Bypassing Tor Exit Blocking with Exit Bridge Onion Services

Zhao Zhang
Georgetown University

Wenchao Zhou
Georgetown University

Micah Sherr
Georgetown University

ABSTRACT

Tor exit blocking, in which websites disallow clients arriving from Tor, is a growing and potentially existential threat to the anonymity network. This paper introduces HebTor, a new and robust architecture for *exit bridges*—short-lived proxies that serve as alternative egress points for Tor. A key insight of HebTor is that exit bridges can operate as Tor onion services, allowing any device that can create outbound TCP connections to serve as an exit bridge, regardless of the presence of NATs and/or firewalls. HebTor employs a micropayment system that compensates exit bridge operators for their services, and a privacy-preserving reputation scheme that prevents freeloading. We show that HebTor effectively thwarts server-side blocking of Tor, and we describe the security, privacy, and legal implications of our design.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; *Network security*; *Software and application security*; • **Networks → Network privacy and anonymity**; *Layering*; *Network protocol design*; *Security protocols.*

KEYWORDS

Tor; Exit; Bridge; Server-side; Blocking

ACM Reference Format:
Zhao Zhang, Wenchao Zhou, and Micah Sherr. 2020. Bypassing Tor Exit Blocking with Exit Bridge Onion Services. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372297.3417245

1 INTRODUCTION

Researchers have long focused on understanding how censors block access *to* anonymity networks [5, 18, 45, 52, 57] and how to best thwart such efforts [6, 12, 16, 17, 26, 27]. Generally, the focus has been on nation-state censors [2, 8] and the techniques they employ to enumerate anonymous relays, distinguish routes that traverse decoy routers, and more generally, curtail unfettered access to the Internet.

Separate from the arms-race that is occurring on the ingress side of anonymity networks—that is, efforts to prevent users from accessing the uncensored Internet—there is the symmetrical case of blocking access *from* anonymity networks. For anonymity networks that use proxies (i.e., relays) to forward their users' traffic, such blocking is trivially achieved by enumerating and preventing access from the anonymity network's egress points (e.g., Tor exit relays). In particular, in the case of Tor, its exit points are publicly advertised.

Tor exit blocking is becoming increasingly common [28, 37, 60], with as many as 20% of popular websites discriminating against users arriving from the Tor network [47]. Although there is anecdotal evidence that Tor transports a disproportionate share of malicious and otherwise unwanted traffic [32], perhaps surprisingly, there is also significant circumstantial evidence that Tor is often not specifically targeted by sites and is unwittingly blocked. In particular, a number of DNS and IP blacklists indiscriminately list (nearly) all Tor relays, including non-exits [28, 37, 47]. Sites that use such blacklists to filter out requests (or equivalently use hosting providers that subscribe to such blacklists) will thus block Tor by default.

The appearance of Tor relays on blacklists presents a potentially existential threat to the network. As more sites and (worse) web hosting providers prevent access from Tor, less of the Internet becomes accessible to Tor's users, making the network increasingly ineffective at providing anonymous browsing and/or censorship circumvention. The consequences of this threat are not entirely hypothetical: to the great annoyance of the maintainers of the Tor Project [33], Cloudflare inserted CAPTCHAs on their hosted sites to users who arrived via the Tor network. (Through some clever engineering [10, 44], Tor users no longer have to solve multiple CAPTCHAs for Cloudflare-hosted websites.) This is indicative of a critical threat to Tor: a wide-scale and unmitigated adoption of a blacklist that contains Tor relays could effectively cripple the utility of the anonymity network.

We recently proposed the notion of ephemeral *exit bridges* for Tor [60] and envisioned short-lived proxies hosted by popular cloud providers such as Amazon or Google. These proxies serve as additional hops in Tor circuits: traffic would exit Tor through existing exit relays, and be further tunneled through these cloud-hosted proxies towards their intended destinations. However, this approach has several practical limitations as it relies on *centralized cloud-based* exit bridges. In short, the service can be easily taken down by the cloud provider.

This paper presents a new, practical, and more distributed architecture for Tor exit bridges to better mitigate the threat of server-side blocking. Rather than rely on centralized cloud-based exit bridges, our architecture uses a network of volunteer-run ephemeral exit bridges that operate for short periods and can be hosted on any network. An important advantage of our architecture is that it makes distinguishing non-anonymous users from exit bridge users far more difficult, since both sets of users can arrive from residential networks, for example.

There are a number of challenges in constructing exit bridges for Tor: (1) ordinary Internet users should have realistically strong incentives to run exit bridges on their computers so that such a

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417245

scheme might be practically and widely deployed; (2) the exit bridge architecture should not make it easier for an adversary to attract a disproportionate share of Tor egress traffic than operating its own exit relay; and (3) more generally, the exit bridge architecture should not inflict additional risks to its users' anonymity over those that are already imposed by Tor.

Our approach towards meeting these challenges is to operate exit bridges as Tor onion services.[1] We call our onion-based exit bridge solutions HebTor (for hidden exit bridges). The volunteer-operated HebTor exit bridges accept connections from end-users via Tor's existing onion services protocol, and then forward the traffic to/from the users' requested destination. In brief, exit bridges are SOCKS proxies, operating as Tor onion services. Because Tor's onion service uses its own rendezvous protocol, exit bridges need not be publicly accessible and can exist behind NATs or firewalls. So long as a device is able to create outbound TCP connections, it can serve as an exit bridge.

We emphasize that *HebTor is not intended to conceal the network addresses of the exit bridges.* Exit bridges reveal their IP addresses whenever they establish TCP connections to the requested destinations. We use Tor's onion services primarily to (1) take advantage of their NAT-piercing properties and (2) avoid making changes to Tor's core protocols.

As a strong incentive to operate an exit bridge, HebTor allows exit bridge operators to earn money. We borrow from our previous design [60] and present simple tasks (e.g., short image labeling problems) that must first be solved in order for a Tor user to use an exit bridge. In short, we trade off a halfdozen seconds of work in favor of *access* (since the requested site would otherwise be inaccessible). Completing these tasks earns micropayments which are transferred to the exit bridge operator.

Unsurprisingly, our design presents a number of interesting technical challenges, including (but not limited to) pairing Tor users with bridge operators, ensuring proper payment and verification, and preventing cheating either by the exit bridge or by the client, all while not endangering the anonymity of the Tor user or the unlinkability of her actions.

2 BACKGROUND

Tor is a network of approximately 6700 volunteer-operated *relays* (i.e., servers) that provides anonymous TCP connections [12, 50] to an estimated eight million daily users [30]. Most commonly, users access Tor through the Tor Browser, which bundles the Tor client software with a modified version of Firefox. Relays also run Tor to manage connection state and handle packet forwarding.

To provide sender anonymity, Tor constructs source-routed paths of (usually) three relays called *circuits*. The ingress point of the anonymity network is typically a Tor *guard* relay, which a user randomly selects and uses consistently for a long period [11]. The second and third hops in a circuit are respectively the *middle* and *exit* relays. Exit relays serve as the egress points of Tor. Relay operators can (and often do) opt not to serve as exits. (We discuss the potential legal ramifications of operating an exit relay in Appendix A.)

Tor circuits use layered encryption to hide the endpoints of anonymous communication. The client's Tor instance agrees on cryptographic keys with each relay, using a telescoping approach to tunnel client-to-relay communication through already-established portions of the circuit.

Onion services. Tor also enables receiver anonymity. Here, a server can receive incoming connections via Tor that are addressed to its *onion address* (a .onion URL) without having to expose its actual network location (i.e., its IP address). To run an onion service, Tor software running on the server selects a number of relays as *introduction points*, and constructs an *onion service descriptor* that lists these introduction points as well as the onion service's public key. The onion service descriptor is then uploaded to a distributed hash table (stored amongst the Tor relays), indexed with the server's onion address, which is derived from the server's public key.

A user with knowledge of the service's .onion address can then retrieve the onion service descriptor, fetched using the distributed hash table via an anonymous Tor circuit. The user then selects a relay of its choosing as a *rendezvous point* (RP) and transmits a one-time secret to the RP via an anonymous Tor connection. The user then sends a message to an introduction point, informing it of the chosen RP and the one-time secret. The introduction point forwards this information to the onion service. Finally, the onion service creates a Tor circuit to the RP along with the one-time secret. Similarly, the user creates a Tor circuit to this same RP, using the identical one-time secret. The RP then relays all further communication between the two communicating parties. Critically, all communication is carried out over Tor circuits, enabling both the client and the server to conceal their respective network locations [51].

Blocking Tor traffic. Tor does not attempt to hide the identities of its exit relays: their IP addresses are publicly advertised by the Tor directory servers and the Tor-operated ExoneraTor service [49] provides a queryable interface of historical records of current and former exit relays. Identifying and blocking traffic from the Tor network is thus trivial, since all Tor traffic must traverse through these egress points.

A growing number of sites either block Tor or discriminate against traffic originating from Tor (for example, by serving CAPTCHAs). The Tor Project catalogues server-side attempts to block Tor—which now number around 300 websites—and identifies several third-party blacklists (e.g., abuseat.org, akismet, and blocked.com) that include the IP addresses of Tor exit relays [37]. Tor contributors have also found instances in which popular hosting providers (namely, Akamai, Bluehost, Incapsula, and Convio) implement Tor-blocking features for some (importantly, not all) of the client websites that they host [37]. In their 2016 study, Khattak et al. confirm that Tor exit blocking is fairly common, with nearly 4% of Alexa top-1000 sites preventing access from Tor [28]. In a follow-up 2017 study [47], Singh et al. find much higher block rates (approximately 20% of tested websites) and identify more than 80 blacklists that contain Tor exits.

