A Paged Domain Name System for Query Privacy

Daniele E. Asoni, Samuel Hitz, and Adrian Perrig

Network Security Group
Department of Computer Science, ETH Zürich
{daniele.asoni,samuel.hitz,adrian.perrig}@inf.ethz.ch

Abstract. The lack of privacy in DNS and DNSSEC is a problem that has only recently begun to see widespread attention by the Internet and research communities, and the solutions proposed so far only look at a narrow slice of the design space. In this paper we investigate a new approach for a privacy-preserving DNS mechanism that hides query information from root name servers and TLD registries. Our architecture lets TLD registries group the DNS records in their zones together into *pages*. Resolvers cache all pages locally, and retrieve only small incremental updates to optimize performance. We show that this strategy is particularly effective given the relatively static nature of TLD zone records. We analyze the privacy guarantees to assess the potential and limitations of our approach; we also evaluate the memory overhead for a resolver, and obtain feasibility guarantees through a prototype implementation of the new functionalities for resolvers and registries.

1 Introduction

The Domain Name System (DNS) [29,30] is a fundamental building block of the Internet, providing host name to IP address translation. Its design has been sufficiently scalable to cope with the Internet's growth, but among its deficiencies is the lack of privacy protection. The DNS security extensions (DNSSEC), which are still far from widespread adoption, have addressed some of DNS's flaws, but privacy has explicitly remained a non-goal in the design of DNSSEC [5]. While the Internet Engineering Task Force (IETF) has recently started considering privacy concerns for DNS more seriously [8], so far only minor improvements have been proposed [9].

Users with very high privacy requirements will resort to an anonymous communication system (ACS) such as Tor [16], which will anonymize not only the DNS lookups, but also the subsequent communications with the hosts whose addresses are obtained through the lookup. This is necessary, for instance, if a user wishes to hide from its own Internet service provider (ISP) what hosts it communicates with. However, communication over Tor comes with harsh performance penalties, so for clients which have some degree of trust in their ISP, a more lightweight solution is desirable. In particular, we identify the main privacy threat in this scenario to be large-scale information collection at the highest levels of the DNS hierarchy: the name servers of the root and of the top-level

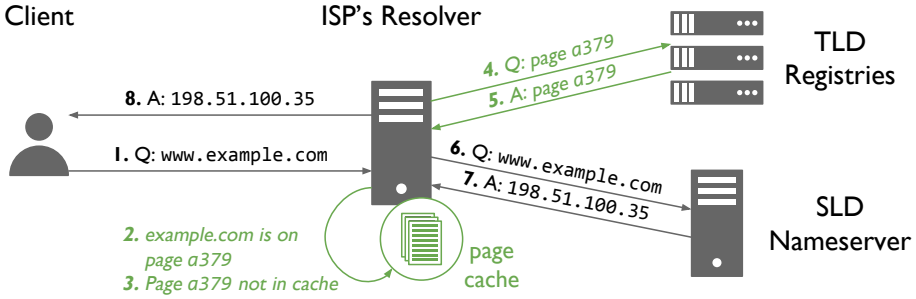

Fig. 1: High-level overview of the PageDNS architecture. Steps 2–5 are specific to PageDNS, while the others are as in DNS. In the example shown, the entire page `a379` is retrieved from the TLD registries, but in practice the page would be cached at the resolver, and at most a (much smaller) incremental update would be retrieved.

domain (TLD) registries. These are centralized observation points that are ideally suited for surveillance by a nation state actor. While a recent proposal [9] would, if accepted, hide query information from the root name servers, there seems to be no possibility in current DNS to hide sensitive information from TLD registries.

Although users gain some privacy by relying on the recursive resolver provided by their ISP, this method is not secure against an adversary who is able to correlate multiple queries through timing. For instance, the adversary may observe that a certain set of domains is always queried together in a short time interval, and thus infer that the same user is responsible: if one of the domains identifies the user (e.g., because it is the user's own website, which is otherwise scarcely visited), then the adversary is able to deanonymize all the other queries as well. Furthermore, users may wish to conceal the fact that a certain domain is being queried at all, a property akin to private information retrieval (PIR) [11,31], which, incidentally, cannot be achieved even if DNS lookups are performed over an ACS.

In this paper we propose a system whose goal is to prevent information leakage to the DNS root and TLD registries, including the information of what domains are actually queried. Our system requires changes to the TLD registries and to the recursive resolvers, but is transparent to clients and to second level domain (SLD) authoritative name servers, which continue to use the traditional DNS protocol. The core idea of the system is to group records in the TLD zones into fixed sets we call *pages*, which are created and maintained jointly by the TLD registries. Recursive resolvers query for entire pages, rather than single records, which provides a basic amount of privacy (see Figure 1 for a high-level overview). We improve the performance of this basic mechanism with optimizations such as full page caching on the recursive resolvers, and we improve its privacy with enhancements such as cover page queries from the recursive resolver to the TLD registries.

1.1 Overview

Our *Paged Domain Name System* (PageDNS) introduces a new way in which recursive resolvers can obtain records from TLD registries in a privacy-preserving manner. The TLD registries collaborate to group the records of all their zones into 2^l ($\simeq 10^5$) sets of records which we call *pages*. Each page contains n records, with $n \simeq 10^4$, assuming a total number of records of around one billion (see Section 4.1). The overwhelming majority of these records are name server records of second level domains (SLDs), i.e., records providing the IP addresses of the name servers authoritative for various SLDs (e.g., `example.com`). To spread the records uniformly across the pages in a way that allows resolvers to easily determine which page stores which record, each page is given a unique l-bit identifier, and each record is assigned to the page whose identifier matches the first l bits of the hash value of the record's domain name.

A recursive resolver with PageDNS support should, for performance reasons, cache all pages locally, de facto mirroring all TLD zones. These cached copies are kept indefinitely, in particular beyond their expiration time. When queried for a domain name (say `www.example.com`), the recursive resolver proceeds as follows. First, it determines which page should store the corresponding SLD record (the NS record for `example.com`). Second, it checks whether it has an up-to-date copy of that page in its cache. If not, the resolver sends a page query to one of the TLD registries, specifying the *version number* of the locally cached copy. The registry then replies with a list of records that have been changed, added, or removed since the specified version. Because the name servers for SLDs are relatively static, as we show in Section 4.1, the size of these *incremental updates* will typically be small. Finally, once the recursive resolver has obtained the NS record for the SLD, it completes the iterative lookup as in DNS, and returns the response to the client.

Replication of popular records. Requesting a page instead of a single domain protects query privacy, because an adversary observing a page request cannot directly determine which domain among the $\sim 10^4$ domains in the requested page is accessed by a client. However, domains have very different popularity, meaning that the probability that a certain page is requested because it contains, e.g., the record for `twitter.com` is much higher than the probability that the page was accessed due to some obscure domain that has its record on the same page. To counter this problem and provide privacy protection even for popular domains we *replicate* records of popular domains across multiple pages: the higher the popularity of a domain, the higher the replication degree of its record. In Section 3.1, we show in particular that an optimal solution is to replicate the $\sim 0.01\%$ most popular domains, with the most popular one being replicated on all pages.

