume 7384 of *Lecture Notes in Computer Science*, pages 79–99, July 2012.

[36] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *18th ISOC Network and Distributed System Security Symposium*, February 2011.

[37] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), March 2007.

[38] C. Mavroforakis, N. Chenette, A. O'Neill, G. Kollios, and R. Canetti. Modular order-preserving encryption, revisited. In *ACM SIGMOD International Conference on Management of Data*, pages 763–777, May 2015.

[39] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar. I know why you went to the clinic: Risks and realization of HTTPS traffic analysis. In *14th Privacy Enhancing Technologies Symposium*, volume 8555 of *Lecture Notes in Computer Science*, pages 143–163, July 2014.

[40] K. Nikitin, L. Barman, W. Lueks, M. Underwood, J.-P. Hubaux, and B. Ford. Reducing metadata leakage from encrypted files and communication with PURBs. *Proceedings on Privacy Enhancing Technologies*, 2019(4):6–33, 2019.

[41] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *10th Workshop on Privacy in the Electronic Society*, pages 103–114, October 2011.

[42] H. H. Pang, X. Xiao, and J. Shen. Obfuscating the topical intention in enterprise text search. In *28th IEEE International Conference on Data Engineering*, April 2012.

[43] A. C. Reed and M. K. Reiter. Optimally hiding object sizes with constrained padding. In *IEEE Computer Security Foundations Symposium*, July 2023.

[44] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, November/December 2001.

[45] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *8th European Symposium on Research in Computer Security*, volume 2808 of *Lecture Notes in Computer Science*, pages 116–131, October 2003.

[46] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *21st IEEE Symposium on Security and Privacy*, pages 44–55, May 2000.

[47] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *23rd IEEE Symposium on Security and Privacy*, pages 19–30, May 2002.

[48] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[49] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium*, pages 143–157, August 2014.

[50] T. Wang and I. Goldberg. Improved website fingerprinting on Tor. In *12th Workshop on Privacy in the Electronic Society*, pages 201–212, November 2013.

[51] J. Yan and J. Kaur. Feature selection for website fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2018(4), October 2018.

[52] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. On flow correlation attacks and countermeasures in mix networks. In *4th Privacy Enhancing Technologies Symposium*, volume 3424 of *Lecture Notes in Computer Science*, pages 207–225, May 2004.