CoStricTor: Collaborative HTTP Strict Transport Security in Tor Browser

Killian Davitt
UCL
London, UK
killian.davitt.17@ucl.ac.uk

Dan Ristea
UCL
London, UK
dan.ristea.19@ucl.ac.uk

Duncan Russell
The Tor Project

Steven J. Murdoch
UCL
London, UK

ABSTRACT

HTTP Strict Transport Security (HSTS) is a widely-deployed security feature in modern web browsing. It is also, however, a potential vector for user tracking and surveillance. Tor Browser, a web browser primarily concerned with online anonymity, disables HSTS as a result of this tracking potential. We present the CoStricTor protocol which crowdsources HSTS data among Tor Browser clients. It gives Tor Browser users increased resistance to man-in-the-middle attacks without exposing them to HSTS tracking. Our protocol adapts other privacy-preserving data aggregation algorithms to share data effectively among users with strong local differential privacy guarantees. The CoStricTor protocol resists denial of service attacks by design through our innovative use of Bloom filters to represent complementary data. Our simulations show our protocol can model up to 150,000 websites, providing 10,000 upgrades to HSTS for users.

1 INTRODUCTION

HSTS (HTTP Strict Transport Security) is an HTTP security feature that forces secure connections on websites that have previously provided a HSTS flag to the user's browser. Currently, the Tor Browser does not support HSTS due to the tracking potential of HSTS. Malicious websites could set arbitrary HSTS flags by directing the browser to load resources from various domains or subdomains, creating a persistent fingerprint allowing multiple visits by the same user to be linked [33]. However, disabling HSTS removes the security protection this feature offers and increases the risk that malicious Tor exit nodes could perform a man-in-the-middle attack.

We construct a protocol which enables users of Tor Browser to effectively crowdsource their HSTS flag status. Users share HSTS flags they receive from websites into a centralized location where it can be distributed to other users for querying. This completely eliminates the potential of HSTS tracking as described in [33] by ensuring that any malicious HSTS flag will be shared among all of the users of the protocol.

To inform our design decisions, we survey the current state of HSTS deployment, including the distribution of expiry times. We use the Majestic Millions index of the most popular websites based on incoming links.

One immediate consideration of a protocol like this is privacy; if users share data on websites they visit, it is essential to introduce a privacy mechanism such that users do not automatically reveal which websites any one user has been browsing. To this end, the core of our protocol involves privacy-preserving data sharing, i.e., a protocol to ensure data can be reliably reported from many clients to a server without leaking the data of any one individual to the server. We have selected the RAPPOR protocol [16] to provide local differential privacy for user submissions. RAPPOR has the necessary flexibility to enable us to use broad concepts from the protocol while also allowing us to alter and build on top of it to adapt it for this specific application. This includes a double Bloom filter construction, which reduces false positives in our protocol and improves its resilience to Sybil attacks. The protocol design prevents it from degrading the user experience even in worst-case scenarios.

We perform a simulation of the protocol which allowed us to select the best parameters for the protocol and demonstrated that it can efficiently model up to 150,000 websites.

The CoStricTor protocol provides additional information to the user in a potential man-in-the-middle downgrade attack. It replaces the standard HTTPS-only warning with a variant of the HSTS warning, informing the user that continuing to the page caries a higher risk than the more common HTTPS-only warning. The protocol does not increase the total number of warnings a user will see. As well as helping users directly, the protocol assists with detecting potential malicious exit nodes in the Tor network, which may help prevent future attacks.

Section 2 provides relevant background information on Tor, Tor Browser and HSTS which are necessary for understanding our protocol and why it is useful. We describe our protocol in section 4, in which we provide a discussion on various building blocks which make up our protocol before describing the detailed processes of the protocol's operation. The simulation which was used to evaluate the protocol is discussed in section 5, and we provide the results which both prove the usefulness of the protocol and describe the appropriate parameters which are needed to operate an implementation of the CoStricTor protocol. We provide a discussion on the usefulness of the protocol as well as its limitations in section 6 Related work and conclusions are given in sections 7 and 8.

Proceedings on Privacy Enhancing Technologies 2024(1), 343–356

https://doi.org/10.56553/popets-2024-0020

2 BACKGROUND

2.1 HSTS

HTTP Strict Transport Security or HSTS is a protocol by which websites can instruct Internet browsers to only interact with them via a secure HTTPS connection [22]. Websites communicate their HSTS policy to browsers by setting the corresponding header in their HTTP response to a user request. When the browser encounters the HSTS header, it will store the HSTS status of the website and only allow future connections to be HTTPS. Forcing HTTPS connections prevents both passive eavesdropping attacks and active man-in-the-middle attacks. If a website known to have HSTS does not support HTTPS, the browser displays a warning page which does not allow the user to continue, due to the high risk of an attack aiming to downgrade the security of the connection.

The HSTS header must include a *max-age* directive which informs the user's browser how long to store the HSTS status. Websites can revoke their HSTS status by setting the expiry time in the header to 0 and waiting for any previously stated expiry time to elapse.

HSTS requires a user to first navigate to a website over a valid HTTPS connection for future connections to be protected by HSTS. To help mitigate this limitation, all major browsers (including Google Chrome, Chromium derivatives, and Mozilla Firefox) include the HSTS preload list [8]. The preload list is distributed with the web browser and updated when the web browser is updated. When initiating a connection, browsers will check both the saved HSTS list and the preload list. The preload list allows website administrators to mark their websites as having HSTS to all users of the browser. The recommended value for HSTS expiry is 2 years and the preload list requires this value for a website to be included [8].

In practice, the overall number of sites submitted to the preload list is low, therefore it is of limited benefit. Many industries and types of websites were found to be poorly represented in the preload list [31]. Although the preload list has grown to approximately 190,000 entries at the time of writing, of which the Tor Browser incorporates 138,000 entries, it scales poorly with the size of the public Internet. Of the top million websites by referring subnets [6], only 14,000 are included in the preload list.

HSTS is an actively recommended security feature for the web. Mozilla web security guidelines include HSTS as "*mandatory for all websites*", as HSTS has high value and low implementation difficulty [27].

2.1.1 HSTS Supercookies. An unfortunate side-effect of HSTS is its potential as a tracking system. By saving the HSTS status of multiple subdomains, a browser can be manipulated to store a unique identifier which will implicitly be sent to the website on subsequent connections. This may be used as a form of web tracking [22, Sec. 14.9]. Such tracking techniques, which eschew the normal controls for client-side data retention, are termed "supercookies" [33].

To create an HSTS supercookie, a website can include dummy resources from multiple subdomains. On a user's first visit, all connections to subdomains will be made over HTTP, identifying the user as a new user. The server can set a unique combination of HSTS flags for the subdomains in its responses to the user. Therefore, each subdomain's HSTS entry can store one bit of identifying information. On subsequent visits, the pattern of HTTPS and HTTP connections to subdomains based on the unique combination of HSTS flags stored in their browser can identify the user.

Tracking through this method becomes even more effective if it is performed by actors whose resources are loaded on multiple websites, such as ad networks or content distribution networks. In this case, not only can a user be identified on one site, they can also be tracked across the web.

2.2 Tor

Tor is a system for anonymous communication over the Internet. Tor nodes, or relays, create an overlay network on the Internet, called the Tor network, through which anonymous network traffic can be routed. Data sent through Tor is encrypted using multiple layers of encryption and sent through nodes which decrypt a single layer at a time. Each layer of encryption only holds routing information for the next relay, hiding the true destination and origin of the packet from the intermediate nodes. This technique is termed *onion routing*. To servers outside the Tor network, connections appear to originate from the last Tor node that traffic was routed through, the *exit node*. This system protects the anonymity of the user and hides their online activity.

The configuration of the Tor network is decided through consensus by a set of *directory authorities* whose IPs are hard-coded into Tor clients [5]. The consensus is approximately 2.8 MB or 580 kB compressed. Once downloaded, consensus data is cached while it is valid and persists between sessions. Whenever a client initiates a new connection, it uses the configuration to pick a new set of relays to route their traffic.

Some nodes in Tor are designated *guard nodes* which are considered more trustworthy than other nodes. These nodes are used as the entry, or first node in a users connection to Tor. Tor clients select a single guard node to be used for a period of time and all connections made will begin with that guard for the duration.

The Tor network has been subject to multiple attacks which aimed to compromise the encryption of outgoing connections. Due to its position, the Tor exit node can perform man-in-the-middle attacks when forwarding packets to the destination server. SSL stripping attacks, in which a man-in-the-middle prevents the upgrade from HTTP to HTTPS have been used to deanonymize communications [28] and to rewrite cryptocurrency transactions [1]. Due to the outsized impact of bad exit nodes, Tor provides a mechanism to report potentially suspicious exit nodes [37].

