Lightweb: Private web browsing without all the baggage

Emma Dauterman
UC Berkeley

Henry Corrigan-Gibbs
MIT

Abstract. This paper proposes lightweb, a new system for private browsing. A lightweb client can browse a web of text-based pages without revealing to anyone—not the network, not the servers hosting the pages—which pages it is reading. Unlike Tor and other anonymizing web proxies, which are inherently vulnerable to traffic-analysis attacks, lightweb's design protects against traffic-analysis attacks by design. While lightweb is expensive in relative terms (hundreds of core-seconds of server computation per page load), we show with microbenchmarks that the total system cost can be inexpensive in absolute terms (comparable to the cost of a Netflix membership). This paper does not present a polished system, but instead aims to spark discussion on radical approaches to a privacy-first web.

1 Introduction

Each time we fetch a piece of information over the web, we leave an indelible digital trace of our activity: the search engine logs our queries, the DNS resolver logs our requests, the ISP logs our TCP flows, the CDN logs our HTTP headers, the routers along the way log our packets, and so on. These logs are ripe for abuse by corporations, governments, and attackers [1–4].

The dominant strategy for bringing privacy to web browsing is to route a client's traffic through an *anonymizing proxy*, such as Tor [23]. The tremendous benefit of these systems is that they are backwards compatible with the web: a client can essentially browse any existing website via Tor.

At the same time, anonymizing proxies have two major limitations. The first is that they are vulnerable to correlation attacks that let an attacker (e.g., at an Internet-exchange point) determine which user is visiting which website [40]. Second, even if an attacker cannot identify the precise destination of a particular user's flow, the attacker can use low-cost traffic-analysis attacks to determine what a user is watching or reading [49]: without extreme amounts of cover traffic, a visit to the media-rich *New York Times* homepage—even over an encrypted link—exhibits a very different traffic signature than a visit to an article page [31].

HotNets '23, November 28–29, 2023, Cambridge, MA, USA

ACM ISBN 979-8-4007-0415-4/23/11.
https://doi.org/10.1145/3626111.3628207

We propose a more radical approach to private web browsing that addresses both shortcomings of anonymizing proxies. Our idea is to sacrifice backwards compatibility with today's web, for the sake of gaining strong (i.e., rigorous, cryptographic) protection against deanonymization attacks [43, 44] and most traffic-analysis attacks.

We center our design around a single "private-`GET`" operation, which lets a client fetch a short, fixed-size named blob of data from a server. The key security property that the private-`GET` operation provides is that *no one—not the network, not the server—learns which data blob the client received*. We envision this private-`GET` operation as being implemented in a new application-layer protocol, the *zero-leakage transfer protocol* (ZLTP), and we show that it is possible to implement this protocol using either private-information-retrieval protocols (providing cryptographic security guarantees, optionally with the assumption that two servers do not collude) or hardware enclaves (which rely on hardware protections for security).

The key challenge here is making private-`GET` operations cheap, even when a single ZLTP server is serving a dataset of hundreds of millions of data blobs. One barrier is that cryptographic private-information-retrieval protocols require per-request computation that scales linearly with the dataset size [10]. While a per-request linear scan over a gigantic dataset seems absurd on its face, we run microbenchmarks to estimate that supporting datasets of 305 GiB or more (the compressed size of the cleaned C4 version of the Common Crawl data set [45, 46]) is possible at the cost of less than one cent per request, or 204 core-seconds of parallelizable computation.

Given ZLTP as a communications substrate, we demonstrate how to build "lightweb," an interactive web-like system for fully private browsing. A user can browse a deployment, consisting of hundreds of millions of lightweb pages or more, *without revealing to anyone—not the network, not the lightweb servers*—which pages the user is visiting. Unlike today's web, lightweb pages are of fixed size and are extremely sparse: they contain no graphics, no videos, and minimal style.

Lightweb, like today's web, consists of pages authored by a potentially unlimited number of publishers. Unlike today's web, a lightweb deployment is administratively centralized: a single content-distribution network (CDN) is responsible for running an entire lightweb "universe." The CDN maintains a logical ZLTP server, running on many thousands of machines potentially, that collectively serve every page in a lightweb universe. Publishers push updates to their lightweb pages to one or more lightweb universes, and the CDNs serve this content up to users.

