Optimally Hiding Object Sizes with Constrained Padding

Andrew C. Reed
United States Military Academy
andrew.reed@westpoint.edu

Michael K. Reiter
Duke University
michael.reiter@duke.edu

Abstract—Among the most challenging traffic-analysis attacks to confound are those leveraging the sizes of objects downloaded over the network. In this paper we systematically analyze this problem under realistic constraints regarding the padding overhead that the object store is willing to incur. We give algorithms to compute privacy-optimal padding schemes—specifically that minimize the network observer's information gain from a downloaded object's padded size—in several scenarios of interest: per-object padding, in which the object store responds to each request for an object with the same padded copy; per-request padding, in which the object store pads an object anew each time it serves that object; and a scenario unlike the previous ones in that the object store is unable to leverage a known distribution over the object queries. We provide constructions for privacy-optimal padding in each case, compare them to recent contenders in the research literature, and evaluate their performance on practical datasets.

I. Introduction

The transmission of objects in a way that hides the sizes of objects transmitted from a network observer—either to hide which of many potential objects is transmitted, or as an ingredient in hiding which sender and receiver are communicating—is a longstanding problem in traffic-analysis defense. Indeed, the sizes of objects transmitted has been shown to single-handedly enable fingerprinting websites or webpages, even in the presence of otherwise sophisticated defenses against this practice (e.g., [30]). Despite the utility of object sizes in traffic analysis, the expense of hiding object sizes is such that substantial threads of research on private communication either do not even attempt to hide object sizes (e.g., in low-latency anonymous messaging [26], [51], [70], [88] or protocols for private video downloads, e.g., [87]) or restrict attention to fixed-sized, small messages (e.g., anonymous microblogging systems [1], [21]).

In this paper we consider a fundamental and practical instance of this problem, in which a benevolent object store enables clients to retrieve its objects. Each client's communication with the object store is encrypted for that client, but the sizes of objects it retrieves is nevertheless revealed to a network observer. This network observer might also be one of the clients of the object store and so can retrieve objects himself. We further allow this observer (and the object store, potentially) to know the frequency of requests for each object. Being benevolent, the object store is willing to pad objects before sending them, so that their sizes do not directly disclose to the network observer the objects others retrieve. However, the benevolence of the object store extends only so far; since padding objects consumes more of its bandwidth, the object store is willing to pad objects only so much. The question we

consider here is: how should the object store pad its objects subject to this constraint, to best hide which object a client retrieves from this network observer?

More specifically, for any object identifier $s \in S$, where S is the set of all object identifiers, let obj_s denote the object with identifier s at the object store, and let $|\mathsf{obj}_s| \in \mathbb{N}$ denote the size of obj_s. Consider random variables S, X, and Y, which take on the object identifier s in a request, the corresponding object's actual size $|\mathsf{obj}_s|$, and the object's padded size when returned, respectively. S is distributed according to a public probability distribution, and so

$$\mathbb{P}(\mathsf{X} = x) = \sum_{s \in S: |\mathsf{obj}_s| = x} \mathbb{P}(\mathsf{S} = s)$$

is also public for each size $x \in \mathbb{N}$. The goal of the object store is to select a padding scheme $\lceil \cdot \rceil$ that pads each object obj_s to a (possibly randomized) size $\lceil \mathsf{obj}_s \rceil$ before sending it, in which case

$$\mathbb{P}(\mathsf{Y} = y) = \sum_{s \in S} \mathbb{P}(\lceil \mathsf{obj}_s \rceil = y \mid \mathsf{S} = s)\mathbb{P}(\mathsf{S} = s)$$

We presume that objects are served in full and cannot be compressed, and so

$$\mathbb{P}(\lceil \mathsf{obj}_s \rceil < |\mathsf{obj}_s|) = 0 \tag{1}$$

for all $s \in S$. Moreover, as mentioned above, the object store is willing to pad each object only so much. We capture this constraint by requiring, for a specified constant $c \geq 1$, that

$$\mathbb{P}(\lceil \mathsf{obj}_s \rceil > c \times |\mathsf{obj}_s|) = 0 \tag{2}$$

for all $s \in S$. An object store might prefer (2) to requiring only that objects be expanded by a factor of at most c in expectation—i.e., that $\mathbb{E}\left(\frac{\lceil \mathsf{obj}_s \rceil}{|\mathsf{obj}_s|}\right) \leq c$, where the expectation is taken with respect to the distribution of S and any random choices of $\lceil \cdot \rceil$—for fairness. Limiting padding overhead only in expectation would still permit some objects to be expanded by more than a factor of c, which might cause some clients to unduly suffer if they retrieve that object more than others or do so over a bandwidth-limited (or priced) connection.

Having adopted constraints (1)–(2), the object store cannot hide *all* objects retrieved from the network observer. For example, an object obj_s for which $c \times |\mathsf{obj}_{s'}| < |\mathsf{obj}_s|$ or $c \times |\mathsf{obj}_s| < |\mathsf{obj}_{s'}|$ for every other object $\mathsf{obj}_{s'}$ will be the only object padded to a value in the range $[|\mathsf{obj}_s|, c \times |\mathsf{obj}_s|]$. So, obj_s will be identifiable to the network observer when returned to a client. Rather than give up and return to today's status quo,

however, the benevolent object store strives to protect client privacy as well as it can (subject to (1)–(2)). In this paper, we presume the measure of client privacy the object store seeks to minimize is the mutual information between S and Y, also referred to as the *information gain*, i.e.,

$$\mathbb{I}(S;Y) = \mathbb{H}(S) - \mathbb{H}(S \mid Y) \qquad (3)$$

where $\mathbb{H}$ denotes entropy. That is, the object store seeks to choose a padding scheme $\lceil \cdot \rceil$ to minimize $\mathbb{I}(S;Y)$ subject to constraints (1)–(2). Note that since $\mathbb{H}(S)$ is a constant, minimizing $\mathbb{I}(S;Y)$ thereby maximizes $\mathbb{H}(S \mid Y)$. This is useful for privacy because $\mathbb{H}(S \mid Y)$ has been shown by Rioul [67, Theorem 12] to lower-bound the guessing entropy $\mathbb{G}(S \mid Y)$ according to

$$\mathbb{G}(S \mid Y) > \frac{2^{\mathbb{H}(S \mid Y)}}{e} + \frac{1}{2} \qquad (4)$$

Thus, minimizing $\mathbb{I}(S;Y)$ increases the expected number of guesses needed by the adversary to guess S given Y.

In this context, our algorithmic contributions, detailed in Sec. III, are as follows:

- In a *per-object padding* scenario in which the object store pads each object a single time and serves this single padded object in response to repeated requests (including possibly one from the network observer), $\lceil obj_s \rceil$ is fixed across retrievals of obj_s and so $\lceil \cdot \rceil$ is, in effect, a function. In this case, we characterize the privacy-optimal scheme $\lceil \cdot \rceil$, i.e., that minimizes $\mathbb{I}(S;Y)$, by showing that it is nondecreasing in the sense that if $|obj_s| \le |obj_{s'}|$ then $\lceil obj_s \rceil \le \lceil obj_{s'} \rceil$. We use this finding to give an explicit algorithm to choose $\lceil \cdot \rceil$ to minimize $\mathbb{I}(S;Y)$ subject to the above constraints. This algorithm computes $\lceil \cdot \rceil$ in $O(\#S^2)$ time, where $\#S$ is the cardinality of S.

- In a *per-request padding* scenario in which the object store pads each object anew before serving it each time, we observe that we can express our problem as an instance of *rate-distortion minimization* as originally considered by Shannon [72]. By doing so, we can apply classic results to compute the privacy-optimal $\lceil \cdot \rceil$ numerically.

- Both of the above contributions require the object store to accurately predict the distribution of S, i.e., the distribution of requests it will receive, in order to compute the optimal padding scheme $\lceil \cdot \rceil$. In cases where the object store cannot do so, e.g., if the adversary can affect S, we give an algorithm to solve for a padding scheme $\lceil \cdot \rceil$ that pads objects to achieve a measure that upper-bounds $\mathbb{I}(S;Y)$ even for an adversarially chosen distribution. Perhaps surprisingly, this algorithm computes $\lceil \cdot \rceil$ in $O(\#S)$ time and so is the most scalable of our algorithms.

In Sec. IV, we empirically evaluate these algorithms using two real-world datasets to compare the security they provide to recently published algorithms for similar goals, for both per-object padding and per-request padding [20], [63]. Our evaluation shows that in terms of both information gain for the adversary and the adversary's practical ability to detect object retrievals as being in classes of interest, our per-request

padding algorithm provided better security than the per-request padding contender, and similarly, our per-object padding algorithm dominated its contenders. Even our algorithm that does not leverage a distribution for S remained competitive, while being intrinsically robust to any mistakes in estimating that distribution that would cripple the other algorithms.

We report our algorithms' performance on these datasets in Sec. V. Only our algorithm for finding the privacy-optimal per-request padding scheme presented scaling challenges for large numbers of objects and large values of c. However, we show that once it computed a solution, this solution could be incrementally adapted in response to object (size) changes much faster than computing the distribution used by $\lceil \cdot \rceil$ from scratch.

Finally, we discuss limitations and possible extensions of our results in Sec. VI. We conclude in Sec. VII. Our implementations and datasets are available on GitHub[1].

II. RELATED WORK

Mutual information as a privacy measure: Mutual information $\mathbb{I}(S;Y)$ between a random variable Y that *will* be disclosed and secret information S that should *not* be disclosed has been used to measure the information leakage from S to Y in numerous contexts for over 70 years (e.g., [24], [28], [34], [37], [48], [50], [59], [69], [71], [81], [82], [86]). Perhaps closest to our work is one such example due to Köpf and Dürmuth [48], who present techniques to thwart timing attacks by minimizing an upper-bound on the mutual information between a sequence of side-channel measurements (of response times) and an encryption key. In their setting, the performance penalty is calculated as the increase to the average response time after delaying (i.e., padding) those times. Since they assume that the channel will be used to send multiple responses, calculating the average overhead across all responses makes sense. As stated in Sec. I, though, we adopt a per-object overhead constraint that, we believe, is more appropriate for our scenario. This, in turn, leads to more efficient runtimes for our algorithms that are most similar to theirs, namely, our per-object algorithm and our unknown distribution algorithm, described in Sec. III-A and Sec. III-C, respectively.

Relative to the above work, our contribution lies in leveraging mutual information specifically to optimize the padding applied to objects to hide the object returned over a network, subject to padding size constraints, a problem for which heuristic solutions continue to be published (e.g., [20], [63], as we will discuss in Sec. IV). As we will show, in some cases we can adapt known methods for minimizing mutual information in other contexts, and in others we develop novel and very efficient algorithms for doing so.

Alternatives to mutual information: Despite the longevity and pervasiveness of mutual information as a privacy measure, it is not without its critics (e.g., [42], [75]). For instance, Smith [75] presents two programs—one that outputs s (a uniformly drawn $8k$-bit integer) when $s \equiv 0 \mod 8$ and another that reveals (only) $s \mod 2^{k+1}$ always—to argue that mutual information is not a good measure of privacy: the first

---

[1] https://github.com/andrewreed/constrained-padding

program yields a lower value for mutual information than the second, even though it can also disclose s completely. Translated to our context, then, a padding scheme $\lceil \cdot \rceil$ that minimizes $\mathbb{I}(S; Y)$ might still allow for some objects to be identified by their sizes. Note, however, that this is already a possibility due to the object store's adoption of (1)–(2) as discussed in Sec. I, and would remain a possibility under these constraints regardless of the privacy definition we adopt.