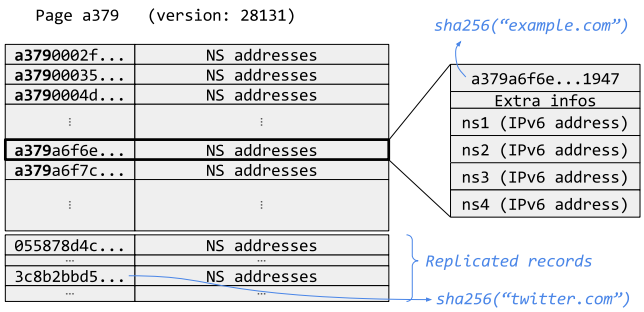

Fig. 2: Structure of a PageDNS page. The highlighted detail on the right shows the structure of a single record for the SLD `example.com`.

2 Design

In this section we describe the details of our Paged Domain Name System (PageDNS). We begin by describing our threat model and privacy goals, and then define the structure of pages, name resolution process, and other protocol aspects.

2.1 Threat Model

We want to prevent (government-level) monitoring which targets DNS query information. In particular, we want to prevent linkability between clients and the queries they make, and, to the extent possible, we also try to hide the fact that a certain name is being queried, i.e., that there is interest by *some* client for a certain name. We exclude the case of a compromised client ISP, which would be able to observe not only the DNS queries of its clients, but also the communications after that, requiring the use an anonymous communication system (ACS) to achieve anonymity. For analogous reasons, we do not consider other types of in-network adversaries. Instead, we aim to provide privacy with respect to the DNS root and TLD registries, which constitute a centralized point which is ideal for mass surveillance.

2.2 Page Structure

PageDNS pages consist of a sequence of records, plus information such as the page's length, expiration time, and version number. Each page is identified by an l-bit hash prefix, and all records whose domains' hash values match the prefix are contained in the page. Additionally, each page has a small separate section which contains the records of replicated domains (whose hashes will typically not match the prefix, see below). Besides the hash of the domain name, each record contains a set of addresses of the domain's name servers, and additional information such as the type of the addresses (e.g., IPv6 or IPv4). The reasons

for indexing records by hash instead of including the domain name are twofold: first, it fixes the length of the identifier; second, it provides a degree of protection against zone enumeration [26]. In Figure 2 the high-level structure is depicted for one sample page, and the details for the record of domain `example.com` are shown. This sample record contains four IPv6 addresses, which in terms of size we consider to be a reasonable upper bound.

Since one IPv6 address is 16 bytes long, and assuming a hash size of 32 bytes[1], we obtain 96 bytes, excluding the additional information, so overall we round the size of a record to 100 bytes for the scope of this discussion. Considering a limit of 1 MB on page size and a total of 10^9 domain names (see Section 4), this implies a total of 10^5 pages, each containing around 10^4 records. We estimate the variance of the size of pages in Appendix B, and we find that the probability of a deviation of the size of over 10% is negligible.

Record Replication. We observe that the popularity of domains has the potential to heavily affect the privacy of page queries. For instance, a query for a page which contains the record of a very popular domain is likely to be due to an access to that domain. To mitigate this problem we adopt record replication for popular domains. In Appendix A, we show analytically that the optimal replication is proportional to the popularity of the domain, assuming that the popularity of domains follows a Zipf distribution.[2] In particular this means that approximately only the top 200,000 domains need to be replicated, with the most popular domain being replicated on all pages (the effects of replication become clearer in our analysis in Section 3.1, and are depicted in Figure 3). This replication has an overall size overhead of 0.23% (with a median of 25 replicated records per page), but it allows to hide accesses even to the most popular domains. Furthermore, our analysis shows that the probability of accessing a certain page because of interest in a domain with low popularity (any non-replicated domain) is lower than the probability that that page is being accessed because of interest in the most popular domain (which is replicated on that page). This means that replication provides effective and relatively uniform privacy protection for less popular domains. These replicas are placed separately on a page (e.g., at the end, see Figure 2) and are identified by their hash value.

Assigning records to pages. Mapping records to the $m = 2^l$ pages is not entirely trivial. For non-replicated domains, we use the first l bits of the hash value of the domain name to determine the page identifier to which the domain is assigned. However, for replicated records another scheme is needed. We propose a scheme

[1] We consider SHA-256 as a reasonable choice for the hash function. While the size could be reduced to 16 bytes while still retaining a negligible collision probability in a non-adversarial setting, a larger size is necessary if we want to have a negligible probability even in a scenario where the adversary actively tries to find a domain name which will result in a collision.

[2] In practice, popularity will vary on a regional basis. We envision that replication may be made region-specific (the non-replicated part of each page would remain the same). We leave a more detailed analysis of these aspects to future work.

based on a pseudo-random permutation (PRP), keyed with the hash value of the domain name: this PRP has as domain the set of all page identifiers (i.e., all integers from 1 to m).[3] For a replicated domain d, the PRP maps all integers between 1 and the replication degree of d, $r(d)$, to the page identifiers on which the replicas should be stored. Since the PRP is keyed with the hash of d, the mapping will be independent from the mapping of replicas for other domains.

Note that this method ensures minimal modifications to the pages as the replication of a domain changes: assuming a domain previously replicated r_{old} times increases its popularity and has to be replicated r_{new} times ($r_{new} > r_{old}$), the first r_{old} replicas will remain the same, and only $r_{new} - r_{old}$ additional pages have to be changed to include the replica. Similarly, if $r_{new} < r_{old}$, the first r_{new} replicas will remain the same, and only $r_{old} - r_{new}$ pages have to be changed to remove the extra replicas.

We point out that for this mechanism to work, the resolvers need to be aware of the replication degree of the domains they look up. To that end, the TLD registries create a special *meta-page* for replication, which lists all the most popular domains (by their hash value), and for each of them it provides the replication degree. This meta-page has a size of about 7 normal pages (7 MB), and is updated less frequently. We show how the TLD registries determine the popularity of domains in Section 2.4. In the next section, we show how clients can resolve a name, including more details about how the case of replicated domains is handled.

2.3 Resolving a Name

Algorithm 1 shows the steps a recursive resolver performs when resolving a fully qualified domain name (FQDN), e.g., `www.example.com`. First, the algorithm splits the FQDN into the SLD (`example.com`) and the remaining part (typically the host name, `www`). On a high-level, the algorithm then retrieves the address of an authoritative name server for `example.com` using PageDNS (Lines 3–30) before completing the lookup for `www.example.com` using traditional DNS. The resolver starts by identifying the replication degree of the SLD. To obtain this information, it retrieves the meta page from the registries which contains all the replicated domains together with their replication degree. As for ordinary pages, the resolvers also keeps the meta page in its cache, and will therefore usually only need to retrieve an incremental update of the meta page. If a domain is not replicated it has a replication degree of 1.

Then the algorithm checks whether it already has a cached copy of a page that contains the record for the domain. To that end, the algorithm computes the possible page identifiers that could contain the record by calling `CalcIdentifier` (Algorithm 2) for all $i \in \{1..\text{ReplDeg}\}$ and checks if the cache contains any of these pages. A possible optimization for this step would be to keep track, for

[3] PRPs of small domain size can be implemented using format-preserving encryption (FPE) schemes, there exist suitable encryption modes that use standard AES block ciphers as a primitive and achieve FPEs of arbitrary domain size.

