HYPHAE: Social Secret Sharing

Isis Agora Lovecruft isis@patternsinthevoid.net
Henry de Valence hdevalence@hdevalence.ca

Provisional Draft, 2017-04-21

Abstract—The Tor network allows millions of people to access the Internet anonymously. Many people cannot access the Internet freely, and use the Tor network to bypass censorship. Many powerful adversaries, including state-level censors, therefore attempt to block access to the Tor network. To access Tor where it is blocked, users can connect to *Tor bridges*, secret entry points to the Tor network. The *bridge distribution problem* is to distribute the secret bridge addresses only to legitimate users, and not to censors – who have far greater resources than any single legitimate user. The Proximax and rBridge schemes propose using a social graph to distinguish legitimate users.

In this document, we review historical censorship of the Tor Network and the Proximax and rBridge proposals for bridge distribution. We also review an anonymous credential scheme proposed by Chase, Meiklejohn, and Zaverucha. We describe HYPHAE, a redesign of the rBridge concept using these credentials, as well as an anonymous micropayment system that may be of independent interest.

1 INTRODUCTION

"Please check the box to ensure you are not a robot" is a phrase all-too-often seen in user interfaces for the distribution of resources which require defenses against spammers, scrapers, Sybil attacks, and the like. The problem is to distinguish legitimate users from illegitimate users – to find a distinguishing property held only (or at least primarily) by legitimate users, and not by illegitimate ones. For instance, many services distinguish by ability to solve a CAPTCHA [VAMM+08]; by monitoring user requests to determine patterns of "malicious" activity [Pri16]; by possession of a scarce resource, such as a phone number; or by ability to pay a computational cost, as in proof-of-work systems. Less common suggestions include blockchain-based micropayments, state-issued "e-Passports," or possession of a trusted platform module [HG11].

A choice of distinguisher is a choice, implicitly or explicitly, of a conceptual model of the capabilities of legitimate and of illegitimate users. This model is often inadequate. For instance, CAPTCHAs rely on a conceptualization of legitimate users as "humans" and illegitimate users as "robots," as if scrapers were not controlled by humans, and as if humans did not use robots (their `User-Agents`) to make requests. Legitimate users who use unusual robots, such as a screen reader for the visually impaired, or who are not considered "human," such as those unable to read English-language instructions, are out of luck.

Sophisticated adversaries make choosing a good distinguisher much more difficult. Our resource is the addresses of *Tor bridges*, secret entrances to the Tor network; our adversaries are state-level censors seeking to learn the bridge addresses and block access to Tor. These adversaries have capabilities far beyond those of any single legitimate user. To distinguish legitimate users, therefore, Proximax [MML11] and rBridge [WLBH13] suggest using a social graph. Censors are more capable, more determined, and have more resources and more human hours than any legitimate user. They can solve CAPTCHAs, purchase scarce resources, or solve proofs-of-work. But legitimate users have friends.

In the rBridge proposal, users are assigned bridges, and earn reputation credits as long as their bridges remain unblocked. When one of a user's bridges is blocked, they can obtain a new bridge as long as they have a reputation for good behaviour. Users with good reputation can invite their friends into the system. Censors who attempt to enter the system and block bridges they receive lock themselves out of getting new bridges. The optimal strategy for a censor, then, is to enumerate bridges over time before blocking all of them at once; under reasonable assumptions, the majority of legitimate users remain unblocked.

However, the proposed rBridge protocol has several shortcomings, as described in Section 7. We therefore propose a redesign with the same overall concept, but with all cryptographic primitives and protocols replaced. We call our scheme *Hyphae*: although the social graph of legitimate users is a complex, interconnected, rhizomatic web, the bridge distributor sees only disconnected strands.

1.1 Organization

In Section 2, we provide a detailed history of attempts to censor access to the Tor network, as well as attempts to counter that censorship; readers familiar with this history can skip Section 2. In Section 3, we review two previous proposals for social bridge distribution, Proximax and rBridge; readers familiar with these can skip Section 3. In Section 4, we review the anonymous credential scheme proposed in [CMZ14]. In Section 5, we use these credentials to build an anonymous micropayment scheme. In Section 6, we describe our redesign of the rBridge concept, which we call *Hyphae*. In Section 7, we compare Hyphae to rBridge. In Section 8, we describe our implementation of Hyphae and its performance. Finally, Appendix A gives explicit descriptions of the Hyphae protocol and Appendix B describes an API.

All implementations provided are open-sourced, freely available, and licenced in the public domain. ♡

1.2 Notation

We use the following notation. $\mathbb{G}$ denotes a group of prime order ℓ, written additively. Since we model $\mathbb{G}$ using an elliptic curve group, we choose ℓ rather than p or q to avoid confusion between the scalars $\mathbb{Z}/\ell\mathbb{Z}$ and the base field for the elliptic curve.

Group elements (points) are denoted with capital letters, A, B, P, Q, etc. Scalars in $\mathbb{Z}/\ell\mathbb{Z}$ are denoted with lower-case letters m, n, b, u, etc. System parameters are denoted by greek letters α, β, ρ, etc.