As an alternative to mutual information, Smith advocates defining leakage as $\mathbb{H}_\infty(S) - \mathbb{H}_\infty(S \mid Y)$, i.e., as the difference between the min-entropy of S and the conditional min-entropy of S given Y. Alvim et al. [3] extend this notion of leakage with g-leakage, which broadens (and subsumes) the min-entropy model first proposed by Smith [75] by introducing a gain function that can be used to model other forms of adversarial guessing. Alvim et al. [3, Theorem 5.1] further show that min-capacity—the supremum of min-entropy leakage across all distributions S—upper-bounds g-leakage for all gain functions. Min-capacity is, in fact, equivalent to the metric that we minimize in Sec. III-C.

Padding to achieve other privacy measures: Other measures of privacy for padding security have been studied in contexts similar to ours [54], [55], notably adaptations of measures initially proposed for ensuring privacy of microdata in statistical databases, namely: k-anonymity [68], [77], which requires that for every possible padded size there must be at least k objects padded to that size; and its generalization ℓ-diversity [57], which requires that no object accounts for more than $\frac{1}{\ell}$ of any padded size's probability. Aside from having critics of their own (e.g., [52]), these measures are orthogonal to mutual information: subject to padding constraints, minimizing mutual information does not necessarily achieve the maximum ℓ for ℓ-diversity, and maximizing ℓ does not necessarily minimize mutual information. Furthermore, as discussed in Sec. I, combining ℓ-diversity with a per-object padding constraint c is problematic, as a given ℓ may not be achievable for a given c. Thus, we see both ℓ-diversity and k-anonymity as ill-suited for our setting.

Differential privacy [29] is another privacy measure that is used when a publicly available database must also protect the privacy of the individuals whose data is contained in the database. Differential privacy does so by adding noise to released data so as to render two databases (that differ in only one element) practically indistinguishable. Normally, this noise is allowed to be either negative or positive, and so would be unsuitable for our setting. However, Case et al. [12] recently analyzed truncated mechanisms to provide (ϵ, δ)-differential privacy in scenarios where noise must be non-negative. In the (ϵ, δ)-differential privacy setting, δ must be very small in order to be a meaningful application of (ϵ, δ)-differential privacy [29, Chapter 2.3]. Unfortunately, small δ leads to very large overhead. For instance, Case et al. [12] show that, for the counting query where sensitivity is one, the truncated discrete Laplace mechanism with $\epsilon = 0.5$ and $\delta = 10^{-6}$ yields an expected overhead of $25\times$ the sensitivity. In our setting, sensitivity will typically be much greater than one as objects can differ greatly in size, and so this mechanism will yield exceptionally large overhead (on average) if (ϵ, δ)-differential privacy must be meaningfully upheld. So, just as with ℓ-diversity and k-anonymity, we find differential privacy

to be impractical for our setting.

Leakage based on communication volume: Kellaris et al. [47] analyzed an "outsourced" (i.e., untrusted) object store that returns some subset of its objects in response to *range queries* on their (encrypted) search keys by clients. They evaluated basic sources of leakage that practical defenses might permit, of which one is *communication volume*. This form of leakage occurs when the object store observes *how many* objects it returns to the client (but not which ones), as systems leveraging ORAMs (e.g., [6], [15], [32]) would typically leak. (See also Naveed [62].) While communication volume leakage bears some similarity to our problem, our study differs in the threat model (our object store is trusted, theirs is not), what is sensitive (search terms in their case, objects in ours), and the nature of the results (they presented attacks, we present defenses).

Generic defenses against communication-volume leakage in the model of Kellaris et al. have been explored recently (e.g., [45], [46]). Lossless defenses (as we require here) provide strong privacy but retrieve an object via multiple fixed-sized retrievals—of total size larger than the original object, and so themselves padded—over multiple rounds of interaction. In contrast, our design does not require multiple rounds or otherwise alter the communication pattern of object retrievals (aside from padding them), and focuses on limiting bandwidth overhead to a maximum multiplicative overhead per object while achieving the best privacy that limit allows against a network observer.

Leakage based on access patterns: The second basic source of leakage analyzed by Kellaris et al. [47] is *access patterns*, in which the object store observes *which* objects it returns to the client, as would be typical of systems based on searchable symmetric and structured encryption (e.g., [14], [16], [18], [44]) or on deterministic and order-preserving encryption (e.g., [4], [66]). Various other works have studied access-pattern leakage and its detrimental effects on privacy against an untrusted object store (e.g., [13], [36], [41], [49]). Defenses against this risk tend to incorporate fake queries (e.g., [35], [60], [65]), again which we eschew here due to their overheads, or ORAMs, whose overheads are even worse (e.g., [17]). Still, most defenses against access-pattern leakage assume all objects are the same size (e.g., [35]), so that object lengths do not leak information. It is exactly this assumption that we seek to relax.

Webpage fingerprinting: The context within which traffic analysis has been most often considered recently is webpage fingerprinting. In this context, a network observer seeks to determine which webpage (or which website) a user is accessing based on features visible to the observer. The variety of features that the observer might leverage is vast [83], but it has been shown that communication volume is particularly powerful for fingerprinting webpages [30]. Webpage fingerprinting has been attempted within, among others, HTTPS traffic [2], [19], [25], [33], [61]; encrypted web-proxy traffic [40], [76]; SSH proxy tunnels [9], [53]; netflow records [22], [84]; packet headers [58]; and Tor traffic [11], [20], [38], [39], [43], [64], [79], [80]. Many (though not all) proposed defenses against webpage fingerprinting exploit protocol features in TCP and/or HTTP (e.g., [56], [76]).

TABLE I: Objects used in algorithm illustrations in Sec. III.

Label	URL (accessed Apr 25, 2021) https://images.unsplash.com/...	Size (B)	Downloads per day
P0	photo-1572095426476-808d659b4ea3	2493855	2.53
P1	reserve/qstJZUtQ4uAjijbpLzbT_LO234824.JPG	3833489	27.92
P2	photo-1583582829797-b2990fb9946b	7929946	5.41
P3	photo-1591672524177-261a7744a2b6	13322074	12.41
P4	photo-1579832888877-74d7a790df36	13589747	1.09
P5	photo-1558136015-7002a0f5e58d	16235142	5.54
P6	photo-1586030307451-dfc64907aaa5	16719886	10.65
P7	photo-1558729923-720bbb76a430	19437984	5.07
P8	photo-1528233090455-e245a0c50575	25905442	2.27
P9	photo-1559422721-1ed9b8d28236	34389677	4.23

While padding web objects has been considered as a defense (e.g., [20], [76]), we know of no work that shows how to pad objects so as to maximize privacy subject to a bandwidth overhead constraint, as we do here. In particular, object padding schemes have been proposed to ensure k-anonymity and ℓ-diversity [54], [55], as well as to reduce mutual information [7]; while these works are more ambitious than ours in attempting to address correlated, sequential retrievals typical of web applications, these proposed defenses do not provably optimize the tradeoff between privacy and overhead. For example, the defense proposed by Backes et al. [7] must be run repeatedly to generate candidate padding schemes, from which an object store would then select a scheme that offers a tradeoff between privacy and overhead that it finds acceptable, but with no guarantee of how close to optimal the resulting scheme is. We instead focus on the simpler scenario of independent object retrievals, for which we are able to provide provably optimal privacy subject to an overhead constraint.

III. ALGORITHMS

In this section we develop algorithms for calculating the padding scheme $\lceil \cdot \rceil$ that optimally achieves privacy subject to padding overhead constraints (1)–(2). We address multiple scenarios: "per-object padding," in which objects are padded once and then provided in response to requests distributed according to the known distribution on S (Sec. III-A); "per-request padding," in which objects can be padded anew in response to each request, with each request again distributed according to the known distribution on S (Sec. III-B); and a third scenario in which the distribution on S is unknown to the object store (Sec. III-C). In all cases, our target is to minimize the mutual information $\mathbb{I}(\mathsf{S}; \mathsf{Y})$ of S and the padded object sizes Y revealed to the attacker. Once $\lceil \cdot \rceil$ has been calculated, each invocation $\lceil \mathsf{obj}_s \rceil$ involves simply sampling from the distribution for Y conditioned on the event $\mathsf{S} = s$ and then padding obj_s to the sampled size, and so each invocation is very efficient. As such, the primary cost we focus on in this paper is the cost of calculating the distribution of Y conditioned on S.

For each algorithm we provide, we illustrate the padding scheme $\lceil \cdot \rceil$ it produces for objects selected from one of the datasets we utilize in our evaluations in Sec. IV–V. We defer detailed discussion of these datasets to that section, but the objects selected for illustration in this section are shown in Table I. We selected these objects to illustrate the

differences among algorithms effectively; we do not claim that the selected objects are representative of the dataset from which we drew them. Central to two of our algorithms is knowing the frequency with which each object is requested, so that we can estimate the distribution of S. For the objects listed in Table I, this information is provided in the "Downloads per day" column.

We reiterate that our threat model permits an attacker to observe the sizes of objects returned in response to others' requests, and to query the object store itself to observe padded objects. However, we assume that these are the only features available to the attacker. In particular, the sizes of all requests (as observable on the network) are the same, and neither others' requests nor the contents of the objects returned to these requests are visible to the attacker. We further assume that the *contents* of objects returned to others' requests have no observable effect on network-level features available to the attacker; i.e., two different objects, if padded to the same size, will be indistinguishable to the attacker.

A. Per-object Padding

The case of per-object padding differs from per-request padding in that $\lceil \mathsf{obj}_s \rceil$ is invoked only once per identifier s. All queries for s then return this once-padded object. As such, we can view $\lceil \cdot \rceil$ as a deterministic function in this case. Per-object padding is appropriate when the expense of padding anew for each query is deemed too costly, or if objects will be diffused via content-distribution networks (CDNs) outside the object store's control.

A classic result (e.g., [23, Theorem 2.4.1]) regarding mutual information is that in addition to (3),

$$\mathbb{I}(\mathsf{S}; \mathsf{Y}) = \mathbb{H}(\mathsf{Y}) - \mathbb{H}(\mathsf{Y} \mid \mathsf{S}) \tag{5}$$

As such, when $\mathsf{Y} = \lceil \mathsf{obj}_\mathsf{S} \rceil$ is a deterministic function of S, as in this case, then $\mathbb{H}(\mathsf{Y} \mid \mathsf{S}) = 0$ and so $\mathbb{I}(\mathsf{S}; \mathsf{Y}) = \mathbb{H}(\mathsf{Y})$. Therefore, to minimize $\mathbb{I}(\mathsf{S}; \mathsf{Y})$ it suffices to minimize $\mathbb{H}(\mathsf{Y})$.

In the remainder of this section, we show how to compute the optimal function $\lceil \cdot \rceil$ for per-object padding. We first prove an important property about the optimal $\lceil \cdot \rceil$ in Sec. III-A1 and then provide an algorithm to compute the optimal $\lceil \cdot \rceil$ efficiently in Sec. III-A2.

1) The Privacy-Optimal $\lceil \cdot \rceil$ is Nondecreasing: In this section we prove that any privacy-optimal scheme $\lceil \cdot \rceil$ for per-object padding is *nondecreasing* in object size, in the sense that $|\mathsf{obj}_s| < |\mathsf{obj}_{s'}| \Rightarrow \lceil \mathsf{obj}_s \rceil \leq \lceil \mathsf{obj}_{s'} \rceil$. To do so, consider any function $\lceil \cdot \rceil$ satisfying padding constraints (1)–(2) that is *not* nondecreasing, i.e., for which there are objects obj_s and $\mathsf{obj}_{s'}$ with $|\mathsf{obj}_s| < |\mathsf{obj}_{s'}|$ but for which $\lceil \mathsf{obj}_s \rceil > \lceil \mathsf{obj}_{s'} \rceil$. Since $|\mathsf{obj}_s| < |\mathsf{obj}_{s'}|$, increasing $\lceil \mathsf{obj}_{s'} \rceil$ to $\lceil \mathsf{obj}_s \rceil$ will not violate our padding constraints (in particular, (2)). Similarly, decreasing $\lceil \mathsf{obj}_s \rceil$ to $\lceil \mathsf{obj}_{s'} \rceil$ will not violate our constraints (in particular, (1)). Below we show that one of these alternatives will decrease $\mathbb{H}(\mathsf{Y})$ and, therefore, $\mathbb{I}(\mathsf{S}; \mathsf{Y})$, showing that $\lceil \cdot \rceil$ cannot be privacy-optimal. To do so, we use the following proposition.