Risks of operating an exit bridge or relay. Traffic exiting an exit bridge could be misattributed as originating from the bridge's operator. We survey the current legal landscape concerning the risks and liabilities of operating Tor exit relays in Appendix A, with a bias towards United States and European law. In brief, the current legal consensus seems to be that "safe harbor" provisions exist in both American and European legal systems, providing prosecutorial

[1] Onion services were previously called hidden services.

and civil immunity for operating services (such as Tor) that merely forward traffic. A more in-depth discussion of the legal risks is provided in the Appendix.

3 RELATED WORK

As discussed above, the Tor Project [37], Khattak et al. [28], and Singh et al. [47] all measure the occurrence of Tor exit blocking, with the latter study finding that such blocking is rampant [28].

Rather than block Tor access outright, Cloudflare instead chose to require Tor users to first complete CAPTCHAs, in an attempt to allow Tor's human users to access their hosted sites while stemming the use of automated scripts that employ Tor [32]. Because the Tor Browser does not by default allow third party cookies, a new CAPTCHA had to be solved for each visited Cloudflare-hosted site. Privacy Pass [10] removed the annoyance of having to solve multiple CAPTCHAs by using a 1-RTT cryptographic protocol to issue a large number of tokens to a Tor user after completing a CAPTCHA. These tokens could then be used to bypass further Cloudflare-imposed CAPTCHAs [9]. Privacy Pass requires the cooperation of the site and/or the site's hosting provider, and does not prevent the inclusion of exit relay IPs on blacklists.

Sites and hosting providers are able to block Tor exits since the anonymity network funnels its egress traffic through a well-defined and enumerable set of exit relays. Peer-to-peer anonymity systems [42, 46] inherently offer greater resistance to server-side blocking since their exit points are distributed across all (or many) of its users, and thus are more difficult to catalogue. Our focus however is on Tor, given its enormous user base [30, 50]. We also note that I2P [58, 62], the only widely deployed peer-to-peer anonymity system, does not directly transit traffic to non-I2P websites and, like Tor, it also has to rely on fixed exit points to reach the public Internet. Conceptually, HebTor can be viewed as a method of providing a distributed egress architecture for Tor.

A number of incentive schemes have been proposed to increase the number of relays in the Tor network. PAR rewards relay operators with virtual coins that they can spend to form fast paths through the Tor network [1]. The "gold star" system similarly rewards relay operators with improved quality-of-service guarantees. BRAIDS [21] uses a partially trusted offline bank to issue tickets to relay operators, which again can be later used to achieve increased performance. And Tortoise [36] provides incentives by enforcing rate limits on Tor to all users except those who run relays. These approaches all impose serious privacy risks, since they significantly decrease the size of the anonymity set of potential senders of "fast" traffic to just those who operate relays. Even if successful at increasing the number of exit relays in the network, Tor includes all relay information in publicly available consensus documents, allowing blacklist maintainers and site operators to easily identify them.

We also provide incentives to increase the size of the Tor network. Compensating bridge operators with cryptocurrency or fiat currency could offset operators' bandwidth costs and justify the risk of forwarding anonymous traffic. Existing proposals in which relay operators can receive proof-of-work (PoW) hashes for cryptocurrency mining from Tor users seem promising. Unfortunately, given the increasing difficulty of mining cryptocurrency [4], it is unclear that such an approach would be profitable for relay operators. TorCoins [19] have been proposed as a new, Tor-focused alt-coin cryptocurrency based on proof-of-bandwidth. Unlike TorCoin, we do not require changes to the core Tor protocol.

Liu et al. define a system of server-specified access controls for Tor, called TorPolice [29]. In TorPolice, *access authorities* issue anonymous capabilities to users after the users complete some proof-of-work (e.g., a CAPTCHA). Users then expend these capabilities to obtain better service. However, unlike HebTor, TorPolice requires the active participation of sites, which we seek to avoid.

We previously proposed an exit bridge architecture for Tor [60] that relies on bridges that are hosted by popular cloud service providers. That design argued that blocking resistance is achieved due to the high collateral-damage of blocking exit bridges: since exit bridges reside in the same IP address ranges as remote desktop-as-a-service offerings (e.g., Amazon's Workspaces), blocking cloud IP addresses *en masse* would also result in the blocking of potential website visitors that used cloud-based remote desktops. However, that approach is dependent upon the cooperation of the cloud provider: a disapproving cloud provider could immediately and entirely disrupt the exit bridge infrastructure by suspending the accounts that operate the bridges.

This paper presents a very different design for exit bridges that removes all reliance on cloud providers. We achieve greater blocking resistance by allowing almost any Internet-connected device to function as an exit bridge, thus making the bridges more difficult to enumerate.

We introduce a novel, privacy-preserving reputation system to enable Tor users to select reputable exit bridges. Anonymous reputation systems have been well studied, with such notable examples as AnonRep [59] and EigenTrust [25]. Unfortunately, we know of no existing privacy-preserving reputation system that is compatible with our requirements, in which a set of anonymous users must collectively form reputation scores for a separate set of exit bridges.

4 OVERVIEW

We begin our presentation of HebTor by introducing our threat model and system goals (§4.1) and describing our intuitions (§4.2), and high level design (§4.3).

4.1 Threat Model and System Goals

We adopt Tor's threat model [12] and consider a non-global adversary that is able to control some portion of the Internet and optionally participate in Tor as a malicious insider, but is not able to observe the entirety of the Tor network. We extend this threat model to cover the various components of HebTor, which we describe in subsequent sections.

HebTor is not designed to strengthen Tor's resiliency to known attacks such as traffic correlation [38, 40], hidden service de-anonymization [3], and denial-of-service (DoS) [22, 23]. While we adopt Tor's threat model, we also inherit the network's existing limitations and vulnerabilities. Put more positively, HebTor also benefits from improvements to the core Tor network. Unless exacerbated by HebTor, we view existing attacks against Tor as orthogonal to HebTor and do not consider them in this paper.

Unlike Tor, we also consider a secondary adversary that attempts to prevent Tor users from accessing a website. This *blocking adversary* could be the website operator, a website's hosting provider, or a contracted firewall service or tool that blocks IPs that appear on a blacklist. Here, the blocking adversary's goal is not necessarily to de-anonymize the requesting user (although HebTor should certainly maintain the anonymity offered by Tor), but rather to discriminate against users arriving from the anonymity network.

Finally, we consider malicious HebTor participants who attempt to game the system either to (1) attract a disproportionate share of exit traffic (e.g., to increase its ability to perform traffic correlation attacks) or (2) earn compensation without performing the requisite traffic forwarding. We note that the former is an attack on anonymity, while the latter targets HebTor's incentive system.

We do not consider external attackers who attempt to disrupt HebTor by conducting DoS against its infrastructure. We note here, however, that such attacks are likely far more difficult to carry out against HebTor than most other Internet services, since HebTor's infrastructure operates as onion services and thus is accessible only through the Tor network. In short, HebTor benefits from the denial-of-service protections provided to Tor onion services.

System goals. HebTor should achieve the following high-level goals:

- *Usability:* Tor users should be able to use HebTor exit bridges without significantly changing how they currently use Tor.
- *Anonymity and unlinkability:* HebTor should not degrade Tor users' anonymity or unlinkability [41].
- *Unblockability:* It should be difficult for a blocking adversary to either enumerate exit bridges or otherwise discriminate against HebTor users.
- *Low overhead:* The use of HebTor should not incur significant performance penalties.
- *Openness.* The system should impose few requirements to operate an exit bridge, allowing most Internet-connected devices to participate as bridges.

4.2 Intuitions

Before presenting the technical aspects of HebTor, we briefly present some of the main intuitions that motivate the system's design.

Server-side blocking of Tor depends on exit enumeration. A primary goal of HebTor is to make it difficult to enumerate its exit points. Blocking Tor's exit relays is fairly straightforward since Tor publishes the network addresses of all of its relays (excluding traditional Tor bridges that serve as alternative ingress points into the network). There are many reasons why publishing a list of relays is desirable (e.g., to enable source routing); however, doing so also makes it trivial to perform exit blocking. A main intuition behind HebTor is that it is much more difficult to block exit points that are (1) ephemeral and (2) resistant to enumeration.

Relatedly, exit points (i.e., exit bridges) that are located in the same autonomous systems and IP ranges as ordinary Internet users (e.g., residential networks, corporate networks, college campuses, etc.) are especially difficult to identify, since these are the same network locations that ordinarily originate web requests. In contrast, exit *relays* that are located on cloud-hosted virtual private servers

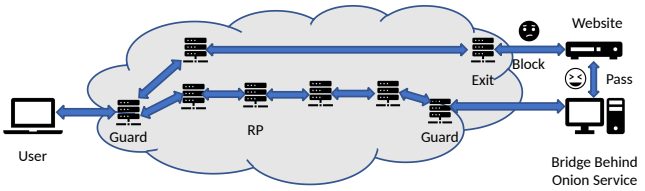


Figure 1: Two attempts to connect to a website that blocks Tor exits. *Top path:* A connection via a traditional Tor circuit, which is blocked by the website. *Bottom path:* A connection via a HebTor exit bridge, which operates both as an onion service and SOCKS proxy.

are fairly easy for a site to identify, as most (but certainly not all[2]) client traffic does not originate from such networks.

Removing barriers will increase exit capacity. Only a small fraction of Internet-connected devices are qualified to serve as Tor exit relays. Exit relays must have static IPs and be publicly accessible; they must be able to accept incoming TCP connections. HebTor is designed to remove such barriers, and enables nearly any Internet-connected device to serve as an exit bridge. (More concisely, the device must be able to create outbound TCP connections and connect—either directly or through a Tor bridge on the ingress side—to the Tor network.) Here, we make use of Tor's onion services, which due to its rendezvous protocol, enables any computer running the Tor software to function as an onion service.