Algorithm 1 FQDN resolution on the recursive resolver.

```
 1: procedure ResolveFQDN(FQDN, MetaPage, Cache, Registry)
 2:     Host, Domain ← DomainSplit(FQDN)
 3:     if Domain ∈ MetaPage then
 4:         ReplDeg ← MetaPage[Domain].ReplDeg
 5:     else
 6:         ReplDeg ← 1
 7:     end if
 8:     for all i ∈ RandomShuffle({1..ReplDeg}) do
 9:         PageId ← CalcIdentifier(Domain, i)
10:         if PageId ∈ Cache then
11:             Page ← Cache[PageId]
12:             break
13:         end if
14:     end for
15:     if not Page or HasExpired(Page) then
16:         if not Page then
17:             k ← RandomChoice({1..ReplDeg})
18:             PageId ← CalcIdentifier(Domain, k)
19:             Page ← Query(Registry, PageId)
20:             Page.Registry ← Registry
21:         else
22:             Page ← Query(Page.Registry, PageId, Page.Version)
23:         end if
24:         Key ← PubKey(Registry)
25:         if not Verify(Page.MerkleRoot.Sig, Key) then
26:             abort()
27:         end if
28:         Cache[PageId] ← Page
29:     end if
30:     NS ← BinSearch(Page, SHA256(Domain))
31:     IP ← CompleteLookup(NS, Host)
32:     return IP
33: end procedure
```

the more highly replicated records that are requested, of the cached pages that contain them, in order to avoid the computation of tens of thousands of hashes. This step can also be optimized when all pages are cached by the resolver.

The page identifier calculation depends on the chosen replica ID (k): if k is 1, the original record for the domain is chosen, and the page identifier determined as the l-bit prefix of the hash of the domain name (Algorithm 2, line 3). If $k > 1$, then the page is determined by applying a PRP with domain $\{1, \ldots, 2^l\}$ keyed with the hash of the domain name to k (line 5).

If the cached page containing the domain has expired, or if no page containing the domain is cached by the resolver, a page query has to be send out. In case a cached page is available but outdated, the resolver can perform an incremental query by attaching the version number of the cached page to the request.

Algorithm 2 Calculating the page identifier for a (possibly replicated) domain.

```
1: procedure CalcIdentifier(Domain, k)
2:     if k == 1 then
3:         PageId ← SHA256(Domain)[0:l]
4:     else
5:         PageId ← PRP_{2^l}(SHA256(Domain); k)
6:     end if
7:     return PageId
8: end procedure
```

Note that we avoid querying multiple registries for the same page (line 22). If, on the other hand, no page is cached, the algorithm needs to download an entire page page. First, a replica ID is chosen uniformly at random from the set {1..ReplDeg}. Then, a page containing the chosen replica is determined using `CalcIdentifier`, and the registry is queried for that page.

Once the page is obtained, the algorithm verifies the page's integrity. For this, the resolver obtains a signed root of a Merkle hash tree (not show in the algorithm). This hash tree is computed by the registries, for every version number, over all the pages. The resolver verifies the signature of the root (using the standard Web PKI), and verifies that the obtained page is in the tree. This mechanism allows resolvers to use gossiping protocols to ensure that the same hash tree root provided by the queried registry for a specific version is seen by all resolver, and across all registries.

If the verification is successful the page is accepted and updated in/added to the cache. The lookup for the record of interest on the page can be done efficiently by a binary search over the domain name hashes. Communication has to be done over TCP or another reliable transport protocol, given the size of the data returned by the registry (similar to what is done today in DNS for large responses [15]); this has the advantage of preventing reflection and amplification attacks [34]. To complete the lookup and obtain the address of the actual host (Algorithm 1, line 31), the resolver sends an ordinary DNS query directly to the name server whose address was obtained through PageDNS.

2.4 Keeping Pages Updated

Popularity estimation for replication. As explained in Section 2.2, popular domains are replicated on multiple pages, according to their popularity. TLD registries have to determine the approximate popularity for all these domains, and replicate the records across the pages accordingly. To determine the popularity, we assume that a large fraction of the resolvers can authenticate themselves to the registries (possibly through some out-of-band mechanism), and then provide, at regular time intervals, the approximate number of requests received for the most popular domains (using some randomization to hide the exact numbers). Based on the reports by the resolvers, the registries can then assess the overall popularity of the most popular records and update the pages and meta page

accordingly. We point out that strong fluctuation in the popularity are possible (the so-called slashdot effect) and would require updating a high number of pages, which is expensive for the registries. For efficiency we therefore allow registries to consider an averaged popularity, computed for example as a moving average, which obviates the need for rapid and expensive updates of a large number of pages. It is important to note that this comes with some privacy cost for accesses to domains whose popularity has recently increased, which become more identifiable. Similarly, regional differences in popularity can also impact the identifiability of queries.

Page updates and authentication. At regular intervals, TLD registries will issue new versions of the pages which need to change as a consequence of updates, insertions, and removals. To authenticate the updated pages (the new versions), each registry constructs a Merkle tree over all the updated pages, and signs the root. When resolvers query for a page, the registries will also provide the signed root of the tree, as well as a proof (which consists of a list of hashes) that the page is part of the tree. We use a binary tree with the pages sorted according to their identifier (every level determines one additional bit): this makes it impossible for a registry to include two pages with the same identifier.

3 Privacy and Security Analysis

In this section we analytically model the privacy guarantees of PageDNS. We start by identifying the ideal replication degree of every domain across pages, depending on their popularity. We then derive the analytic expression of the probability that an adversary is able to correctly guess the *target domain* (i.e., the domain the client is accessing) depending on the domain's popularity, given that the adversary is able to observe the page requests made by the recursive resolver queried by the client. We use this probability distribution as the main metric for the efficacy of PageDNS, and we analyze the impact of replication, *page fingerprints* (one website access causing multiple PageDNS page requests) and of *cover* page queries (retrieving additional pages to provide extra privacy). Cover page queries also model the fact that the DNS queries by other clients of the same resolver cause additional page requests, as well as the fact that the resolver can autonomously update expired pages when idle.

3.1 Replication

Our goal in PageDNS is to hide a client's target domain, and ideally we would like to hide it independently of its popularity. As discussed in Section 2.2, to hide target domains with high popularity we have to replicate their records across multiple pages. To determine how much each record should be replicated depending on its popularity, we try to optimize for two goals. First, we want to keep the total number of replicas to a minimum. Second, denoting with $\mathcal{I}_x$ the *identifiability* of a domain x, i.e., how easily x can be guessed based on a page

request (we provide a formal definition in Section 3.2), we want to minimize the maximum ratio $\mathcal{I}_x/\mathcal{I}_y$ for any two domains x and y. We solve this problem analytically in Appendix A under the assumption that domain popularity follows a Zipf distribution with parameter s (Jung et al. [23] show that this is the case, and that $s = 0.91$). We obtain the following *replication function* that maps a domain's rank $k \in \{1, \ldots, N\}$ to the replication degree of that domain:

$$r(k) = max\{1, Rk^{-s}\} \tag{1}$$

where R is the maximum replication degree. Note that $r(k) = 1$ means that only one record exists for the domain of rank k. Evidently $R \leq m$, where m denotes the total number of pages; the optimal choice in terms of privacy is $R = m$.

3.2 Identifiability

When accessing a certain domain for web browsing, it is likely that a number of additional domains have to be looked up by the client: web pages contain external content from CDNs, from advertisement providers, or from user tracking sites (e.g., `google-analytics.com` [1]). While some of these might be safely blocked by the browser, others are necessary for correctly displaying a page.

For simplicity in our analysis we consider the case where each domain corresponds to a single website (e.g., this can be the index `www` webpage). We define the *page fingerprint* of a domain as the set of pages which are requested due to an access to the website corresponding to that domain. Different domains may have the same page fingerprint; we also note that, owing to replication, the same target domain may have many different possible fingerprints. Intuitively, the size of a fingerprint and the replication degree of the domains in the fingerprint determine its uniqueness: a relatively unique fingerprint can undermine the protection provided by PageDNS. In this section we investigate how identifiable queries are according to the popularity of the domain, depending on the replication degree, on page fingerprinting, and the amount of cover page queries.