Proposition 1: Let f be a function that is defined on the interval $R \subseteq \mathbb{R}$, and that has a negative second derivative. For

all $z, z' \in R$ such that $z \leq z'$ and for any $\epsilon > 0$ such that $z - \epsilon \in R$ and $z' + \epsilon \in R$:

$$f(z) + f(z') > f(z - \epsilon) + f(z' + \epsilon)$$

Proof: Since a negative second derivative implies a decreasing first derivative,

$$\frac{f(z) - f(z - \epsilon)}{z - (z - \epsilon)} > \frac{f(z' + \epsilon) - f(z')}{(z' + \epsilon) - z'}$$

and the result follows by rearranging terms. ∎

To use Prop. 1, consider a partition $\{B_y\}_{y \in \mathbb{N}}$ of the objects based on their padded sizes, i.e., so that $B_y = \{s \in S : \lceil obj_s \rceil = y\}$. If we let $p_y = \sum_{s \in B_y} \mathbb{P}(S = s)$ and $h(p) = -p \log_2 p$ for $p \in [0, 1]$, then

$$\mathbb{H}(Y) = \sum_{y \in \mathbb{N}} h(p_y) \qquad (6)$$

Note that the second derivative of $h(p)$ is $h''(p) = -\frac{1}{p \ln(2)}$, which is negative for $p \in [0, 1]$.

Now, return to considering a function $\lceil \cdot \rceil$ that is *not* nondecreasing, i.e., there are objects obj_s and $obj_{s'}$ with $|obj_s| < |obj_{s'}|$ but $y > y'$ where $y = \lceil obj_s \rceil$ and $y' = \lceil obj_{s'} \rceil$. If $p_y \leq p_{y'}$, then setting $\epsilon = \mathbb{P}(S = s)$ and applying Prop. 1 yields $h(p_y) + h(p_{y'}) > h(p_y - \epsilon) + h(p_{y'} + \epsilon)$. In other words, decreasing $\lceil obj_s \rceil$ from y to y' reduces $\mathbb{H}(Y)$. Otherwise (i.e., $p_y > p_{y'}$), setting $\epsilon = \mathbb{P}(S = s')$ and applying Prop. 1 shows that increasing $\lceil obj_{s'} \rceil$ from y' to y reduces $\mathbb{H}(Y)$. Either case reveals that the initial function $\lceil \cdot \rceil$, which is not nondecreasing, cannot be privacy-optimal, either, and so any privacy-optimal function $\lceil \cdot \rceil$ must be nondecreasing.

2) Finding the Privacy-Optimal $\lceil \cdot \rceil$: That the privacy-optimal $\lceil \cdot \rceil$ must be nondecreasing allows us to leverage dynamic programming to compute the solution. In general, to leverage dynamic programming, it is necessary to express our optimization problem recursively, i.e., so that the optimal solution is computed by combining optimal solutions to overlapping subproblems (e.g., [27, Chapter 6]).

Let $[i, j] = \{i, \ldots, j\}$, $[i] = [1, i]$, and $\#S$ be the cardinality of set S. Because the privacy-optimal $\lceil \cdot \rceil$ is nondecreasing as a function of object size, there is a total order $\psi : [\#S] \to S$ of S by object size—i.e., a bijection satisfying $|obj_{\psi(k)}| \leq |obj_{\psi(k+1)}|$ for $k \in [\#S - 1]$—such that each block B_y of the partition induced by $\lceil \cdot \rceil$ is of the form $B_y = \{\psi(k)\}_{k \in [i,j]}$ where $|obj_{\psi(j)}| \leq y \leq c \times |obj_{\psi(i)}|$. We equivalently denote this block by $B_{[i,j]}$, and analogously denote $p_{[i,j]} = \sum_{s \in B_{[i,j]}} \mathbb{P}(S = s)$.

Now consider the function H defined recursively as $H(0) = 0$ and, for $0 < j \leq \#S$,

$$H(j) = \min_{i \leq j : |obj_{\psi(j)}| \leq c \times |obj_{\psi(i)}|} \left(H(i-1) + h(p_{[i,j]}) \right) \qquad (7)$$

Then, computing $H(\#S)$ computes the privacy-optimal padding scheme $\lceil \cdot \rceil$ for the per-object padding case: for the value of i minimizing the right hand side of (7), we set $\lceil obj_{\psi(k)} \rceil = |obj_{\psi(j)}|$ for all $k \in [i, j]$. (Note that no other objects will be padded to this size, as otherwise (7) could be minimized further, as shown in Sec. III-A1.) The recursion

s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(a) $c = 2$

s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(b) $c = 3$

Fig. 1: $\mathbb{P}(Y = y \mid S = s)$ produced by POP (Sec. III-A).

(7) exhibits the properties needed for dynamic programming to be effective because $H(i)$ can be leveraged in the calculation of $H(j)$ for every $j > i$. That is, $H(i)$ can be computed only once and saved to be looked up when needed. As a consequence, dynamic programming (e.g., [27, Chapter 6]) yields an algorithm that runs in time $O((\#S)^2)$. In the following sections, we refer to this algorithm as Per-Object Padding (POP).

When applied to the objects listed in Table I, POP produces the conditional distributions shown in Fig. 1. Fig. 1a shows the case $c = 2$, and Fig. 1b shows the case $c = 3$. (We believe these values for c are larger than would typically be tolerated in practice, but we use large values here to illustrate the effects of the algorithm in light of the small number, but diverse sizes, of the objects in Table I.) Because POP produces a deterministic padding scheme $\lceil \cdot \rceil$, each row includes only one nonzero entry. It is perhaps most insightful to consider how these distributions change as c moves from $c = 2$ to $c = 3$. For example, POP prescribes that P0 be padded to the size of P1 when $c = 2$, leaving P2 isolated. In contrast, when $c = 3$, POP instead prescribes padding P1 to the size of P2, leaving P0 isolated. It is simple to confirm that under the padding constraints (1)–(2), at least one of P0, P1, and P2 must be left isolated by a deterministic padding scheme when $c \leq 3$, but when allowed, POP prefers to isolate P0 because it is requested less often (see Table I).

B. Per-request Padding

In the per-request padding scenario, the padding scheme $\lceil \cdot \rceil$ can be calculated as a special case of *rate-distortion minimization* proposed by Shannon [72] (see also [8, Sec. IV]). Specifically, using our terminology, rate-distortion minimization solves for a scheme $\lceil \cdot \rceil$ minimizing $\mathbb{I}(S; Y)$ subject to the constraint

$$\mathbb{E}(D(S, \lceil obj_S \rceil)) \leq d$$

where $D : S \times \mathbb{N} \to [0, \infty]$ is a *distortion function*, $d \in \mathbb{R}$ is a positive constant, and the expectation is computed relative to the distribution of S and random choices made by $\lceil \cdot \rceil$. Written explicitly, this expected value is

$$\mathbb{E}(D(S, \lceil \text{obj}_S \rceil)) = \sum_{s \in S} \sum_{y \in \mathbb{N}} \mathbb{P}(S = s)\mathbb{P}(\lceil \text{obj}_s \rceil = y)D(s, y)$$

Specifying $D(s, y) = \infty$ for any s such that $\mathbb{P}(S = s) > 0$ implies that a solution for rate-distortion minimization, if it exists, must set $\mathbb{P}(\lceil \text{obj}_s \rceil = y) = 0$. As such, specifying $D(s, y) = \infty$ for any $y < |\text{obj}_s|$ and any $y > c \times |\text{obj}_s|$ suffices to enforce (1)–(2). Specifying $D(s, y) = 0$ for $|\text{obj}_s| \leq y \leq c \times |\text{obj}_s|$ then provides maximum flexibility in choosing $\lceil \cdot \rceil$ to minimize $\mathbb{I}(S; Y)$.

Having reduced our problem to rate-distortion minimization, we can employ a known algorithm for that problem to solve ours. Typically this problem must be solved numerically because the optimization problem is nonlinear. An iterative algorithm that converges to the optimal solution is due to Blahut [10] and independently Arimoto [5] (see also [85]). For our purposes, this algorithm works by iteratively computing values $\{\langle u_t(y), v_t(y, s) \rangle\}_{t \geq 0}$ as follows for each $s \in S$ and each y, $|\text{obj}_s| \leq y \leq c \times |\text{obj}_s|$, where β is a parameter:

$$
\begin{aligned}
v_{t+1}(y, s) &= \frac{u_t(y) \exp(-\beta \times D(s, y))}{\sum_{y'} u_t(y') \exp(-\beta \times D(s, y'))} \\
&= \begin{cases} \frac{u_t(y)}{\sum_{y':D(s,y')=0} u_t(y')} & \text{if } D(s, y) = 0 \\ 0 & \text{otherwise} \end{cases} \\
u_{t+1}(y) &= \sum_{s \in S} \mathbb{P}(S = s)v_{t+1}(y, s)
\end{aligned}
\tag{8}
$$

(8) holds since in our case, $D(s, y) \in \{0, \infty\}$ for all s, y.

As t grows, the values $u_t(y)$ and $v_t(y, s)$ converge to $\mathbb{P}(Y = y)$ and $\mathbb{P}(Y = y \mid S = s)$, respectively, for the optimal $\lceil \cdot \rceil$ for per-request padding, provided that $v_0(y, s)$ is initialized to be nonzero for any y for which $\mathbb{P}(Y = y \mid S = s)$ in the privacy-optimal $\lceil \cdot \rceil$ is nonzero. So, in all empirical tests reported in this paper, we initialized $v_0(y, s)$ by computing every possible padded size for obj_s, i.e.,

$$Y(s) = \left\{ y' \;\middle|\; \begin{array}{l} |\text{obj}_s| \leq y' \leq c \times |\text{obj}_s| \;\wedge \\ \exists s' \in S : (\mathbb{P}(S = s') > 0 \wedge |\text{obj}_{s'}| = y') \end{array} \right\}$$

and set

$$
\begin{aligned}
v_0(y, s) &= \begin{cases} \frac{1}{\#Y(s)} & \text{if } y \in Y(s) \\ 0 & \text{otherwise} \end{cases} \\
u_0(y) &= \sum_{s \in S} \mathbb{P}(S = s)v_0(y, s)
\end{aligned}
$$

We terminated the algorithm once $\tilde{\mathbb{I}}_t(S; Y) - \tilde{\mathbb{I}}_{t+1}(S; Y) < \Delta$, where $\tilde{\mathbb{I}}_t(S; Y)$ is the value of $\mathbb{I}(S; Y)$ obtained using $\mathbb{P}(Y = y) = u_t(y)$ and $\mathbb{P}(Y = y \mid S = s) = v_t(y, s)$ in (5), and where Δ is a threshold to indicate when the algorithm can cease iterating (as the per-iteration improvement to $\mathbb{I}(S; Y)$ has become sufficiently small). In the sections that follow, we refer to this algorithm as <u>Per-R</u>equest <u>P</u>adding (PRP).

PRP produces the conditional distributions shown in Fig. 2 when applied to the objects in Table I. The most immediately noticeable difference from Fig. 1 is that Fig. 2 includes values

s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.19	0.00	0.00	0.00	0.81	0.00
P4	0.00	0.00	0.00	0.00	0.19	0.00	0.00	0.00	0.81	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.14
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.14
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(a) $c = 2$

s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	0.39	0.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	0.72	0.00	0.00	0.00	0.00	0.28	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19	0.00	0.81
P4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19	0.00	0.81
P5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19	0.00	0.81
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19	0.00	0.81
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19	0.00	0.81
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(b) $c = 3$

Fig. 2: $\mathbb{P}(Y = y \mid S = s)$ produced by PRP (Sec. III-B).