Relative to traditional Tor exit relays, exit bridges require far less bandwidth capacity since their use is required only for sites that block Tor. The Tor Metrics Portal reports that the average Tor user's throughput is approximately 0.200 Mbps [50]. A moderately-provisioned ISP offers 100Mbps/100Mbps, and thus a single exit bridge hosted on such a network could support 500 simultaneous clients, which conservatively assumes all such clients are constantly communicating at their maximum rate. In general, we believe that exit bridges pose little threat of adding congestion to Tor, since the Tor network itself is much more likely to impose a performance bottleneck. Additionally, should a particular exit bridge offer poor performance, Tor's native congestion control mechanisms will prevent it from causing congestion in the core Tor network. More generally, by offloading some egress traffic onto additional infrastructure, the introduction of exit bridges increases the overall exit capacity of the Tor network.

Incentives help. HebTor provides incentives for users to operate exit bridges. In brief, the more Tor users who use an operator's exit bridge, the more money is earned by that bridge operator. Unlike Tor exit relays, since HebTor supports short-lived ephemeral exit bridges, users may be willing to operate bridges during off-hours or, if ample bandwidth exists, throughout the day. By compensating exit bridge operators with actual fiat currency, our hope is that a sufficient number of Internet users will contribute their bandwidth and grow the HebTor network of exit bridges.

[2]There are important exceptions here, including VPN exit points and, as noted by Zhang et al. [60], cloud-hosted remote desktops.

4.3 High Level Design

HebTor consists of four components: (1) a broker that assigns exit bridges to requesting users; (2) a Human Task Provider which serves easily solvable tasks (e.g., image labeling tasks) to users and produces a payment when a task is successfully completed; (3) a pool of exit bridges that operate as Tor onion services; and (4) a small Tor Browser extension and accompanying software that is installed on client machines. We envision that this latter component can be packaged with Tor.

The broker and Human Task Provider are assumed to be honest-but-curious. HebTor is robust against malicious exit bridges.

At a high level, HebTor operates as follows: a Tor user who cannot access a site due to Tor exit blocking is presented with a notice by the Tor Browser, and is optionally redirected to the HebTor broker's onion site. The broker presents the user with a human-solvable task (e.g., an image labeling task), produced by the Human Task Provider. Completing this task yields a payment that will ultimately compensate the exit bridge operator. The user is then provided with an exit bridge. The user's Tor Browser extension then automatically configures the exit bridge, and the user's request is then routed through Tor to the exit bridge's onion service. In more detail, the exit bridge operates a SOCKS proxy (as an onion service), which then relays the traffic towards the final destination. An example HebTor workflow is presented in Figure 1. We emphasize that all HebTor communication (including with the Human Task Provider) occurs over anonymous Tor circuits, with the exception of the final "hop" between the exit bridge and the website. A more thorough explanation of HebTor is provided in the following section.

5 IMPLEMENTATION

At a high-level, HebTor's primary aim is to allow Tor users to reliably route their traffic to a destination website through a set of voluntarily participating bridges while still preserving Tor users' anonymity. To achieve this goal, HebTor employs a set of protocols to (1) curate a pool of bridge operators and their reputation (to prevent abuse), (2) fairly assign and compensate a bridge operator to forward users' traffic (i.e., to provide incentives), and (3) create a secure channel to tunnel traffic between the Tor user and the bridge operator (to preserve anonymity and unlinkability).

5.1 Typical Workflow

A typical workflow of HebTor involves three main sets of participants: a broker hosted as a centralized trusted onion service; Tor users who want to bypass Tor-exit blocking; and bridge operators who willingly contribute their bandwidth and CPU resources to tunnel traffic between Tor users and destination websites.

Register and advertise bridge (§5.2). HebTor employs the broker to curate all volunteering bridges. As the first step for a new bridge operator, it needs to register itself with the broker and start the actual bridge as a hidden onion service. The use of onion services allows the bridge operator to contribute even when it does not have a static IP address or is behind a NAT. Once the bridge is online, it advertises its onion address to the broker. The broker maintains a pool of the onion addresses of all advertised bridges.

BO: bridge operator; BR: broker; HTP: human task provider

```
1:  function operator_register([BO, BR, HTP])
2:      (BO.ECC+, BO.ECC−) = BO.ecc_key_gen()
3:      [BO.ECC+]sig(BO.ECC−) → HTP
4:      HTP.init_record([BO.ECC+]sig(BO.ECC−))
5:      [BO.ECC+]sig(BO.ECC−) → BR
6:      if BR.h_verify(h_challenge(BR, BO)) then
7:          BR.init_record([BO.ECC+]sig(BO.ECC−))
8:          BR.success → BO
9:      else
10:         BR.reject → BO
11:         ABORT
```

Figure 2: Bridge operator registration

Request bridge (§5.3). HebTor includes a TorBrowser extension that allows Tor users to use HebTor's service. The extension prompts Tor users whether to request a bridge when they encounter a Tor-exit blocking site.

Assign and compensate bridge (§5.4). Upon receiving a request from the user, the broker selects a bridge from its advertised bridge pool and assigns the bridge to the user. The selected bridge operator is compensated for its contribution to the system. While the compensation originates from the Tor user, it should avoid jeopardizing the anonymity of the user. In addition, given the monetary incentive, the broker should assign the bridges according to some fairness policy. In HebTor, the probability that a bridge operator is selected is proportional to its reputation.

Create HebTor circuit and forward traffic (§5.5). The Tor user is notified of the bridge assignment which includes the onion address of the bridge and the broker's signature to prove the authenticity of the assignment. The Tor user contacts the bridge and presents the signed assignment to initiate a HebTor circuit for tunneling traffic between the Tor user and the destination site.

Update reputation (§5.6). The reputation of each bridge is updated as the bridge forwards traffic. The reputation should reflect the quality of service that the Tor user receives during an active session (e.g., the ratio of successful vs. failed requests). In HebTor, the browser extension and the local relay collect a QoS metric for each minute of the service and report it to the broker. The broker then updates the bridge's reputation after the session concludes.

5.2 Bridge Registration and Advertisement

Bridge registration. Users who are willing to contribute idle bandwidth and CPU resources may register at the broker as a bridge operator. Figure 2 presents the pseudocode of the registration. A bridge operator is uniquely identified in HebTor by its public/private key pair; the public key is submitted to the broker and will be used as the bridge operator's permanent identifier (Line 7). To defend against malicious bridge operators who create multiple identities (for example, for the purpose of reseting a low reputation), the bridge operator is required to complete a "human task" before the registration (Line 6). Human tasks are discussed in more detail in §5.4. For a successful registration, a clean reputation record will be generated at the broker and linked with the bridge operator's

BO: bridge operator; BR: broker

1: **function** bridge_advertise([BR, BO])
2: $BO.onAddr$ = BO.gen_onion_address()
3: $BO.msg = [BO.ECC^{+}, BO.onAddr]_{sig(BO.ECC^{-})}$
4: $BO.msg \rightarrow BR$
5: **if** BR.sig_verify($BO.msg, BO.ECC^{+}$) **then**
6: BR.advertise($BO.ECC^{+}, BO.onAddr$)
7: $BR.success \rightarrow BO$
8: **else**
9: $BR.reject \rightarrow BO$
10: $ABORT$

Figure 3: Bridge advertisement

identifier. Finally, the bridge operator registers with the Human Task Provider in order to receive payment for its contribution as an exit bridge (Lines 3-4). All communication between the bridge operator and the broker occurs over anonymous Tor circuits; the broker is itself a Tor onion service.

Bridge advertisement. After the registration, the bridge operator further provides the onion address that hosts the actual bridge for forwarding network traffic between the Tor user and the destination site. Figure 3 presents the pseudocode of this process. More concretely, the bridge operator sends a signed message containing the onion address and its identifier (i.e., its public key) to the broker through Tor. The broker will then add this onion address to the advertised bridge pool upon signature verification.

To some extent, the broker operates similarly to Tor's directory service except that the broker records bridges' *onion addresses* rather than *IP addresses*. In addition, unlike Tor relays' IP addresses which are publicly accessible, discovering an exit bridge's onion address requires *human* effort. By only providing onion addresses and requiring human effort for accessing the addresses, an adversary cannot easily enumerate all bridges.

5.3 Bridge Request

HebTor provides a TorBrowser extension and a local relay that are installed on the user's computer. The extension locally maintains a user-specified blacklist that includes destination sites that block traffic from Tor exits. The user may modify the blacklist manually, e.g., by adding a new entry to the blacklist when she experiences difficulty accessing a site. (Although not implemented in our initial release, the blacklist could be managed by a third party monitoring service, which could operate similarly to distributed techniques to detect misbehavior at exit relays [56]. We leave such integration to a future release of HebTor.)