Limitations. While lightweb provides strong privacy guarantees, it still has limited functionality in comparison to the traditional web: lightweb is not a drop-in replacement for the web. Lightweb content is read-only and pages must be relatively small—on the order of kilobytes. Even though we show that retrieving a data blob costs less than a cent in server computation, server-side computation costs are still vastly higher in lightweb than in the traditional web. Furthermore, lightweb is still susceptible to restricted timing attacks; an attacker that controls the network can see when a client fetches a webpage and how many pages the client fetches, even if the attacker cannot see which pages the client is fetching. Despite these limitations, lightweb offers a path forward for providing strong privacy guarantees that anonymizing proxies cannot achieve.

2 The zero-leakage transfer protocol (ZLTP)

In this section, we introduce ZLTP, a new client-server application-layer protocol that underlies lightweb. A ZLTP server holds a list of key-value pairs where each key is an arbitrary string, and each value is a fixed-length binary blob. The ZLTP API exposes a single "private-`GET`" operation to the client, which has the type signature `GET(key)->value`.

A ZLTP session begins with a client connecting to a ZLTP server. The server indicates to the client the size of the fixed-length blobs it is serving, and the client and server then negotiate which cryptographic mode of operation they will use. For each `GET` request the client makes, the client and server exchange some mode-of-operation-specific protocol messages. At the end of each exchange, the client holds the value (data blob) corresponding to its query key.

2.1 Security goal

What makes ZLTP non-trivial is that even an attacker who controls the ZLTP server should should learn *no information whatsoever* about which key-value pair the client fetched. In particular, the ZLTP server must never have access to the client's request key in plaintext.

Depending on which mode of operation is in use, this security property holds under one or more of these assumptions:

- *Cryptographic:* Some computational problem (e.g., factoring) is intractable.
- *Non-collusion:* An attacker controls at most k of n servers.
- *Hardware:* A hardware enclave correctly protects secrets.

Non-goals. ZLTP does not hide the number or timing of client requests. Similarly, ZLTP does not guarantee availability or provide integrity against malicious servers, or availability against network faults.

2.2 Modes of operation

We envision two main modes of operation, requiring different assumptions and computational resources:

Private information retrieval. The highest security mode of operation for ZLTP uses cryptographic protocols for private information retrieval [15]. These schemes allow a client to fetch a key-value pair [14] from a server while hiding all information from the server about which pair the client fetched. Typical schemes for private information retrieval require that, for each query, the servers perform a linear scan over all of the key-value pairs [10]. (Otherwise, an adversary controlling the server can learn which key-value the user is not retrieving.) To minimize the cost of this linear scan, a ZLTP server using private information retrieval as its mode of operation would typically use a very small blob size.

Our prototype uses one of the fastest known private-information-retrieval schemes [12]. This scheme has very low communication cost: for a single key-value lookup, the upload is logarithmic in the size of the key space, and the download is linear in the size of retrieved value. The downside is that this scheme requires the client to communicate with two non-colluding servers to fetch a blob. When using this mode of operation therefore, the ZLTP client must establish sessions with two ZLTP servers; security holds as long as the attacker can compromise at most one of them. (Schemes whose security rests only on cryptographic assumptions also exist, but these have higher communication and computation costs [7, 35].)

Hardware enclaves + oblivious RAM. While private information retrieval provides strong privacy guarantees, the server-side linear scan over all stored data limits performance. A faster mode of operation allows the client to make private key-value lookups by communicating with a server-side hardware enclave (e.g. Intel SGX), which uses an oblivious-RAM scheme to privately access a large local store in untrustworthy memory [19, 38, 48]. The hardware enclave must use an oblivious-RAM protocol to access its local storage to ensure that the memory-access patterns do not leak which key-value pairs a client is requesting. By using an oblivious RAM scheme tailored to hardware enclaves, the enclave can hide its memory-access patterns while supporting many clients. This approach has best-possible communication costs and appealingly low server-side computational costs: both polylogarithmic in the number of key-value pairs. At the same time, a slew of attacks on the security of hardware enclaves makes relying on them for data protection somewhat risky [13, 47, 50, 53, 54, 56].