< 1.0, which permits $\lceil \cdot \rceil$ to sample each response size from the distribution in the row of the requested object. (Note that each row sums to 1.0, i.e. the calculation of $v_t(y, s)$ followed by $u_t(y)$, as per (8), yields a valid probability distribution.) Whereas POP was forced to isolate either P0 or P2 when $c \leq 3$, PRP spreads probability mass to ensure that no object is isolated; i.e., for any feasible response size y, there are at least two objects that response could possibly represent. Moreover, PRP spreads probability mass in the most effective way possible to minimize $\mathbb{I}(S; Y)$.

C. Unknown Query Distribution

There are cases in which it is inconvenient or even impossible for the object store to estimate the distribution for S. This might occur because the adversary can affect that distribution or simply because the distribution is likely to vary in unpredictable ways over time. Issa et al. [42] advocate that the Sibson mutual information of order infinity, denoted $\mathbb{I}_\infty(S; Y)$, be used in place of $\mathbb{I}(S; Y)$ to describe information leakage when the distribution of the private random variable S is complex or outside the defender's control, where

$$\mathbb{I}_\infty(S; Y) = \log_2 \sum_y \max_{s:\mathbb{P}(S=s)>0} \mathbb{P}(Y = y \mid S = s)$$

This alternative is attractive because $\mathbb{I}_\infty(S; Y) \geq \mathbb{I}(S; Y)$ [73], [78] and, moreover, $\mathbb{I}_\infty(S; Y) = \mathbb{I}(S; Y)$ for distributions of S and Y meeting certain conditions [42, Lemma 2]. As such, $\mathbb{I}_\infty(S; Y)$ is a conservative estimate of information leakage in the absence of knowledge of the distribution for S (except those s for which $\mathbb{P}(S = s) > 0$) and, Issa et al. argue, has other advantages as a measure of information leakage, as well. Indeed, as mentioned in Sec. II, $\mathbb{I}_\infty(S; Y)$ is equivalent to min-capacity, which Alvim et al. [3, Theorem 5.1] have shown upper-bounds g-leakage for all gain functions and all distributions S. Thus, minimizing $\mathbb{I}_\infty(S; Y)$ provides robust privacy guarantees to the object store.

In our scenario, constructing $\lceil \cdot \rceil$ that minimizes $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ subject to overhead constraints (1)–(2) is very efficient. First, note that for any s' for which $\mathbb{P}(\mathsf{S}=s')>0$, constraints (1)–(2) imply

$$\sum_{y:|\mathsf{obj}_{s'}|\leq y\leq c\times|\mathsf{obj}_{s'}|}\mathbb{P}\big(\mathsf{Y}=y\mid\mathsf{S}=s'\big)=1$$

and so

$$\sum_{y:|\mathsf{obj}_{s'}|\leq y\leq c\times|\mathsf{obj}_{s'}|}\max_{s:\mathbb{P}(\mathsf{S}=s)>0}\mathbb{P}\big(\mathsf{Y}=y\mid\mathsf{S}=s\big)\geq 1$$

As such, for any $S^*\subseteq S$ such that

$$\forall s,s'\in S^*,s\neq s':$$
$$[|\mathsf{obj}_s|,c\times|\mathsf{obj}_s|]\cap[|\mathsf{obj}_{s'}|,c\times|\mathsf{obj}_{s'}|]=\emptyset \qquad (9)$$

we know that $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})\geq\log_2\#S^*$.

Now, for any $S^*\subseteq S$ define the *anchors* $A(S^*)$ inductively as follows: $A(\emptyset)=\emptyset$ and, for $S^*\neq\emptyset$,

$$A(S^*)=\{a(S^*)\}\cup A(S^*\setminus B(S^*)) \qquad \text{where} \qquad (10)$$
$$a(S^*)=\arg\max_{s\in S^*}|\mathsf{obj}_s| \qquad \text{and} \qquad (11)$$
$$B(S^*)=\{s'\in S^*:|\mathsf{obj}_{a(S^*)}|\leq c\times|\mathsf{obj}_{s'}|\} \qquad (12)$$

In words, the anchors of S^* include the largest object of S^* (see (11)) and the anchors of the set remaining after removing those objects for which this anchor is no more than c times larger (see (10), (12)).

Consider calculating the anchors $A(S)$ of the full set S, and for each $B(S^*)$ calculated in the induction, pad each object in $B(S^*)$ to the size of anchor $a(S^*)$; i.e.,

$$\lceil\mathsf{obj}_s\rceil=|\mathsf{obj}_{a(S^*)}|$$

for each $s\in B(S^*)$. This padding scheme respects constraints (1)–(2) and yields $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})=\log_2\#A(S)$, since there are only $\#A(S)$ values y to which objects are padded and

$$\max_{s:\mathbb{P}(\mathsf{S}=s)>0}\mathbb{P}\big(\mathsf{Y}=y\mid\mathsf{S}=s\big)=1$$

for each such y. Moreover, since (9) is satisfied with $S^*=A(S)$, we know that this value of $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ is the minimum. This algorithm, which produces $\lceil\cdot\rceil$ in time linear in $\#S$, is denoted <u>P</u>adding <u>w</u>ith<u>o</u>ut a <u>D</u>istribution (PwoD) in the following sections.

Applying PwoD to the objects in Table I yields the conditional distributions shown in Fig. 3. As in the case of POP, PwoD is forced to isolate either P0 or P2 when $c\leq 3$ (since both produce deterministic padding schemes), and PwoD does so in the same way as POP. The distributions resulting when $c=3$, shown in Fig. 3b, are also identical to those produced by POP (Fig. 1b). When $c=2$, however, PwoD prescribes that P3–P9 be padded differently than POP prescribes, as can be seen by comparing Fig. 3a to Fig. 1a. This is due to the fact that PwoD iterates through an object list in reverse-order by size and greedily assigns objects to the current anchor. Thus, in Fig. 3a PwoD sets P9 as the first anchor and then pads P8 and P7 to P9's size before arriving at P6 and creating the next anchor.

s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P4	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(a) $c=2$

s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(b) $c=3$

Fig. 3: $\mathbb{P}\big(\mathsf{Y}=y\mid\mathsf{S}=s\big)$ produced by PwoD (Sec. III-C).

IV. SECURITY EVALUATION

In this section, we begin by describing the datasets that we created to support all of our security and performance tests. Next we list the padding algorithms to which we compare, and we describe how they work when used in our setting. We then evaluate each algorithm's ability to reduce $\mathbb{I}(\mathsf{S};\mathsf{Y})$ for the two datasets. We conclude this section with a security assessment that evaluates each algorithm's ability to hinder a network attacker in an operational setting.

A. Datasets

For our tests we use two datasets: a dataset of NodeJS[2] packages, and a dataset of Unsplash[3] images.

NodeJS Packages: To create our dataset of NodeJS packages, we first referenced a list[4] of all packages available on the NodeJS Package Manager (NPM) registry. We then used NPM's Application Programming Interface (API) to retrieve the weekly download statistics of each package, for the week of Feb. 13-19, 2021. Finally, we sent HTTP HEAD requests for each package to obtain the tarball size (in bytes) of its most current version (as of Feb. 19, 2021). Overall, this dataset contains the name, tarball size, and weekly download statistics for 423,450 packages.[5]

Unsplash Lite: To create our Unsplash Lite dataset, we first referenced Unsplash's freely available "Unsplash Lite Dataset 1.1.0" dataset[6]. This dataset includes the URL of

---

[2]https://nodejs.org/en/

[3]https://unsplash.com/

[4]https://github.com/nice-registry/all-the-package-repos, accessed Feb. 19, 2021.

[5]Due to limitations of NPM's API, we did not retrieve statistics for scoped packages (i.e., packages whose name begins with an @ symbol). Also, we did not include packages with 0 weekly downloads. Overall, our starting list contained 993,825 packages, and our final dataset is comprised of 423,450 packages.

[6]https://unsplash.com/data

each image in the dataset, as well each image's cumulative downloads since the image was uploaded to Unsplash. Given this information, we were able to issue HTTP HEAD requests for each image in the dataset, as well as compute each image's average downloads per day (based on the difference between the dataset's creation date and the image's upload date). In total, our Unsplash Lite dataset contains the URL, file size (in bytes), and average daily downloads for 24,997 images.

B. Padding Algorithms to Which We Compare

In this section we briefly introduce the other padding algorithms to which we compare. These algorithms appeared within the past five years in a well-regarded, peer-reviewed venue focusing on privacy technologies, and so we take them as representative of modern approaches for padding objects to make their retrieval harder to detect by a network observer. The padding schemes produced by each of these algorithms for the dataset in Table I are shown in App. A.

D-ALPACA: Cherubin et al. [20] proposed padding algorithms to defend against website fingerprinting attacks and, so, that seek to address leakage resulting from the retrieval of objects hyperlinked in a webpage, subject to padding overhead constraints. Distilled down to our simpler scenario, though, their padding algorithms result in natural contenders for comparison. One of these, called D-ALPACA, is deterministic and so is suitable as a per-object padding algorithm. In brief, D-ALPACA sets $\lceil \mathsf{obj}_s \rceil$ to be the smallest multiple of σ that is $\geq |\mathsf{obj}_s|$, where σ is an input parameter. For our setting, we set $\sigma = \mathsf{floor}\big((c-1) \times |\mathsf{obj}_{s_{\min}}|\big)$, where $\mathsf{floor} : \mathbb{R} \to \mathbb{N}$ is the floor function and where $s_{\min}$ is the identifier of the smallest object in the object store. This, then, ensures that D-ALPACA does not violate c for any $s \in S$. Note that D-ALPACA is insensitive to the distribution of S.

PADMÉ: Nikitin et al. [63] proposed a per-object padding algorithm called PADMÉ. Like ours, PADMÉ seeks to limit leakage about which object obj_s is returned from the size $\lceil \mathsf{obj}_s \rceil$ of the returned object, while also limiting padding overhead. The notion of leakage that PADMÉ is designed to limit, however, is simply the total number of valid padding sizes over all objects; in particular, PADMÉ calculates per-object padding sizes independent of the distribution of S. Specifically, PADMÉ pads an object obj_s to size $\lceil \mathsf{obj}_s \rceil = b - (b \bmod 2^{E-S})$ where $E = \mathsf{floor}(\log_2 |\mathsf{obj}_s|)$, $S = \mathsf{floor}(\log_2 E) + 1$, and $b = |\mathsf{obj}_s| + 2^{E-S} - 1$. Nikitin et al. show that this scheme limits "leakage" to $O(\log \log M)$ bits if the largest object is of size M, with a padding overhead of $c \approx \frac{1}{2 \log_2 |\mathsf{obj}_s|} < 1.12$.

P-ALPACA: Cherubin et al. [20] also proposed a randomized algorithm called P-ALPACA that is suitable as a per-request padding algorithm. When applied to our setting, P-ALPACA pads the response to a request s so that

$$\mathbb{P}(\lceil \mathsf{obj}_s \rceil = y) = \mathbb{P}\big(|\mathsf{obj}_\mathsf{S}| = y \mid |\mathsf{obj}_s| \leq |\mathsf{obj}_\mathsf{S}| \leq c \times |\mathsf{obj}_s|\big)$$
$$= \frac{\sum_{s':|\mathsf{obj}_{s'}|=y} \mathbb{P}(\mathsf{S}=s')}{\sum_{s':|\mathsf{obj}_s| \leq |\mathsf{obj}_{s'}| \leq c \times |\mathsf{obj}_s|} \mathbb{P}(\mathsf{S}=s')}$$

for each y, $|\mathsf{obj}_s| \leq y \leq c \times |\mathsf{obj}_s|$. In particular, the most probable padded size for obj_s is the most probable unpadded size in the interval $[|\mathsf{obj}_s|, c \times |\mathsf{obj}_s|]$.