2.2.1 Ethics of Tor. The system proposed in this paper will reduce Tor's vulnerability to an attack that is being actively exploited for criminal purposes, however in principle the same attack could be applied by law-enforcement against activities that are unlawful in their jurisdiction. This tension raises ethical issues for research in anonymous communications and more generally for information security. Studies on this topic include the United Nations Human Rights Council finding that anonymous communication is fundamental to the human rights to freedom of expression and privacy, despite the potential of harm [38]. The American Association for the Advancement of Science similarly conclude that anonymous communication is morally neutral as a fundamental precept [4]. It is also important to consider that criminals have a wide range of

options for anonymity, such as through botnets and bullet-proof hosting, so any weakness introduced in Tor would impose limited constraints on criminals while severely harming the interests of legitimate users of anonymous communication systems. Despite the popular association between Tor and criminality, Jardine *et al.* [24] estimate only a small proportion of Tor traffic is potentially malicious (6.7%), with the proportion being lower in countries that impose restriction on political speech and freedom of expression . While onion services draws public attention for their potential to be abused for criminal purposes, our paper does not affect this feature of Tor. In any case, according to the Internet Watch Foundation, less than 1% of domains hosting child sexual abuse material (CSAM) are .onion domains hosted on the Tor network [23]. For these reasons we consider this research to be ethically justified.

2.3 Tor Browser

Tor Browser is a web browser which routes users' Internet traffic through the Tor network. Tor Browser is built on Firefox and integrates further features that increase the privacy of the user. By default, it hard-codes identifying information the browser exposes to servers, which creates a standard fingerprint shared by all Tor browser users. To further impede tracking attempts, it deletes local data between Tor sessions, including HSTS flags. Although this behaviour prevents the use of HSTS supercookies to track Tor Browser users across sessions [30], it negates the intended role of storing HSTS flags long-term.

As of version 11.5, Tor Browser defaults to HTTPS and displays a warning page before allowing users to continue over an insecure HTTP connection [35].

2.4 Differential Privacy

Differential privacy is a definition of privacy first introduced by Dwork *et al.* in 2006 [15] and it has been widely adopted in privacy-preserving data collection and manipulation.

Differential privacy is a property of a mechanism for aggregate data retrieval. It protects any individual's privacy by making any observed output of a mechanism operating on the data almost as likely as the output of the mechanism if the individual's data was not present. More formally, mechanism M is said to be ϵ-differentially private if the following inequality holds for any two neighboring databases D and $\hat{D}$ that differ in one item and for all subsets S of the possible outputs of M, i.e., $S \subseteq \text{Range}(M)$:

$$\Pr[M(D) \in S] \leq e^{\epsilon} \cdot \Pr[M(\hat{D}) \in S]$$

The privacy parameter ϵ, also termed the privacy budget provides a quantifiable measure for the relative privacy loss.

An important property of differential privacy is *immunity to post-processing*. Differentially private data can be released in full without any further degradation of privacy, regardless of any auxiliary data or computational resources an adversary might have.

Differential privacy mechanisms are implemented by introducing random perturbations, such as adding randomly sampled noise, to create uncertainty over the data which produced an observed result. In the centralized setting, the data will be aggregated first and then have a DP mechanism applied to produce the desired results, thus it requires a trusted collector; in the local setting, data is perturbed to ensure differential privacy before being sent to a collector, thus dispensing with the need for a trusted collector.

The idea of perturbing responses before they are collected predates differential privacy by decades [40]. Randomized response was introduced to avoid the bias inherent in research into sensitive topics. In the local setting, differential privacy ensures that even if the aggregator and all participants bar one collude, they cannot compromise the value submitted by an honest participant. This comes at a cost to utility as the overall noise introduced by local differential privacy is higher than in the centralized setting.

2.5 RAPPOR

RAPPOR [16] is a protocol for private aggregate data collection introduced by Erlingsson *et al.* in 2014. The protocol was integrated into the Google Chrome browser [9] but was subsequently removed.

RAPPOR has two phases. Firstly, a reporting phase in which user data is encoded into a Bloom filter, which is randomized for local differential privacy, and sent to the server where it is aggregated. Secondly, a decoding phase in which the aggregated reports are processed to obtain the relevant statistics.

2.5.1 Reporting. RAPPOR aggregates reports which encode data over a specified time period. A client will report a particular value so it can be tallied, without revealing it to the server. To send a value to the aggregation server, it is inserted into an empty Bloom filter B of size k.

RAPPOR provides privacy by randomizing the Bloom filter locally in two steps. It first generates a *permanent randomized response* (PRR) B', which will be used for all further reporting of the same data. See [16] for details on how PRR is computed using the privacy parameter f.

Before reporting to the server, the value is again perturbed in an *instantaneous randomized response* (IRR) S. The IRR is computed on every reporting on the value by initializing S with all 0 bits and then setting the value of individual bits S_i as follows:

$$P(S_i = 1) = \begin{cases} \text{q}, & \text{if } B'_i = 1. \\ \text{p}, & \text{if } B'_i = 0. \end{cases} \tag{1}$$

where the probabilities p and q are parameters of the RAPPOR protocol. The client will finally send the IRR S to the server, where the set bits are aggregated with all other responses.

This system preserves the deniability of each report from randomized response and introduces longitudinal privacy so that even over multiple reports of the same value, the reports cannot be correlated and an averaging attack cannot be performed to infer the underlying value. Our protocol omits the PRR step, as described in section 4.3.1.

2.5.2 Decoding. Upon receiving reports from clients, the server tallies these reports to get the overall number of bits set, accumulating the individual instances of Bloom filters into one counting Bloom filter The true counts of these bits in the filter can then be estimated (see the appendix of [16] for more details). From the estimated counts RAPPOR produces a best-fit distribution over a dictionary of known values using regression. Our protocol omits the regression step, as described in Section 4.3.4.

2.5.3 Hash Functions. RAPPOR uses multiple hash functions to insert the data into Bloom filters. A higher number of hash functions h reduces the chance that two distinct elements map to exactly the same bits, reducing false positives – but decreases the maximum number of items that can be reliably stored before false positives start increasing again. The same set of h hash functions is used to check membership of the set. Our protocol does not deviate from RAPPOR in how it uses hash functions. As we expect many more insertions compared to the size of the filter, the optimal value of h is always 1, therefore, we omit h from further discussion. For additional information on h, see [16].

2.5.4 Privacy. Reporting data through RAPPOR, as used in CoStricTor, provides local differential privacy with

$$\epsilon = \log(\frac{q(1-p)}{p(1-q)})$$

Using higher values of p and lower values of q introduces more noise to reports but reduces accuracy. For additional information on ϵ, see [16].

3 DATA COLLECTION

To properly evaluate our protocol, we required accurate and up-to-date data on the deployment of HSTS. We performed a survey of popular websites to gain insight into the current state of HSTS on the web.

In particular, we aimed to answer two specific questions:

- Of the most popular websites, what percentage deploy HSTS?
- Of these, what expiry times are most commonly used?

For surveying the most popular websites, we used the Majestic Million list of websites [6]. This is a freely available index, based on the number of other domains referring to the host. We then deployed the zgrab utility [34] to perform a minimal scan: a single HTTP request for the root of each website, producing minimal load on the services. The only data stored was HSTS headers. As a one-off scan with a single request, an opt-out mechanism would not have had any effect. This would have been offered if we planned any further scans. However, we omitted to provide information to web administrators about the purpose of our scan, deviating from best practices. We deemed there to be no chance of overwhelming the network infrastructure of these websites with our scan due to their popularity.

We observe that approximately 15% of the top 1 million websites have HSTS enabled. For HSTS to be effective, the expiry time must be long enough for users to return to the website. A proportion of websites surveyed have expiry times that are too short for this to occur. This can be partially explained by some websites being in the process of rolling out HSTS. Best practices recommend expiry times start small – a few minutes – to avoid bad configurations being stored by browsers, then increased gradually to the recommended 2 years [8]. Further to this, 9322 of the sites surveyed do have HSTS enabled but with an expiry time of 0, thereby entirely disabling HSTS. The vast majority of HSTS enabled websites have an expiry time of either 6 months, 1 year or 2 years.

This data, as we will discuss later, is critical in properly evaluating our protocol.

Table 1: Frequency of HSTS expiry times

Expiry time (seconds)	Equivalent to	Occurrences
31536000	1 year	70,014
63072000	2 years	19,248
15768000	6 months	10,087
0	HSTS disabled	9,322
15552000	180 days	7,654
300	5 minutes	7,399
7889238	91 days	5,295
120	2 minutes	2,861
2592000	30 days	2,840
15724800	182 days	2,614
86400	1 day	1,584
10886400	126 days	1,535
16000000	185 days	1,451
16070400	186 days	1,420
43200	12 hours	1,214
31557600	1 year, 6 days	1,168
604800	7 days	1,067
600	10 minutes	835
3600	1 hour	803
31556926	1 year, 5 days	722
31622400	1 year, 1 day	665

4 PROTOCOL

As long as there are a significant number of websites accessible only over HTTP, Tor Browser will permit unencrypted traffic between the exit node and destination server, introducing the risk of tampering or eavesdropping. HSTS was designed to protect against these risks. To protect users' anonymity, Tor Browser deletes local data after each session, thus HSTS is effectively disabled for Tor Browser users.