2.3 Beyond the web

Private web browsing is only one potential application of ZLTP. Developers could use ZLTP to build other privacy-preserving desktop or mobile applications without needing to implement the cryptographic building blocks or potentially even host their own servers. The development of ZLTP servers could enable developers to focus on the application layer of privacy-preserving applications without having to worry about the implementation of the core cryptographic primitives.

3 The lightweb architecture

We now describe how to use ZLTP to build lightweb, a web-like environment that completely hides, from the network and from the lightweb servers, which pages a user has visited.

3.1 System overview

A lightweb deployment is centered around a content *universe*, a collection of millions or billions of lightweb pages hosted

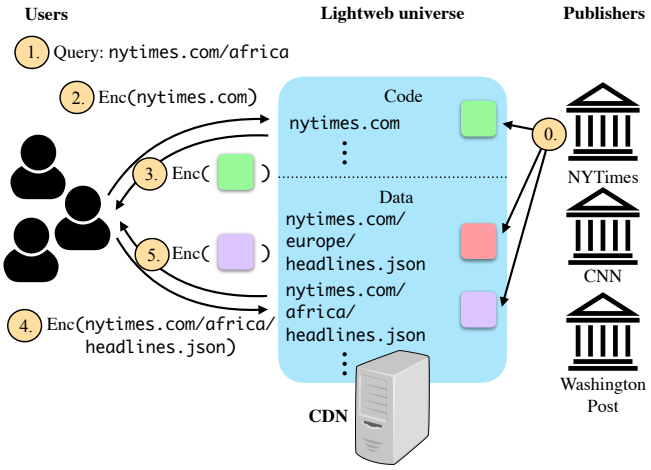

Figure 1: Lightweb system architecture. First, publishers upload content (a root code blob and many data blobs). Then, to query for a path, the client first queries for the code blob (if not already cached), and then queries for data blobs (above the client only queries for a single data blob, but in reality, the client would query for a small, fixed number of data blobs).

on a single content distribution network (e.g., Akamai), managed within a single administrative domain. Multiple lightweb universes can co-exist and can "peer" with each other to share content; we defer that discussion to Section 3.5 and for now, assume that all lightweb content lives within a single universe.

The content-distribution network (CDN) hosting a lightweb universe maintains a single logical ZLTP server serving all of the lightweb pages within its universe. In practice, a CDN's logical ZLTP server would be comprised of thousands of physical machines configured for fault-tolerance, as a large-scale web service is today. Each CDN chooses which ZLTP modes of operation to support, based on the cost tolerance and privacy demands of its users.

Lightweb publishers (`cnn.com`, `wikipedia.org`, etc.) produce content as:

- a single root "code" blob that contains a blob of JavaScript code and style information and
- a large number of "data" blobs that contain relatively small JSON data objects.

(We discuss page structure and dynamic content further in Sections 3.2 and 3.3.) Since clients download these blobs via ZLTP, all code blobs in the universe must have a single fixed size (e.g., 1 MiB) and all data blobs in the universe must have a single fixed size as well (e.g., 4 KiB). To expose content to the lightweb users, the publisher pushes its code and data blobs to one or more of the CDNs managing lightweb universes.

Every data blob within a CDN's lightweb universe has a unique path, such as `nytimes.com/world/africa/2023/06/headlines.json`. The only constraint on the path format is that it must have a valid domain as the top-level path component; otherwise, the path may have any format.

The separation of page content into code blobs and data blobs is primarily a performance optimization: by breaking pages into individual blobs, publishers can eliminate redundancy across pages in their site and reduce the amount of data stored at the CDN. This in turn reduces the CDN's computational cost, especially when the CDN serves content via ZLTP's high-security private-information-retrieval mode of operation.

By convention, a single publisher controls all of the content beneath a particular top-level path component. For example, the New York Times might control all paths beginning with `nytimes.com/*`. The CDN is responsible for managing ownership of path prefixes within a universe.

3.2 Anatomy of a lightweb browsing session

To browse lightweb, a user first downloads the lightweb client, which is essentially a minimal web browser that speaks the ZLTP protocol.

1. Connect to a CDN. The first step to browsing lightweb is for a user to connect to the CDN for a particular universe. The client opens two ZLTP sessions with the CDN: one for fetching the large code blobs and one for fetching the small data blobs.