C. Mutual Information

To more systematically compare the security offered by the six padding algorithms we consider, we applied them to the two datasets described in Sec. IV-A and computed the mutual information $\mathbb{I}(\mathsf{S};\mathsf{Y})$ of the distribution resulting from each. Fig. 4a shows the results for the NodeJS dataset, and Fig. 4b shows the results for the Unsplash dataset. Each bar indicates the value for $\mathbb{I}(\mathsf{S};\mathsf{Y})$ that was achieved by each algorithm, when calculated using the distribution of the given dataset and the value of c on the horizontal axis. Each error bar extends to $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ for each algorithm's padding scheme, which upper-bounds the worst-case $\mathbb{I}(\mathsf{S};\mathsf{Y})$ if the distribution for S the defender assumed was incorrect. The algorithms are ordered left-to-right roughly according to security, i.e., from lower security to higher (since lower numbers indicate better security). The one exception is PADMÉ, for which a bar is added to the left of the cluster at the value of c that its padding scheme achieved on the corresponding dataset ($c = 1.09$ in Fig. 4a and $c = 1.03$ in Fig. 4b). Note that the y-axis in both Fig. 4a and Fig. 4b begins at 5 bits.

As these figures show, D-ALPACA was not competitive with the other algorithms in our tests. Indeed, it performed so poorly that portions of its bars were clipped in both Fig. 4a and Fig. 4b so that they would not distort the graph so as to obscure the differences among the bars for other algorithms. PADMÉ performed convincingly better than D-ALPACA (for the c values it yielded on the two datasets) but was nevertheless inferior to other alternatives in terms of mutual information of the distribution it produced.

Of the remaining algorithms, PRP consistently produced the lowest $\mathbb{I}(\mathsf{S};\mathsf{Y})$, which is not surprising since it is designed specifically to minimize $\mathbb{I}(\mathsf{S};\mathsf{Y})$. (So is POP, but POP is constrained to produce a padding scheme where only one padded size is possible for each object, since $\lceil \cdot \rceil$ is constrained to be deterministic.) However, PRP also produced the highest $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ (of these four algorithms), which suggests that by fine-tuning the padding scheme to the assumed distribution for S, there is a risk of making security more sensitive to errors in that assumption. On the other end of the spectrum, PwoD achieves the best $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ in all cases; again, this is unsurprising since it was designed to optimize this measure. And, while $\mathbb{I}(\mathsf{S};\mathsf{Y})$ of the distribution it produces is generally the worst of these four algorithms, it does not differ by much. For this reason, and since PwoD does not rely on knowing the distribution for S, it appears to be an attractive choice for these datasets.

Another observation from these graphs is that POP outperformed P-ALPACA in most of the tests, producing a lower $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ in all but one case ($c = 1.1$ in Fig. 4b) and a $\mathbb{I}(\mathsf{S};\mathsf{Y})$ value that is no higher than (and often lower than) P-ALPACA's. This dominance is perhaps unexpected, since POP is restricted to produce a deterministic padding scheme $\lceil \cdot \rceil$ whereas that produced by P-ALPACA need not be.

Finally, although P-ALPACA remained competitive for these two datasets, we note that it provides no guarantees regarding optimality when used on other datasets. To demonstrate this, consider an object store that serves two objects, obj_s and $\mathsf{obj}_{s'}$, with $|\mathsf{obj}_s| = 100\mathrm{B}$ and $|\mathsf{obj}_{s'}| = 101\mathrm{B}$. If these objects are requested uniformly at random (i.e., each

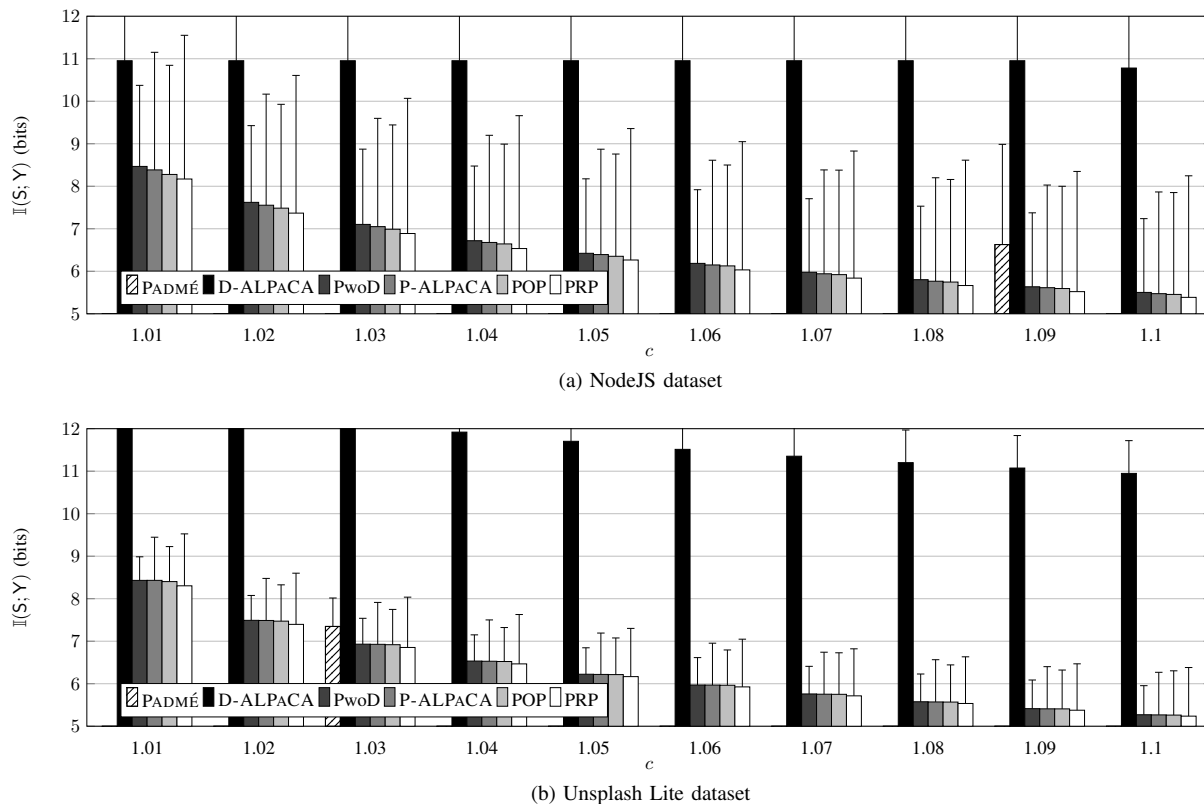(a) NodeJS dataset

(b) Unsplash Lite dataset

Fig. 4: Per-algorithm mutual information. Error bars extend to $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$. Lower values indicate better security.

50% of the time), then $\mathbb{H}(\mathsf{S}) = 1$ bit and, if left unpadded, $\mathbb{I}(\mathsf{S};\mathsf{Y}) = 1$ bit. Now, for $c = 1.01$, all of our algorithms—PRP, POP, and PwoD—will set $\lceil \mathsf{obj}_s \rceil = |\mathsf{obj}_{s'}|$ for every request of obj_s, resulting in $\mathbb{I}(\mathsf{S};\mathsf{Y}) = 0$ bits. P-ALPaCA, however, will only set $\lceil \mathsf{obj}_s \rceil = |\mathsf{obj}_{s'}|$ for half of obj_s's requests, resulting in $\mathbb{I}(\mathsf{S};\mathsf{Y}) \approx 0.31$ bits. Thus, here we see that P-ALPaCA performs worse than our algorithms and indeed is far from optimal (in a relative sense).

D. Attacker's Recall and Precision

One of the criticisms sometimes levied against mutual information as a measure of privacy is that it is difficult to interpret operationally [42]. In this section, therefore, we evaluate the extent to which the various algorithms we consider interfere with a network attacker attempting to achieve a specific operational goal, namely to infer whether a request s is a member of a target set $S^* \subseteq S$, based on the size of the object returned in response to the request. Specifically, for the observed size y, the adversary returns *true* iff $\mathbb{P}\big(\mathsf{S} \in S^* \mid \lceil \mathsf{obj}_\mathsf{S} \rceil = y\big) \geq \tau$ for a tunable parameter $0 \leq \tau \leq 1$. For a given setting of τ, the adversary's *recall* is the fraction of requests for some member of S^* for which the adversary returns *true*, and the adversary's *precision* is the fraction of its *true* declarations for which the requested object is in S^*. We consider two different sets S^*, described below.

Vulnerable NodeJS Packages: For the NodeJS packages, we envision an adversary that wants to detect if *any* vulnerable package was retrieved/installed by a server. If

so, the adversary might then hope to exploit the server via a program such as Metasploit[7]. To define a subset S^* of "vulnerable" packages, we use a dataset provided by Ferenc et al. [31] containing 88 NodeJS packages that, at the time of its publication, contained vulnerable functions and whose source code was available on GitHub. Note that these packages may no longer be vulnerable. Still, we use this dataset as a way to define a subset S^* of "vulnerable" packages based on their vulnerability at that time.

Unsplash Lite Nature Collection: The Unsplash Lite dataset includes a table that contains user-created collections of photos contained within the dataset. We used a collection of 256 photos named *Nature* to create the adversary's subset S^* of interest. We envision that such an adversary might want to send the user targeted ads, in which case knowledge of a downloaded picture might help the adversary to tailor the selection of ads shown to the user.

We allow the adversary to know the object sizes and retrieval distribution for S, as well as the padding algorithm employed by the object store. For confidence thresholds $\tau \in \{0.0, 0.05, 0.10, \ldots, 1.0\}$, we calculate this adversary's recall and precision for each algorithm using $c \in \{1.01, 1.03, 1.05, 1.07, 1.09\}$. The recall-precision curves that result are depicted in Fig. 5 and Fig. 6.

Intuitively, a curve closer to the upper right-hand corner of each plot indicates that the adversary did better at detecting

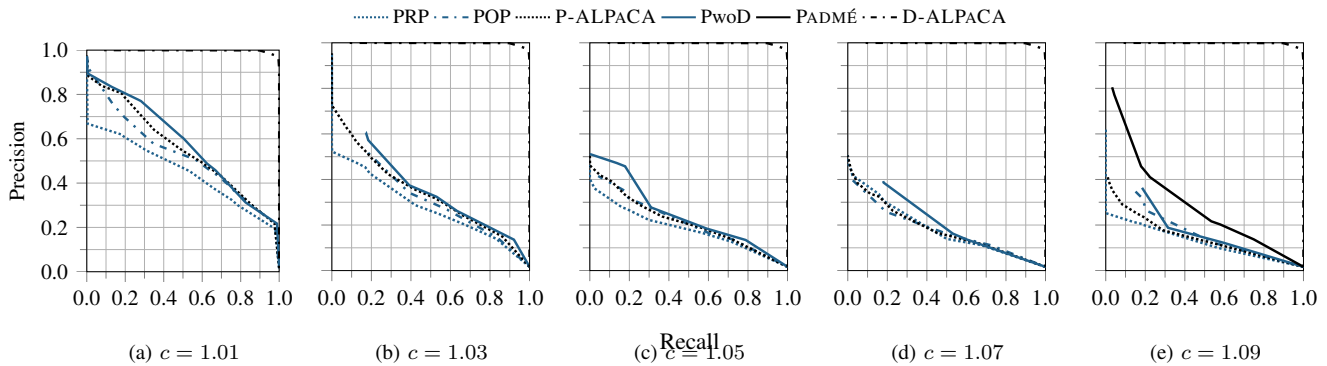---

[7]https://www.metasploit.com/

513

Precision

(a) $c = 1.01$ (b) $c = 1.03$ (c) $c = 1.05$ **Recall** (d) $c = 1.07$ (e) $c = 1.09$

Fig. 5: Adversary's recall and precision for detecting vulnerable NodeJS packages.