Once a request towards an exit-blocking site has been detected, the TorBrowser extension first checks if a valid bridge instance exists. If it does, this request will be encapsulated in a SOCKS5 session and forwarded to the bridge using the local relay (which itself forwards the SOCKS5 connection over Tor). If no valid bridge instance exists, the user will be redirected to a local page asking whether to request a new bridge to visit this site. Should the user choose to, she will be redirected to the broker's onion site for a bridge assignment after completing a "human task". These human task challenges prevent an adversary from enumerating all bridges,

BO: bridge operator; BR: broker; HTP: human task provider; U: user

1: **function** bridge_assignment([BR, U, HTP])
2: $(U.ECC_S^{+}, U.ECC_S^{-}) = U.ecc\_key\_gen()$ // session key for accessing a website
3: $U.ticketParam = U$.get_ticket() // blind ticket to bypass extra human tasks
4: $[U.ECC_S^{+}, U.ticketParam]_{sig(U.ECC_S^{-})} \rightarrow BR$
5: **if** U.ticketParam is not a list of unsigned tickets **then**
6: // U.ticketParam contains a ticket
7: **if** !BR.sig_verify(U.ticketParam, $BR.ticketKey^{+}$) **then**
8: $BR.reject \rightarrow U$
9: $ABORT$
10: **else** // U.ticketParam contains tickets to sign
11: **if** ! BR.h_valid(h_challenge(BR, U)) **then**
12: $BR.reject \rightarrow U$
13: $ABORT$
14: $BR.tickets = BR$.rsa_sign_tickets($U.ticketParam$)
15: $BR.tickets \rightarrow U$
16: // now U either presents a valid ticket in ticketParam or passes the extra HTP verification
17: $BO = BR$.random_select_advertised_bridge()
18: PoP = h_challenge($BO.ECC^{+}, U$)
19: **if** ! BR.h_verify(PoP) **then**
20: $BR.reject \rightarrow U$
21: $ABORT$
22: $BR.PoA = [BO.onAddr, PoP]_{sig(BR.ECC^{-})}$
23: BR.log_assign($U.ECC_S^{+}, BO.ECC^{+}, BO.onAddr$)
24: $[BR.PoA, BR.success] \rightarrow U$

Figure 4: Bridge request and assignment

at the cost of imposing an extra burden to users. In §5.7, we present an unlinkable ticket scheme based on RSA blind signature to reduce such burdens.

5.4 Bridge Assignment and Compensation

Figure 4 presents the pseudocode of the bridge assignment process, which considers the following three aspects:

Biased bridge assignment. Given the pool of advertised bridges and their corresponding reputation scores, the broker randomly selects one at random, biased by the bridges' reputations (Line 17). More specifically, the probability of a bridge being selected fits the following distribution:

$$Pr[\text{ bridge}_i \text{ is selected}] = (\text{score}_i)/\textstyle\sum_{j=1}^{n}(\text{score}_j)$$

where n is the number of available bridges and score_i is bridge_i's reputation score. We add 1 to each score to normalize the score range from $[-1, 1]$ to $(0, 2]$ to avoid negative probabilities.

Compensation through hCaptcha. Our current implementation uses hCaptcha, a publicly available service provided by Intuition Machines, Inc. [20], as its Human Task Provider. hCaptcha accepts machine learning related data labeling tasks from third party companies, and encapsulates tasks into CAPTCHA challenges which

BO: bridge operator; BR: broker; U: user

1: **function** bridge_usage ([BR, BO, U])
2: $U.PoA \rightarrow BO$
3: BO.sig_verify($U.PoA, BR.ECC^{+}$)
4: **if** ! BO.h_verify($U.PoA.PoP$) **then**
5: $BO.reject \rightarrow U$
6: $ABORT$
7: $BO.credential = BO$.gen_socks5_cred()
8: $BO.credential \rightarrow U$
9: **while** $U.session\ is\ valid$ **do**
10: $U \leftrightarrow BO$ // tunnel traffic
11: **for** every minute **do**
12: $U.tag = U$.gen_measurement_tag()
13: $[U.ECC_S^{+}, U.tag]_{sig(U.ECC_S^{-})} \rightarrow BR$

Figure 5: Bridge Usage

can be distributed via websites. A solution of an hCaptcha is trivially translated to a result of the corresponding data labeling task. hCaptcha gathers these results (e.g., across many successful CAPTCHAs) and sends them back to the third party companies for compensation; a small portion of this compensation will be rewarded back to the website that serves hCaptcha. Effectively, hCaptcha is similar to Google's reCAPTCHA, but offers payment to the hosting website (in our case, the broker or bridge operators) when the human solvable tasks are successfully completed. In §5.8, we present a brief financial analysis and show that a bridge operator who contributes 10Mbps of bandwidth can receive $3.55 per day.

HebTor leverages hCaptcha to allow anonymous payment to the bridge operators (Lines 18-19). One key strength of hCaptcha is that no contact or payment information is required, and the identity of the payer (i.e., the Tor user) is completely oblivious to the payee (i.e., the bridge operator). Note that hCaptcha currently does not provide signatures on Proof of Payment (PoP), which slightly deviates from the ideal case of our protocol. (That is, we compensate for the lack of signatures on PoPs by having the verifying party explicitly request proof-of-payments using hCaptcha's API over Tor.)

Proof of assignment. The broker receives feedback from Tor users about the bridges, which in turn affects the bridges' reputations. This presents an opportunity for a malicious user to increase or decrease a bridge's reputation by providing spurious feedback. To minimize the impact of malicious feedback, HebTor verifies that the broker randomly assigns bridges to users; this randomization prevents a malicious user from targeting a specific bridge. Once the bridge assignment is decided, the broker generates a signed proof of assignment (PoA) that contains the selected onion address and PoP, and sends the PoA back to the Tor user (Lines 22-24). The Tor user then presents the proof of assignment to the bridge operator such that the bridge operator can verify that the assignment is indeed made by the broker.

5.5 HebTor Circuit

Figure 5 presents the pseudocode of HebTor's circuit creation and reputation update process. Once a bridge receives the PoA from the user, it verifies the authenticity of the PoA (Lines 2-3), and then spawns a SOCKS5 server with a newly generated credential and sends back the credential to the user (Lines 7-8). With this credential, the user can spawn HebTor circuits towards the exit blocking sites via the bridge. We call this the *HebTor circuit* to differentiate it from traditional Tor circuits. We argue in §6 that the HebTor circuit provides at least the same user anonymity as a Tor circuit.

An illustration of a regular HebTor circuit is shown in Figure 1. A SOCKS5 proxy is hosted and configured on the bridge as an onion service, the local relay serves as the local end point of a HebTor circuit and listens on a local port to forward traffic between the Tor Browser and the bridge. The Tor Browser would consider the local relay as an ordinary SOCKS5 server and the bridge would deem the local relay as the source of SOCKS5 requests.

Compared with a three-hop regular Tor circuit, HebTor circuits contain 7 hops, which means the network latency is roughly doubled. An optimization can be achieved by using Tor's newer Single Onion Service protocol [55] on the bridge side, which removes hops between the rendezvous point (RP) and the bridge, making the length of a Single Onion Service HebTor circuit reduced to just four hops. We expect that this will decrease the latency overhead, as it resembles the performance penalty caused by using an ingress bridge. Note that a user's anonymity is not harmed when the exit bridge operates as a Single Onion Service since it still uses a three-hop anonymous Tor circuit.

To achieve unlinkability, different bridges will be used for different sites requested by the Tor user. HebTor implements SOCKS5 routing at the local relay. When a SOCKS5 request comes from the TorBrowser, the local relay recognizes the destination site and selects the corresponding bridge to build the HebTor circuit.

5.6 Reputation Update

The goal of the reputation system is to allow the broker to favor reliable bridges when assigning them to handle users' requests. To achieve this goal, the broker maintains a reputation score for each registered bridge. The intuition is to assign a higher reputation score to an honest bridge with a good service history while penalizing a freeloading bridge that rarely forwards data.

Feedback tag. Reputation scores are calculated from user feedback. Once a request is proxied through bridges, HebTor's browser extension will query the local relay to learn whether the request is successful. The local relay knows the number of successful requests ($\#_{\text{success}}$) and the number of failed requests ($\#_{\text{fail}}$) during each minute. If $\#_{\text{success}} - \#_{\text{fail}} \geq 0$, an "up" vote will be generated; otherwise a "down" vote will be cast. The vote will then be signed and sent to the broker via Tor.

Reputation scores. The broker maintains, for each bridge and bridge session, a list of up/down votes. A *session* score, computed for each of a bridge's session, is defined as the average of the session's up/down votes, where up votes are counted as +1 and down votes as −1. Finally, the bridge's overall reputation is the median of its associated session scores. In brief, an exit bridge's reputation is the median of its users' average up/down scores. We discuss the robustness of reputation scores against manipulation in §6.

5.7 Unlinkable Ticket Scheme based on RSA Blind Signatures

As currently described, a user needs to pass two rounds of HTP challenges: one for sending a request to the broker (this is needed to prevent a malicious user from enumerating all bridges); the other for compensating the bridge operator for the contributed bandwidth and CPU resources. Such frequent HTP challenges negatively impacts user experience. As an optimization, HebTor uses an unlinkable ticket scheme based on RSA blind signatures to allow users to bypass the HTP challenges required for sending bridge requests to the broker.

At a high-level, the broker will assign *tickets* to a user when it sends a request for the first time (the user still needs to complete the HTP challenge this time), such that the user can use these verifiable tickets to bypass HTP challenges in the future.

Blind signature. More concretely, a ticket is a tuple (m, s) where m is a random number generated by the user and s is the broker's signature for m. To maintain unlinkability between a user's requests (and tickets), the user generates a blinding factor r, then sends the blinded message $m' = m * r^e (mod\ n)$ to the broker, where n, e are generated from the broker's RSA public ticket key. The broker then signs m' and returns signature s' back to the user. The user can easily recover the actual signature s for m from s' using the inverse of r. When the user needs to spend the ticket, he presents (m, s), and the broker can verify that s is its signature for m. Note that the broker cannot link s with s' since it does not know r.

The blind signature scheme allows a user to have multiple tickets after completing one round of HTP challenge: the user can simply generate a list of m' and let the broker sign each m' in the list at the same time. In this way, the user can hold multiple valid tickets for its use in future requests.

Key rotation. To prevent a malicious user from accumulating tickets, the tickets are made to expire after a specific period of time, by letting the broker rotate its ticket key at a predetermined frequency. For example, a set of ticket key pairs may only be valid for signature generation for 1 hour and for signature verification for 2 hours. In this case, if a user presents an expired ticket, the verification will fail and the user has to complete the HTP challenge again.