We use Camenisch-Stadler [CS97] notation for zero-knowledge proofs: $\pi_{\text{label}} = \text{NIPK}\{(x, y, \ldots) : \texttt{statements}\}$ denotes that "π_{label} is a non-interactive proof-of-knowledge of secrets $x, y, \ldots$, which statisfy some `statements`, in zero-knowledge."

2 Censorship of the Tor Network

The Tor network is a low-latency anonymity network. Servers in the network are called *relays* (or *nodes*). The Tor network is semi-centralised: relays submit *descriptors* containing connection information, cryptographic keys, and other statistics and settings to a set of privileged relays, called *Directory Authorities*, which vote upon their views of relays in the network. The Directory Authorities then distribute a single, authoritative view of relays in the network, called a *consensus*, to clients attempting to connect to the network. After receiving the consensus, a client builds a path, called a *circuit*, through the network by encrypting their traffic to each relay (or *hop*) in the circuit in layers, like an onion: hence the names *onion encryption* and *onion routing*. This *circuit-layer* is telescoping, end-to-end encryption, and it lies inside an outer, superencrypting *link-layer*, which is implemented as point-to-point (i.e. non-telescoped) TLS connections between relays.

2.1 IP- and DNS-based blocking

Because the Tor consensus is publicly distributed, there is nothing preventing a censor from simply blocking client connections to the Directory Authorities themselves, or downloading the consensus and blocking connections to all the relays in it.

To combat censorship of the Tor network, Mathewson and Digledine introduced introduced Tor *bridge relays*, or more succinctly, *bridges* [DM06]. Bridges are Tor relays which are not listed in the consensus and are thus secret ingress points to the Tor network. This system requires a *bridge line* to be distributed out-of-band by a *bridge distributor*. The bridge line contains the information necessary to connect to a bridge, which includes the IP address and port, and optionally key material or other configuration data.

The introduction of Tor bridges eventually forced censors to switch to deep packet inspection (DPI) to continue censoring Tor effectively. In 2008, China began blocking access to The Tor Project's website by conducting a substring search for `torproject.org` on unencrypted HTTP traffic [WC12], which was trivially circumventable by requiring users to use Transport Layer Security (TLS) when accessing the website. The next stage of the arms race came in 2009, when China upgraded their DPI infrastructure to conduct IP-based blocking of all public relays in the Tor consensus [WC12]. The solution to this move, Tor bridges, had already existed for several years. In March 2011, a significant decrease from users in China to Tor bridges occurred [WC12]. While the mechanism is as-yet unknown, [Loe11] suggests this to be the result of Chinese network administrators manually discovering some of the most popular Tor bridges and adding specific DPI rules to block those bridge by IP.

2.2 TLS Fingerprinting and Active Probing

Eventually, many adversaries' infrastructures were improved to conduct DPI using more sophisticated distinguishers, which were frequently related to Tor's idiosyncratic usage of the TLS protocol.

In its original implementation, Tor used TLS in ways which presented adversaries with trivial distinguishers: disabling all unused certificate fields and extensions, sending an unique client ciphersuite list indicating actual cipher preferences, sending a two-certificate chain containing a long-term server identity key and an ephemeral link-key, and conducting the ephemeral agreement with the 1024-bit Diffie-Hellman (DH) group specified in RFC2409 §6.2 [Mat12]. In 2008, Tor switched to using the same ciphersuite list as Firefox 3 in order to "blend in" [Din08], which required feigning support for the nonstandard `SSL3_RSA_FIPS_WITH_3DES_EDE_CBC_SHA` cipher [Mat12]. At this time, Tor also altered relay link-handshake behaviour to instead send a single-certificate—rather than a two-certificate chain—and then have clients immediately renegotiate [Mat12]. This usage of TLS renegotiation caused further problems a year later: "When Marsh Ray's attack against renegotiation came out in 2009, and everybody who could possibly turn off renegotiation did so, our use of renegotiation stood out even more, especially since we had to keep doing it even when built with versions of OpenSSL that didn't support RFC 5746." [Mat12]

In October 2010, an anonymous user posted packet captures on Tor's bug tracker which showed that China had begun actively probing to distinguish Tor bridge user's traffic, pretending to speak the Tor protocol all the way up until the client's first `BEGIN_DIR` command (a tunnelled directory request for information on a relay, used to fetch the bridge's descriptor) and then "hanging up" [APKD10] [WL12]. Extensive testing revealed various curiosities in the configuration and behaviours of the Great Firewall of China, including that neither OpenSSL `s_client` connections nor excessively unordinary certifications would trigger an active probe from a the GFW [Wil11]. In December 2011, the Great Firewall of China began censoring connections to the Tor Network with a much cheaper distinguisher: Tor's unique TLS ciphersuite list as sent in the `CLIENT_HELLO` [KMD+11]. The solution, implemented in 2012, was automated retrieval of the current Firefox TLS ciphersuite preference list, by regexing over the directories of Firefox and OpenSSL source-code [Din12] [MF17].

At the beginning of 2011, Iran had also begun blocking Tor, based on the above-mentioned unusual choice of RFC2409 Diffie-Hellman parameters within the link-layer TLS handshake. The Tor Project developers responded by switching to the same DH parameters as were in currently

use by Apache's `mod_ssl` [Din11a]. The arms race with Iran continued throughout 2011, with Iran later updating their DPI infrastructure to distinguish traffic by the unusual expiration times of Tor's TLS certificates, and the Tor Project developers promptly releasing a new version with more realistic certificate expiration timestamps in the `SERVER_HELLO` [Din11c] [Din11b].