To measure the privacy risk of domain queries, we analytically determine the probability of an adversary correctly guessing a target domain with a certain rank, given that the adversary is able to observe the page fingerprint resulting from the client's access to the target domain. We assume that domains are distributed according to a perfect Zipf distribution with parameter $s = 0.91$ [23]. This implies that every domain has a unique rank, and we will therefore often use the rank of a domain to refer to the domain itself. For instance, we use (lowercase) k to indicate a specific domain of rank k, where the set of possible values of k is $\mathcal{K} = \{1, \ldots, N\}$.

We define a random variable K indicating the rank of a domain chosen according to the Zipf distribution. We also define a stochastic process F that maps each domain k to its possible fingerprints. More precisely, F maps each domain k to a random variable that has as possible values all the sets of pages that can be k's fingerprint—we assume that for any replicated domain the resolver chooses one of the possible pages uniformly at random. We will also, as a slight abuse

of notation, consider the application of F to the random variable K, $F(K)$: this represents another stochastic process, the possible outcomes of which are determined by first drawing a domain k from K, and then applying F to k. Finally, we also consider a random variable $T = T(k)$, which represents the choice of the cover page queries when querying for domain k.[4] For ease of notation we will often write the argument of the stochastic processes as subscript, e.g., F_K for $F(K)$ or T_k for $T(k)$.

We can now provide the definition of the *identifiability* of domain k, which denotes the probability of the adversary correctly guessing k having observed one of k's fingerprints.

Definition 1 (Identifiability). *We define the identifiability of a domain k as follows:*

$$\mathcal{I}_k = \Pr(K = k \mid F_K \cup T_K = F_k \cup T_k) \tag{2}$$

Intuitively, this models a rational adversary that has no prior information about the preferences of the client. The adversary observes a set of page requests coming from a resolver, and assumes that they are due to an access to an unknown domain K chosen by the client according to the Zipf distribution. The adversary then determines, for all $k' \in \mathcal{K}$, the probability that $K = k'$ given the observed set of pages. This probability for $k' = k$ (where k is the domain actually accessed by the client) is the identifiability of k.

We now show how the identifiability can be expressed in a form that allows us to compute it. For the definition of the basic notation see Section 3.2. First, we apply Bayes theorem.

$$\begin{aligned} \mathcal{I}_k &= \Pr(K = k \mid F_K \cup T_K = F_k \cup T_k) \\ &= \frac{\Pr(F_K \cup T_K = F_k \cup T_k \mid K = k)\Pr(K = k)}{\sum_{k' \in \mathcal{K}} \Pr(F_K \cup T_K = F_k \cup T_k \mid K = k')\Pr(K = k')} \end{aligned} \tag{3}$$

From Appendix A, $\Pr(K = k) = f(k)$ (Zipf distribution). We can rewrite Equation 3 as follows, where for a random variable X we use notation X' to indicate another random variable with the same distribution.

$$\mathcal{I}_k = \frac{\Pr(F_k \cup T_k = F'_k \cup T'_k) f(k)}{\sum_{k' \in \mathcal{K}} \Pr(F_{k'} \cup T_{k'} = F_k \cup T_k) f(k')} \tag{4}$$

Denoting with A the event $F_k \cup T_k = F'_k \cup T'_k$ and with B the event $F_{k'} \cup T_{k'} = F_k \cup T_k$ (for $k' \neq k$), we rewrite the equation as follows.

$$\mathcal{I}_k = \frac{\Pr(A) f(k)}{\Pr(A) f(k) + \sum_{k' \in \mathcal{K} \setminus \{k\}} \Pr(B) f(k')} \tag{5}$$

If we consider random variable $L_k = F_k \cup T_k$, it can be seen that the values it assumes (sets of pages) are all equiprobable, since all pages in F_k are chosen

[4] The pages in T are chosen uniformly at random; the only dependency that T has from k is for its size. For instance, $|T|$ may be chosen such that the total number of page requests is higher than or equal to a given minimum.

uniformly at random among the possible replications of each domain, and the cover pages in T_k are chosen uniformly at random among the remaining pages. Therefore, denoting with $d(X)$ the possible values (range) of a random variable X, we have that $\Pr(A) = 1/d(L_k)$. With the assumption that the size of the fingerprint is equal to constant q for all domains, and assuming also a constant number of cover pages t, we can rewrite the probability as follows.[5]

$$\Pr(A) = \frac{1}{d(L_k)} = \frac{1}{r(k)^q m^t} \tag{6}$$

Note that $r(k)$ is the replication degree of k (Equation 1). There is an important assumption behind this equation, which is that all pages in the fingerprint of a domain have the same popularity as the domain itself. We call this *popularity inheritance*, and the rationale behind it is that if a domain is very popular and requires access to another domain, then the other domain will be requested at least as often. However, this means that for non-popular domains we might be overestimating the identifiability, since non-popular domains may very well include contents from popular domains. We leave it to future work to make the calculation of the identifiability with fingerprints for unpopular domains more realistic.

To compute probability $\Pr(B)$, we note that it is actually independent of the value of k', as long as $k' \neq k$. It can be shown[6] that $\Pr(B)$ is simply equal to the probability of guessing a randomly chosen set of pages of size $q + t$ (lottery-winning probability). We rewrite the probability as follows.

$$\Pr(B) = \frac{1}{\binom{m}{q+t}} \simeq \frac{1}{m^{q+t}/(q+t)!} = \frac{(q+t)!}{m^{q+t}} \tag{7}$$

Finally we rewrite Equation 5 as follows:

$$\mathcal{I}_k \simeq \frac{\frac{1}{r(k)^q m^t} f(k)}{\frac{1}{r(k)^q m^t} f(k) + (1 - f(k)) \frac{(q+t)!}{m^{q+t}}} \tag{8}$$

Limitations of the identifiability metric. Because of how the identifiability is defined, our results are in a sense *averaged* over all possible assignments of records to pages (i.e., over all possible hash functions). This means that in a concrete instantiation, there might be pages which are worse for privacy. In Appendix B we determine the distribution of the number of ordinary records and of the number of replicas per page. Our results show that the number of ordinary records will in the worst case be less than 10% below the average, which would not significantly impact the identifiability. However, the number of replicas will be 7 at

[5] We are slightly approximating the exact value in Equation 6, ignoring the fact that the pages in T are chosen from the set of all m pages *excluding* those that are already part of the fingerprint.

[6] To formally show this step, one needs to average out the probability over all possible replica sets that could be assumed by all domains, i.e., over all possible hash functions (or all possible sets of domain names of size N).

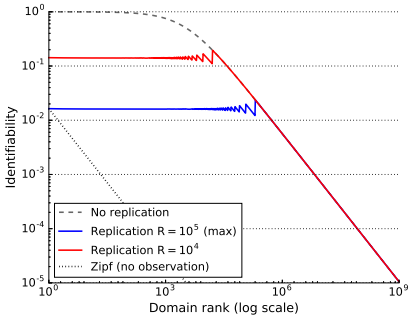

Fig. 3: Identifiability of domains according to their rank, depending on the maximum replication allowed. The figure also shows the Zipf distribution, which is equal to the identifiability *prior* to any observation of pages by the adversary. We point out that the sawtooth pattern is due to rounding in the replication function.