2. Fetch code blob. When the user requests a page at a particular lightweb path, such as `nytimes.com/2023/06/25/uganda`, the browser extracts the domain (`nytimes.com`) and makes a private-`GET` request for the code blob for this domain via its code-only ZLTP session with the CDN. Since the code blobs are large and expensive to fetch, we only allow each domain to host a single code blob, and the client aggressively caches the code blobs. We would expect code blobs to change very rarely—once every few days at most.

3. Fetch data blobs. Once the client has the code blob for its desired domain, the client executes a `main` JavaScript routine contained in the code blob, passing in the path `/2023/06/25/uganda` as an argument. A domain's code blob can then make a small, fixed number of private-`GET` requests over the client's data-only ZLTP session with the CDN.

Within a universe, the number of data blobs fetched per page view must be fixed. Otherwise, the client could leak information about which pages it is visiting via the number of private-`GET` requests made in a given time period. The number of data `GET` requests per page must be small enough to give tolerable performance.

4. Render content. Once the client receives the responses to all of these private-`GET` requests, the browser may render the page. To do so, the browser passes the fetched data blobs to the running JavaScript code, which renders the page content. The data blobs may contain arbitrary JSON objects, so the publisher can choose whether the data blobs include textual content, style information, additional code, or other data. The only hard restriction is on the number and size of the blobs.

Many of the client-side JavaScript features that today's web provides are available in lightweb: client-side interaction, local storage, and so on. (As today, the lightweb browser enforces domain separation on local storage and other client-side state.)

Differences with standard web environment. The primary difference between the lightweb JavaScript environment and a conventional web one is that once the browser has fetched one code page and a fixed number of data blobs (e.g., five), the browser permits no other network interaction until the user browses to a new page. The browser loads no images, media, or style content, apart from what can fit into the data-blob fetches.

In this way, a network attacker only learns:

- which universe a user is connected to (leaked via IP headers),
- when the user has visited a new domain (leaked via a code-page fetch), and
- when the user visits a new page or follows a hyperlink (leaked via data-page fetches).

It is possible in principle to infer some limited information about the user's browsing behavior by the number and timing of their page visits [34]. For example, a user fetching a page every five minutes in the morning might be most likely to be reading the news. But even this leakage is modest.

Domain-specific code should be small in order to cache it at the client and to make ZLTP requests performant. We envision publishers using regular expressions to parse paths in order to avoid enumerating all possible paths in the client-side code. CDNs can host domain-specific code in a separate "universe" from the other key-value pairs. This separation can improve ZLTP performance and only reveals when a user is visiting a path with a domain where the code is not cached locally.

3.3 Dynamic content

One severe limitation of lightweb is that it fundamentally cannot support dynamic page content requiring large amounts of server-side content. That said, limited but useful forms of dynamic content are possible with lightweb. In particular, because the domain's code determines which data blobs to fetch as a function of the page requested and local user data, publishers can support customized and personalized content as long as it does not require too much server state. For example, the `weather.com` lightweb page could prompt the user for their postal code and cache it in local storage. Later on, when the user visits `weather.com`, the page could could use the user's cached postal code to automatically fetch a per-postal-code data blob containing up-to-date weather information for their location.

Lightweb can also support access control by allowing web publishers to control the set of users who can view content. We would like to only allow some users to access web content, but we don't want the CDN to know each user's permissions with each domain. To solve this problem, the CDN can simply store an encryption of the data. When the client makes an account with the publisher outside of lightweb, it obtains cryptographic key(s) that it can use to decrypt data for that publisher that correspond to its permissions. The publisher can periodically rotate keys in order to revoke users' access as necessary, and clients can query the publisher periodically for updated keys. The publisher could also use broadcast encryption to allow clients to update their keys based on membership changes [25, 41] While an attacker that has compromised the network or the publisher's server can determine that a user has an account with a given publisher, it learns no information about the individual pages that the user is visiting.

For applications, such as online banking, for which it is acceptable (or even necessary) for the server to know the client's identity, a lightweb page can link out to a standard web page. A user that clicks this link will no longer have the privacy guarantees provided by lightweb, but will be able to perform their desired task (e.g. transferring funds).

3.4 Monetizing lightweb pages

Paywalls. Publishers place content behind paywalls in order to incentivize users to pay to view content. Lightweb can support paywalls via the access-control mechanisms described above.