The goal of the CoStricTor protocol is to reduce this risk through the browser forcing encrypted HTTPS connections when the website supports this. Specifically, the protocol will allow Tor Browser to record the HSTS and HTTP status of websites in a series of privacy-preserving Bloom filters which are shared with other Tor Browser users, such that future visitors to the website can force the use of encrypted HTTPS when possible. Not only does this approach mitigate the weaknesses introduced by Tor Browser disabling persistent HSTS records, but it also goes further than HSTS by protecting Tor Browser users visiting a website for the first time, much like the preload list does.

As we discuss our changes and adaptations from the original RAPPOR protocol it is essential to note the fundamental differences in the layout of our usage. The typical usage of RAPPOR, as described by the authors, is one where data from a *known set* is submitted to the protocol.

In contrast to this, our protocol features an unbounded and unknown set of possible strings, website domains and subdomains. The strings are not known on configuration, but the relevant domain is known to the user when making a query. This setup reduces some complexity in that we do not need to reconstruct the distribution of all strings using regression. This also changes the role of the central server in our setup. A typical RAPPOR server performs regression on this data to produce useful statistics. Our protocol moves the decoding process back to the user so that the user is both a reporter of data and a decoder. This is helpful in maintaining the privacy of users' queries on the data.

4.1 Basic Overview

There are three actors in our protocol: the user (acting as a reporter and a decoder), the Tor guard nodes, and the Tor directory authorities.

As decoder, a user must first download the current protocol state, which they can obtain alongside the Tor consensus. Whenever the user visits a website, they query the protocol state. If the protocol indicates the website has been reported as having HSTS, the browser must behave as if it previously encountered an HSTS flag for the website. Due to the probabilistic nature of the protocol, we allow users to override this restriction by clicking through a warning page. To ensure that the protocol does not permit fingerprinting Tor Browser users, all user must participate in this phase to present identical information across all clients.

Once the website has loaded, if the site does provide HSTS flags, the user's second role begins. Upon seeing this HSTS flag sent by the server, the protocol encodes the domain name as a submission for the server and perturbs the data according to the privacy parameters to preserve the user's privacy. At the start of the session, the client randomly selects a guard node that is distinct from the user's current guard. All reports for the duration of the session are sent to this node over a new Tor circuit. The guard nodes periodically report their collated reports to a directory authority where it can become part of the latest consensus. The use of a single node for reports allows the use of anonymous tokens for submissions, which mitigates the risk of denial of service, as detailed in 4.3.7. Opting out of the reporting phase is possible but lower volumes of reports will degrade the performance of the protocol.

The nature of Bloom filters, combined with the perturbation introduced to achieve local differential privacy, results in a protocol that is probabilistic. Domain names which are reported many times will likely manifest as true positives in queries; domains which are reported less may be lost in the noise. It is not a design goal of this protocol to guarantee full knowledge of websites' HSTS flags to users. Instead, we aim to increase the number of users who benefit from HSTS and increase their security and privacy, while not compromising user privacy.

4.2 Threat Model

Similarly to the original threat model against Tor presented in [14], we assume that an adversary can compromise some proportion of nodes. This includes malicious Tor exit nodes that can perform SSL downgrading connections which compromise users' security and privacy. The primary goal of CoStricTor is to provide HSTS to Tor Browser users which can protect against malicious exit nodes without compromising privacy.

Due to the local differential privacy of the protocol, any individual submission does not divulge which domain was visited by the user to the server or to any other user. The immunity to post-processing property of differential privacy means that no adversary can perform further processing on the distributed filters or the individual submissions to reveal the domains that were visited by any individual user.

Although more trusted than other nodes, we assume a small proportion of guard nodes can be subverted. Submissions in the protocol are distributed randomly across all the guard nodes in the network, as clients select them uniformly at random. Directory authorities can therefore expect a similar number of submissions from each guard, any major deviation from this average will be dropped by the directory authority and that data will not be recorded by the protocol.

The Tor network assumes that a majority of directory authorities are honest. Our protocol differs in that a single malicious directory authority can do damage to the protocol proportional to the amount of data reported to them. Just as directory authorities can detect anomalies in the data reported by guard nodes, directory authorities can detect issues in data reported to them by other directory authorities.

In considering the robustness of our protocol, a key failure scenario is denial-of-service (DoS) attacks. False reporting to the protocol could prevent users of the protocol from loading legitimate sites that do not offer HTTPS. DoS attacks through HSTS are a security consideration in the HSTS standard [22, Sec. 14.5] but our protocol introduces another vector.

Although Sybil attacks are problematic for any data aggregation protocol, the anonymity of the Tor Browser makes Sybil attacks on our protocol difficult to detect and prevent. We must, therefore, integrate strong denial-of-service protection, to avoid degrading the protocol's usefulness.

We use two distinct denial of service protections in our protocol. We employ Privacy Pass [11] to prevent adversaries from reporting large amounts of false data in a denial of service attack. This is a rate limiting technique which stops opportunistic attackers from disabling the protocol. We detail this protection in 4.3.7. We also integrate DoS protection into the core design of the protocol to mitigate the damage a DoS attack will have if it is successful. This is achieved by focusing on reducing the rate of false positives overall, which is detailed in sections 4.3.5 and 4.3.6.

4.3 Protocol Design

4.3.1 Privacy. Tor Browser is designed to protect anonymity and does not save data between sessions which includes HSTS flags. This effectively disables HSTS for Tor Browser users. Our protocol creates a common set of HSTS flags across the entire Tor Browser userbase, so HSTS cannot be used as a tracking vector but users benefit from the security and privacy HSTS provides.

To ensure protection for multiple submissions from a single user, the RAPPOR protocol uses a permanent randomized response which is stored long-term and used for reporting on the same data over time. This is not possible when reporting data with our protocol,

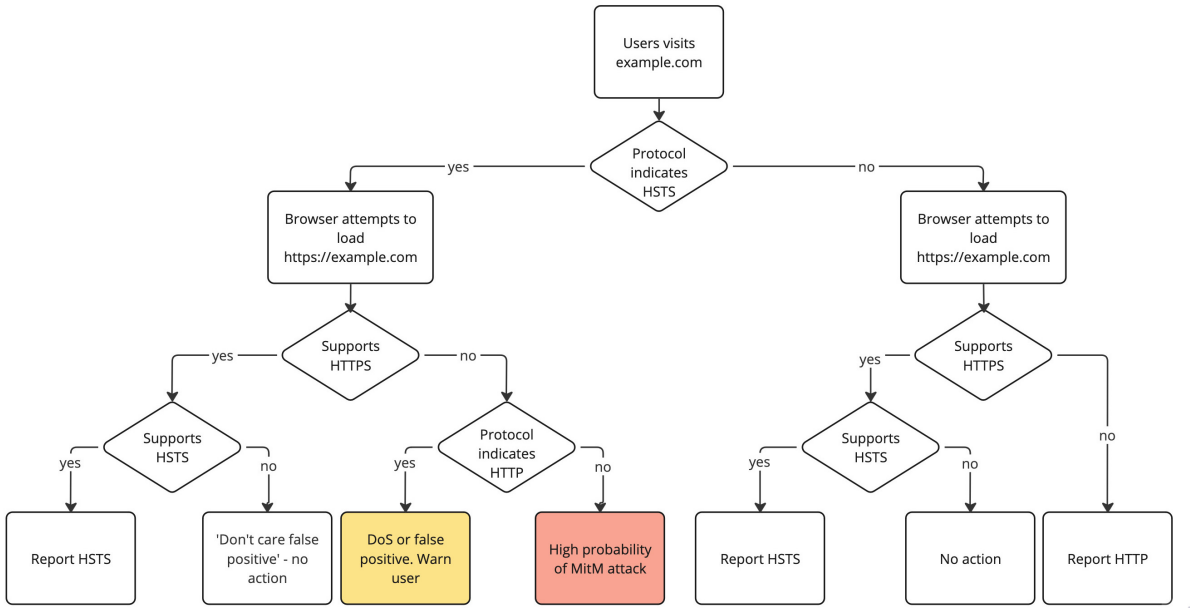

Figure 1: Protocol workflow

as Tor Browser removes user data between sessions. This is, in fact, not an issue precisely because of the anonymity provided by Tor. Different protocol submissions cannot be linked to the same user as, even within the same session they will be made over different Tor circuits.

Thus, it is not possible for an adversary to defeat the differential privacy guarantees over time and the longitudinal privacy protection offered by PRR is not needed. Therefore, our protocol is equivalent to 'One-Time RAPPOR' from [16], despite multiple submissions of the same data from a single user being possible, and provides the same local differential privacy.

Local differential privacy protects each submission sent to the server. As the aggregate data is differentially private and immune to post-processing, it can be distributed to users. The user performs queries locally, eliminating the need for a private mechanism to query the central data and allowing multiple queries to be performed without a cost to privacy.