Precision

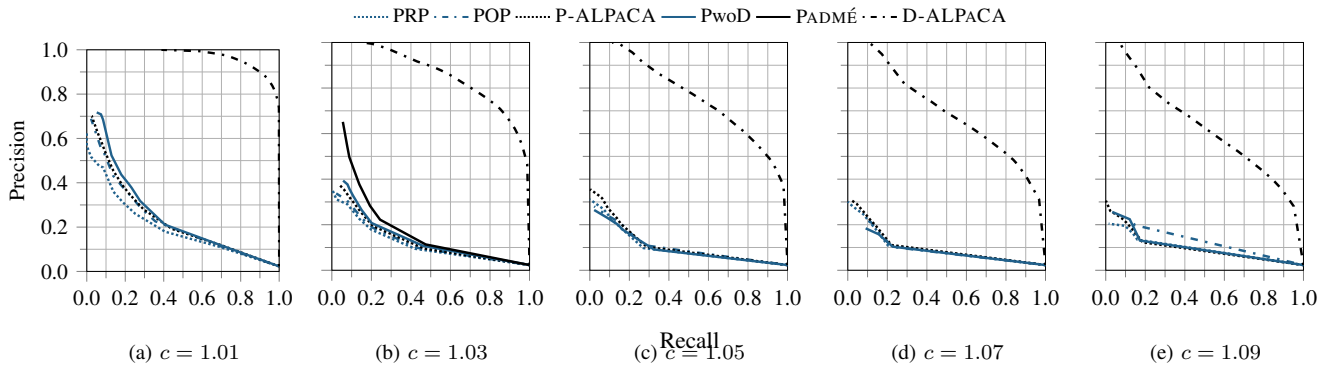(a) $c = 1.01$ (b) $c = 1.03$ (c) $c = 1.05$ **Recall** (d) $c = 1.07$ (e) $c = 1.09$

Fig. 6: Adversary's recall and precision for detecting the Unsplash Lite *Nature* collection.

requests in S^*. In that light, we see that better security provided by an algorithm when measured using mutual information (i.e., Fig. 4) translated reasonably well into diminished precision and/or recall in these tests. Indeed, our per-request padding algorithm (PRP) defended as well or better than the competitor for per-request padding (P-ALPACA), and our per-object padding algorithm (POP) consistently outperformed the per-object padding competitors (D-ALPACA and PADMÉ). Only PwoD was outperformed by P-ALPACA in some cases, but P-ALPACA did so by leveraging the distribution of S; if that distribution were unknown or incorrect, presumably the protection offered by P-ALPACA would decay.

We also computed the precision-recall curve per vulnerable NodeJS package and per *Nature* photo in the Unsplash Lite dataset, to assess the extent to which individual objects could still be identified by the attacker based on their sizes when returned. To visualize these results, we reduced the precision-recall curve for each $s \in S^*$ to a single number—the area under the curve[8] (AUC)—and then plotted the distribution of AUC values per algorithm.

Figs. 7–8 show distributions for the vulnerable NodeJS packages and the *Nature* photos, respectively. The more bottom-heavy the distribution, the more members of S^* the algorithm protects. As such, it is easy to see from these figures that our algorithms performed better than P-ALPACA, particularly at smaller values of c, i.e., $c = 1.01$ (Fig. 7a)

and $c = 1.03$ (Fig. 7b) for the NodeJS dataset and $c = 1.01$ (Fig. 8a) for the Unsplash dataset. D-ALPACA and, to a lesser extent, PADMÉ were not competitive with the other algorithms.

Overall, we attribute the success of our algorithms to (4). Specifically, minimizing $\mathbb{I}(\mathsf{S};\mathsf{Y})$ has the effect of increasing the lower-bound of $\mathbb{G}(\mathsf{S} \mid \mathsf{Y})$, i.e., by minimizing $\mathbb{I}(\mathsf{S};\mathsf{Y})$ one would expect that the adversary who observes padded object sizes would require more guesses, on average, to correctly determine object identities. Since a precision-recall assessment models an adversary that only makes a single guess, it stands to reason that our techniques would excel in these tests.

V. PERFORMANCE EVALUATION

In this section, we report on the runtime performance of PRP, POP, PwoD, and P-ALPACA in computing their padding schemes. We did not evaluate the runtime performance of either D-ALPACA or PADMÉ, as they do not need to compute a padding scheme prior to usage. We conclude this section with an analysis of the overall bandwidth increase imposed by each padding scheme.

A. Implementation

Code Overview: We implemented all four algorithms in Python. Additionally, we used Cython to optimize POP, PRP, and P-ALPACA. By using Cython, we were able to implement each algorithm's core routines in "pure C" using only C arrays and C data types. Furthermore, we leveraged the multi-processing API OpenMP to parallelize both PRP and

---

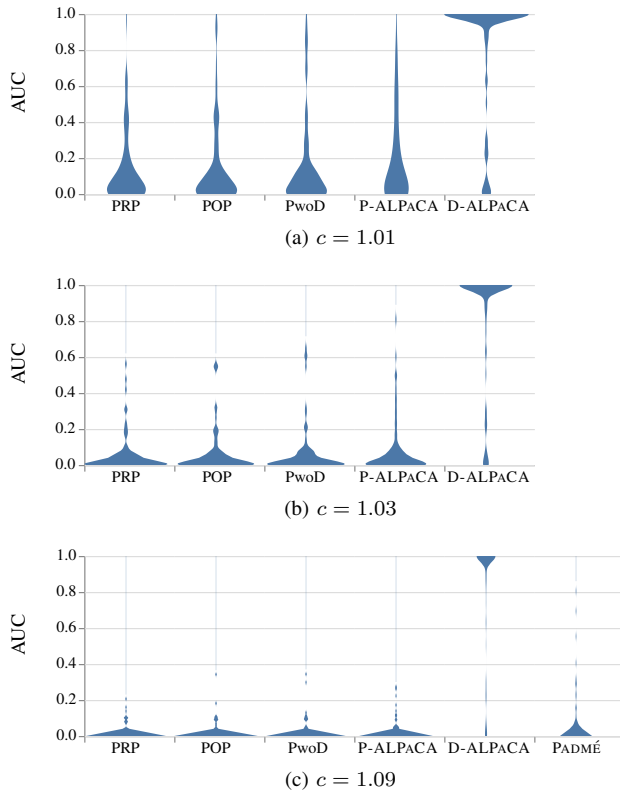[8]We extended each curve to the left to meet the vertical axis at its maximum precision.

(a) $c = 1.01$

(b) $c = 1.03$

(c) $c = 1.09$

Fig. 7: Distributions of precision-recall AUC for vulnerable NodeJS packages.
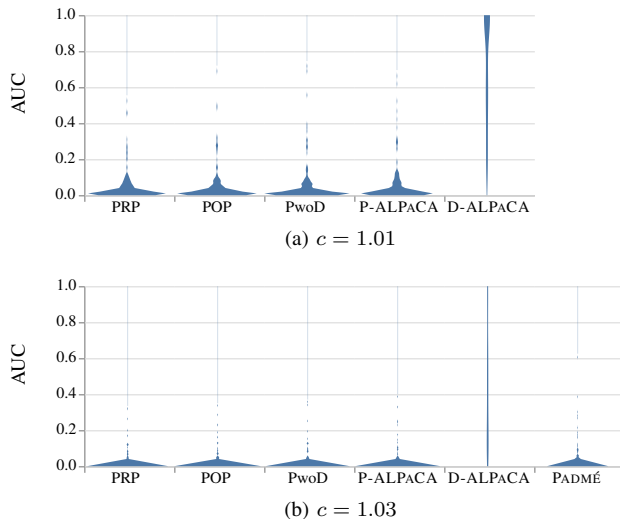

(a) $c = 1.01$

(b) $c = 1.03$

Fig. 8: Distributions of precision-recall AUC for Unsplash Lite *Nature* photos.

P-ALPACA so that they used all of the available processors on our evaluation platform (see Sec. V-B).

PRP Improvements: Our implementation of PRP improved upon the underlying algorithm in the following two ways:

- $\tilde{\mathbb{I}}_t(\mathsf{S};\mathsf{Y})$ **calculations.** In our tests, calculating $\tilde{\mathbb{I}}_t(\mathsf{S};\mathsf{Y})$

for every $t \geq 0$ (and specifically the needed logarithms) was a performance bottleneck, and so our implementation of PRP calculated only $\tilde{\mathbb{I}}_{10t}(\mathsf{S};\mathsf{Y})$, instead. As such, the algorithm terminated after the first iteration $10(t+1)$ for which $\tilde{\mathbb{I}}_{10t}(\mathsf{S};\mathsf{Y}) - \tilde{\mathbb{I}}_{10(t+1)}(\mathsf{S};\mathsf{Y}) < 10\Delta$.

- **Incremental update.** Despite this optimization, PRP was still considerably slower than the alternatives (see Sec. V-D), so much so that recalculating the distributions for use by $\lceil \cdot \rceil$, e.g., after changes to (the sizes of) objects, could be a considerable disruption to serving those changed objects. We anticipated, however, that incrementally updating these distributions after only a few object changes would be much faster, suggesting that their continuous maintenance could be adequately responsive, even if computing them from scratch would not be.

  To this end, we implemented two versions of PRP. The first, PRP$_{\text{init}}$, implemented PRP as described in Sec. III-B and was used to calculate the initial padding scheme for a given dataset and a given c. The second, PRP$_{\text{inc}}$, allowed for faster updates to a padding scheme as object sizes were changed. PRP$_{\text{inc}}$ did this by retaining the final $u_t(y)$ and $v_t(y, s)$ values from PRP$_{\text{init}}$ (or a previous invocation of PRP$_{\text{inc}}$) for values s and y unaffected by the newly changed objects, using these values to initialize $u_0(y)$ and $v_0(y, s)$. For other values of y and s, $u_0(y)$ and $v_0(y, s)$ were initialized as in Sec. III-B. After this initialization step, PRP$_{\text{inc}}$ iterated in the same way as PRP$_{\text{init}}$, stopping once $\tilde{\mathbb{I}}_{10t}(\mathsf{S};\mathsf{Y}) - \tilde{\mathbb{I}}_{10(t+1)}(\mathsf{S};\mathsf{Y}) < 10\Delta$.

Inputs: Our two datasets were each converted to a table, implemented as a Pandas DataFrame, with columns *Object ID*, *Object Size*, and *Retrievals* (per unit time). For each run of an algorithm, we supplied the algorithm with the table for the dataset being tested and a value for c. We also provided PRP$_{\text{init}}$ and PRP$_{\text{inc}}$ with a third input: the value for Δ to indicate when these algorithms should halt iterating and return their padding scheme. For the NodeJS dataset we set $\Delta = 1 \times 10^{-3}$, and for the Unsplash dataset we set $\Delta = 5 \times 10^{-4}$. We arrived at these values for Δ by observing, for each dataset, at what point smaller values for Δ yielded minimal reductions to $\mathbb{I}(\mathsf{S};\mathsf{Y})$ at the expense of increasing runtimes.

Outputs: PRP$_{\text{init}}$, PRP$_{\text{inc}}$, and P-ALPACA each returned their scheme for $\mathbb{P}(\mathsf{Y} = y \mid \mathsf{S} = s)$ as a compact, linear array that only contained the scheme's nonzero values. These algorithms also returned the auxiliary information needed to support sampling from this array on a per-request basis. POP and PwoD each returned a Python dictionary that mapped each object's original size to its padded size.

B. Evaluation Platform

Our evaluation platform was an Ubuntu 20.04.1 LTS virtual machine running on VMware Workstation 15.5 Pro. The host had a quad-core (eight logical processors) 2.80GHz Intel Core i7-7700HQ CPU and 32GB of RAM; the virtual machine was provided six logical processors and 20GB. That said, memory usage was not a factor for any of the algorithms with the values of c tested.