Advertising. Lightweb is compatible with online ads. The simplest way to achieve this is to have a publisher embed subject-relevant ads directly into their site's static content. Ad targeting is also possible in principle: the site's code could fetch different ads from the CDN based on the user's local state (browsing history, postal code, inferred interests, etc.).

3.5 Multiple universes and peering

Up to now, we have described lightweb in terms of a single universe within a single administrative domain. Having multiple universes, one on each of the major CDNs, would improve both fault tolerance and economic competition. Each universe could decide which lightweb content to make available to its users—much as different cable TV providers offer different packages to users. The more content a universe contains, the more computationally expensive it is to respond to each ZLTP request, so a single CDN could even offer multiple universes to its users, with varying cost-coverage trade-offs.

For example, a single CDN could group its pages into "small", "medium", and "large" universes where each universe has a different fixed page size. These different universes would allow a CDN to accommodate large pages without adding overhead for fetching small pages, although the CDN (and an attacker observing the network) would learn whether the user is fetching a page from the small, medium, or large universe.

To allow lightweb content to be available across multiple universes managed by multiple CDNs, the CDNs managing these universes could peer with each other. If a publisher uploads content to one CDN, the CDN would push the content to all of its peers. To make this possible, CDNs would have to agree on the assignment of lightweb domain names to owners (e.g., using today's domain-name registration system) so that each domain has the same owner in each universe.

4 Discussion: Who pays?

Private-`GET` requests are the core primitive that makes lightweb possible. And the nature of private-`GET` requests makes the economics around the lightweb look very different from those of the standard web.

- First, as we show in Section 5, private-`GET` requests are costly. Servicing a single ZLTP private-`GET` request requires tens or hundreds of CPU core-seconds of computation, at least when using the cryptographic mode of ZLTP operation. (We we argue in Section 5 that the absolute cost is still reasonable.)
- Second, the cost of a private-`GET` request for a lightweb page scales *linearly* with the total number of pages in the lightweb. So if a publisher decides to post more news articles on their site, the cost of every user's private-`GET` requests increases.
- Third, the cost of adding a page to a lightweb universe is independent of the popularity of a page: adding a page to `cnn.com` is as costly to the system as adding a page to `poodleclubofamerica.org`, even if one site receives 1000× more traffic than the other.

For users who make on average 50 daily page requests where each page request results in 5 `GET` requests for data blobs, we estimate that the monthly per-user cost for a universe of 360M data blobs with blob size at most 0.9 KiB each to be roughly $15 (comparable to the cost of a Netflix membership).

We would expect CDNs to charge both users and publishers for access to a universe via a variety of billing models, much as Internet service providers and large-scale CDNs bill today.

Some CDNs could choose to charge publishers proportionally to the number of queries received for their domain. In order to privately collect data on the number of queries received for each domain, the CDN could use a system for the private collection of aggregate statistics [5, 11, 16, 22, 39].

5 Evaluation

In the evaluation, we aim to answer the following questions:

- What are the costs of ZLTP in two-server private-information-retrieval mode for a small data shard? (Section 5.1)
- How do these costs scale up to a larger dataset? (Section 5.2)

Implementation and experiment setup. We implement a small-scale prototype of ZLTP with two-server private information retrieval in C++. Two-server private information retrieval provides security in the event that an attacker can compromise at most one of the two (logical) servers. We use Google's distributed point function library for two-server private information retrieval [28]. We also use vector AVX instructions to accelerate the data scan. We run experiments on Amazon AWS EC2 instances. We use a c5.large instance with 2 vCPUs and 4 GiB of memory.

Dataset. We use the C4 dataset [45, 46], a cleaned version of the common crawl, to approximate the contents of lightweb. While the C4 dataset does not include the entire contents of the web, it provides a reasonable approximation of the subset of web content that developers might push to lightweb. The C4 dataset is roughly 305 GiB compressed, contains 360M pages, and the average compressed page size is roughly 0.9 KiB.

We derive parameters that allow us to approximate a small shard of the C4 dataset, which we can then use to estimate ZLTP performance at a larger scale. We set the maximum data blob size to 4 KiB based on the fact that the average compressed page size is 0.9 KiB; any values longer than this can be broken up and retrieved separately (i.e. the user can click a "next" link if she wants to read more). We run our microbenchmarks on dummy values of the maximum blob size, as the server cost is primarily determined by DPF evaluation and the cost of scanning over all stored values. We note that in a real deployment with heterogeneous-sized values, we would lose some space by packing values.