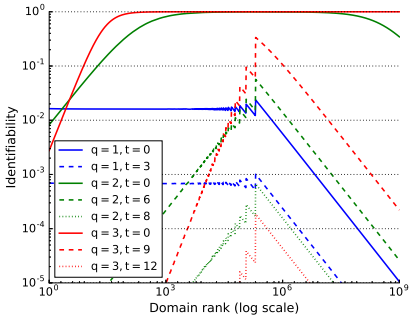

Fig. 4: Identifiability of domains according to their rank, depending on the size of the fingerprint q and on the number of cover queries t. As can be seen, fingerprints can be highly effective for identifying a website access. However, with a number of cover queries between twice and three times the size of the fingerprint, strong privacy can still be guaranteed.

the minimum with high probability, the median being 25, so this could have an impact: in particular, a page with few replicas would cause higher identifiability for the popular records on it (closer to the cases with little or no replication in Figure 3). For the non-replicated records, the worst case happens for a somewhat popular record to be on a page with few replicas and few other records of similar or higher popularity. Even in such an (unlikely) case, the replicas alone will still provide privacy protection.

3.3 Results

We use identifiability (Definition 1) as a metric to measure the effectiveness of PageDNS, showing in particular how the *identifiability curve* (obtained from all possible values of k) varies depending on the replication degree, on the use of cover page queries, and on page fingerprinting.

Identifiability and replication. Considering the basic scenario without fingerprinting or cover queries, we study the effectiveness of the basic mechanism of PageDNS. Figure 3 shows the identifiability curves for two maximum replication degrees, $R = 10^4$ and $R = m = 10^5$, the latter being the highest possible replication degree where the most popular record is replicated on all m pages. The figure also shows for comparison the case where no replication is used, and it can be seen how replication is indeed able to achieve its goal of hiding requests to popular domains. We have plotted also the prior knowledge of the adversary, i.e., the identifiability of domains when the adversary does not observe the requested pages (this is simply the Zipf distribution).

	Min	Median	95th	Max
considering all SLDs	1	12	47	238
without top 100	1	4	24	211
without top 1,000	1	2	8	172

Table 1: Distribution of the fingerprint size q as observed loading 10,971 web-pages. The total number of SLDs seen over all page loadings is 20,777. The table shows how most domains in the fingerprints are common (e.g., advertisement, analytics, social media) by showing how the fingerprint size is reduced when we exclude the most common 100 or 1,000 SLDs.

Fingerprinting and cover queries. In Figure 4 we show the impact on identifiability of page fingerprinting and of cover queries, considering small fingerprint sizes. We consider a fingerprint size of q, including the page of the main domain, and an amount of cover queries t. As can be seen, without cover queries the use of page fingerprinting by an adversary can be very effective, leading to complete privacy loss in many cases. Fortunately, we find that cover queries are sufficient to compensate for this loss. In particular, an amount of cover queries of three times the size of the fingerprint appears to be enough to provide a privacy level lower than the basic one obtained for $q = t = 0$ (even for $q > 3$, which is not shown in the figure).

We have also analyzed the size of fingerprints in practice: we have logged all SLDs that appear in HTTP GET/POST requests for 10,971[7] out of the 20,000 most popular domains (according to Alexa [35]) by automatically loading all these pages in a browser and relaying all requests made by the browser through a custom proxy. This gave us a list of unique SLDs for each domain loaded by the browser.

The results, reported in Table 1, show that indeed many websites require external content from a large number of domains, but also that many of these domains are common across different websites. Indeed, if we discard the most popular 100 SLDs, the median number of additional DNS queries performed is 4, and it drops to only 2 when discarding the top 1,000. Still, there appear to be certain websites which require an exceptionally large number of SLD lookups. While we expect that in most cases these accesses could be hidden due to the large number of queries being constantly performed by the resolver of a medium-large ISP, and due to the fact that a number of page requests can be avoided as fresh page copies are still in the resolver's cache, we cannot in general provide strong guarantees for such websites. To be secure, clients would need to be made privacy-aware, and restrict the number of SLD queries per page (or perhaps space them over a longer period of time). We leave a more detailed investigation of these possibilities to future work.

[7] The number of accessed domains is almost half of the 20,000 we consider: this is because many of them did not have a `www` host, and also due to some restrictions we imposed on the loading time.

Total	Min	Max	Mean	Median	95th
16205	1	32	5.40	4	30

Table 2: Number of changes of authoritative name servers for the 3000 domains that had at least one change over the monitored 25 days.

3.4 Security Against Active Attacks

In previous sections we have analyzed the privacy guarantees of PageDNS with the assumption that the adversary is able to see the page queries made by recursive resolvers, but does not perform any active attack (i.e., deviating from the protocol). However, it is possible that an adversary may try to improve its ability to identify the domains accessed by a client by the use of active attacks. In particular, the adversary could modify the records for some domains he wishes to monitor to point to a honeypot server under his control: he could use a different server for every connecting resolver, and would therefore be able to link accesses to one of honeypot servers to a resolver, revealing that one of the resolver's clients has an interest in the monitored domain (this type of attack is sometimes called *split world attack*).

In PageDNS this attack is prevented by requiring registries to authenticate every new set of page versions they create through a Merkle hash tree, and by having resolvers gossip about the roots of the trees obtained from all the registries that they contact. Additionally, domain owners can monitor the pages distributed by PageDNS to ensure that the information contained in them is correct. This public auditability property also provides significantly higher integrity guarantees than those achieved in plain DNS. Since we assume that the adversary wishes to avoid detection, this scheme ensures that active attacks of this kind are prevented.

4 Evaluation

In this section we present an evaluation of the computational overhead for maintaining the PageDNS, as well as the memory overhead for registries using PageDNS.

4.1 Cost of Maintaining the PageDNS

Frequency of Authoritative NS changes. Records in PageDNS pages only contain name servers of SLDs (Section 2.2) to limit the total number of records in PageDNS, but also to ensure that pages will not change too frequently. The frequency of page changes affects both TLD registries (cost of creating the updated pages) and resolvers (cost of downloading incremental updates in order to keep the local cache updated).

To evaluate how frequently authoritative name servers of SLDs change, we monitored the authoritative name servers of the 100,000 most popular domains (according to the Alexa Top 1M domains list [35]) over 25 days in July 2017. Out of the 100,000 monitored domains, we could resolve the authoritative name server for 75,622 domains[8]. 72,622 of these, or 96%, did not change their authoritative name servers over the 25 days. For the remaining 3,000 (4%) domains, Table 2 shows statistics about the number of name server changes. From these results we can calculate the expected number of changes C to the name servers for each domain per day:

$$E[C] = \frac{16205}{75622} \cdot \frac{1}{25} \approx 0.0086 \tag{9}$$

Thus, there are expected $10{,}000 \cdot 0.0086/24 \simeq 3.6$ updates per page per hour. If new page versions are created, e.g., every 4 hours, less than 15 records would need to be changed per page on average between two versions. Out of our monitored domains which where updated, only around 5% had, right before the change, a TTL lower than 4 hours, so only a small number of domains might suffer from higher inconsistencies than with DNS's caching. Furthermore, for all planned updates, domain owners could schedule an update with their registrar, ensuring that the update will be included by the registry at a specified version, at a specified time.