In June 2012, Ethiopia began blocking all TLS connections, including Tor traffic, by matching on the `TLS1_TXT_DHE_RSA_WITH_AES_256_SHA` string in the `CLIENT_HELLO` and dropping those packets [KSWF12]. Amusingly, if the client split the ciphersuite list across TLS cells [Win12a], or chose `TLS1_TXT_DHE_RSA_WITH_AES_128_SHA` instead [LLKW12], Ethiopia's DPI infrastructure was again unable to censor the traffic [Win12b]. Curiously, at the same time that Ethopia had begun fingerprinting the TLS `CLIENT_HELLO`, Kazakhstan followed suit, using the same fingerprint from Ethopia as well as an unconfirmed server-side TLS `HELLO_DONE` fingerprint. The unconfirmed fingerprint ceased to work to censor Tor in Kazakhstan soon after, presumably because Kazakhstan mis-re-configured their DPI infrastructure [SKWF12].

Finally, another mechanism to enumerate bridges was described in a 2012 paper [LLY+12]. Since bridge users select randomly from public Tor relays when constructing circuits, a middle relay can compare incoming connections against the public consensus to learn the addresses of bridges. The solution, originally proposed in 2011 along with a description of the problem, is *bridge guards* [ML15]. Tor's loose-source routing allows a relay to add arbitrary hops in the middle of a Tor circuit; bridges should choose a guard relay and relay their traffic through it. This feature is currently in development.

2.3 Pluggable Transports

The numerous list of distinguishers in TLS implementation and usage for Tor client traffic in the preceding section is by no means exhaustive. The domain names presented by bridges in their Tor TLS certificates are derived from the certificate public key, and as such are random. Another potential traffic distinguisher is the characteristic packet-size distribution, due to Tor's 512-byte cell size. In an effort to avoid these hazards, Kadianakis proposed *pluggable transports*, or *PTs* [KMDA16]. The original idea of pluggable transports was to provide a composable, or "pluggable", link layer which provided traffic obfuscation. The first proof-of-concept pluggable transport protocol, obfs2 [KM11b], used encryption to achieve computational indistinguishability from random, however for the goal of mere *obfuscation* rather than *computational hiding*, which was provided by the circuit-layer encryption.

The client and server, upon connecting, each sent the other a cryptographic seed in the clear. Both parties concatenated the two seeds together, hashed with SHA-256 a fixed number of times, and used the result as an AES-128 key and initialisation vector (IV) [KM11a]. While it would have been possible for a censor conducting DPI to re-derive the key and decrypt the obfs2 traffic to reveal the circuit-layer encryption, without a distinguisher to identify obfs2 traffic, their DPI infrastructure would have had to do this work for every unidentified connection.

The intended long-term goal was to build pluggable transports which mimicked, and hopefully were indistinguishable from, other protocols, in order to use other Internet traffic as cover traffic. For instance, one could imagine a PT which might make Tor traffic look like HTTP traffic.

However, most other Internet traffic, such as HTTP browsing, has incredible statistical complexity, and it seems implausible that it is possible to mimic, e.g., a human browsing the web without trivial distinguishers. This intuition, combined with several unsuccessful attempts in at protocol mimicry, suggest that pluggable transports which attempt protocol mimicry will always have distinguishers, in much the same way that Tor was never fooling anyone when it tried to look like Apache.

In contrast to efforts at protocol mimicry, the obfs4 [Ang14] pluggable transport protocol tries to look like random data, by using Tor's NTor circuit-layer handshake [GSU12] with public keys obfuscated to appear as uniformly random 32-byte strings, via the Elligator 2 mapping [BHKL13]. Once an elliptic curve Diffie-Hellman (ECDH) key exchange occurs, the obfs4 protocol wraps its traffic inside a layer of authenticated encryption.

We believe that obfs4, or any other pluggable transport which

1) uses a handshake design which is uniformly random *and* without distinguishers across multiple connections;
2) uses some pre-shared key material for authentication to the server; and
3) (optionally) encrypts to the pre-shared key, starting with the client's first message;

is essentially the ideal design for obfuscation.

2.4 The Bridge Distribution Problem

A pluggable transport can make traffic indistinguishable from random, but it cannot hide the destination of the traffic. Using pluggable transports to avoid censorship therefore requires preventing the censor from learning the addresses of the bridges. However, legitimate users should be able to learn the addresses of bridges (and other associated data) in order to bypass censorship. Distinguishing legitimate users from censors, and distributing secrets only to legitimate users, is the *bridge distribution problem*.

Tor bridges are historically distributed via a web interface which asks the user to solve a custom-implementation CAPTCHA, as well as via an email interface [LMFL13]. Of course, state-level censors are perfectly capable of purchasing email addresses in bulk, as well as circumventing a simple CAPTCHA. Several proof-of-work systems have been proposed informally over the last few years, including requiring clients to submit micropayments in some cryptocurrency, using one-out-of-many proofs [GK15], using proof-of-stake systems, and using proofs-of-work with asymmetric difficulties for the client and server [BK16].