5.1 Microbenchmarks

Server computation. We run microbenchmarks on a small data shard in order to estimate the costs of ZLTP on a large dataset. With 1 GiB of memory and an output domain of size 2^{22}, each request takes approximately 167ms of computation. Of those 167ms, roughly 64ms are spent in DPF evaluation and the other 103ms are spent scanning over the data. By setting the output domain to size 2^{22}, we guarantee that if there are roughly 2^{20} key-value pairs (roughly the number we'd expect with 1 GiB of storage and an average value size of 0.9 KiB), the probability of collision is at most 1/4 when the ZLTP server is almost at capacity (if this happens, then the publisher can simply select another key name). We could decrease this probability by increasing the DPF output domain or by using cuckoo hashing and probing several locations per request.

Batching requests to increase throughput. Because the majority of the overhead is due to the cost of scanning over the data, we batch together requests, which increases latency (page-load time) but improves throughput. By batching 16 requests together, we spend on average 167ms of computation per request for a total latency of 2.6s and a throughput of 6 requests/s. (Because we are in the two-server setting, each request must be processed at two different servers, and so the effective per-query computation time of the entire system is actually 334ms.) In contrast, by only processing one request at a time, we achieve a latency of 0.51s and a throughput of 2 requests/s.

Communication. The communication overhead depends on the size of the DPF key and the bucket size returned to the client. The DPF key size is roughly $(\lambda + 2)d$ where λ is the security parameter ($\lambda = 128$) and 2^d is the size of the output domain ($d = 22$). Our implementation sets the output bucket size to 4 KiB. Together, the total communication per request is 13.6 KiB (including the 2× overhead for two-server private information retrieval).

5.2 Estimated costs for scaling up ZLTP

We use the measured costs for a small data shard to estimate the costs for the C4 dataset with 305GiB and 360M webpages (for approximating costs, we assume that no space is lost due to packing different-sized webpages). To scale up from 1 GiB with a single c5.large data server, we consider a deployment of

Dataset	Total size	# pages	Avg. page size	vCPU sec	Request cost	Communication
C4	305 GiB	360M	0.9 KiB	204	$0.002	15.9 KiB
Wikipedia	21 GiB	60M	0.4 KiB	10	$0.0001	14.9 KiB

Table 2: Summary of estimated costs of running ZLTP on the C4 dataset and Wikipedia. The dataset size and page size include compression. We base request cost on the cost of an AWS c5.large instance.

305 c5.large data servers, each managing 1 GiB of the dataset. Such a deployment would also need several front-end servers to intercept incoming client requests, route them to the data servers, and combine the results before routing to the servers.

Distributing DPF evaluation. In this setting, the cost of the data scan is still the same at each individual machine as in our microbenchmarks. By having the front-end server process the client's DPF key before sending the DPF key to the data servers, the DPF evaluation can also be the same. DPF evaluation is done by building a tree, and so the front-end server can build the top part of the tree and then, for each sub-tree, send the sub-tree root to the corresponding server. The cost for the data server of completing the DPF evaluation from that point is the same as the cost of evaluating the DPF key for the smaller domain.

Server computation and AWS cost. In Table 2, we summarize the estimated costs for the C4 dataset as well as the smaller Wikipedia dataset. For this architecture on the C4 dataset, we shard each request across 305 c5.large instances, each of which must perform 167ms of computation per request, has 2 vCPUs, and costs $0.085 per hour. Therefore, each request requires 1.7 vCPU minutes of computation at the data servers, resulting in a per-request cost of $0.001 Due to the overhead of the two-server setting, the system-wide computation cost is 3.4 vCPU minutes and $0.002 per request. We do not consider the cost of the front-end coordinators or egress communication, which will also add to the system cost. The request latency (page-load time) is lower-bounded by 2.6s, the latency of an individual data server, but would likely be higher due to network latency, front-end server latency, and data-server stragglers.

For comparison, Google Fi charges $10/GiB [27], and so the cost to load the 22.4 MiB New York Times homepage is $0.218, suggesting that users are willing to pay $0.218 to load a page. Loading data via ZLTP is roughly two orders of magnitude more expensive than the traditional web: loading 4 KiB (our ZLTP value size) costs $0.002 with ZLTP and $0.000038 with Google Fi.