C. Runtime Test Procedure

For PRP$_{\text{init}}$, POP, P-ALPACA, and PwoD, for each dataset and for $c \in \{1.01, 1.02, \ldots, 1.1\}$, we measured the time for each algorithm to calculate its padding scheme. Each such test was conducted ten times, and we report the average of these measurements.

To test PRP$_{\text{inc}}$, for each dataset we first created a list of the ten most frequently retrieved objects. Then, we simulated each object's size being increased by 25% and measured the time for PRP$_{\text{inc}}$ to update the padding scheme provided by PRP$_{\text{init}}$. We report the average of ten such tests for each value of c. We took this approach to test PRP$_{\text{inc}}$ because, in general, updating the size of frequently requested objects led to longer runtimes than when updating the sizes of infrequently requested objects. Thus, our test for PRP$_{\text{inc}}$ was designed to estimate its worst-case runtime.

D. Runtime Results

The runtime results are depicted in Figs. 9–10. The relative standard deviations for all algorithms were less than 18%, except for P-ALPACA, which ranged up to 124%. Our use of OpenMP in P-ALPACA led to high variance since its runtimes were so low.

Though each algorithm took less time on the Unsplash dataset, Fig. 10 looks largely similar to Fig. 9. In other words, the relative performance between each algorithm remained unchanged between the two datasets. The absolute runtime differences on the NodeJS and Unsplash datasets were due mainly to their sizes—the NodeJS dataset contains about $17\times$ more objects.

Overall, PwoD outperformed the other algorithms, as its runtime remained constant and negligible for a given dataset. We then see P-ALPACA, POP, PRP$_{\text{inc}}$, and PRP$_{\text{init}}$, in that order. Furthermore, we see from both Fig. 9 and Fig. 10 that PRP$_{\text{init}}$ required significantly more time to compute its initial padding scheme for each value of c than the other algorithms required. With the addition of PRP$_{\text{inc}}$, though, PRP required considerably less time to maintain its padding scheme as object sizes were changed. The addition of PRP$_{\text{inc}}$ therefore put PRP's steady-state runtime much closer to that of the other algorithms.

Finally, we attribute the slight decrease in PRP$_{\text{init}}$'s runtime between $c = 1.09$ and $c = 1.1$ in Fig. 10 to our decision to calculate only $\tilde{\mathbb{I}}_{10t}(\mathsf{S}; \mathsf{Y})$, rather than each $\tilde{\mathbb{I}}_t(\mathsf{S}; \mathsf{Y})$. To confirm this, we conducted the Unsplash runtime test a second time while calculating each $\tilde{\mathbb{I}}_t(\mathsf{S}; \mathsf{Y})$. Though not depicted, this test resulted in increasing runtimes for PRP$_{\text{init}}$, from a low of 0.75 seconds at $c = 1.01$, to a high of 3.25 seconds at $c = 1.1$. Additionally, this test revealed that, by calculating only $\tilde{\mathbb{I}}_{10t}(\mathsf{S}; \mathsf{Y})$, PRP$_{\text{init}}$ took 14 extra iterations at $c = 1.09$, whereas it only took 6 extra iterations at $c = 1.1$. Thus, we conclude that, although calculating $\tilde{\mathbb{I}}_{10t}(\mathsf{S}; \mathsf{Y})$ resulted in overall reduced runtimes for PRP$_{\text{init}}$, it had the effect of causing PRP$_{\text{init}}$ to iterate more times than necessary to reach its termination condition, which impacted some values of c more than others.
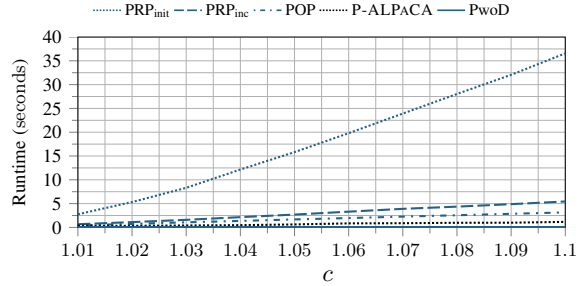

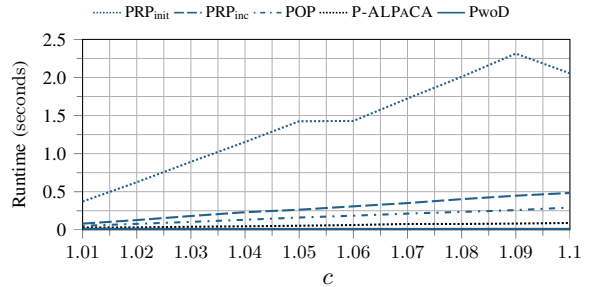Fig. 9: Runtimes on the NodeJS dataset.

Fig. 10: Runtimes on the Unsplash Lite dataset.

E. Bandwidth Increase Analysis

Our final evaluation analyzed the overall bandwidth increase incurred by each of the padding algorithms for $c \in \{1.01, 1.02, \ldots, 1.1\}$. For this analysis, we calculated the multiplicative increase to bandwidth that the object store would have incurred over an arbitrary time interval if all of its objects had been retrieved according to the distribution S. The results for the NodeJS dataset are depicted in Fig. 11. Note that Fig. 11's y-axis starts at 1, which represents sending objects with no padding, i.e., this is the object store's baseline average bandwidth. Unsplash is not shown, as the only significant difference was that PADMÉ yielded a point at $(1.031, 1.011)$, compared to $(1.093, 1.022)$ in Fig. 11.

Two trends stand out in Fig. 11. First, D-ALPACA had a negligible effect on the object store's overall bandwidth since, for all values of c, the chosen input parameter σ (see Sec. IV-B) was relatively small compared to the majority of object sizes in the dataset. Second, PRP, PwoD, POP, and P-ALPACA resulted in similar increases to the object store's overall bandwidth, despite producing different padding schemes.

Finally, as mentioned in Sec. IV-B, PADMÉ is not a tunable algorithm, and so it yielded a point rather than a line for this analysis. Thus, at approximately $c = 1.09$, PADMÉ caused this object store's bandwidth to increase by 2.25%, whereas our algorithms increased this object store's bandwidth by 4.5%. Note, though, that PADMÉ's 2.25% savings in bandwidth (4.5% minus 2.25%) came with a $\mathbb{I}(\mathsf{S}; \mathsf{Y})$ that was roughly 18% higher than our algorithms (see Fig. 4a).

VI. Discussion

We see primarily two opportunities for improving on our results. First, perhaps the most significant limitation of our
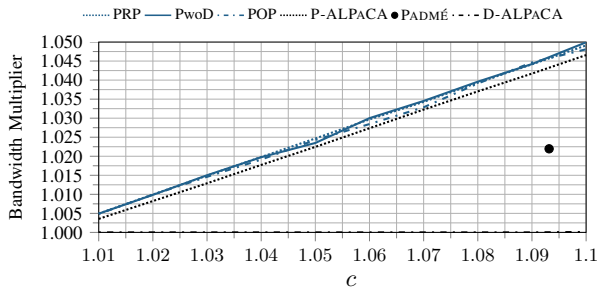
Fig. 11: Bandwidth increase for the NodeJS dataset.

results is that the privacy metric we optimize measures privacy only for *independent* object retrievals. In several contexts, most notably web browsing, there are dependencies among objects retrieved due to hyperlinking, a fact that has been used in several previous works to fingerprint webpages (e.g., [19], [40], [61]). This leakage could be reduced by transcluding *at the server* those objects that the client-side browser would typically transclude when assembling a webpage (images, scripts, stylesheets, etc.); the assembled page could then be padded at the server as a single object. Some statistical dependencies among pages would nevertheless remain due to hyperlinks followed manually by the user.

Second, while our performance analysis of PRP in Sec. V touched on the need to update padding distributions in response to object (size) changes, we believe this topic warrants further consideration. An object store that supports updates to its objects' sizes will need to recalculate object padding distributions in response to those updates before serving the new objects, if it is to protect its clients' privacy from the attacker we consider. If objects are updated frequently, then these updates might therefore need to be batched and remain hidden from clients until after the object store recalculates the necessary padding distributions. Evaluating the best balance among padding algorithm, batch size, and length of the recalculation time window for a given type of object store is a topic of future work.

VII. Conclusion

Object size is a particularly potent feature for traffic analysis, and one that encryption does nothing to obscure. In this paper we provided algorithms for computing padding schemes suitable for various scenarios, in which the object store responds to every request with the same (padded) object copy; in which the object store pads each object anew before serving it; and in which the object store has no knowledge of (or little confidence in its knowledge of) the distribution of object requests it will receive. In each case we provided an algorithm for constructing a padding scheme $\lceil \cdot \rceil$, subject to a padding overhead constraint, that minimizes the information gain $\mathbb{I}(\mathsf{S};\mathsf{Y})$ about the object identity S based on the padded size Y of the object returned or, in the last case, an upper bound $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ on the information gain that is robust to any query distribution. Our empirical analysis using datasets of NodeJS packages and of Unsplash Lite photos suggested that our algorithms provide better privacy than competitors from recent literature, and do so efficiently.

References

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

[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*, Mar. 2019.

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

[4] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Dossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with Cipherbase," in *6th Biennial Conference on Innovative Data Systems Research*, Jan. 2013.

[5] S. Arimoto, "An algorithm for computing the capacity of arbitrary discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 14–20, Jan. 1972.

[6] G. Asharov, T.-H. H. Chan, K. Nayak, R. Pass, L. Ren, and E. Shi, "Locality-preserving oblivious RAM," in *Advances in Cryptology – EUROCRYPT 2019*, ser. Lecture Notes in Computer Science, vol. 11477, May 2019, pp. 214–243.

[7] 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*, Feb. 2013.

[8] T. Berger and J. D. Gibson, "Lossy source coding," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2693–2723, Oct. 1998.

[9] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted HTTP streams," in *5th Privacy Enhancing Technologies Symposium*, ser. Lecture Notes in Computer Science, vol. 3856, May 2005, pp. 1–11.

[10] R. Blahut, "Computation of channel capacity and rate-distortion functions," *IEEE Transactions on Information Theory*, vol. 18, no. 4, pp. 460–473, Jul. 1972.

[11] 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*, Oct. 2012, pp. 605–616.

[12] B. Case, J. Honaker, and M. Movahedi, "The privacy-preserving padding problem: non-negative mechanisms for conservative answers with differential privacy," *arXiv preprint arXiv:2110.08177*, 2021.

[13] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *22nd ACM Conference on Computer and Communications Security*, Oct. 2015, pp. 668–679.

[14] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Advances in Cryptology – CRYPTO 2013*, ser. Lecture Notes in Computer Science, vol. 8042, Aug. 2013, pp. 353–373.

[15] 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*, Feb. 2019.

[16] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *3rd International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, vol. 3531, Jun. 2005, pp. 442–455.

[17] Z. Chang, D. Xie, and F. Li, "Oblivious RAM: A dissection and experimental evaluation," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, Aug. 2016.

[18] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Advances in Cryptology – ASIACRYPT 2010*, ser. Lecture Notes in Computer Science, vol. 6477, Dec. 2010, pp. 577–594.

[19] 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.

[20] G. Cherubin, J. Hayes, and M. Juarez, "Website fingerprinting defenses at the application layer," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 186–203, 2017.

[21] 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.

[22] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, and M. K. Reiter, "On web browsing privacy in anonymized netflows," in *16th USENIX Security Symposium*, Aug. 2007, pp. 339–352.

[23] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Wiley-Interscience, 2006.

[24] I. Csiszar and J. Korner, "Broadcast channels with confidential messages," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 339–348, May 1978.

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

[26] ——, "The traffic analysis of continuous-time mixes," in *4th Privacy Enhancing Technologies Symposium*, ser. Lecture Notes in Computer Science, vol. 3424, May 2004.

[27] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. McGraw-Hill Education, 2006.

[28] N. Ding and P. Sadeghi, "A submodularity-based clustering algorithm for the information bottleneck and privacy funnel," in *20th IEEE Information Theory Workshop*, 2019, pp. 279–283.