However, as noted in the introduction, none of these are adequate methods to distinguish legitimate users from censors, who have more human resources, more computational power, more money, and more resolve than any single legitimate user. In contrast, the Proximax [MML11] and

rBridge [WLBH13] designs attempt to distinguish legitimate users from censors using something that should be easy for legitimate users and difficult for censors: the ability to make friends. For this reason, they attempt to use placement in a social graph, and a reputation system, to distinguish legitimate users from censors.

3 Previous Bridge Distribution Schemes

Previous work towards solving the distribution problem can be divided into roughly two categories: that which attempts to solve the problem directly, and that which attempts to sidestep the problem using ephemeral bridges.

3.1 Orthogonal problems and solutions

FlashProxy was a service run at Stanford University which used Javascript and the WebSockets API to turn visitors to a site into "flash proxies," or ephemeral channels for other censored users [Fif17a]. The FlashProxy authors defined a *rendezvous protocol* as follows:

Definition 1 (Rendezvous Protocol)**.** *"A rendezvous protocol lets a user in the censored region send and receive a small amount of information (a few bytes) from the circumvention system to outside the censored region, for the purpose of introducing a user to a proxy. Rendezvous protocols are designed for low-rate traffic and are intended to be difficult to block." [FHE+12]*

The Snowflake pluggable transport was later developed as an effort to use the FlashProxy concept with an improved NAT traversal algorithm using WebRTC [Han17].

FlashProxy and Snowflake attempt to sidestep the bridge distribution problem: if there is no way to distinguish legitimate users from censors, then one can, at least, attempt to distribute secrets faster than the censor can block them [FHE+12].

Domain fronting is a circumvention technique that hides the actual remote endpoint from a censor by tunneling actual requests in the `Host:` headers inside HTTPS requests to a third-party domain. [FLH+15] The third-party should be economically infeasible for the censor to block, e.g. a large content distribution netork (CDN) which delivers others services, such that blocking it would damage commercial viability. In the context of Tor bridges, the domain fronting technique has been used in the meek pluggable transport, which tunnels Tor client traffic through Google AppEngine, Microsoft Azure, and Amazon S3 to a Tor bridge [Fif17b].

However, using domain fronting to tunnel the client traffic itself has many problems, including poor scalability from high financial costs, and a reliance on the beneficence of large CDN providers. Building on the techniques of FlashProxy and Snowflake [Han16], we use domain fronting as our rendezvous protocol, but not for client traffic.

3.2 Proximax

Proximax [MML11] introduced two key ideas: *social distribution* and *bevaviour-based incentivisation*. In Proximax, some privileged subset of users are designated as *registered users*. Registered users are assigned proxies, which they distribute to their friends, who distribute them to their friends, and so on. The number of end users for a given proxy and how long the proxy lasts before being blocked determine the *efficacy* of the proxy's corresponding registered user. Registered users with higher efficacy are then given more proxy resources, and those with the highest efficacy are given the ability to invite new registered users.

However, if a proxy is blocked, Proximax has the "feature" of locking out the entire tree of users beneath that proxy's registered user: "Similar to the RICO Act in the legal system, once a subnode is suspicious the whole subtree is equally suspicious." [MML11] The wisdom of modelling software systems on U.S. anti-racketeering law aside, Proximax does not attempt to preserve privacy of registered users, or the social connections between users sharing the same proxy, all of whom are linkable to a particular registered user.

3.3 rBridge

In the rBridge scheme [WLBH13], users are given bridges and a reputation score. While a user's bridges remain in use and unblocked, the user earns additional points for this good behaviour. Users may purchase new bridges with their points. Users with a sufficiently good reputation may receive invite tokens from the distributor, allowing them to invite their friends.

The first key idea introduced in rBridge, extending Proximax's concepts of social distribution combined with behaviour-based incentivisation, is that of utilising a *user reputation* system. Where in Proximax a user is incentivised to invite a new user based on the predicted behaviour of the new user, and elimination from the system is based on social ties, the opposite is true of rBridge: invites are based solely on social ties and continued participation in the system is based on demonstration of good-behaviour. By blocking bridges, a censor forfeits their ability to earn points. This forces a censor to trade off more extensive enumeration of bridges with more extensive blocking of the bridges it learns.

The rBridge authors model three censorship strategies: *aggressive blocking*, wherein a censor immediately blocks all bridges they learn; *conservative blocking*, wherein a censor keeps some bridges online to earn points, while blocking others; and *event-driven blocking*, wherein a censor attempts to learn many bridges over time, before blocking all of them at some critical event. The most effective strategy for a censor is event-driven blocking, but under the assumption that only a small fraction of the initial userbase of the system is malicious, the majority of users maintain access to bridges.

The second key idea introduced in rBridge is a protocol for implementing the reputation-based incentive structure with privacy preservation, since it would be obviously unacceptable for the distributor to actually collect and maintain data on bridge users in censored areas. We discuss the cryptographic design of this protocol in Section 7.

We use the term *rBridge concept* to refer to the incentive structure and overall system design of rBridge described in Section 4 of [WLBH13], and use the term *rBridge protocol* to refer to the privacy-preserving protocol described in Section 5 of [WLBH13]. Hyphae, described in Section 6 of this paper, makes slight changes to the rBridge concept while replacing the rBridge protocol.