Communication. In the setting where each of the 305 machines maintains a shard with data from its own output domain of size 2^{22}, the client-to-server communication is 7.9 KiB, and the server-to-client communication is 8 KiB for a total of 15.9 KiB.

Looking forward. Although ZLTP is substantially more expensive than non-private web requests, as the cost of computation continues dropping, ZLTP will become cheaper. In 2003, $1 bought 8 CPU hours, and in 2008, $1 bought 128 CPU hours (adjusted for inflation), a 16× increase [26]. This change suggests that in 5 years, we could potentially see the dollar cost of a ZLTP request drop by an order of magnitude.

6 Related work

The predominant technique for privately browsing the web today is to use an anonymizing proxy such as Tor [23]. The iCloud private relay system is another anonymizing proxy service that uses non-colluding servers to ensure that no single party can learn both a user's IP address and the webpage that a user is requesting [8]. These proxies are vulnerable to a wide array of traffic-analysis attacks [9, 32, 33, 40, 42, 51]. A number of private messaging systems defend against traffic-analysis attacks [7, 17, 18, 24, 36, 36, 37, 52, 55]. However, these systems are generally high-latency, and they focus on private messaging or posting rather than private browsing. Another way to eliminate these traffic-analysis attacks would be for the user to connect to a CDN distributing fixed-size webpages (similar to lightweb) via an anonymizing proxy. A serious drawback of this approach is that the CDN knows all webpage requests for many users and so can run a deanonymization attack to map users to requests [43, 44]. The ZLTP protocol defends against both traffic-analysis and deanonymization attacks. Kesdogan [34] discuss using private information for web browsing at a theoretical level; we give a concrete architecture (showing how to ensure pages have equal length, etc.) and preliminary implementation results.

Another line of work uses private information retrieval as a building block for privacy-preserving systems. Examples of these include systems for private media delivery [29], private search [6, 20, 30], private aggregation queries [21, 57], and private messaging [7]. Tiptoe [30] privately searches over webpages; users could then access their search results using lightweb.

7 Conclusion

We introduce lightweb, a system for web-like fully private web browsing. Crucially, no one, not the network and not the CDN, learn what page the user is requesting. Lightweb is built on ZLTP, which supports a private-`GET` operation. While lightweb is not a drop-in replacement for today's web, lightweb has the potential to afford users privacy guarantees that are not possible to achieve using traditional techniques (i.e. Tor) for privately browsing the web.

Acknowledgements. We thank the anonymous HotNets reviewers for their thoughtful feedback. Alexandra Henzinger and Eric Rescorla gave suggestions that improved the presentation. We would also like to thank Raluca Ada Popa for her generous support. This work was funded in part by gifts from Capital One, Facebook, Google, Mozilla, NASDAQ, and MIT's FinTech@CSAIL Initiative. We also received support under NSF Award CNS-2054869 and from the Sky lab at UC Berkeley. Emma Dauterman was supported by a NSF Graduate Research Fellowship and a Microsoft Ada Lovelace Research Fellowship.

References

[1] NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say, October 2013.

[2] Government push for Yahoo's user data set stage for broad surveillance, September 2014.

[3] In NSA-intercepted data, those not targeted far outnumber the foreigners who are, July 2014.

[4] Google Might Owe You Money. Here's How to Get It., June 2023.

[5] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *Security and Cryptography for Networks: 13th International Conference, SCN 2022, Amalfi (SA), Italy, September 12–14, 2022, Proceedings*, pages 516–539. Springer, 2022.

[6] Ishtiyaque Ahmad, Laboni Sarker, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Coeus: A system for oblivious document ranking and retrieval. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 672–690, 2021.

[7] Sebastian Angel and Srinath TV Setty. Unobservable communication over fully untrusted infrastructure. In *OSDI*, volume 16, pages 551–569, 2016.

[8] Apple. About iCloud Private Relay, August 2023. https://support.apple.com/en-us/102602.

[9] Lamiaa Basyoni, Noora Fetais, Aiman Erbad, Amr Mohamed, and Mohsen Guizani. Traffic analysis attacks on tor: A survey. In *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, pages 183–188. IEEE, 2020.