In addition, to prevent double spending of tickets, once a ticket (m, s) is received by the broker, the broker should put m into a hash table, indicating that the ticket (identified by m) has been used. If m appears again, the broker can detect such collision in the hash table and serve a HTP challenge instead. Since expired tickets will automatically fail upon ticket key rotation, we can remove entries from the hash table 2 hours after their insertion.

5.8 Financial Analysis

In this section, we present a brief financial analysis of (1) the size of the "market", that is, the total potential revenue that could be earned per day, and (2) the per-person incentive, that is, the total potential revenue that could be earned by a single bridge operator.

Market cap. Here we assume a 100% penetration rate—users who cannot access a site that blocks Tor will all choose to use HebTor. As of January 17, 2020, the aggregated bandwidth observed at Tor's exits is 51.64 Gbps [50]. The block rate for Tor's web traffic is 4.8% [60]. Assuming a 30MB bandwidth cap for each bridge, the total bridges needed per day is 456,878, translating to $456.88 (USD) revenue per day based on hCaptcha's pay rate of $0.001 per solve.

Per-person incentive. Here we assume that a bridge operator is willing to contribute 10% of its bandwidth to run the bridges. Assuming a 100Mbps residential network and a 30MB/15mins bridge configuration, then the number of bridges that can be concurrently supported is 37. Therefore, the total bridges that could be supported per day is 3,552, translating to $3.55 per day per bridge operator.

Given the $456.88 market cap and $3.55 revenue per bridge operator, the market can support up to 128 bridge operators who operate at their maximum capacity.

6 SECURITY

In this section we discuss several important security properties of HebTor, and provide informal sketches of their correctness.

Mandatory task completion. Our incentive design requires that users perform some task (for example, an image labeling task) that raises some revenue, which in turn is directed (as payments for service) to the exit bridge operator. We first argue that:

Claim: A user cannot use an exit bridge without first performing the human task, so long as the broker is not malicious.

Sketch: End users interface with the exit bridge using the bridge_usage procedure defined in Figure 5. This is the only mechanism by which the user can access the exit bridge. In step two of bridge_usage, the bridge operator checks the signature of the PoA signed by the broker; if the signature is not valid, bridge_usage terminates. In step three of bridge_usage, the bridge operator calls the h_verify function on the proof of payment (PoP). If h_verify returns $\perp$ (i.e., fails to verify), then bridge_usage terminates and the user cannot use the exit bridge.

A user can use the bridge if and only if (1) the PoA is properly signed by the broker (see line 22 in bridge_assignment in Figure 4) and (2) the PoA contains a valid proof of payment (PoP). Note that the Human Task Provider's signature over the PoP is verified by the broker in line 19 of bridge_assignment.

In summary, to use an exit bridge, a user needs to provide a PoA signed by the broker, which it can only obtain by successfully completing the bridge_assignment procedure, which in turn depends on receiving a valid signed PoP from the Human Task Provider. ■

Difficulty of enumerating bridges. Although we do not attempt to provide anonymity to exit bridge operators, HebTor is most effective when it is difficult for any party to easily enumerate the IP addresses of exit bridges. Such enumeration allows exit bridges to quickly appear on blacklists.

Claim: HebTor reveals the IP address of an exit bridge only to (1) Tor relays with which that exit bridge directly communicates and (2) sites accessed by the exit bridge.

Sketch: Communication between the bridge operator, the broker, the Human Task Provider, and the end user all occur over anonymous Tor connections. The bridge operator does not communicate its IP address in either the operator_register, bridge_advertise, bridge_assignment, or bridge_usage procedures (see Figures 2 through 5). That is, no component of the HebTor infrastructure

(not including Tor relays) learns or stores the IP address of the exit bridge, other than the exit bridge itself.

Therefore, the only exposure of the exit bridge's IP address occurs when it directly communicates over IP. This happens in two instances: when the exit bridge uses Tor (i.e., in its communications with a Tor relay) and when it forwards data on behalf of the end-user to the requested destination (i.e., website). ■

Unlinkability across requests. Unlinkability requires that an adversary should not be able to distinguish whether two or more requests are related [41]. Unlinkability is critical to maintaining anonymity: by way of example, consider a user who uses an anonymity service to connect to two websites, and then discloses its identity (e.g., by posting a message over an unencrypted HTTP connection) to exactly one of the two sites. An adversary who observes traffic at the anonymity network's egress point and can link the user's two anonymous connections can then trivially infer the sender of the otherwise anonymous communication stream.

In the context of HebTor, our goal is to prevent any party (internal or external to HebTor) from determining whether two sites accessed via exit bridges originated from the same end-user. We group all of the web objects associated with a site into our notion of a single "request"; this corresponds to the behavior of the Tor Browser, which uses a separate Tor circuit to achieve unlinkability between different browser tabs and windows. That is, like Tor, we aim to provide unlinkability between a client's requests of two or more websites.

We first argue that the ticket protocol achieves unlinkability.

Claim: A party who has access to two HebTor tickets cannot determine whether those tickets were issued to the same or different users.

Sketch: A HebTor ticket contains a single randomly chosen number m selected by the user who initially generates the (unsigned) ticket. Let m_i denote the random number present in ticket t_i.

The user sends only blinded tickets to the broker; the RSA blind signature scheme guarantees that the broker does not learn the blinded random value m_i selected by the user, for any ticket t_i. An honest-but-curious (semi-honest) broker who obeys the protocol will return a blind signature to the user, who then unblinds the signature to obtain the signed ticket (i.e., m along with its accompanying signature). Importantly, the honest-but-curious broker cannot link two tickets t_x and t_y as originating from the same user, since (1) it never learns the random numbers m_x and m_y and (2) m_x and m_y are independent and identically distributed random values. Since the broker periodically rotates signing keys, it can determine whether t_x and t_y originated during the same key period.

A malicious broker can attempt to watermark a given requestor's tickets by returning invalid signatures (e.g., signatures over values chosen by the broker). However, a user detects such misbehavior by verifying that the returned signature, once unblinded, is over the user-provided (blinded) input m_i. ■

Claim: HebTor achieves unlinkability between a client's requests of two or more websites.

Sketch: For each requested website, the user initiates the bridge_assignment procedure, obtaining a new proof-of-assignment (PoA), set of tickets, and (potentially) a new exit bridge (see §5.5). It is easy to show that, by construction, two or more PoAs are unlinkable: they contain the onion address of an exit bridge (chosen independently, with replacement, for each PoA) and a proof-of-payment (PoP). The PoP is also unlinkable across sessions, as it contains only a signed session public key, which is used only for the given session. Above, we previously argued that two or more tickets are unlinkable.

Each instantiation of bridge_assignment occurs using a new Tor circuit between the user and the broker. Additionally, the user does not use consistent identifiers or credentials when communicating with the broker, and relies only on ephemeral session keys. Because the broker cannot identify the party with whom it is communicating (since communication occurs over a dedicated Tor circuit) and there is no identifying information about the user in an invocation of bridge_assignment, for any two invocations of bridge_assignment, the broker cannot distinguish between two different users calling bridge_assignment and the same user calling the procedure twice.

We next argue that a bridge operator cannot distinguish between two connections originating from the same user and two connections each originating from a different user. Interactions between the user and the exit bridge are governed by the bridge_usage procedure. Here, the requesting user provides only a PoA—which we argued above is unique for every requested site.

In bridge_assignment, the user interacts with a Human Task Provider via the h_challenge procedure. The challenge is served to the user via a Tor circuit, and hence the challenge service does not learn the user's identity. Additionally, since a new Tor circuit is used for every invocation of bridge_assignment and Tor provides unlinkability across circuits, then the challenge service cannot determine whether two challenges are associated with the same or different users.

Finally, the broker can attempt to link user requests using the reputation tags that are sent to the broker (see line 13 of bridge_usage). The tags consist of the user's assessment of the exit bridge (expressed as +1 or −1), a time index, and the user's session public key; the tag is additionally signed using the user's session private key. As intended by design, tags can thus be linked across a particular session (i.e., request for a web site). However, since a new session key is used for each site request, the broker cannot distinguish whether two tags from two different sessions are produced by the same or different users. ■

Robustness of reputation scores. The probability that an exit bridge is assigned to a user is proportional to that exit bridge's reputation score. We next argue that reputation scores are robust against manipulation.

Claim: If (1) the broker is honest, (2) n honest clients and α colluding malicious clients are connected to a malicious exit bridge, and (3) the exit bridge forwards traffic only for $m < n/2$ honest clients, then α must be greater than $\lceil \frac{n}{2} - m \rceil$ for the exit bridge to obtain a positive reputation.

Sketch: As computed by the broker, a bridge's reputation score is the median of the average of measurements taken by the bridge's users. The users' measurements are communicated in line 13 of the bridge_usage procedure. Importantly, note that the broker only accepts measurements from users who have been assigned to that

bridge, since the user's session key ECC_S^+ is recorded by the broker in line 23 of bridge_assignment and the measurements are signed by the corresponding session private key ECC_S^- (line 13 of bridge_usage).

It follows from the statement of the claim that the malicious exit bridge will not forward traffic for $\hat{m} = n - m$ users. Each of the $\hat{m}$ honest users will contribute a score of -1 to the broker.

To obtain an overall positive reputation score, it then follows that $\frac{m+\alpha}{m+\hat{m}+\alpha} > 0.5$, since the reputation score is the median of the user-contributed averages. Here, we conservatively assume that all served honest users and all malicious users assigned to the bridge will contribute positive rankings of $+1$ to the broker. Hence, the fraction of positive scores $(m+\alpha)$ to total scores $(m+\hat{m}+\alpha)$ must be greater than 0.5. Substituting in $n = m + \hat{m}$ to the above inequality, we obtain $\alpha > \lceil \frac{n}{2} - m \rceil$. ∎

The implications of the above claim is that as more honest users (n) are assigned to the malicious exit bridge, the adversary must either serve a large fraction of the honest users (m) to achieve a positive reputation, or must operate a large number of malicious HebTor users who then need to be assigned to the malicious bridge and successfully complete the human task. At the extreme, if the adversary does not forward traffic for any honest clients, then to earn a positive reputation, it needs to operate at least half of the clients that are assigned to its malicious bridge.