4 Keyed-Verification Anonymous Credentials from Algebraic MACs

In [CMZ14], Chase, Meiklejohn, and Zaverucha proposed an efficient keyed-verification anonymous credential scheme. In contrast to a setting where a user may need to present a credential to multiple parties or other users, a *keyed-verification* credential can only be verified by the issuer of the credential. As the authors note, this is suitable for "any setting in which the party controlling access to a resource also manages the accounts of authorized parties," and allows the use of efficient symmetric primitives to construct the credentials.

In particular, they describe two algebraic message authentication codes (MACs) and use each of them to construct keyed-verification anonymous credentials. The first MAC is a generalization of an algebraic MAC described by Dodis, Kiltz, Pietrzak, and Wichs [DKPW12], and has a security proof in the generic group model (GGM). The second is slightly less efficient, but admits a security proof under the decisional Diffie-Hellman (DDH) assumption. We choose the first, and repeat the construction of [CMZ14] here.

4.1 The Algebraic MAC

Fix a group $\mathbb{G}$ of prime order ℓ, written additively, and let A, B be generators of $\mathbb{G}$ so that $log_B(A)$ is unknown. An issuer's secret key is the vector

$$(\widetilde{x_0}, x_0, x_1, \ldots, x_n) \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^{n+2};$$

they may also publish *issuer parameters*

$$(X_1, \ldots, X_n) \leftarrow (x_1 A, \ldots, x_n A),$$
$$X_0 \leftarrow x_0 B + \widetilde{x_0} A,$$

which are used later to construct anonymous credentials.

The MAC works as follows. To tag a vector of attributes $(m_1, \ldots, m_n) \in (\mathbb{Z}/\ell\mathbb{Z})^n$, the issuer selects $b \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ and computes

$$P \leftarrow bB$$
$$Q \leftarrow \sum_{i=1}^{n} x_i m_i P = \left(\sum_{i=1}^{n} b x_i m_i\right) B.$$

The tag is then (P, Q). To verify a purported tag (P, Q) on attributes $(m_1, \ldots, m_n)$, the issuer checks that the nonce $P \neq 0$, then recomputes the MAC $Q' \leftarrow \sum_i x_i m_i P$ and checks that $Q = Q'$. Notice that although the attributes are authenticated, the tag itself is malleable: (tP, tQ) is a valid tag for the same attributes for any non-zero $t \in \mathbb{Z}/\ell\mathbb{Z}$.

4.2 Credential Issuance

To issue a credential with attributes $(m_1, \ldots, m_n) \in (\mathbb{Z}/\ell\mathbb{Z})^n$, the issuer creates a tag (P, Q) on those attributes, and returns the tag, together with a proof that the tag was created correctly:

$$\begin{aligned}\pi_{\text{ClearIssue}} = \text{NIPK}\{(\widetilde{x_0}, x_0, x_1, \ldots, x_n) : \\ X_i = x_i A \quad \forall i = 1, \ldots, n \\ \wedge\ X_0 = x_0 B + \widetilde{x_0} A \\ \wedge\ Q = x_0 P + \sum_{i=1}^{n} x_i (m_i P)\}.\end{aligned}$$

The user accepts the credential $(P, Q, m_1, \ldots, m_n)$ if π_{issuance} verifies. This ensures that a dishonest issuer cannot segment users by issuing to different users credentials with respect to different secret keys.

4.3 Blinded Issuance

It is also possible for the user to request issuance of a credential on attributes hidden from the issuer, as follows. Let the set of hidden indices be $\mathcal{H} \subseteq [1, \ldots, n]$, and let $\mathcal{H}^c$ be its complement, the set of non-hidden indices, so that $\mathcal{H} \cup \mathcal{H}^c = [1, \ldots, n]$.

First, the user generates an ephemeral ElGamal keypair (d, D) as $d \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$, $D \leftarrow dB$. For each $i \in \mathcal{H}$, the user then computes ElGamal encryptions

$$e_i \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z},$$
$$\text{Enc}_D(m_i B) \leftarrow (e_i B, m_i B + e_i D).$$

The user then sends D, $\{(i, m_i)\}_{i \notin \mathcal{H}}$, and $\{(i, \text{Enc}_D(m_i B))\}_{i \in \mathcal{H}}$ to the issuer, as well as a proof that the hidden attributes were correctly computed:

$$\begin{aligned}\pi_{\text{UserBlinding}} = \text{NIPK}\{(d, \{(e_i, m_i)\}_{i \in \mathcal{H}}) : \\ \text{Enc}_D(m_i B) = (e_i B, m_i B + e_i D) \quad \forall i \in \mathcal{H} \\ \wedge\ D = dB\}.\end{aligned}$$

The issuer then verifies $\pi_{\text{UserBlinding}}$, and uses the homomorphic property of ElGamal ciphertexts to compute an encryption of the MAC, as follows. First, the issuer selects a nonce:

$$b \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}, \qquad P \leftarrow bB.$$