4.3.2 Expiry of HSTS. HSTS flags come with an expiry time, and as such, it is necessary to accommodate this in our protocol. As the presence of a site in the aggregate counting Bloom filter is probabilistic, and it contains no other information other than the existence of the URL in the set, we have no way to remove websites from the set and no direct way of storing the expiry time stated by a website. The length of expiry time is effectively the longest time the user can go without using the website before the HSTS flag disappears and need to be reapplied. Conversely, the expiry time is also the shortest amount of time a website will have to wait after disabling HSTS before they can switch off HTTPS. In general, disabling HSTS and HTTPS is likely an infrequent occurrence, but it is mandated by the HSTS specification and so must be supported by our protocol.

Expiry times are handled by storing HSTS data for a set period of time, then discarding it. We first decide on a protocol-wide HSTS expiry time which will be used by all entries. We then divide the expiry time into two epochs. Expiry time is implemented by storing two different parallel instances of the aggregate data, each representing one epoch's worth of HSTS entries. Filter set A represents the current epoch and all user submissions are aggregated into it; filter set B represents the previous epoch and it is read-only. When checking if a domain is present, the protocol queries both sets of filters A and B. If the domain occurs in either filter, it is considered present. At the end of an epoch, the read-only filter B is discarded, filter set A is marked as read-only, and a fresh empty set for the new epoch is initialized.

In this system, once a website administrator turns off HSTS, it will disappear from the protocol after at most 2 epochs. The protocol will not produce spurious warnings for websites that have disabled HSTS, assuming the web administrators act according to the specification.

Websites with an HSTS expiry time shorter than the epoch length will not be reported, as they may expire or their administrators may disable HSTS before the epoch ends.

We note that this system effectively makes HSTS expiry time discrete. Expiry time in the protocol is either 0 (not recorded) or is equal to the epoch length. The worst case outcome of this is that users may receive less protection if a site is not visited enough between epochs for it to register. Even if the site administrator has a set a longer expiry time, it will not be represented. This however leaves users no worse off, as without the protocol they would not be able to make use of HSTS at all. Similarly, users visiting sites with expiry times lower than the epoch will not receive HSTS protection, but once again they are no worse off as they would not receive protection without the protocol.

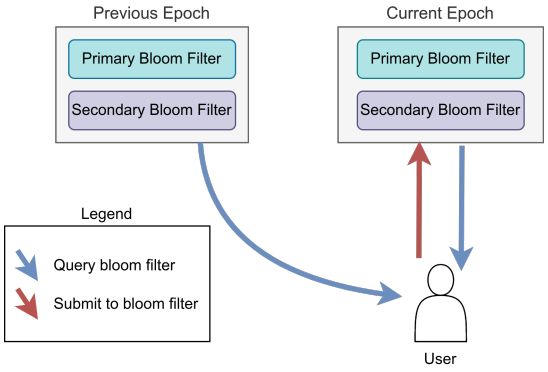

Figure 2: Protocol data is stored in two different epochs in order to model HSTS expiry times

The HSTS data we collected shows an epoch value of 30 days provides a good balance between gathering sufficient data and not recording websites with a low expiry time.

4.3.3 Adapting Protocol Parameters to Observed Data over Time. The protocol parameters can be adjusted when a new filter set is initialized at the start of an epoch to better reflect the observed data. Critically, the protocol adapts the Bloom filter sizes based on the number of submissions in the previous epoch, which is used as an estimate of submissions in the new epoch. By using the formula derived from our simulation data we can then calculate the appropriate filter sizes from the estimate.

4.3.4 Decision Function. Our protocol needs to output a boolean decision when queried with the domain of a website. In order to produce results, however, a typical RAPPOR instance fits a distribution over the full set of candidate values, which are known ahead of time. This allows for an estimation of the data that was submitted.

In contrast, our protocol works on website domains which are drawn from a theoretically unbounded set of possibilities. Practically, however, only a finite set will be visited by users of the protocol during an epoch, though the set is not known ahead of time. Only the current website being queried client-side is known. Due to these limitations, our protocol must define a more conservative decision function which estimates whether a single domain is in the set of sites reported.

In a setting without noise and no hash conflicts, checking that the count is positive would suffice. Although the mean noise introduced by differential privacy to each count is eliminated from the counts, some counts will be artificially inflated. To account for this, we define the decision function d. It takes a count of bits n, the number of insertions to the filter i, and the estimated number of distinct websites reported w and outputs a Boolean decision. Additionally, a decision multiplier c is introduced to allow for tuning the function's output to be more or less sensitive to noise.

$$d(n, i, w) = n > \frac{i}{w} \cdot c$$

The decision multiplier is set empirically based on the simulation results described in Section 5.

4.3.5 False Positives. Bloom filters are at the core of our protocol. They are probabilistic data structures which introduce a rate of false positives. This is further exacerbated by the noise introduced for differential privacy. In the case of this protocol, a false positive indication would mean a user who erroneously believes a website has enabled HSTS when, in fact, the website does not set an HSTS flag.

A false positive generates two possible situations. The website may have HTTPS available: this is the less problematic situation; while the website may never have mandated HTTPS for all its visitors, the user can still access the website over HTTPS and they will not have any interruption to their browsing experience. In fact, as Tor Browser has recently implemented HTTPS-Only Mode by default, there will be no difference to users' browsing experience for sites which implement HTTPS. We term this situation a 'don't care false positive'.

Scenario two is that the website does not, in fact, support HTTPS. This is an undesired outcome for the protocol. According to the HSTS specification, the website should now be blocked and fully inaccessible to the user. We term this outcome a significant false positive. An attempted DoS attack would aim to produce this outcome for target websites. Given that our protocol has a false positive rate, we instead insert a warning page which can be overridden by the user. This allows us to inform the user that they are likely at risk, but also gives them the option to continue if this is a clear false positive.

Careful parameter selection can reduce false positives while maintaining strong utility of the protocol in upgrading websites to HTTPS. Section 5 describes the results of simulating the protocol and, in particular, the effects varying different parameters have on the results.

4.3.6 Double Bloom Filter Construction. To further reduce false positives, CoStricTor not only records sites which report HSTS usage, but also separately records sites which do not implement HTTPS. Thus, the protocol separately collects data for domains with HSTS and domains without HTTPS in two aggregate counting Bloom filters. We term the aggregate filter that collects HSTS data as the 'primary filter'; the aggregate filter for non-HTTPS domains is termed the 'secondary filter'.

Recording complementary data is possible as websites are divided into three disjoint sets:

(1) Websites that support HTTPS and HSTS
(2) Websites that support HTTPS but do not have HSTS enabled
(3) Websites that do not support HTTPS

Note that websites that support HTTPS but do not have HSTS enabled, which are the majority of websites, do not need to be tracked as they can only produce don't care false positives. Therefore, by capturing two reduced subsets of all possible websites, we can efficiently create complementary data that can be used to detect false positives and is resilient to fake submissions.

The addition of the secondary counting Bloom filter changes the submission process by making users submit websites without HTTPS, alongside those with HSTS. The secondary filter is used for confirmation when a result of the primary filter blocks the user from accessing a non-HTTPS site, that is, when a query to the protocol indicates a website as having HSTS by querying the

primary filter but the website does not support HTTPS when the user attempts to load it. In this scenario, the secondary filter is queried. If the site is found in the secondary filter, that is, if it has been reported as having HTTP previously, CoStricTor concludes that a false positive must have indeed occurred. If the website is not found in the second filter, the protocol concludes that HSTS was present previously and that a man-in-the-middle downgrade attack is occurring. This allows for an appropriate warning to be displayed to the user.

The double filter construction also increases resistance to fake submissions. As described in 6.1.1 detection of sites in the secondary filter attackers ability to block users from sites. The second filter introduces a more refined approach to potential false positives, practically reducing the number of significant false positives, and produces a better failure state.

4.3.7 Denial of Service. All submissions to the protocol must be done through the Privacy Pass [29] mechanism, currently an IETF draft. This mechanism issues a set number of anonymous tokens after a challenge is completed. Challenges can be interactive, such as CAPTCHA, or non-interactive, such as attestation or authentication, neither of which are suitable for Tor Browser due to privacy concerns. An alternative to CAPTCHAs, which disrupt the user experience and introduce a barrier to legitimate submissions, is computational puzzles. Client puzzle introduce a minor computational cost to submissions to mitigate DoS [25]. These have the advantage of being able to run in the background and imposing a low cost to legitimate browsing patterns while an attempt at a DoS will require solving a large number of puzzles.

The Tor guard node selected by the user for the session will issue anonymous tokens, which are redeemed to submit data. This effectively rate limits users who submit to the protocol, slowing down adversaries who wish to submit bad data. It also has the additional affect of forcing users to submit to the same guard node, which limits adversaries ability to spread their attack across multiple nodes, making their attacks more detectable. The number of tokens issued for each challenge (CAPTCHA or client puzzle) can be adjusted to trade-off increased protection with worse user experience.

4.3.8 Warning Users of HSTS Failures. When CoStricTor determines that a website has HSTS based on the primary filter but a secure connection cannot be established, two possible situations arise that require informing the user. The secondary filter, which encodes the lack of HTTPS, is used to discriminate between the two situations that elicit a warning.