Update Costs per Registry. According to Verisign's Domain Name Industry Brief [37] (cf. also ICANN's monthly report for `.com` [22]), the DNS has reached a size of 335M domain names across all TLDs with an increase of about 15 million per year over the last few years (∼5%). This is considering only the higher level domains to which TLDs delegate, e.g., `example.org` or `example.co.uk`, excluding subdomains like `cs.example.com`. We therefore set, for our analyses in this paper, the number of SLDs $N = 10^9$.[9]

Given the total number of SLDs and the update frequency we calculated above (Eq. 9), we find that the TLD registries need to perform, overall, around 100 page updates per second, assuming as above that each page is updated every 4 hours. We expect this to be well within the capacity of TLD registries; furthermore, we note that if PageDNS were widely used, the root name servers would see a significantly reduced query load, meaning that their space resources could also be spent to assist the registries (this would be particularly easy in cases where the same company manages both some root name servers and TLDs, which is the case for instance for Verisign). Still, we assess the feasibility of these update with our prototype implementation in Section 4.3, and find that even with low-end hardware this update frequency can be sustained.

[8] The reason almost 25% of domains were not resolved is that for our monitoring we kept low timeouts, and excluded the domains which frequently resulted in time-outs.

[9] With a growth rate of ∼5% the number of domains and thus the number of pages doubles approximately every 14 years. We expect that the available bandwidth and computing power can easily keep up with the growth of PageDNS.

4.2 Memory Overhead for Resolvers

We assume that resolvers will locally cache the set of all pages, corresponding to all records in the TLD zones. Given our assumptions (Section 4.1) of 10^9 SLD records, distributed across 10^5 pages of about 1 MB of size each, we have that the total storage requirement for a resolver is 100 GB (excluding optimizations such as compression). While this could entail non-negligible upgrading costs for the ISPs managing the resolvers, in particular for ISPs of small size, we believe that by the time PageDNS would reach widespread deployment, the cost of this memory upgrade would be bearable even for smaller ISPs. Initially, ISPs will still have an incentive for adoption as PageDNS would allow them to provide a privacy-preserving lookup service to their clients, and it would in all likelihood also be a faster lookup service than in todays DNS, since a significant fraction of queries made by clients would be for up-to-date pages, and thus the resolver can directly query the SLD name server.

4.3 Prototype implementation

We have implemented a prototype of PageDNS in Python to obtain some preliminary performance results and assess the feasibility of our system. Our code defines both a TLD registry and a resolver, for a total of almost 6K LOCs. Our evaluation of this prototype was made running a registry instance and a resolver instance on two Amazon AWS instances (in Ireland and in Germany, respectively), each with an Intel Xeon CPU with 2.53GHz and 2GB of RAM.

The registry is implemented as a server providing pages over HTTP using a RESTful API. The pages are represented as plain text for human readability, and for our evaluation we have not implemented optimizations to compress the size of pages. In our implementation, the registry lazily computes incremental updates between the cached version of the resolver, specified in the query, and the last available version. These incremental updates are cached by the registry until the new page versions are generated.

We use this setting to evaluate the latency of a page query, both in the case of cold cache, in which the entire page has to be downloaded (this should happen only in exceptional cases), and in the case of the download of an incremental update. The results are averaged over 1000 queries. The time to download an entire page consisting of 10,000 records is 789ms on average, while for an incremental update of one version the required time was 41ms. This last value is in the same order of magnitude as a request to Google Public DNS for NS records.

We have also implemented and evaluated the page-updating functionality of registries, offering a RESTful API for domain owners to communicate their updates to the registry. The time needed for one page update was on average 68ms, meaning that a registry can perform around 14 page updates per second using one low-end AWS instance. In practice, we expect registries to deploy significantly more powerful machines. We leave a more comprehensive evaluation of the registries' performance using PageDNS to future work.

5 Related Work

In response to the revelations about the NSA's mass surveillance programs [2], the IETF took a stance considering such surveillance practices as an attack [17], and began analyzing the problem of how to defend against it [6]. One of the identified threat vectors is DNS [8], but the countermeasures proposed so far are relatively weak. The simplest (but also weakest) of these proposals calls for query minimization [9], which would only hide some information from root and TLD servers.[10] Another proposal (a now expired IETF draft) aims to extend DNS with the option to encrypt queries between the recursive resolver and the authorities [39]. Recently also an academic proposal by Zhu et al. called T-DNS [41] was submitted as an RFC [21]: their suggestion is to use TLS between clients and recursive resolvers (and possibly between recursive resolvers and authorities) to protect against network eavesdroppers.

Outside the IETF other solutions have been devised which are similar in scope. DNSCurve [7] allows clients or recursive resolvers to establish secure channels to the authoritative resolvers using efficient cryptography. A related system is DNSCrypt [13], which offers similar guarantees. Both of these systems have seen some adoption, e.g., they are supported by OpenDNS [3]. These systems, as well as the RFCs currently under examination at the IETF, are easily deployable, and could be used complementarily to PageDNS, since their threat model is orthogonal to ours.

Other researchers aimed to protect against stronger adversaries. Zhao et al. [40] suggest a simple approach called *range queries*, which consists in the client sending extra "dummy" queries to the resolver, in order to hide the real query. Federrath et al. [18], however, show that range queries are vulnerable to semantic intersection attacks. In this same paper, the authors propose another system, based on a combination of broadcasting and sending the queries over an anonymous communication system (ACS): popular records are broadcast to all clients, while in order to retrieve less popular entries the clients have to query a resolver through a mixnet or an onion routing system. It is unclear however how these systems can effectively guarantee privacy against a malicious ISP, which will inevitably see the communications following the lookup, thus apparently nullifying the efforts to anonymize the queries. We believe that, to protect against a malicious ISP, an ACS necessarily has to be used.

Our goal in PageDNS of hiding the information of what records are being queried is analogous to that of private information retrieval (PIR) [10,25], which leverages either multiple non-colluding servers or computationally expensive cryptography to significantly reduce communication costs. Unfortunately, even with recent improvements [14,4], PIR has remained too costly in terms of computation to be used for a critical application like DNS. In PageDNS we do not try to trade off lower communication costs for additional computation: instead, we show how domain-specific aspects, such as the low variability of TLD

[10] It appears that Verisign, Inc. was able to obtain a patent [28] on this technology, and it is unclear what this will mean for its adoption.

zones, paired with extensive caching, allow us to achieve privacy properties close to those of PIR, but without its prohibitive performance penalties. Another related direction regarding weak PIR was taken by Toledo et al. [36], who shown how privacy guarantees can be increased through the use of anonymous communication systems, or by leveraging multiple servers.

Other related projects aim to push DNS entries to multiple entities, such as recursive resolvers, across the Internet. For instance, Cohen and Kaplan [12] propose proactive caching of records, and similarly Handley and Greenhalgh [20] also advocate for pushing records to thousands of name servers for higher robustness. Kangasharju and Ross [24] take an even more radical stance, proposing a new design for DNS involving distributed servers storing the complete DNS database. This is perhaps the closest to PageDNS, although without the goal of privacy. However, in our system we put a much stronger emphasis on efficiency and deployability: in particular, PageDNS pages only contain TLD zones, not the entire DNS database, which would be orders of magnitude larger, and change more frequently.