The issuer then computes a partial MAC on the revealed attributes

$$Q_{\mathcal{H}^c} \leftarrow \left(x_0 + \sum_{i \notin \mathcal{H}} m_i x_i\right) P,$$

and encrypts it using randomness $s \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ to get

$$\text{Enc}_D(Q_{\mathcal{H}^c}) \leftarrow (sB, Q_{\mathcal{H}^c} + sD).$$

Let

$$Q_{\mathcal{H}} = \left(\sum_{i \in \mathcal{H}} m_i x_i\right) P$$

be the partial MAC on the hidden attributes, which the issuer cannot compute directly. Instead, the issuer uses the $\text{Enc}_D(m_i B)$ to compute

$$\text{Enc}_D(Q_{\mathcal{H}}) \leftarrow \sum_{i \in \mathcal{H}} x_i b \, \text{Enc}_D(m_i B),$$

and adds the two encryptions to get

$$\text{Enc}_D(Q) \leftarrow \text{Enc}_D(Q_{\mathcal{H}}) + \text{Enc}_D(Q_{\mathcal{H}^c}) = \text{Enc}_D(Q),$$

where $Q = (x_0 + \sum_i x_i m_i)P = Q_{\mathcal{H}} + Q_{\mathcal{H}^c}$.

To prove to the user that all this was done correctly, the issuer constructs the proof

$$\begin{aligned}\pi_{\text{BlindIssue}} = \text{NIPK}\{(\widetilde{x_0}, x_0, x_1, \ldots, x_n, s, b, \{t_i\}_{i\in\mathcal{H}}) :\\ X_i = x_i A \quad \forall i = 1, \ldots, n\\ \wedge\ X_0 = x_0 B + \widetilde{x_0} A\\ \wedge\ P = bB\\ \wedge\ T_i = bX_i \wedge T_i = t_i A \quad \forall i \in \mathcal{H}\\ \wedge\ \text{Enc}_D(Q)[0] = sB + \sum_{i\in\mathcal{H}} t_i \,\text{Enc}_D(m_i B)[0]\\ \wedge\ \text{Enc}_D(Q)[1] = sD + \sum_{i\in\mathcal{H}} t_i \,\text{Enc}_D(m_i B)[1]\\ + x_0 P + \sum_{i\notin\mathcal{H}} x_i (m_i P)\},\end{aligned}$$

where $t_i = bx_i$, $T_i = t_i A$ are auxiliary variables introduced to avoid proving statements involving secret products.

The issuer then sends P, $\text{Enc}_D(Q)$, T_i, and $\pi_{\text{BlindIssue}}$ to the user, who verifies the proof and decrypts $\text{Enc}_D(Q)$ to obtain the credential $(P, Q, m_1, \ldots, m_n)$.

4.4 Credential Presentation

To present a credential $(P_0, Q_0, m_1, \ldots, m_n)$ with an inital tag (P_0, Q_0) to the issuer while hiding some subset of the attributes indexed by $\mathcal{H} \subseteq [1, \ldots, n]$, the user proceeds as follows. First, the user uses the malleability of the tag to re-randomise their credential by computing

$$t \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z} \qquad (P, Q) \leftarrow (tP_0, tQ_0),$$

so that the issuer cannot trivially link credential presentations. The user then forms Pedersen commitments [Ped01] to the hidden attributes

$$z_i \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}, \qquad C_{m_i} \leftarrow m_i P + z_i A, \qquad \forall i \in \mathcal{H},$$

a Pedersen commitment to the MAC

$$z_Q \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}, \qquad C_Q \leftarrow Q + z_Q A,$$

and an error factor

$$V \leftarrow \sum_{i\in\mathcal{H}} z_i X_i - z_Q A$$

(using the issuer parameters X_i). Finally, the user creates a proof that these were generated correctly

$$\begin{aligned}\pi_{\text{CredShow}} = \text{NIPK}\{(\{(m_i, z_i)\}_{i\in\mathcal{H}}, z_Q) :\\ C_{m_i} = m_i P + z_i A \quad \forall i \in \mathcal{H}\\ \wedge\ V = \sum_{i\in\mathcal{H}} z_i X_i - z_Q A\},\end{aligned}$$

and sends $\{(i, m_i)\}_{i\notin\mathcal{H}}$, $\{(i, C_{m_i})\}_{i\in\mathcal{H}}$, C_Q, and π_{CredShow} to the issuer. The user may also optionally include additional statements about the attributes m_i in the proof π_{CredShow}.

To verify the credential presentation, the issuer attempts to recompute the MAC on the committed attributes and compares to the commitment to the MAC:

$$V' \leftarrow \left(x_0 + \sum_{i\notin\mathcal{H}} x_i m_i\right) P + \sum_{i\in\mathcal{H}} x_i C_{m_i} - C_Q.$$

If both parties are honest,

$$\begin{aligned}V' &= \left(x_0 + \sum_{i\notin\mathcal{H}} x_i m_i\right) P\\ &\quad + \sum_{i\in\mathcal{H}} x_i (m_i P + z_i A) - Q - z_Q A\\ &= \left(x_0 + \sum_{i=1}^{n} x_i m_i\right) P - Q + \sum_{i\in\mathcal{H}} z_i X_i - z_Q A\\ &= 0 + V,\end{aligned}$$

so this is exactly the error factor caused by randomness in the commitments. The issuer attempts to verify π_{CredShow} using V'. If the proof fails, the presentation is rejected; otherwise, the issuer now has the attributes and commitments $\{(i, m_i)\}_{i\notin\mathcal{H}}$, $\{(i, C_{m_i})\}_{i\in\mathcal{H}}$.