If a website is present in the primary filter but missing in the secondary filter, the protocol concludes that there is a risk that HTTPS was stripped. The protocol presents a full-page warning to the user similar to the browser's built-in HSTS warning. Because of the probabilistic nature of the protocol, we propose using a modified version that allows the user to continue to the site, unlike the typical HSTS warning which blocks access. It will briefly describe to the user that this warning is based on crowdsourced data and that there is a low, but non-zero possibility of errors.

In the scenario that the website is present in both filters the protocol does not take any action and the standard HTTPS-only mode warning page is displayed.

4.3.9 Reporting Bad Exit Nodes. An additional utility provided by the protocol is identifying potential bad Tor exit nodes. Whenever a site is detected as having HSTS and it does not support HTTPS, this indicates a potential SSL stripping attack. This attack may have been perpetrated by a bad exit node and thus each detection counts as some evidence an exit node is bad. By reporting the identity of every exit node involved, Tor's operators can identify bad exit nodes by the volume of reports received. Due to the random assignment of exit nodes to users, there will be a baseline level of reporting due to SSL attacks external to Tor and false positives. If exits nodes will have a proportionally higher level of their traffic reporting SSL stripping, it serves as strong evidence that the node is conducting an attack. Reports of exit node identities leak none of the users' information and thus can be reported via a Tor circuit to a centralized server operated by the Tor Project.

4.4 Protocol Specification

We now describe the behavior of our protocol, including the scenarios in which the protocol activates, as well as the exact details of how the protocol operates. Figure 1 is a flowchart of the decisions the protocol makes as a user visits a website, showing when decoding and submission occur as well as when warning pages are displayed.

4.4.1 The Query Algorithm in Detail. The initial phase of the protocol occurs at the start of a Tor Browser session. Just as the Tor Browser receives the latest consensus data from the directory authorities, the browser shall also receive the latest protocol data for detecting HSTS.

This data contains a set of 4 counting Bloom filters of length k containing noisy counts along with the count of submissions for each counting Bloom filter. The current count is necessary for the client to adjust the data to compensate for the added noise. Upon receiving this data, the browser performs the following correction for each of the 4 filters, where N is the current insertion count for that filter, c is the initial filter data, and t is the adjusted data.

$$t_i = \frac{c_i - pN}{q - p}$$

Performing this correction step compensates for the differential privacy noise added, and gives us more accurate counts. This concludes the steps required upon starting a Tor Browser session.

The main query process occurs when the user visits a website. At first, Tor Browser will automatically check if the domain is present in the HSTS preload list. If it is present, the protocol will not run as neither checking nor submitting that website will be of benefit. Otherwise, the browser will continue by running CoStricTor.

Before the request is sent to retrieve the web page, the browser checks for the presence of the domain in the primary counting Bloom filters of both epochs. To do this, the relevant counts are obtained from the filters extracting the counts that match the bits where a domain would be inserted. These approximate the number of submissions for the domain.

The protocol's decision function uses the counts to determine if they indicate the presence of the item in the aggregate counting filter.

As discussed earlier, this function is as follows:

$$d(n, i, w) = n > \frac{i}{w} \cdot c$$

where n is the count, w is an estimate of distinct websites submitted, i is the total number of insertions and c is a constant correction factor which is applied. The estimated number of distinct websites submitted is calculated by matching the number of submissions against a standard zipfian curve. As we assume all website visits follow this zipfian distribution, the estimate will be the highest rank on the curve that will appear at least once given the number of insertions.

If the website does not appear in either of the primary Bloom filters the connection continues as normal. If however, the website is present, we assume the site has HSTS. Subsequently, the browser will attempt to load the webpage via a HTTPS connection. If the website loads correctly, we need not take any further action. If the site does not load via HTTPS, the protocol must now employ the secondary counting Bloom filters. If the domain is present in either of the secondary filters, this indicates some form of false positive result. We cannot rely on the output of the protocol in this instance so we direct the user to the standard HTTPS-Only mode warning page. If the domain is not present in either of the secondary filters, we have confirmed that this website usually implements HSTS and thus the user may be at risk of a man-in-the-middle attack. We display a custom warning page to the user informing them of the risk of proceeding to the webpage.

4.4.2 The Reporting Algorithm in Detail. There are two scenarios which need to be reported in CoStricTor: the user loads a domain that has HSTS enabled or a website that does not support HTTPS. When either of the scenarios occur, the protocol checks if the domain exists in the HSTS preload list and ignores it if present. The protocol will also not report any HSTS websites that have an HSTS expiry time shorter than the length of an epoch.

A report will then only be made if the domain has not already been reported this session. This is prevent over-reporting when a website is visited many times.

To construct a domain report for either scenario, the protocol initializes an empty Bloom filter. The appropriate domain is inserted into the Bloom filter. Local differential privacy is ensured by perturbing each bit of the Bloom filter according to the privacy values p and q. This process is identical to constructing the IRR in RAPPOR.

This report is then sent through a new Tor circuit to the users *secondary guard node*. This guard node is randomly selected at the beginning of the Tor Browser session and must be distinct from the users current guard in order to prevent submissions from being linked to the user. The client constructs a new Tor circuit for every report sent to this guard node in order to stop different submissions being linked to the same user. Every report received by a guard is added to their counting Bloom filter data structure containing occurrences of each of the bits in the bit array for the current epoch. The Bloom filters corresponding to the previous epoch are read-only. As part of Tor's normal operations, guard nodes will query a directory authority server to obtain the latest consensus periodically. While obtaining the latest version of the consensus the guard node can now report their Bloom filter data structures to this directory authority, and the directory authority will provide to the guard the latest complete version of the Bloom filter data it has obtained from the other directory authorities. Directory authorities vote every hour on the latest version of the consensus and at this point they will also collectively sum the Bloom filter data they have received to obtain the definitive Bloom filter data. This version of the Bloom filter data is then distributed as part of the consensus to Tor clients, via their guard node.

5 EVALUATION

To find appropriate parameters for CoStricTor and to evaluate its effectiveness, we simulated the protocol. The simulation was based off the sample implementation of RAPPOR [20] but rewritten in Golang to favor simulation speed and adapted to our protocol design. It has been open-sourced [13].

It allows the testing of all system parameters and provides us with specific results on two key metrics: number of sites correctly upgraded by the protocol, and number of sites with incorrect warnings displayed by the protocol. The simulation parameters are: Bloom filter size, the number of reports, the number of websites from which reports can be drawn, the number of additional sites checked, and the privacy parameters p and q, from which the overall ϵ of the system is computed. The simulation uses the Majestic Million list of the one million most popular domain names.

The simulation uses a Zipf distribution to estimate the popularity of different websites which can be submitted. Informally, we know that certain websites are extremely popular and see many more visitors, whereas a huge number of sites are far less popular and are very seldom visited. In [21], the actual distribution is estimated to be a power log distribution. This is also backed up by [26] which confirms the ubiquity of power log distributions in Internet measurements, but, more specifically, the applicability of a Zipf distribution, which is a special case of power log, to website popularity. Whenever a user submission is being simulated, we draw websites from our list of domain names through a Zipf distribution so that websites appear in submission in a manner that mirrors the real world. Finally, to better simulate potential false positives which occur for websites that are never submitted, the simulation also checks an additional list of websites not submitted by any user against the counting Bloom filters produced by the protocol. For example, if we are considering 1,000 websites in the protocol, we can run a simulation over this data, and then check an additional 10,000 websites to check for false positives.

Our protocol simulation works as follows: the n most popular websites are taken from the Majestic Million list [6]. We use estimated proportions of HSTS and HTTPS sites to randomly designate websites as HSTS, HTTPS, or HTTP. We then randomly sample websites from the list with replacement. Samples simulate user submissions to the protocol. If the sampled site has been designated as having HSTS, we add this to our local Bloom filter exactly as described in the protocol, and likewise if the website was designated as having HTTP only.

After these steps, our Bloom filter data can be queried to simulate users' results when using the protocol.

5.1 Simulation Parameters

5.1.1 p, q the Differential Privacy Parameters. These parameters directly affect the amount of noise added to user responses. They must be set at a level that gives the appropriate differential privacy guarantees, while retaining utility in the submissions. We evaluate a range of different ϵ values to ascertain accuracy of the protocol with different privacy guarantees.

5.1.2 Number of Sites Modeled. The number of sites is a key number in our parameter selection. The more sites represented by the protocol, the larger Bloom filter is needed. The key difficulty is that the system can never know exactly how many sites are currently being represented. By definition of this being a privacy-preserving system it is not possible to know this number, but we can estimate it using the total number of submissions to the protocol in a period of time.

5.1.3 Estimated Number of Submissions. The number of submissions the protocol receives will vary over time. To simulate the protocol, we must provide an accurate estimation of the number of submissions that will be received in one epoch. Our simulation runs exactly one epoch of the protocol, and so we must first decide the epoch length in order to estimate the number of submissions. Epoch times that are too long may result in website administrators being unable to effectively remove their HSTS status. Epoch times that are too short provide less utility. Destroying the protocol's data very frequently means that the density of submissions for any one website will be lower, and a lower number of websites will be recorded by the protocol as having HSTS. In the simulation, we use 30 days as our epoch length. From our earlier data gathering, we determine that 83% of sites reporting HSTS use an expiry time of over 30 days. Additionally, 30 days of submissions from users provides plenty of time to allow representation of a large number of sites. From this, we can now estimate the number of submissions the protocol receives in one month. Given that the protocol is intended to be integrated into Tor Browser, we can assume all its users would use the protocol to report websites. In a period of 30 days, Tor receives approximately 90 million connections from users. We assume that each of these connections visits 1 website. This is a weak assumption as many users will visit more than one website. Therefore, over a 30-day period, we can expect approximately 90 million queries to the protocol. This is the figure used to run simulations on the protocol.