[10] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20*, pages 55–73. Springer, 2000.

[11] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.

[12] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[13] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. SGXPECTRE: Stealing intel secrets from SGX enclaves via speculative execution. In *EuroS&P*. IEEE, 2019.

[14] Benny Chor, Niv Gilboa, and Moni Naor. *Private information retrieval by keywords*. 1997.

[15] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[16] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, pages 259–282, 2017.

[17] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[18] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350, 2010.

[19] Emma Dauterman, Vivian Fang, Ioannis Demertzis, Natacha Crooks, and Raluca Ada Popa. Snoopy: Surpassing the scalability bottleneck of oblivious storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 655–671, 2021.

[20] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. Dory: An encrypted search system with distributed trust. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 1101–1119, 2020.

[21] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468. IEEE, 2022.

[22] Alex Davidson, Peter Snyder, EB Quirk, Joseph Genereux, Benjamin Livshits, and Hamed Haddadi. Star: Secret sharing for private threshold aggregation reporting. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 697–710, 2022.

[23] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[24] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, Dan Boneh, et al. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *USENIX Security Symposium*, pages 1775–1792, 2021.

[25] Amos Fiat and Moni Naor. Broadcast encryption. In *Advances in Cryptology—CRYPTO'93: 13th Annual International Cryptology Conference Santa Barbara, California, USA August 22–26, 1993 Proceedings 13*, pages 480–491. Springer, 1994.

[26] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.

[27] Google. About google fi plans. https://support.google.com/fi/answer/9462098?hl=en.

[28] Google. An implementation of incremental distributed point functions in c++. https://github.com/google/distributed_point_functions.

[29] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 91–107, 2016.

[30] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, , and Nickolai Zeldovich. Private web search with Tiptoe. In *29th ACM Symposium on Operating Systems Principles (SOSP)*, Koblenz, Germany, October 2023.

[31] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42, 2009.

[32] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348, 2013.

[33] Ishan Karunanayake, Nadeem Ahmed, Robert Malaney, Rafiqul Islam, and Sanjay K Jha. De-anonymisation attacks on tor: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2324–2350, 2021.

[34] Dogan Kesdogan, Mark Borning, and Michael Schmeink. Unobservable surfing on the world wide web: is private information retrieval an alternative to the MIX based approach? In *Privacy Enhancing Technologies*, 2003.

[35] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings*

38th annual symposium on foundations of computer science, pages 364–373. IEEE, 1997.

[36] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 406–422, 2017.

[37] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 711–725, 2018.

[38] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296. IEEE, 2018.

[39] Dimitris Mouris, Pratik Sarkar, and Nektarios Georgios Tsoutsos. Plasma: Private, lightweight aggregated statistics against malicious adversaries with full security. *Cryptology ePrint Archive*, 2023.

[40] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195. IEEE, 2005.

[41] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings 21*, pages 41–62. Springer, 2001.

[42] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.

[43] Sai Teja Peddinti and Nitesh Saxena. Web search query privacy: Evaluating query obfuscation and anonymizing networks. *Journal of Computer Security*, 22(1):155–199, 2014.

[44] Albin Petit, Thomas Cerqueus, Antoine Boutet, Sonia Ben Mokhtar, David Coquil, Lionel Brunie, and Harald Kosch. SimAttack: private web search under fire. *Journal of Internet Services and Applications*, 7(1):1–17, 2016.

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. C4 model. https://huggingface.co/datasets/c4.

[46] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019. https://arxiv.org/abs/1910.10683.

[47] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Crosstalk: Speculative data leaks across cores are real. In *Security & Privacy*. IEEE, 2021.

[48] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. Zerotrace: Oblivious memory primitives from intel sgx. *Cryptology ePrint Archive*, 2017.

[49] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *USENIX Security Symposium*, pages 1357–1374, 2017.

[50] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-privilege-boundary data sampling. In *CCS*. ACM, 2019.

[51] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: Routing attacks on privacy in tor. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 271–286, 2015.

[52] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.

[53] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings fo the 27th USENIX Security Symposium*. USENIX Association, 2018.

[54] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *Security & Privacy*. IEEE, 2020.

[55] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152, 2015.

[56] Stephan Van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. In *Security & Privacy*. IEEE, 2019.

[57] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *NSDI*, pages 299–313, 2017.