Since the commitments C_{m_i} are now authenticated, in the sense that the issuer knows they are commitments to the attributes of a credential it previously issued, they can then be used for further proof statements.

XXX describe parallel presentation of multiple credentials with shared attributes, establishing consistency at presentation instead of later.

4.5 Spend-Once Tokens

The credential scheme of [CMZ14] described in Sections 4.1–4.4 creates credentials which can be shown arbitrarily many times with absolute unlinkability. For bridge distribution, and other applications, tokens with spend-once semantics are required. (Here and throughout the paper, we use "token" to mean a credential that can only be spent once). To add spend-once semantics to these credentials, we use a database of nonces.

When requesting a token, the user selects a random nonce and requests blinded issuance of a credential containing the nonce, as well as other attributes. When presenting the token, the user reveals the nonce to the issuer, who stores it in a database of spent tokens. With our parameter choices, each nonce (a random scalar) fits into 32 bytes, so that the database of spent tokens can be implemented efficiently using, e.g., a bloom filter in front of a trie, or Redis.

Since the nonce is hidden from the issuer at issuance, the credential presentation is absolutely unlinkable from the credential creation; since the issuer is the verifier, a database of spent token identifiers does not impose additional centralization requirements; and since the underlying credentials are secure against forgery, a misbehaving user cannot alter the nonce and re-spend the token.

To prevent users from sharing tokens, we can anonymously bind a token to a user's identity as follows. Each user is given an account credential, with a random userid assigned by the issuer. The userid is then included as a hidden attribute on each token; whenever a user requests or uses a token, they present their account credential, and prove in zero knowledge that the userid of the account credential is consistent with the userid of the token.

Finally, we note that our token system bears resemblance to 1997 proposal by Stubblebin, Syverson, and Goldschlag called Unlinkable Serial Transactions (UST), wherein a user would exchange a blinded token while viewing a resource or receiving some service (a "transaction") from a provider

[SSG97] [SSG99]. When the user is finished with their transaction, the provider issues a new blinded token for the user's next transaction. The UST system also allowed detection of shared tokens, when multiple users try to use the same token simultaneously.

5 Hyphae Micropayments

Giving users points for good behaviour which they can anonymously exchange for services effectively entails an anonymous micropayment system. Rather than leaving that system implicit in our protocol, we describe it explicitly and seperately, in the hope that it is of independent interest.

Our system is highly restricted, allowing only a star topology for payments. Users may only receive credits from the server, and may only spend those credits at the server. However, this model is sufficient to build a privacy-preserving system where users pay for server resources they use, as long as the value of those resources is higher than the cost of payment processing. For instance, a web browser such as Brave could have users buy credits using money, and then anonymously spend those credits at a payment server in proportion to users' website usage; the payment server would then pay each website in money proportionate to the credits received.

Name	Description	Issuance	Presentation
u	User ID	Revealed	Hidden

Credential 1: Account Credential $\text{cred}_{\text{user}}$

Name	Description	Issuance	Presentation
u	User ID	Hidden	Hidden
n	Wallet Nonce	Hidden	Revealed
w	Wallet Balance	Hidden	Hidden

Credential 2: Wallet Token $\text{tok}_{\text{wallet}}$

Double-spend prevention is achieved by revealing the nonce n when the user wants to spend the token; the server stores a database of nonces. Applications expecting a high transaction frequency could prevent database growth by using key epochs to expire old tokens.

We assume in the rest of this section that each user has already been issued a credential $\text{cred}_{\text{user}}$ which has an attribute u containing a unique user id, and has already been issued a token $\text{tok}_{\text{wallet}}$ with matching u and $w = 0$, and hidden nonce n. This can be done when inducting new users into the system, for example, as in Section 6.1.

The scheme could be simplified slightly in other contexts by dropping the user credential $\text{cred}_{\text{user}}$; we use it to bind each wallet token to a long-term identity and enforce consistency among all of a user's tokens.

The maximum balance Ω is a system parameter with $\Omega = 3^\omega$. To ensure that balance amounts are in the range $[0, \Omega]$, we use the Back-Maxwell rangeproof described in [PBF+17]. For parameters m, n, orthogonal generators A, B, and input value v, it produces a Pedersen commitment to v and a proof that v lies in the interval $[0, m^n - 1] \subset \mathbb{Z}/\ell\mathbb{Z}$. The proof has size proportional to $1 + mn$, and is most efficient when $m = 3$.

5.1 Updating a User's Wallet

A user with wallet balance w can update their balance to $w' = w + c$ using the following protocol. Here, we assume that the wallet balances w, w' are hidden from the credential issuer, but the credit amount c is revealed. We allow c to be negative, giving one protocol for credit and debit transactions.

It is also possible to hide the credit amount c, as is done in Section 6.3, but we do not consider that scenario in this section because it requires proving that c has some protocol-specific relation to some other credentials.