5.1.4 Estimated Proportion of HSTS Sites. Once again, the actual number of HSTS sites in the population of websites recorded in our protocol is a key figure. The proportion of HSTS sites in our population determines how much of our "estimated number of sites" affects the size of our Bloom filter. For example, we may have 100,000 sites in our population, but if only 1% deploy HSTS our Bloom filter only needs space for 1000 entries. For our simulations, we use the figure 15% which we derive from the proportion of websites with HSTS our data collection revealed.

5.1.5 Estimated Proportion of Non-HTTP Sites. As above, this proportion determines how many of the sites in the population are recorded by our secondary Bloom filter. Consequently, a higher proportion of non-HTTP sites will require a larger secondary Bloom filter. Considering the distribution of site popularity, our protocol is much more likely to deal with popular websites and less likely to be faced with less popular sites. Given that a strong majority of popular websites use HTTPS, we can expect a relatively low percentage of sites considered to have no HTTPS available. For our simulation, we estimate the proportion of non-HTTP sites to be 20% [39]. A mismatch between this estimate and real proportion would lead to a poorly sized secondary Bloom filter, affecting accuracy of false positive detection. However, a real deployment of the protocol can dynamically update the secondary filter size based on the number of submissions received in the previous epoch.

5.1.6 Bloom Filter Size. Larger Bloom filter sizes, result in fewer false positives overall. If the filter size is larger for the same number of entries, there will by definition be less overlap in the bits of the Bloom filter and fewer false positives accordingly.

5.1.7 Decision Function Correction. As discussed, the protocol requires a decision function which includes a multiplicative correction term. We estimate the correction term by running the simulation with various candidates for the correction term until we arrive at the value which produces the best results. These values are 0.045 for querying the primary filter and 0.07 for querying the secondary filter. The values scale with with the proportion of websites considered, meaning that, for the simulation, the secondary filter has a higher value but a lower actual threshold. This biases the simulation towards avoiding false positives. The result is that false positives in the primary filter can be mitigated by the secondary filter more easily.

5.2 Results of the Simulation

We run the simulation with a range of different numbers of sites being considered. In a simple case, we can consider 500 of the top websites. Using modestly sized Bloom filters we can easily provide accurate results with a high number of upgrades, and a very low number of false positives. The primary goal of our simulation was to maximize the number of sites considered without sacrificing accuracy. As we discussed, there is effectively an infinite number of sites that could be visited, but, in practice, we can expect that only a small number of sites are visited with high regularity and by many users. The higher the number of sites considered, the closer we can approximate the actual set of sites that the protocol might be asked to record in a real deployment. The primary restricting factor on this is the number of submissions made to the protocol in a certain time period.

The first task of the simulation is to establish which size of Bloom filter is appropriate for the protocol. We can see in Figure 3 the results of the simulation when considering 50,000 websites while varying the size of the filter as well as values of ϵ. At very low Bloom filter sizes performance is poor, but quickly increases to a maximum. As filter size increases beyond this, the number of upgrades slowly declines despite improving false positive rates. This can be explained by observing that, for a fixed p, the number of 0 perturbed into 1 increases roughly with the size of the filter. Increasing the size of the Bloom filter increases the overall chance that, any zero bit in a report is perturbed. This implies an optimal filter size exists for any particular setup, given an accurate estimation of

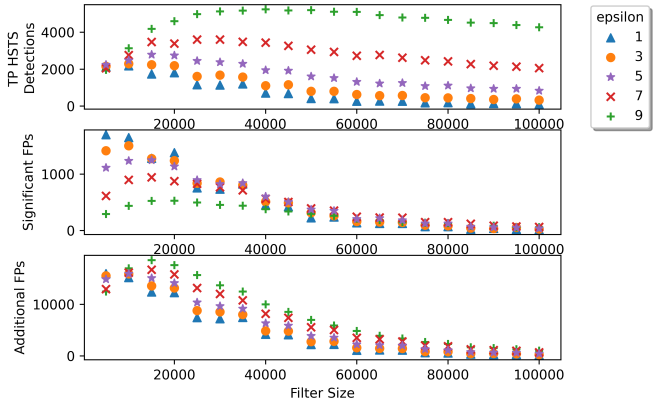

Figure 3: Increasing filter size gives more accurate results until an optimal point past which accuracy declines

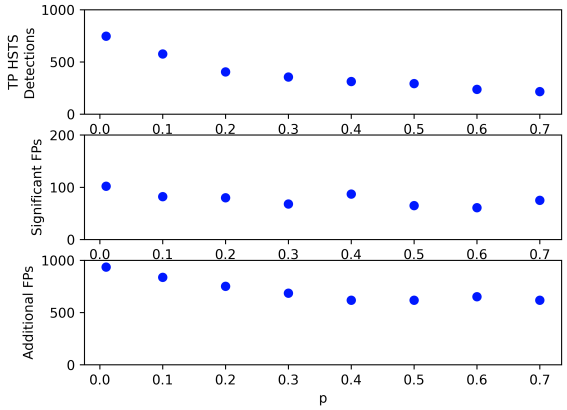

Figure 4: For the same ϵ, differing combinations of p and q produce negligible difference

the number of sites being covered and the number of submissions that will happen.

The filter size affects the amount of data which must be distributed to every Tor client as well as the size of the data which is reported to guard nodes. From Figure 3 we estimate 40,000 as a typical filter size for the system. We conducted additional simulations to extract the exact Bloom filter data of several runs of the protocol. This allows us to check the actual size of the data which needs to be transmitted. Our benchmarks show the total size of the 4 filters when transmitted is on average 130 kB for a filter size of 20,000 and 260 kB for filter size of 40,000, when compressed using the standard Brotli algorithm; the mean size of compressed submissions is 1.4 kB and 2.8 kB, respectively.

The next consideration is the privacy parameters. Differential privacy is calculated in the same way as RAPPOR: $\epsilon = \log(\frac{q(1-p)}{p(1-q)})$ from which we derive

$$p = \frac{q}{(1-q)e^{\epsilon} + q}$$

Recall that p is the probability of any true 0 bit being reported as a false 1, and q is the probability of any true 1 bit being reported as a 1. We explore the space of values of p and q for a set value of ϵ. The results of these simulations (Figure 4) show little variation, which indicates that the choice of p and q need only achieve a desired value of ϵ and that p and q are not a target for optimization for a fixed ϵ.

In order to produce a clear idea of the number of sites that we can accurately represent, we compare the maximum number of sites we can consistently represent for different values of ϵ. For this, we must also obtain the best-suited value for filter size for each combination of ϵ and number of sites being represented. We define a function f which produces the optimal value of Bloom filter size s for a given ϵ and w, the number of sites being considered, from a set of candidates sizes S, using the outputs of multiple runs of the simulation:

$$f(w, S, \epsilon) \rightarrow s$$

In this context, we can consider the simulation a function that given w and a pair of values, $s \in S$ and ϵ, produces values: u, the number of HSTS upgrades, and d, the number of HTTP sites that are falsely blocked.

If the protocol returned perfectly accurate results we could set f to return the maximum number of upgrades as long as false positives are zero, but this is, of course, impractical. By allowing a small number of false positives, we can obtain much much better results for number of upgrades. The optimal definition of f depends on the associated costs of a spurious warning page and the benefits of an HSTS upgrade. We consider finding an optimal definition for Tor Browser to be future work. Choosing a conservative definition of f can keep false positives very low, but greatly restricts the number of upgrades we can provide. Similarly, we could also define f to allow much more false positives which would allow a much greater number of upgrades but would face users with a much greater number of false warning pages.

To evaluate the protocol, we choose the following definition of f, which ensures that the number of false positives is at most a quarter of the number of upgrades.

$$f(w, S, \epsilon) = \underset{s \in S}{\text{argmax}}_u \ \text{simulation}(w, s, \epsilon), \text{ subject to } d < \frac{u}{4}$$

5.2.1 Final Results. By applying f to a range of inputs we obtain data showing the best results our protocol can produce for any value of ϵ and the number of sites being considered. In Figure 5 we can see the results from epsilon values 1–9 and sites values up to 160,000. This data is the culmination of our protocol evaluation and shows the best results we can achieve given our assumptions.

It is important to note that even in an ideal situation, the number of sites upgraded does not scale linearly with the number of sites considered. This is a result of the Zipfian distribution of popular sites, which exhibits a long tail.