Researchers have also investigated new approaches to name resolution that are fundamentally different from DNS, based on distributed architectures that are not structured hierarchically, which can provide privacy. One such approach by Lu and Tsudik [27], called PPDNS, adds privacy on top of CoDoNS [33], an alternative naming system based on distributed hash tables (DHTs). The scheme also uses computational PIR to reduce communication overhead, but the ensuing computational costs strongly limit the size of the range and thus also the privacy guarantees, and leave the system vulnerable to denial of service attacks. The GNU Name System (GNS) [38,19] is another scheme based on DHTs, but because it uses a fully peer-to-peer approach it does not provide global naming consistency, and is thus quite different from today's DNS. Pappas et al. [32] have analyzed more generally DHT-based designs for DNS, and arrived at the conclusion that compared to the current DNS they are inferior in terms of performance and availability, except in terms of protection against some specific denial-of-service attacks.

6 Conclusions

We explore the design space of the solutions to the scarcely studied problem of privacy-preserving DNS lookups, and we identify a yet unexplored but promising direction. We propose an architecture, PageDNS, which aims to hide query information from root name servers and TLD registries. PageDNS lets TLD registries group together the name server records in their zones into pages; recursive resolvers retrieve entire pages rather than single records, which provides a first level of privacy protection. Additionally, we design a number of optimizations and enhancements to make the architecture more efficient, such as full caching of pages at the resolver, and incremental updates, which reduce the overhead. PageDNS requires significant changes to resolvers and TLD registries, and a certain memory overhead for resolvers, but it provides privacy properties close to

those of PIR, and it may even speed up the average DNS query, since effectively the resolvers will be caching all TLD zones. Furthermore, name is incrementally deployable by TLD registries, and does not need to be adopted by all resolvers. Since PageDNS is orthogonal to other privacy solutions, it can be combined with other approaches to achieve different tradeoffs in efficiency and privacy. These are interesting directions for future work.

References

1. Google Analytics Solutions. `https://www.google.com/analytics`. Retrieved on Sept. 22, 2017.
2. NSA spying on americans. "`https://www.eff.org/nsa-spying`". Retrieved on Sept. 22, 2017.
3. OpenDNS. `https://www.opendns.com/`. Retrieved on Sept. 22, 2017.
4. Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. In *PETS*, 2016.
5. R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, 2005.
6. Richard Barnes, Bruce Schneier, Cullen Jennings, Ted Hardie, Brian Trammell, Christian Huitema, and Daniel Borkmann. Confidentiality in the face of pervasive surveillance: a threat model and problem statement. RFC 7624, 2015.
7. Daniel J. Bernstein. DNSCurve: usable security for DNS. `https://dnscurve.org/`. Retrieved on Sept. 22, 2017.
8. Stéphane Bortzmeyer. DNS privacy considerations. RFC 7626, 2015.
9. Stéphane Bortzmeyer. DNS query name minimisation to improve privacy. RFC 7816, 2016.
10. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *IEEE FOCS*, 1995.
11. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6), 1998.
12. Edith Cohen and Haim Kaplan. Proactive caching of DNS records: Addressing a performance bottleneck. In *IEEE/IPSJ International Symposium on Applications and the Internet (SAINT)*, 2001.
13. Frank Denis and Yecheng Fu. DNSCrypt. `https://dnscrypt.org/`, 2011. Retrieved on Sept. 22, 2017.
14. Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally Robust Private Information Retrieval. In *USENIX Security*, 2012.
15. John Dickinson, Sara Dickinson, Ray Bellis, Allison Mankin, and Duane Wessels. DNS Transport over TCP - Implementation Requirements. RFC 7766, 2016.
16. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.
17. Stephen Farrell and Hannes Tschofenig. Pervasive monitoring is an attack. RFC 7258, 2014.
18. Hannes Federrath, Karl-Peter Fuchs, Dominik Herrmann, and Christopher Piosecny. Privacy-preserving DNS: Analysis of broadcast, range queries and mix-based protection methods. In *ESORICS*, 2011.
19. Christian Grothoff, Matthias Wachs, Monika Emert, and Jacob Appelbaum. NSA's MORECOWBELL: knell for DNS. Technical report, GNUnet e.V., 2015.

20. Mark Handley and Adam Greenhalgh. The Case for Pushing DNS. In *HotNets*, 2005.
21. Si Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, 2016.
22. ICANN. .com Monthly Registry Reports. `https://www.icann.org/resources/pages/com-2014-03-04-en`. Retrieved on Sept. 22, 2017.
23. Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS performance and the effectiveness of caching. *IEEE/ACM Transactions on Networking*, 10(5), 2002.
24. Jussi Kangasharju and Keith W. Ross. A replicated architecture for the Domain Name System. In *IEEE INFOCOM*, 2000.
25. Eyal Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *IEEE FOCS*, 1997.
26. Ben Laurie, Geoffrey Sisson, Roy Arends, and David Blacka. DNS security (DNSSEC) hashed authenticated denial of existence. RFC 5155, 2008.
27. Yanbin Lu and Gene Tsudik. Towards plugging privacy leaks in the domain name system. In *IEEE P2P*, 2010.
28. Danny McPherson and Eric Osterweil. Providing privacy enhanced resolution system in the domain name system. US Patent 8,880,686 B2, 2014.
29. Paul Mockapetris. Domain names – concepts and facilities. RFC 1034, 1987.
30. Paul Mockapetris. Domain names – implementation and specification. RFC 1035, 1987.
31. Rafail Ostrovsky and William E. Skeith, III. A Survey of Single-Database PIR: Techniques and Applications. In *PKC*, 2007.
32. Vasilcios Pappas, Dan Massey, Andreas Terzis, and Lixia Zhang. A comparative study of the DNS design with DHT-based alternatives. In *IEEE INFOCOM*, 2006.
33. Venugopalan Ramasubramanian and Emin Gün Sirer. The design and implementation of a next generation name service for the Internet. In *ACM SIGCOMM*, 2004.
34. Christian Rossow. Amplification hell: revisiting network protocols for DDoS abuse. In *NDSS*, 2014.
35. Alexa the Web Information Company. Alexa top 500 global sites. `http://www.alexa.com/topsites`, 2016.
36. Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost ε-private information retrieval. *PoPETS*, (4), 2016.
37. Verisign, Inc. The domain name industry brief. 14(2), 2017. `https://www.verisign.com/assets/domain-name-report-Q12017.pdf`. Retrieved on Sept. 22, 2017.
38. Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. A censorship-resistant, privacy-enhancing and fully decentralized name system. In *International Conference on Cryptology and Network Security (CANS)*, 2014.
39. Wouter Wijngaards and Glen Wiley. Confidential DNS. Internet Draft draft-wijngaards-dnsop-confidentialdns-03, 2015.
40. Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. Analysis of privacy disclosure in DNS query. In *International Conference on Multimedia and Ubiquitous Engineering (MUE)*, 2007.
41. Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-oriented DNS to improve privacy and security. In *IEEE Symposium on Security and Privacy*, 2015.

A Replication Function

Let $\mathcal{P}$ be a page of PageDNS containing n records, and let k' be the rank of a record in $\mathcal{P}$. For ease of notation, we write $k' \in \mathcal{P}$, and in general we will often use a domain's rank to refer to the domain. We assume a total of N domains, thus $k' \in \{1, \ldots, N\}$, where $k' = 1$ is the highest rank. We consider a random variable K indicating the rank of a domain chosen at random (by a generic client) according to a Zipf distribution with parameter $s = 0.91$, i.e., such that:

$$\Pr(K = k) = f(k; s, N) = \frac{1}{k^s H_{N,s}} \quad \text{with } H_{N,s} = \sum_{k=1}^{N} \frac{1}{k^s} \tag{10}$$

To simplify the notation, we will write $f(k)$ to mean $f(k; s, N)$ and H to mean $H_{N,s}$.