[29] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[30] 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*, 2012, pp. 332–346.

[31] R. Ferenc, P. Hegedus, P. Gyimesi, G. Antal, D. Bán, and T. Gyimóthy, "Challenging machine learning algorithms in predicting vulnerable javascript functions," in *7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2019, p. 8–14.

[32] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43, no. 3, May 1996.

[33] R. Gonzalez, C. Soriente, and N. Laoutaris, "User profiling in the time of HTTPS," in *16th Internet Measurement Conference*, Nov. 2016, pp. 373–379.

[34] P. K. Gopala, L. Lai, and H. El Gamal, "On the secrecy capacity of fading channels," *IEEE Transactions on Information Theory*, vol. 54, no. 10, pp. 4687–4698, Oct. 2008.

[35] P. Grubbs, A. Khandelwal, M.-S. Lacharité, L. Brown, L. Li, R. Agarwal, and T. Ristenpart, "PANCAKE: Frequency smoothing for encrypted data stores," in *29th USENIX Security Symposium*, Aug. 2020.

[36] 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.

[37] D. Gunduz, E. Erkip, and H. V. Poor, "Lossless compression with security constraints," in *IEEE International Symposium on Information Theory*, Jul. 2008, pp. 111–115.

[38] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium*, Aug. 2016, pp. 1187–1203.

[39] 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*, Nov. 2009, pp. 31–42.

[40] A. Hintz, "Fingerprinting websites using traffic analysis," in *2nd Privacy Enhancing Technologies Symposium*, ser. Lecture Notes in Computer Science, vol. 2482, Apr. 2002, pp. 171–178.

[41] 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*, Feb. 2012.

[42] I. Issa, A. B. Wagner, and S. Kamath, "An operational approach to information leakage," *IEEE Transactions on Information Theory*, vol. 66, no. 3, Mar. 2020.

[43] 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*, Nov. 2014, pp. 263–274.

[44] S. Kamara and T. Moataz, "SQL on structurally-encrypted databases," in *Advances in Cryptology – ASIACRYPT 2018*, ser. Lecture Notes in Computer Science, vol. 11272, Dec. 2018, pp. 149–180.

[45] ——, "Computationally volume-hiding structured encryption," in *Advances in Cryptology – EUROCRYPT 2019*, ser. Lecture Notes in Computer Science, vol. 11477, May 2019, pp. 183–213.

[46] S. Kamara, T. Moataz, and O. Ohrimenko, "Structured encryption and leakage suppression," in *Advances in Cryptology – CRYPTO 2018*, ser. Lecture Notes in Computer Science, vol. 10991, Aug. 2018, pp. 339–370.

[47] 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*, Oct. 2016.

[48] B. Kopf and M. Durmuth, "A provably secure and efficient countermeasure against timing attacks," in *22nd IEEE Computer Security Foundations*, Jul. 2009.

[49] 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.

[50] S. K. Leung-Yan-Cheong and M. E. Hellman, "The Gaussian wire-tap channel," *IEEE Transactions on Information Theory*, vol. 24, no. 4, pp. 451–456, Jul. 1978.

[51] 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*, ser. Lecture Notes in Computer Science, vol. 3110, Sep. 2004, pp. 251–265.

[52] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and ℓ-diversity," in *23rd IEEE International Conference on Data Engineering*, Apr. 2007.

[53] M. Liberatore and B. N. Levine, "Inferring the source of encrypted HTTP connections," in *13th ACM Conference on Computer and Communications Security*, Oct. 2006, pp. 255–263.

[54] 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*, vol. 11, no. 6, pp. 538–552, Nov.-Dec. 2014.

[55] 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*, ser. Lecture Notes in Computer Science, vol. 7384, Jul. 2012, pp. 79–99.

[56] 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*, Feb. 2011.

[57] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "ℓ-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, Mar. 2007.

[58] G. Macia-Fernandez, Y. Wang, R. Rodriguez-Gomez, and A. Kuzmanovic, "ISP-enabled behavioral ad targeting without deep packet inspection," in *29th IEEE Conference on Computer Communications*, Mar. 2010.

[59] A. Makhdoumi, S. Salamatian, N. Fawaz, and M. Médard, "From the information bottleneck to the privacy funnel," in *14th IEEE Information Theory Workshop*, 2014, pp. 501–505.

[60] 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*, May 2015, pp. 763–777.

[61] 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*, ser. Lecture Notes in Computer Science, vol. 8555, Jul. 2014, pp. 143–163.

[62] M. Naveed, "The fallacy of composition of oblivious RAM and search-able encryption," Cryptology ePrint Archive, Report 2015/668, 2015, https://eprint.iacr.org/2015/668.

[63] 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*, vol. 2019, no. 4, pp. 6–33, 2019.

[64] 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*, Oct. 2011, pp. 103–114.

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

[66] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *23rd ACM Symposium on Operating Systems Principles*, Oct. 2011, pp. 85–100.

[67] O. Rioul, "Variations on a theme by Massey," *IEEE Transactions on Information Theory*, vol. 68, no. 5, pp. 2813–2828, May 2022.

[68] P. Samarati, "Protecting respondents' identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, Nov./Dec. 2001.

[69] L. Sankar, S. R. Rajagopalan, and H. V. Poor, "Utility-privacy tradeoffs in databases: An information-theoretic approach," vol. 8, no. 6, pp. 838–852, Jun. 2013.

[70] A. Serjantov and P. Sewell, "Passive attack analysis for connection-based anonymity systems," in *8th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, vol. 2808, Oct. 2003, pp. 116–131.

[71] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct. 1949.

[72] ——, "Coding theorems for a discrete source with a fidelity criterion," in *Institute of Radio Engineers, International Convention Record*, vol. 7, 1959, pp. 142–163, reprinted as [74, pp. 325-350].

[73] R. Sibson, "Information radius," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 14, pp. 149–160, 1969.

[74] N. J. A. Sloane and A. D. Wyner, Eds., *Claude E. Shannon: Collected Papers*. IEEE Press, 1993.

[75] G. Smith, "On the foundations of quantitative information flow," in *12th International Conference on Foundations of Software Science and Computational Structures*, ser. Lecture Notes in Computer Science, vol. 5504, 2009, pp. 288–302.

[76] 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*, May 2002, pp. 19–30.

[77] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[78] S. Verdú, "α-mutual information," in *Information Theory and Applications Workshop*, Feb. 2015, pp. 1–6.

[79] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *23rd USENIX Security Symposium*, Aug. 2014, pp. 143–157.

[80] T. Wang and I. Goldberg, "Improved website fingerprinting on Tor," in *12th Workshop on Privacy in the Electronic Society*, Nov. 2013, pp. 201–212.

[81] A. D. Wyner, "The wire-tap channel," *Bell System Technical Journal*, vol. 54, no. 8, pp. 1355–1387, Oct. 1975.

[82] H. Yamamoto, "Coding theorems for Shannon's cipher system with correlated source outputs, and common information," *IEEE Transactions on Information Theory*, vol. 40, no. 1, pp. 85–95, Jan. 1994.

[83] J. Yan and J. Kaur, "Feature selection for website fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, Oct. 2018.

[84] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter, "Browser fingerprinting from coarse traffic summaries: Techniques and implications," in *6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, vol. 5587, Jul. 2009, pp. 157–175.

[85] R. W. Yeung, "The Blahut–Arimoto algorithms," in *Information Theory and Network Coding*. Springer, 2008, pp. 211–228.

[86] X. Yin and H. An, "A novel algorithm with reduced mutual information for smart meter privacy protection," in *5th IEEE International Conference on Cloud Computing and Big Data Analytics*, Apr. 2020.

[87] X. Zhang, J. Hamm, M. K. Reiter, and Y. Zhang, "Statistical privacy for streaming traffic," in *26th ISOC Network and Distributed System Security Symposium*, Feb. 2019.

[88] 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*, ser. Lecture Notes in Computer Science, vol. 3424, May 2004, pp. 207–225.

	y							
s	2493855	4987710	9975420	14963130	17456985	19950840	27432405	34913970
P0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
P4	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
P5	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P6	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(a) $c = 2$

	y					
s	4987710	9975420	14963130	19950840	29926260	34913970
P0	1.00	0.00	0.00	0.00	0.00	0.00
P1	1.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	1.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	1.00	0.00	0.00	0.00
P4	0.00	0.00	1.00	0.00	0.00	0.00
P5	0.00	0.00	0.00	1.00	0.00	0.00
P6	0.00	0.00	0.00	1.00	0.00	0.00
P7	0.00	0.00	0.00	1.00	0.00	0.00
P8	0.00	0.00	0.00	0.00	1.00	0.00
P9	0.00	0.00	0.00	0.00	0.00	1.00

(b) $c = 3$

Fig. 12: $\mathbb{P}\big(\mathsf{Y} = y \mid \mathsf{S} = s\big)$ produced by D-ALPaCA [20].

APPENDIX

In this appendix we illustrate the padding schemes produced by each of PADMÉ, D-ALPaCA, and P-ALPaCA for the dataset in Table I, to enable comparisons to the analogous results for our algorithms shown in Figs. 1–3.

A. D-ALPaCA

Fig. 12 illustrates $\mathbb{P}\big(\mathsf{Y} = y \mid \mathsf{S} = s\big)$ when D-ALPaCA is applied to the objects listed in Table I. The values of y (i.e., the column headings) listed in Fig. 12 differ from those in Figs. 1–3 because D-ALPaCA pads objects to sizes that are not necessarily the sizes of other objects, unlike POP, PRP, and PwoD. Fig. 12a illustrates that D-ALPaCA is not particularly effective for the objects listed in Table I when $c = 2$, as retrievals of six of the ten objects can be identified immediately based on the sizes to which they are padded. Only once $c = 3$ is a majority of those objects padded to sizes where multiple objects share the same padded size (Fig. 12b).

B. PADMÉ

Unlike the other algorithms considered here, PADMÉ is not tunable to different padding factors c. Moreover, as shown in

						y					
s	2555904	3866624	7995392	13369344	13631488	16252928	16777216	19922944	26214400	34603008
P0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
P4	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
P5	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Fig. 13: $\mathbb{P}\big(\mathsf{Y} = y \mid \mathsf{S} = s\big)$ produced by PADMÉ [63].

						y					
s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.08	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	0.29	0.66	0.06	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.34	0.03	0.15	0.29	0.14	0.06	0.00
P4	0.00	0.00	0.00	0.00	0.04	0.23	0.43	0.21	0.09	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.24	0.45	0.22	0.10	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.59	0.28	0.13	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.44	0.20	0.37
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.35	0.65
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(a) $c = 2$

						y					
s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.08	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	0.84	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	0.13	0.31	0.03	0.14	0.27	0.13	0.00	0.00
P3	0.00	0.00	0.00	0.30	0.03	0.13	0.26	0.12	0.06	0.10
P4	0.00	0.00	0.00	0.00	0.04	0.19	0.37	0.18	0.08	0.15
P5	0.00	0.00	0.00	0.00	0.00	0.20	0.38	0.18	0.08	0.15
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.48	0.23	0.10	0.19
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.44	0.20	0.37
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.35	0.65
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

(b) $c = 3$

Fig. 14: $\mathbb{P}\big(\mathsf{Y} = y \mid \mathsf{S} = s\big)$ produced by P-ALPACA [20].

Fig. 13, PADMÉ is unfortunately ineffective when applied to the objects listed in Table I—each of the objects is padded to a different size. In part this is due to the small number of objects or, more precisely, the low density of object sizes.

C. P-ALPACA

The effect of P-ALPACA on the objects in Table I is shown in Fig. 14. Among all of the algorithms we consider, P-ALPACA permits the widest variety of possible padding sizes for each object, for the objects in Table I: e.g., when $c = 3$, only the largest object can be padded to only a single size, and some objects can be padded to up to six distinct sizes.