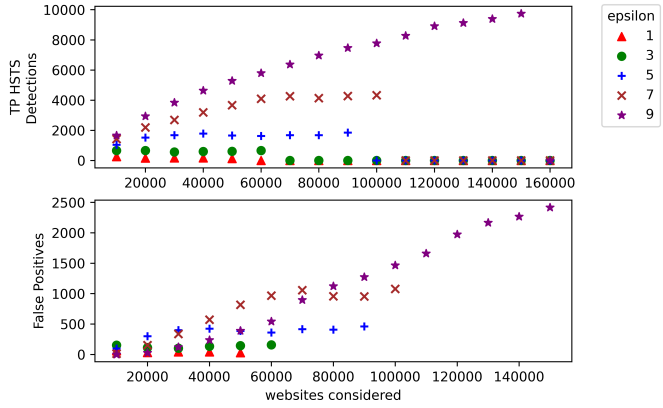

Figure 5: Depending on the value of ϵ, the protocol can only a model a certain numbers of websites before false positives are too high. In this figure an excessive ratio of false positives to upgrades reduces the upgrade number to 0.

6 DISCUSSION

The simulations show that the protocol can model up to 150,000 websites with reasonable differential privacy guarantees, and potentially much more if ϵ is relaxed further. The protocol provides 10,000 upgrades to HSTS with 2500 false positives. The ratio of upgrades to false positives can of course be reduced with a more conservative definition of the selection function f. The limiting factor is the number of submissions; with a greater number of users of the system, we can accurately model more websites which means we can more closely approximate the actual number of websites that are visited by users. The protocol will also perform better as the number of HSTS enabled sites continues to increase.

The protocol can bring quantifiable security benefits to Tor Browser users. When CoStricTor accurately presents a site as having HSTS, it provides all of the benefits that HSTS usually brings, including removing the ability for an adversary to perform SSL stripping attacks. Although Tor Browser now implements HTTPS-Only Mode by default, if an adversary attempts an SSL stripping attack, the user will be presented with a more accurate warning page indicating to them that the site once had HTTPS and no longer does, replacing the HTTPS-only mode warning which simply states that the site currently does not have HTTPS. Users will never be faced with more warning pages than they would see without using the protocol.

HTTPS-Only warnings have relatively frequent occurrence as a user browses the web and they can only provide a non specific warning to users, sites which do not implement HTTPS are not necessarily malicious and often the best course of action for the user is to click through the warning. In contrast, a HSTS warning from our protocol shows the user that there is a real danger from the site they are about to visit. The relatively low false positive rate ensures that users can have a high confidence there is a danger from this site and it is unlikely to be something they should click through.

We note that not only does our protocol overcome Tor Browser's inability to use HSTS, it also protects Tor Browser users visiting a website for the first time. HSTS, by itself, employs a 'trust on first use' paradigm where users are not protected the first time they visit a website. CoStricTor overcomes this issue since other users may have already reported the website as implementing HSTS.

The benefits of the protocol come at a cost to increased traffic on the Tor network to aggregate and distribute the filters, and additional complexity in operating a guard node. The communication cost depends on the filter size. Our benchmarks show a user needs to initially download 130 kB of data for filters of size 20,000 or 260 kB of data for filters of size 40,000, as these sizes provide adequate performance. For comparison, the compressed size of the consensus is approximately 580 kB. Although this increases the size of the initial download, the filter set for the previous epoch is read-only and non-specific, therefore can be cached across sessions, halving the size of subsequent downloads to update the filter set for the current epoch. The overall increase in Tor network traffic from submissions and filter downloads is not significant compared to the total traffic over the Tor network which is over 250 Gbit/s [36], although it increases the load on directories and directory mirrors. Additional traffic between guard nodes and directory authorities is also relatively minor.

6.1 Limitations

6.1.1 DoS Attacks Can Render the Protocol Ineffective. The CoStricTor protocol is resilient to some forms of denial-of-service attacks. Opportunistic attackers will be deterred by the Privacy Pass mechanism. Attackers with more resources can still overcome this rate limit and attempt to perform a DoS attack.

There are two potential attacks the CoStricTor system. Attackers can either submit false entries to the primary Bloom filter, or to the secondary Bloom filter. Erroneous entries in the primary filter can make users avoid visiting certain websites by producing false warning pages on non-HTTPS websites. This attack can only be successful if the website being targeted has not been sufficiently submitted to the secondary filter, otherwise any false results generated from the primary filter will be disregarded, which makes it ineffective against popular websites. Erroneous entries in the secondary Bloom filter can negate the utility of the protocol and potentially allow SSL stripping attacks to occur. This second kind of attack cannot be avoided, but is made more difficult by the implementation of the Privacy Pass mechanism as described earlier. If an attacker sends many false submissions to the secondary filter about a website, the protocol will believe that this website does not implement HTTPS and therefore any hits in the primary filter are false positives. The Privacy Pass mechanism means that an adversary must complete a challenge to make submissions. Although the number of submissions that can be made per challenge solved can be varied, we estimate that 20 submissions would be sufficient for a typical browsing session. Using our earlier assumptions about Tor user activity, we estimate that in order to cause a website to appear in the secondary Bloom filter, 315,000 submissions are needed to be made. Given 20 submissions and assuming CAPTCHA challenges, we calculate that an adversary must complete 15,750 CAPTCHAs in order to perform this attack. At a rate of $2.99 per 1000 CAPTCHAs [2], it would cost an adversary $47.10 to perform the attack. A secondary filter that was entirely full of false reports

would reduce the protocol's utility to zero. In this scenario, no utility is gained from the protocol, but crucially users are no worse off than if the protocol was not present. An attack on the primary Bloom filter would require 202,500 submissions and would cost approximately $30.27. Any attack performed would only be effective for a single epoch and would need to be repeated after it expires. This additional cost must be borne by the attacker to perform a SSL stripping attack whereas previously without the use of the protocol there was no additional cost or barrier for performing attacks.

Malicious guard nodes could also be used to perform these attacks by avoiding the Privacy Pass restrictions. Currently, there are approximately 3,000 guard nodes, this means that any one malicious guard node can only alter approximately $\frac{1}{3000}$ of the protocols data, giving them limited power over the protocol. Similarly, a malicious directory authority has the ability to have approximately $\frac{1}{9}$ effect on the protocol. Large deviations from the expected data, such as a large spike for a single value, can be detected and discarded, helping mitigate harm to the protocol by limiting the impact of malicious nodes and DAs.

6.1.2 Only Effective for Very Popular Sites. As we aim for a strong level of differential privacy for users of the system, the protocol by definition cannot show a site as having HSTS unless there are a large number of reports made. Additionally, as we have discussed, the Zipfian distribution of website visits means that the most popular websites receive the vast majority of traffic, and the number of visits falls dramatically as we consider more websites. The result of both of these factors means that only a certain number of popular websites can be modeled by the protocol. Regardless of any other parameters we use for the protocol, the number of sites we can positively report as having HSTS is limited by the level of differential privacy we require and the popularity of websites we want to represent. This also means that there is no risk that users will be identified by their visiting a very unpopular/niche site, as there will not be enough reports of this site to cause it to be represented by the protocol.

6.2 Future Work

6.2.1 Real-world Evaluation. To establish the viability of the protocol, it was evaluated through a set of simulated scenarios. A full evaluation of the protocol would require real-world usage data. Although the ultimate aim, deploying the protocol in Tor Browser without real-world testing is not practical. As an intermediate step, we aim to create a web browser add-on that runs the protocol. This will enable us to test different aspects of the system. For example, does the protocol introduce any noticeable increase in latency as the system computes the HSTS status of websites?

6.2.2 Warning Pages. This system introduces additional warning pages which can be shown to users as they browse the web. There is significant research on creating effective warnings for common web browsing scenarios [3, 18, 19, 32]. SSL/TLS errors cause the browser to display well-studied, thoughtful, and effective warnings which accurately inform the user and enable them to make the best decision. The new warnings introduced by the protocol must also conform to this high standard and we wish to perform a user study to show users are correctly informed to make the correct decisions.

6.2.3 Appropriate Proportions of False Positives. The CoStricTor protocol provides HSTS data to users which is correct with a certain probability. False positives can always occur, however, there are also trade-offs which must be decided when implementing the protocol. We outlined the function f which decides the best Bloom filter size for a given set of other parameters. We aim to configure f for a real Tor Browser deployment by conducting a qualitative survey of Tor experts. This work will allow us to set a value that provides the most benefit for users, whilst also not burdening users with tedious warning pages. This is crucial not just for user experience, but also to avoid introducing warning fatigue which can devalue the usefulness of our warning pages if they are displayed too often.

7 RELATED WORK

Some attempts have been made in the literature to adapt RAPPOR for different purposes. In particular, in [17] the authors attempt to use RAPPOR with an unknown set of strings through the reporting of n-grams of submitted strings. This technique, while effective, is unnecessary in our scenario, since our candidate strings become known when the user needs to query the protocol.