Now we try to analytically express the probability that an adversary would assign to k' being the target domain having observed a request to $\mathcal{P}$, which is equal to the probability that $K = k'$ given that K is restricted to $\mathcal{P}$. By applying Bayes theorem, we obtain the following equation:

$$\Pr(K = k' \mid K \in \mathcal{P}) = \frac{\Pr(K \in \mathcal{P} \mid K = k') \Pr(K = k')}{\sum_{k \in \mathcal{P}} \Pr(K \in \mathcal{P} \mid K = k) \Pr(K = k)} \tag{11}$$

Probability $\Pr(K \in \mathcal{P} \mid K = k)$ is equal to 1 if k is not replicated, since we are assuming that $k \in \mathcal{P}$. More generally, if k has a replication degree of $r(k)$ (i.e., the record for domain k exists on $r(k)$ pages), then the probability of choosing the replica in $\mathcal{P}$ is $1/r(k)$. We can therefore rewrite Equation 11 as follows:

$$\Pr(K = k' \mid K \in \mathcal{P}) = \frac{f(k')/r(k')}{\sum_{k \in \mathcal{P}} f(k)/r(k)} \tag{12}$$

Now let k'' be another domain on the same page, i.e., $k'' \in \mathcal{P}$. Ideally, we would like the replication function to be such that $\Pr(K = k' \mid K \in \mathcal{P}) = \Pr(K = k'' \mid K \in \mathcal{P})$ for all possible choices of k' and k''. Unfortunately, it is possible to see that the only scenario where this could theoretically be achieved is one where the number of pages is equal to the number of domains, and the cost of replication would be excessive (the total size of all PageDNS pages would increase by almost a hundredfold). Instead, we try to get the ratio of those probabilities as close to 1 as possible. Since we also want to minimize the cost of replication, we do not replicate the least popular domain (i.e., $r(N) = 1$): replication should only help to reduce the probability in Equation 11 for high-rank domains, to get it closer to the probability of the more unpopular domains. It is reasonable therefore for the ratio to be at its maximum when $k' = 1$ and $k'' = N$.

$$\rho_{MAX} = \frac{\Pr(K = 1 \mid K \in \mathcal{P})}{\Pr(K = N \mid K \in \mathcal{P})} = \frac{f(1)/r(1)}{f(N)/r(N)} \tag{13}$$

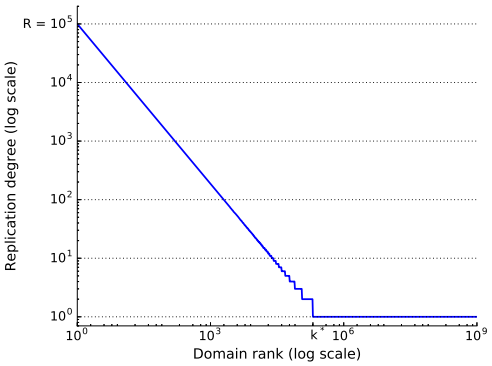

Fig. 5: Replication degree of domain names according to their rank.

Denoting with R the replication degree of the most popular domain ($r(1) = R$), and since $r(N) = 1$, Equation 13 becomes the following:

$$\rho_{MAX} = \frac{f(1)/R}{f(N)} = \frac{H^{-1}/R}{N^{-s}H^{-1}} = \frac{N^s}{R} \tag{14}$$

All other domains should be replicated in order not to increase this ratio further. From this requirement, we obtain the following bound $\forall k$.

$$\frac{\Pr(K = k \mid K \in \mathcal{P})}{\Pr(K = N \mid K \in \mathcal{P})} \leq \rho_{MAX} \tag{15}$$

$$\implies \quad \frac{f(k)/r(k)}{f(N)/r(N)} = \frac{k^{-s}H^{-1}/r(k)}{N^{-s}H^{-1}} = \frac{N^s}{k^s r(k)} \leq \frac{N^s}{R} \tag{16}$$

$$\implies \quad r(k) \geq \frac{R}{k^s} \tag{17}$$

We derived the bound in Equation 17 for the worst case of the page containing both the most and the least popular domains, so by applying it generally to the replication for all k-s we ensure that on no page there will be two domains for which the ratio of their identification probabilities (Equation 11) exceeds ρ_{MAX}. Furthermore, with the approximation that the denominator in Equation 12, $\sum_{k \in \mathcal{P}} f(k)/r(k)$, has the same value for all pages, the bound in Equation 17 actually guarantees the following for any two pages $\mathcal{P}$, $\mathcal{P}'$:

$$\forall k \in \mathcal{P}, \forall k' \in \mathcal{P}' \quad \frac{\Pr(K = k \mid K \in \mathcal{P})}{\Pr(K = k' \mid K \in \mathcal{P}')} \leq \rho_{MAX} \tag{18}$$

Since we desire to minimize the cost of replication, we try to match the bound of Equation 17 as closely as possible (rounding it to the nearest integer), with the additional constraint that replication of any domain be at least 1. Thus the replication function we use is the following:

$$r(k) = max\{1, round(Rk^{-s})\} \tag{19}$$

In Figure 5 we plot the function. Note how only the most popular domains with rank from 1 to k^* are replicated: these will all have approximately (because of rounding) the same identification probability, while for less popular domains the probability will be lower. We also point out that in our scenario $R \leq m$, where m is the number of pages, and that the best (lowest) probabilities are obtained for the equality: in this case, we have the most popular domain replicated on all pages. For realistic values ($m = 10,000$ and $s = 0.91$), we obtain $k^* \simeq 200,000$.

B Page-Size Variance

We can think of the size of a page P as the sum of N random variables X_k, each assuming value 1 if the k-th domain is assigned by the hash function to page P, and value 0 otherwise. The size of page P is thus $X = \sum_{k=1}^{N} X_k$. Assuming that the hash function behaves as a random function, and considering a set of m pages, we can easily compute the expected value of the size of the generic page P as follows:

$$\mu = \mathrm{E}\left[\mathrm{X}\right] = \sum_{k=1}^{N} \mathrm{E}\left[\mathrm{X_k}\right] = \sum_{k=1}^{N} \frac{1}{m} = \frac{N}{m} \tag{20}$$

Since X is the sum of independent random variables with values in the set $\{0, 1\}$, we can apply the multiplicative Chernoff bound to estimate the probability that the size of a specific page will deviate from the expected value μ by a certain factor $(1 - \delta)$ (we aim to find a lower bound). The bound has the following form.

$$\Pr(X \leq (1 - \delta)\mu) \leq e^{-\frac{\delta^2 \mu}{2}} \tag{21}$$

Considering for the parameters the values $N = 10^9$ and $m = 10^5$, as we have done throughout the paper, we obtain from Equation 20 that $\mu = 10^4$. Setting $\delta = 0.1$ for a deviation of at least 10% from the mean, Equation 21 yields the following bound:

$$\Pr(X \leq 0.9\mu) \leq e^{-\frac{10^{-2} 10^4}{2}} = e^{-50} \simeq 2 \cdot 10^{-22} \tag{22}$$

We see from these numbers that the probability of having pages significantly smaller than the average is clearly negligible. Another Chernoff bound can be used to find similar limitations for the probability of pages to be 10% larger than the average.

Replicas distribution. To determine the distribution of the number of replicas per page, we find that Chernoff bounds are not effective, as they do not allow us to rule out extreme cases such as having only 2 or 3 replicas on some page. Instead, we use a simulation over 10^6 pages, and find that the median is 25 records per page, though it can be as low as 7 in exceptional cases.