Non-degradation of anonymity. Our goal is to enable Tor users to access sites that they would otherwise be unable, without degrading their anonymity. We first consider the case in which the broker is honest-but-curious, and then explore the ramifications of a malicious broker.

Claim: If the broker is not malicious, HebTor offers similar anonymity to that of Tor.

Sketch: All communication between the user and the broker, and between the user and exit bridge, are conducted over Tor. As argued above, HebTor sessions are unlinkable and the HebTor protocols do not include identifying information about the user. A curious broker therefore cannot discern the identity of the user.

The user and the exit bridge communicate via SOCKS5. Since the exit bridge operates as an onion service, this communication is both end-to-end encrypted and, in the case of the exit bridge, self-authenticating [12].

The user may be assigned to an malicious exit bridge. This happens with a probability that is proportional to the reputation of the exit bridge. As shown above, the reputation system is difficult to manipulate: the adversary needs to operate at least half of the total clients that connect to its malicious bridge if it does not forward any traffic for honest clients.

Of course, the adversary can operate bridges with high reputations by participating in the HebTor network and forwarding traffic. The probability of a user selecting a malicious bridge is thus proportional to that bridge's contribution towards the network's sum of reputation scores. This is not dissimilar from traditional Tor exit relays, which are selected proportional to relays' bandwidth contributions to the network.

If a malicious bridge is chosen, it cannot trivially identify the client's network location, since the client communicates with the bridge via Tor. However, if the traffic between the client and the website is not end-to-end encrypted (e.g,. using TLS), then the bridge can learn the user's identity if it is revealed in the contents of the communication. Operating the egress point also significantly increases the adversary's ability to de-anonymize users using traffic correlation attacks [38, 40], which have been shown to be problematic for Tor [24]. Overall, the risks of selecting a malicious exit bridge are analogous to those from selecting a malicious exit relay, when the broker is not malicious. ∎

A *malicious* broker cannot directly learn the identities of the requesting users, since users communicate only via anonymous Tor connections. However, a malicious broker can assign only malicious exit bridges to requesting clients. This is roughly equivalent to having clients always select a malicious exit relay under Tor, and thus incurs additional susceptibility to eavesdropping and traffic correlation attacks, as described above.

As potential future work, we could increase HebTor's resilience to malicious brokers by using a more distributed model, akin to Tor's authoritative directory architecture. Here, the general concept would be to have exit bridges register with multiple brokers, who would then vote on a signed consensus document. The information theoretic private information retrieval technique suggested by Mittal et al. [35] could then be used to allow users to efficiently obtain an exit bridge from multiple brokers, while protecting against selective corruption attacks in which a malicious broker purposefully returns malicious bridges. We leave the exploration of improving HebTor's protection against a malicious broker as future work.

7 LIMITATIONS

HebTor bypasses server-side blocking of Tor by permitting any Internet-connected device to operate as an egress point. Our technique is targeted at IP-based blocking of exit relays.

A limitation of our design is that it does not completely avoid fate sharing. Just as Tor exit relays can be added to an IP blacklist, so can the IP addresses of exit bridges. However, unlike exit relays, exit bridges reside on end-user devices, and thus are more likely to be on dynamically assigned IP addresses; this increases the potential collateral damage of blocking such IPs since overblocking prevents future potential customers from accessing the site. Additionally, because they are located on potentially residential/broadband networks, it is difficult for a website operator to distinguish between normal client traffic originating from the residential ISP and Tor traffic that egresses through the residential ISP. Where such distinction is possible, HebTor does not eliminate the threat of fate sharing, since the blocking of an exit bridge disrupts the communication of all connected Tor users.

As with traditional (ingress) bridges, HebTor bridges are also susceptible to enumeration. Bridge enumeration is an open problem for Tor, but it is worth emphasizing that enumeration attacks are usually enabled by an adversary with a large workforce—e.g., a nation-state's intelligence service. Such human resources would unlikely be available to website or blacklist operators. Additionally, HebTor offers some protection against automated enumeration since learning an exit bridge requires first solving an hCaptcha. Finally, we envision that our incentive scheme will (hopefully) provide a fresh flow of volunteers to make the effectiveness of such enumeration attacks short-lived.

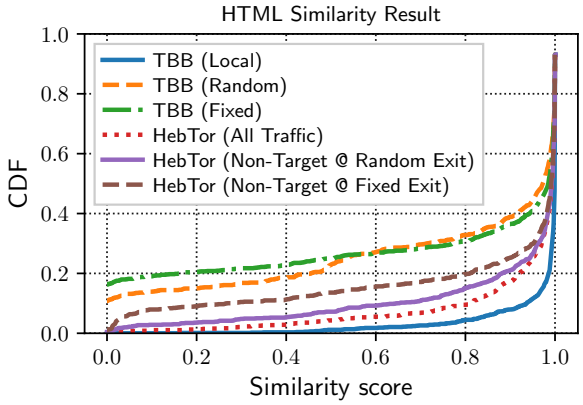


Figure 6: Distribution of HTML similarity scores for various browsing configurations.

We also note that operating a Tor exit bridge poses similar legal risks to operating a Tor exit relay. We discuss the current legal landscape of running a Tor egress point in Appendix A.

8 EVALUATION

Our evaluation aims to answer two main questions: (1) whether HebTor helps Tor users bypass Tor-exit blocking and (2) what are the performance penalties of using HebTor.

8.1 Experimental Setup

Bridge configuration. The HebTor exit bridge runs on an Ubuntu 18.04 virtual machine, connected to the Internet through a U.S.-based home broadband network with an advertised bandwidth capacity of 100 Mbps. We use microsocks [43] as the backend SOCKS5 server on the bridge, which is hosted as a Tor onion service.

We perform experiments using both the classic onion service scheme and the newer Single Onion Service design [55] that decreases latency by removing three hops from the classic onion scheme, at the cost of sacrificing receiver anonymity.

User configuration. The simulated user also runs on an Ubuntu 18.04 virtual machine, with Firefox and TorBrowser installed. We use Selenium [48] as a controller to simulate user browsing behavior. Experiments are conducted over the live Tor network.

Workload. We select the first 1000 websites from the Alexa Top sites list as the destination websites throughout our evaluation.

8.2 Functionality Evaluation

To evaluate HebTor's ability to bypass Tor exit blocking, we use *HTML similarity score* [31] to measure the similarity between HTML pages fetched directly (without Tor) versus pages retrieved through Tor or an exit bridge. The similarity score is between 0 to 1; a higher score indicates higher similarity between two pages. A similarity score of 1 indicates two identical pages. If a website blocks traffic from Tor exits, we should expect a low similarity score between the directly fetched webpage and the (empty) page retrieved via an ordinarily Tor circuit.

We consider the following configurations:

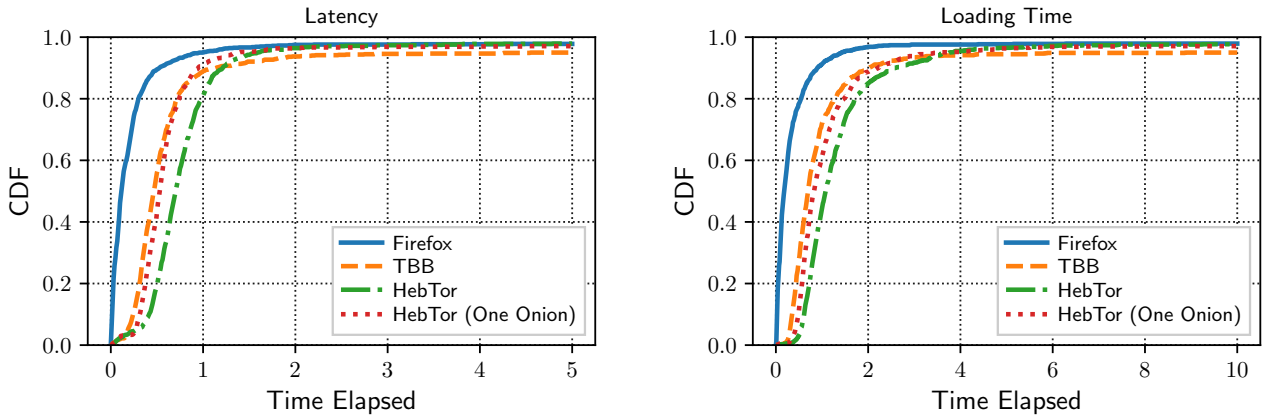
- TBB (Local) uses a modified version of the Tor Browser that communicates to the website directly without using Tor. We opt for this modified version of the Tor Browser over Firefox since the Tor Browser has a number of unique features (e.g., a restricted Javascript engine) that, if ignored (e.g., in the case of Firefox) would introduce artificial errors in our similarity scores, since all other configurations use the Tor Browser. We perform two TBB (Local) fetches to minimize the impact of dynamic content. The comparison result of these two fetches is used as our baseline HTML similarity score. Note that for the following configurations, their similarity scores are also calculated against the first TBB (Local) fetch.
- TBB (Fixed) uses TorBrowser and visits the destination through the Tor network. To minimize the effects of localized web content, we fix the exit relay during the entire experiment, and ensure this relay is in the same geographic region as the exit bridge used in other configurations.
- TBB (Random) uses TorBrowser and visits the destination through Tor. The exit relay is randomly selected. This is the default Tor configuration used by Tor users.
- HebTor (All Traffic) uses TorBrowser and visits the destination through HebTor. *All* requests, including retrieving images and other objects, are tunneled through the same exit bridge.
- HebTor (Non-Target @ Random Exit) uses TorBrowser and visits the destination through HebTor. Requests for non-target hosts are routed through a random Tor exit relay. This is HebTor's default scenario: a bridge is responsible only for routing a given user's traffic to a specific host that blocks Tor, while all other requests are routed via Tor exit relays.
- HebTor (Non-Target @ Fixed Exit) uses TorBrowser and visits the destination through HebTor. Requests for non-target hosts are routed through same fixed Tor exit relay.