Private data collection in browsers has seen significant research. Prio [10], in use in Mozilla Firefox, uses multiple partially trusted servers for storing shares of the reported data, which can then be aggregated to reveal the aggregate data. Prio is limited to only encoding numerical data which can be aggregated between servers. Prochlo [7] is the successor to RAPPOR in Google Chrome designed to improve utility on many types of queries. Prochlo's design is a departure from RAPPOR as it has a three-step pipeline, with each step intended to be performed by a separate entity. STAR [12] is a system for threshold data aggregation intended for use in the Brave browser. Each item becomes available to the collector only if it is submitted at least k times. The threshold is based on k-of-n secret sharing of an encryption key derived from the data using a verifiable oblivious pseudo-random function (VOPRF) run by a third party. Similarly to CoStricTor, STAR can use the TOR network to prevent linking submissions to individuals.

8 CONCLUSIONS

CoStricTor, the protocol we have presented provides real value for Tor Browser users by allowing users to share HSTS data for a large number of popular websites, introducing these protections where they were previously unavailable. This protects against malicious Tor exit nodes attempting to downgrade connections and tamper with data, where previously Tor Browsers' lack of HSTS support left users vulnerable to attacks. We show that false positive rates are low and users will be infrequently interrupted by erroneous warning pages. The protocol also maintains user privacy which is a key concern of Tor users. Our simulations show that accurate results can be given for over 150,000 websites with 10,000 HSTS upgrades, depending on the level of differential privacy required. We also provide a valuable detection system for bad Tor exit nodes, preventing future attacks and assisting with the health of the Tor network as whole.

ACKNOWLEDGMENTS

This work was supported by EPSRC (EP/R045178/1 Human Data Interaction: Legibility, Agency, Negotiability). Killian Davitt and Steven J. Murdoch are funded by the Royal Society (RGF\EA\80191 and UF160505), Dan Ristea is funded by UK EPSRC grant EP/S022503/1 supporting the CDT in Cybersecurity at UCL. Duncan Russell is employed by the Tor Project. We also wish to thank the anonymous reviewers as well as the anonymous shepherd for their guidance.

REFERENCES

[1] Isabela . 2020. Tor security advisory: exit relays running sslstrip in May and June 2020 | Tor Project. https://blog.torproject.org/bad-exit-relays-may-june-2020/
[2] 2captcha.com. 2023. Captcha prices. https://web.archive.org/web/20230818150417/https://2captcha.com/pricing
[3] Devdatta Akhawe and Adrienne Porter Felt. 2013. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, Boston, MA, 257–272. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/akhawe
[4] Al Teich, Mark S. Frankel, Rob Klin. 1999. Anonymous Communication Policies for the Internet: Results and Recommendations of the AAAS Conference. *The Information Society* 15, 2 (May 1999), 71–77. https://doi.org/10.1080/019722499128538
[5] Alexander Færøy. 2023. Tor directory protocol, version 3. https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt
[6] Majestic Analytics. 2022. Majestic Million. https://majestic.com/reports/majestic-million
[7] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, Shanghai China, 441–459. https://doi.org/10.1145/3132747.3132769
[8] Chromium. 2022. HSTS Preload List Submission. https://hstspreload.org/
[9] Chromium Design Documents. 2018. Rappor (Randomized Aggregatable Privacy Preserving Ordinal Responses). https://web.archive.org/web/20180309120825/https://www.chromium.org/developers/design-documents/rappor
[10] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)}*. USENIX Association, Boston, MA, 259–282. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs
[11] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (June 2018), 164–180. https://doi.org/10.1515/popets-2018-0026
[12] Alex Davidson, Peter Snyder, E. B. Quirk, Joseph Genereux, Benjamin Livshits, and Hamed Haddadi. 2022. STAR: Secret Sharing for Private Threshold Aggregation Reporting. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Los Angeles, U.S.A., 697–710. https://doi.org/10.1145/3548606.3560631 arXiv:2109.10074 [cs].
[13] Killian Davitt and Dan Ristea. 2023. CoStricTor. https://github.com/KillianDavitt/CoStricTor
[14] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The Second-Generation Onion Router:*. Technical Report. Defense Technical Information Center, Fort Belvoir, VA. https://doi.org/10.21236/ADA465464
[15] Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming (Lecture Notes in Computer Science)*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.). Springer, Berlin, Heidelberg, 1–12. https://doi.org/10.1007/11787006_1
[16] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Scottsdale Arizona USA, 1054–1067. https://doi.org/10.1145/2660267.2660348
[17] Giulia Fanti, Vasyl Pihur, and Úlfar Erlingsson. 2016. Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries. *Proceedings on Privacy Enhancing Technologies* 2016, 3 (July 2016), 41–61. https://doi.org/10.1515/popets-2016-0015
[18] Adrienne Porter Felt, Alex Ainslie, Robert W. Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. 2015. Improving SSL Warnings: Comprehension and Adherence. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, Seoul Republic of Korea, 2893–2902. https://doi.org/10.1145/2702123.2702442
[19] Adrienne Porter Felt, Robert W. Reeder, Hazim Almuhimedi, and Sunny Consolvo. 2014. Experimenting at scale with google chrome's SSL warning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Association for Computing Machinery, New York, NY, USA, 2667–2670. https://doi.org/10.1145/2556288.2557292
[20] Google. 2017. rappor implementation. https://github.com/google/rappor
[21] Benjamin Greschbach, Tobias Pulls, Laura M. Roberts, Philipp Winter, and Nick Feamster. 2017. The Effect of DNS on Tor's Anonymity. In *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. https://doi.org/10.14722/ndss.2017.23311
[22] Jeff Hodges, Collin Jackson, and Adam Barth. 2012. *HTTP Strict Transport Security (HSTS)*. Request for Comments RFC 6797. Internet Engineering Task Force. https://doi.org/10.17487/RFC6797 Num Pages: 46.
[23] Internet Watch Foundation. 2022. The Anual Report 2022. (2022).
[24] Eric Jardine, Andrew M. Lindner, and Gareth Owenson. 2020. The potential harms of the Tor anonymity network cluster disproportionately in free countries. *Proceedings of the National Academy of Sciences* 117, 50 (Dec. 2020), 31716–31721. https://doi.org/10.1073/pnas.2011893117
[25] Ari Juels and John Brainard. 1999. Client Puzzles: A Cryptographic countermeasure against connection depletion attacks. In *Proceedings of NDSS '99*. 151–165.
[26] Aniket Mahanti, Niklas Carlsson, Anirban Mahanti, Martin Arlitt, and Carey Williamson. 2013. A tale of the tails: Power-laws in internet measurements. *IEEE Network* 27, 1 (Jan. 2013), 59–64. https://doi.org/10.1109/MNET.2013.6423193
[27] Mozilla. 2018. Web Security. https://infosec.mozilla.org/guidelines/web_security
[28] nusenu. 2020. How Malicious Tor Relays are Exploiting Users in 2020 (Part I). https://nusenu.medium.com/how-malicious-tor-relays-are-exploiting-users-in-2020-part-i-1097575c0cac
[29] Tommy Pauly, Steven Valdez, and Christopher A. Wood. 2023. *The Privacy Pass HTTP Authentication Scheme*. Internet Draft draft-ietf-privacypass-auth-scheme-10. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-privacypass-auth-scheme Num Pages: 23.
[30] Mike Perry, Erin Clark, Steven Murdoch, and Georg Koppen. 2018. The Design and Implementation of the Tor Browser [DRAFT]. https://2019.www.torproject.org/projects/torbrowser/design/
[31] J. V. Roig and Eunice Grace Gatdula. 2019. HSTS Preloading is Ineffective as a Long-Term, Wide-Scale MITM-Prevention Solution: Results from Analyzing the 2013 - 2017 HSTS Preload List. https://doi.org/10.48550/arXiv.1905.04436 arXiv:1905.04436 [cs].
[32] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. 2009. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *18th USENIX Security Symposium*. USENIX Association, Montreal, Canada, 18.
[33] Paul Syverson and Matthew Traudt. 2018. HSTS Supports Targeted Surveillance. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)}*, Vol. 8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)}. USENIX Association, Baltimore, MD. https://www.usenix.org/conference/foci18/presentation/syverson
[34] The ZMap Team. 2022. ZGrab 2.0. https://github.com/zmap/zgrab2
[35] Tor Browser. 2022. Releases: Tor Browser 11.5. https://www.torproject.org/releases/tor-browser-11-5/
[36] Tor Metrics. 2023. Traffic. https://metrics.torproject.org/bandwidth-flags.html?start=2022-04-26&end=2023-06-12
[37] Tor Project. 2023. Bad relays. https://community.torproject.org/relay/community-resources/bad-relays/
[38] United Nations Human Rights Council. 2015. *Report of the Special Rapporteur on the promotion and protection of the right to freedom of opinion and expression*. Technical Report A/HRC/29/32. United Nations. https://ap.ohchr.org/documents/dpage_e.aspx?si=A/HRC/29/32
[39] W3Techs. 2023. Usage Statistics of Default protocol https for Websites, May 2023. https://w3techs.com/technologies/details/ce-httpsdefault
[40] S. L. Warner. 1965. Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias. *J. Amer. Statist. Assoc.* 60, 309 (March 1965), 63–66. https://doi.org/10.1080/01621459.1965.10480775 Place: United States.