We use the same fixed exit relay in the TBB (Fixed) and HebTor (Non-Target @ Fixed Exit) scenarios.

For each destination website, we first check whether the returned HTTP status code reported by Selenium is valid, and then check whether the HTML body is wrapped properly. Invalid status codes or incomplete HTML bodies will directly lead to a similarity score of 0. We note that additional objects may still be partially loaded, since we rely on Selenium's returned status code corresponding to the webpage's main HTML content.

Figure 6 shows the cumulative distribution of similarity scores for the configurations described above. We observe that the similarity scores of HebTor, especially HebTor (All), closely track the baseline, and consistently and notably outperform those of TBB.

In addition, we further observe that there is a significant gap between TBB (Fixed) and TBB (Random)—the former has a much higher prevalence of low similarity scores. We attribute this to our selection of the fixed exit relay, which is a high-performing and longstanding exit relay that is thus more likely to appear on IP blacklists than the relays we select randomly. This also applies to the gap between HebTor (Non-Target @ Fixed Exit) and HebTor (Non-Target @ Random Exit).



(a) Latency **(b) Loading Time**

Figure 7: Performance overheads.

In summary, we find that HebTor is able to achieve much greater similarity scores (relative to direct communication) due to its ability to access sites that would otherwise be inaccessible to Tor users.

8.3 Performance Evaluation

To evaluate the performance overhead, we measure the **latency** and **loading time** when visiting the Alexa Top 1000 websites.

We focus on the performance of the Tor exit bridges rather than on the cost of contacting the broker. The broker's responsibility is to assign bridges and bookkeep the bridges' reputations; the broker is not part of the circuit (depicted in Figure 1) and does not impose any additional delays. We anticipate that the cost in contacting the broker to be within a few seconds, which is mainly due to solving an hCaptcha to obtain a proof-of-assignment (PoA). Setting up a HebTor circuit further takes some time from the arrival of the PoA to the start of HTTP request, including an onion-service lookup and one round-trip communication of the PoA submission and SOCKS5 credentials retrieval, followed by a normal SOCKS5 handshake. The delay caused by the circuit setup is measured to be between 2.5 to 4.0 seconds. However, this startup cost can be eliminated by having the user construct a pool of ready-to-use HebTor circuits; this is analogous to Tor's construction of Tor circuits, which are established at start time.

We use the PerformanceTiming API [61] available both on Firefox and TorBrowser. **Latency** is defined as the return time of the first-byte of response, which is $T_{\text{RespStart}} - T_{\text{ReqStart}}$, and **loading time** is defined as total time for the data transmission, which is $T_{\text{RespEnd}} - T_{\text{ReqStart}}$. We consider the start and end timestamps of the server's response regardless of whether the reply indicates a "block" or not. On the user side, we deactivate the use of guards to eliminate the bias introduced by using a fixed guard.

We consider the following four configurations:

- Firefox uses the Firefox browser to visit the destination website via direct IP communication;
- TBB uses the Tor Browser to visit the destination website through a Tor circuit;
- HebTor uses the Tor Browser to visit the destination website through a HebTor circuit; and
- HebTor (One Onion) uses the Tor Browser to visit the destination website through HebTor using the Single Onion Service scheme.

For each destination, we collect T_{latency} and T_{loading} for all four configurations. Figure 7 shows the cumulative distribution of latency (T_{latency}) and page loading time (T_{loading}). We observe that the performance of our system with the Single Onion Service is similar to that of Tor's, with an increase in median of 0.08s for T_{latency} and 0.13s for T_{loading}. HebTor with an ordinary three-hop Hidden Service is slower, as expected—the increases in median T_{latency} and T_{loading} are respectively 0.25s and 0.40s—showing the tradeoff between performance and server anonymity.

9 CONCLUSION

HebTor bypasses Tor exit blocking by reassigning the job of exiting the Tor network from a collection of fixed exit relays (as in Tor) to a network of more ephemeral run-from-anywhere exit bridges. The key insight to HebTor is that exit bridges can join the Tor network as onion services, allowing nearly any Internet-connected device to function as an exit bridge. We provide an incentive structure that compensates Internet users for operating HebTor bridges, and a reputation system to load balance requests and prevent freeloading exit bridges. Our implementation is available at https://github.com/GUSecLab/tor-exit-relays.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and especially our shepherd Yixin Sun for their valuable comments and suggestions. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contracts Nos. FA8750-19-C-0500 and HR0011-16-C-0056 and the National Science Foundation (NSF) under grants CNS-1453392, CNS-1513734, CNS-1527401, CNS-1704189, and CNS-1718498. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or NSF.

REFERENCES

[1] Elli Androulaki, Mariana Raykova, Shreyas Srivatsan, Angelos Stavrou, and Steven Bellovin. 2008. PAR: Payment for Anonymous Routing. In *Privacy Enhancing Technologies Symposium (PETS)*.

[2] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. 2013. Internet Censorship in Iran: A First Look. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.

[3] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. 2013. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *IEEE Symposium on Security and Privacy (Oakland)*.

[4] BTC.COM. 2019. Bitcoin Difficulty. Available at https://btc.com/stats/diff.

[5] Sam Burnett and Nick Feamster. 2015. Encore: Lightweight Measurement of Web Censorship with Cross-origin Requests. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 653–667.

[6] R. Clayton, S. Murdoch, and R. Watson. 2006. Ignoring the Great Firewall of China. In *Privacy Enhancing Technologies (PET)*.

[7] Court of Justice of the European Union. 2016. Judgment of the Court in the case of Tobias Mc Fadden v. Sony Music Entertainment Germany GmbH. Available at http://curia.europa.eu/juris/document/document.jsf?text=&docid=183363.

[8] M. Crete-Nishihata, R. Deibert, and A. Senft. 2013. Not by Technical Means Alone: The Multidisciplinary Challenge of Studying Information Controls. *IEEE Internet Computing* 17, 3 (2013), 34–41.

[9] Alex Davidson. 2017. Privacy Pass - "The Math" (Cloudflare Blog Post). https://blog.cloudflare.com/privacy-pass-the-math/.

[10] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proceedings on Privacy Enhancing Technologies (PoPETS)* 2018, 3 (2018), 164–180.

[11] Roger Dingledine, Nicholas Hopper, George Kadianakis, and Nick Mathewson. 2014. One Fast Guard for Life (or 9 Months). In *Privacy Enhancing Technologies Symposium (PETS)*.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium (USENIX)*.

[13] Electronic Frontier Foundation (EFF). 2014. The Legal FAQ for Tor Relay Operators. Available at https://www.torproject.org/eff/tor-legal-faq.html.en.

[14] European Parliament. 1998. Directive 98/34/EC of the European Parliament and of the Council of 22 June 1998 Laying Down A Procedure For The Provision of Information in the Field Of Technical Standards and Regulations and of Rules on Information Society Services. , 37–48 pages. OJ L 204, 21.7.1998.

[15] European Parliament. 2000. Directive 2000/31/EC of the European Parliament and of the Council of 8 June 2000 on Certain Legal Aspects of Information Society Services, In Particular Electronic Commerce, In The Internal Market. , 16 pages. OJ L 178, 17.7.2000.

[16] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. 2002. Infranet: Circumventing Web Censorship and Surveillance. In *USENIX Security Symposium (USENIX)*.

[17] David Fifield. 2019. meek. https://trac.torproject.org/projects/tor/wiki/doc/meek.

[18] Arturo Filasto and Jacob Appelbaum. 2012. OONI: Open Observatory of Network Interference. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.

[19] Mainak Ghosh, Miles Richardson, Bryan Ford, and Rob Jansen. 2014. A TorPath to TorCoin: Proof-of-Bandwidth Altcoins for Compensating Relays. In *Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETS)*.

[20] Intuition Machines, Inc. 2020. hCaptcha - Earn money with a captcha. https://haptcha.com/.

[21] Rob Jansen, Nicholas Hopper, and Yongdae Kim. 2010. Recruiting New Tor Relays with BRAIDS. In *ACM Conference on Computer and Communications Security (CCS)*.

[22] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. 2014. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *Network and Distributed System Security Symposium (NDSS)*.

[23] Rob Jansen, Tavish Vaidya, and Micah Sherr. 2019. Point Break: A Study of Tor's Resilience to Bandwidth Denial-of-Service Attack. In *USENIX Security Symposium (USENIX)*.

[24] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users Get Routed: Traffic Correlation on Tor By Realistic Adversaries. In *ACM Conference on Computer and Communications Security (CCS)*.

[25] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. 2003. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *World Wide Web Conference (WWW)*.

[26] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David P. Mankins, and W. Timothy Strayer. 2011. Decoy Routing: Toward Unblockable Internet Communication. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.

[27] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M. Swanson, Steven J. Murdoch, and Ian Goldberg. 2016. SoK: Making Sense of Censorship Resistance Systems. *Proceedings on Privacy Enhancing Technologies (PoPETS)* 2016, 4 (July 2016), 37–61.

[28] Sheharbano Khattak, David Fifield, Sadia Afroz, Mobin Javed, Srikanth Sundaresan, Damon McCoy, Vern Paxson, and Steven J Murdoch. 2016. Do You See What I See? Differential Treatment of Anonymous Users. In *Network and Distributed System Security Symposium (NDSS)*.

[29] Zhuotao Liu, Yushan Liu, Philipp Winter, Prateek Mittal, and Yih-Chun Hu. 2017. TorPolice: Towards Enforcing Service-defined Access Policies for Anonymous Communication in the Tor Network. In *International Conference on Network Protocols (ICNP)*.

[30] Akshaya Mani, T Wilson Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. 2018. Understanding Tor Usage with Privacy-Preserving Measurement. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*.

[31] Edgar Marca. 2018. HTML Similarity Tool. Available at https://github.com/matiskay/html-similarity.

[32] Matthew Prince. 2016. The Trouble with Tor (Cloudflare Blog Post). https://blog.cloudflare.com/the-trouble-with-tor/.

[33] Mike Perry. 2016. The Trouble with CloudFlare. https://blog.torproject.org/trouble-cloudflare/.

[34] Tomáš Minárik and Anna-Maria Osula. 2016. Tor Does Not Stink: Use and Abuse of the Tor Anonymity Network from the Perspective of Law. *Computer Law & Security Review* 32, 1 (2016), 111–127.

[35] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. 2011. PIR-Tor: Scalable Anonymous Communication using Private Information Retrieval. In *USENIX Security Symposium (USENIX)*.

[36] Brad Moore, Chris Wacek, and Micah Sherr. 2011. Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race with Tortoise. In *Annual Computer Security Applications Conference (ACSAC)*.

[37] Multiple authors. 2020. List Of Services Blocking Tor. Available at https://trac.torproject.org/projects/tor/wiki/org/doc/ListOfServicesBlockingTor.

[38] Steven J. Murdoch and George Danezis. 2005. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy (Oakland)*.

[39] Nassim Nazemi. 2012. DMCA Sec. 512 Safe Harbor for Anonymity Networks Amid a Cyber-Democratic Storm: Lessons from the 2009 Iranian Uprising. *Nw. UL Rev.* 106 (2012), 855.

[40] Lasse Øverlier and Paul Syverson. 2006. Locating Hidden Servers. In *IEEE Symposium on Security and Privacy (Oakland)*.

[41] Andreas Pfitzmann and Marit Köhntopp. 2001. Anonymity, Unobservability, and Pseudonymity–A Proposal for Terminology. In *Designing Privacy Enhancing Technologies*.

[42] Michael K. Reiter and Aviel D. Rubin. 1998. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security* 1, 1 (1998), 66–92.

[43] rofl0r. 2020. MicroSocks: Multithreaded, Small, Efficient SOCKS5 Server. https://github.com/rofl0r/microsocks.

[44] Mahrud Sayrafi. 2018. Introducing the Cloudflare Onion Service (Blog post). Available at https://blog.cloudflare.com/cloudflare-onion-service/.

[45] Andreas Sfakianakis, Elias Athanasopoulos, and Sotiris Ioannidis. 2011. CensMon: A Web Censorship Monitor. In *USENIX Workshop on Free and Open Communication on the Internet (FOCI)*.

[46] Clay Shields and Brian Neil Levine. 2002. Hordes: A Multicast Based Protocol for Anonymity. *Journal of Computer Security* 10, 3 (2002), 213–240.

[47] Rachee Singh, Rishab Nithyanand, Sadia Afroz, Paul Pearce, Michael Carl Tschantz, Phillipa Gill, and Vern Paxson. 2017. Characterizing the Nature and Dynamics of Tor Exit Blocking. In *USENIX Security Symposium (USENIX)*.

[48] The Selenium Project. 2018. The Selenium Project. Available at https://www.seleniumhq.org/.

[49] Tor Project. 2020. ExoneraTor. Available at https://metrics.torproject.org/exonerator.html.

[50] Tor Project. 2020. Tor Metrics Portal. Available at https://metrics.torproject.org/.

[51] Tor Project. 2020. *Tor Rendezvous Specification (Version 3)*. Tor protocol specification rend-spec-v3. Available at https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt.

[52] Michael Carl Tschantz, Sadia Afroz, Anonymous, and Vern Paxson. 2016. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *IEEE Symposium on Security and Privacy (Oakland)*.

[53] U.S. Government. 1998. *Digital Millennium Copyright Act (DMCA)*. U.S. Code Title 17, Chapter 5, §512. Available at https://www.law.cornell.edu/uscode/text/17/512.

[54] Keith D Watson. 2012. The Tor Network: A Global Inquiry into the Legal Status of Anonymity Networks. *Wash. U. Global Stud. L. Rev.* 11 (2012), 715.

[55] Tim Wilson-Brown, John Brooks, Aaron Johnson, Rob Jansen, George Kadianakis, Paul Syverson, and Roger Dingledine. 2015. *Rendezvous Single Onion Services*. Tor protocol specification 260. Available at https://gitweb.torproject.org/torspec.git/tree/proposals/260-rend-single-onion.txt.

[56] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. 2014. Spoiled Onions: Exposing Malicious Tor Exit Relays. In *Privacy Enhancing Technologies Symposium (PETS)*.

[57] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China is Blocking Tor. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.

[58] Bassam Zantout and Ramzi Haraty. 2011. I2P Data Communication System. In *International Conference on Networks (ICN)*.
[59] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. 2016. AnonRep: Towards Tracking-Resistant Anonymous Reputation. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.
[60] Zhao Zhang, Tavish Vaidya, Kartik Subramanian, Wenchao Zhou, and Micah Sherr. 2020. Ephemeral Exit Bridges for Tor. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.
[61] Zhiheng Wang. 2012. Navigation Timing. https://www.w3.org/TR/navigation-timing/#sec-window.performance-attribute.
[62] zzz (Pseudonym) and Lars Schimmer. 2009. Peer Profiling and Selection in the I2P Anonymous Network. In *PET-CON 2009.1*.

A LEGAL RISKS OF OPERATING AN EXIT BRIDGE (OR EXIT RELAY)

Operating an exit bridge incurs risks that are analogous to those of running an exit relay. Traffic can be misattributed as originating from the bridge rather than being relayed through it. We briefly review the legal risks, according to the most current literature, of forwarding traffic on behalf of an anonymity network. We emphasize that we are synthesizing others' legal opinions in this appendix, and are not formulating any new legal theories of our own. Our survey is admittedly biased towards the legal systems in the United States and in Europe.

Minárik and Osula [34] present the most comprehensive analysis of the legality of anonymity systems, with a particular focus on Tor. Their legal analysis is based on European Law, and in particular, on case law from the European Court of Human Rights and the Court of Justice of the European Union. In their analysis of the legality of operating Tor exit relays, they found the most pertinent law is Article 12 of the E-Commerce Directive [15]. Article 12 gives safe harbor (i.e., legal immunity) to a "service provider" that "…(a) does not initiate the transmission; (b) does not select the receiver of the transmission; and (c) does not select or modify the information contained in the transmission…" [15]

Minárik and Osula conclude that operators of Tor exit relays—and we believe, by equivalent arguments, operators of exit bridges—clearly meet criteria (a)–(c). The question, which was unanswered at the time of their 2016 legal analysis, was whether exit relay operators could be considered service providers, which under European law, must be "normally provided for remuneration" [14]. That is, the question of whether safe harbor protections applied to exit relay operators in Europe hinged on whether the fact that Tor exit relay operators did not charge for their services negated their ability to be considered service providers. This question was addressed by the Court of Justice of the European Union later in 2016, which found that renumeration is not required for the service provider [7].

There is no law in the United States that specifically governs the use of Tor or other anonymity networks [54]. Exit relay operators however frequently receive copyright infringement complaints under the Digital Millennium Copyright Act (DMCA) [53]. The Electronic Frontier Foundation (EFF) and others have argued that Tor operators fall under the DMCA's safe harbor provisions [13, 39], which provide immunity so long as the communication is not modified in transit, the communication did not originate from the relay, and no copy of the communication is stored [53]. While several relay operators have received DMCA complaints, EFF reports that no one has been sued or prosecuted solely for running an exit relay in the United States [13].

B OTHER FUNCTIONS REFERENCED IN THIS PAPER

We list all other supplemental functions we referenced in this paper with their brief pruposes here:

General Processes

- **ecc_key_gen()** generates an ECC public/private key pair.
- **sig_verify(msg, sig, pubKey)** verifies whether the signature `sig` for a message `msg` is signed by `pubKey`.
- **h_challenge(payee, user)** verifies `user` is indeed a human using challenges created by HTP. A proof-of-payment (PoP) signed by HTP will be returned if the user passes the challenge and a payment will be accounted towards the `payee`.
- **h_verify(PoP)** verifies whether a proof-of-payment (PoP) is valid. A valid PoP indicates a piece of human work has been confirmed.

Tor User's Processes

- **U.get_ticket()** retrieves a valid unblinded ticket. If no ticket is available, a list of unsigned tickets will be returned (see Section 5.7).
- **U.gen_measurement_tag()** generates QoS measurement tags during an active traffic forwarding session (see Section 5.6).

Human Task Provider (HTP)'s Processes

- **HTP.init_record(BO.id)** registers the bridge operator (`BO`) on HTP using the BO's. identifier ($\texttt{BO.ECC}^{+}$).

Broker's Process

- **BR.init_record(BO.id)** initializes a new record for bridge operator ($\texttt{BO.ECC}^{+}$), with a default reputation.
- **BR.advertise(BO.ECC^{+}, BO.onAddr)** adds `BO`'s onion address to the broker's advertising pool; `BO`'s reputation will be referenced.
- **BR.log_assign(session, BO.id, BO.onAddr)** marks the bridge with onion address `BO.onAddr` as assigned, and the user in `session` has the right to send QoS measurement tag which may impact the `BO`'s reputation.

Bridge Operator's Process

- **BO.gen_socks5_cred()** generates a clean SOCKS5 credential for the Tor user to connect to the bridge.