To request that the server add c points to a user's wallet, the user proceeds as follows. The user re-randomises the tags on $\text{cred}_{\text{user}}$ and $\text{tok}_{\text{wallet}}$ to obtain $P_{\text{cred}_{\text{user}}}, Q_{\text{cred}_{\text{user}}}$ and $P_{\text{tok}_{\text{wallet}}}, Q_{\text{tok}_{\text{wallet}}}$. Next, they prepare presentations of $\text{cred}_{\text{user}}$ and $\text{tok}_{\text{wallet}}$. Since credential presentation produces Pedersen commitments to the hidden attributes (cf. Section 4), the user now has commitments

$$C_{u,\text{cred}_{\text{user}}} = uP_{\text{cred}_{\text{user}}} + z_{u,\text{cred}_{\text{user}}} A$$
$$C_{u,\text{tok}_{\text{wallet}}} = uP_{\text{tok}_{\text{wallet}}} + z_{u,\text{tok}_{\text{wallet}}} A$$

to the u values of $\text{cred}_{\text{user}}$ and $\text{tok}_{\text{wallet}}$ (which are equal for honest users), and a commitment

$$C_{w,\text{tok}_{\text{wallet}}} = wP_{\text{tok}_{\text{wallet}}} + z_w A$$

to the current wallet balance.

The user then prepares an issuance request for a new wallet token with userid u, balance $w' = w + c$, and nonce $n' \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$, producing an ElGamal keypair (d, D) and encryptions

$$\text{Enc}_D(uB) \leftarrow (e_u B, uB + e_u D)$$
$$\text{Enc}_D(n'B) \leftarrow (e_{n'} B, n'B + e_{n'} D)$$
$$\text{Enc}_D(w'B) \leftarrow (e_{w'} B, w'B + e_{w'} D).$$

Next, the user constructs a Back-Maxwell range proof that the new wallet balance $w' \in [0, \Omega]$, producing a commitment $C_{w',\text{range}} = w'B + z_{w'} A$ and a proof π_{Range} that $C_{w',\text{range}}$ commits to a value in $[0, \Omega]$.

Finally, the user sets $C_{w',\text{tok}_{\text{wallet}}} \leftarrow C_w + tP_w$ and constructs

$$\begin{aligned}\pi_{\text{UserCredit}} = \text{NIPK}\{(u, w, w', n', e_u, e_{w'}, e_{n'}, d, z_w, z_{u,1}, z_{u,2}) :\\ \text{Enc}_D(uB) = (e_u B, uB + e_u D)\\ \wedge\ \text{Enc}_D(n'B) = (e_{n'} B, n'B + e_{n'} D)\\ \wedge\ \text{Enc}_D(w'B) = (e_{w'} B, w'B + e_{w'} D)\\ \wedge\ D = dB\\ \wedge\ C_{u,\text{cred}_{\text{user}}} = uP_{\text{cred}_{\text{user}}} + z_{u,\text{cred}_{\text{user}}} A\\ \wedge\ C_{u,\text{tok}_{\text{wallet}}} = uP_{\text{tok}_{\text{wallet}}} + z_{u,\text{tok}_{\text{wallet}}} A\\ \wedge\ C_{w',\text{range}} = w'B + z_{w'} A\\ \wedge\ C_{w',\text{tok}_{\text{wallet}}} = w' P_{\text{tok}_{\text{wallet}}} + z_w A\},\end{aligned}$$

proving that the encryptions are well-formed, that the user id is consistent, and that the new balance is $w' = w + c$. The user sends the presentation proofs for $\text{cred}_{\text{user}}$ and $\text{tok}_{\text{wallet}}$, along with D, $\text{Enc}_D(uB)$, $\text{Enc}_D(w'B)$, $\text{Enc}_D(n'B)$, $C_{w',\text{range}}$, π_{Range}, and $\pi_{\text{UserCredit}}$ to the issuer.

The issuer verifies the range proof π_{Range}, verifies the presentation proofs to obtain authenticated commitments

$C_{u,\text{cred}_{\text{user}}}$, $C_{u,\text{tok}_{\text{wallet}}}$, C_w, recomputes $C_{w'} \leftarrow C_w + cP_w$, and uses these to verify $\pi_{\text{UserCredit}}$.

If the request verifies correctly, the issuer uses $\text{Enc}_D(uB)$, $\text{Enc}_D(n'B)$, and $\text{Enc}_D(w'B)$ to issue a new wallet token with attributes (u, n', w').

6 The Hyphae Protocol

In this section we describe the Hyphae protocol, our redesign of the rBridge concept for social bridge distribution. Hyphae retains the overall incentive structure of rBridge, but replaces the protocol. An overview of the rBridge concept can be found in Section 3.3, and a comparison between Hyphae and rBridge can be found in Section 7.

We use the anonymous credential scheme of [CMZ14] described in Sections 4.1–4.4, and the micropayment scheme described in Section 5. Each user has a user credential $\text{cred}_{\text{user}}$, as well as a wallet token $\text{tok}_{\text{wallet}}$ holding their reputation credit. The distributor issues bridge tokens $\text{tok}_{\text{bridge}}$, which represent a user's knowledge of a given bridge at some time. Users can later redeem their bridge tokens to increase their reputation credit. Finally, users can use their reputation to purchase invite tokens to invite their friends into the system.

The protocol uses multiple credential types, defined inline. The distributor generates a secret key and issuer parameters for each credential type (user credentials, bridge tokens, wallet tokens, etc.); we assume throughout the protocol description that each user has in hand the distributor's issuer parameters for each credential type.

6.1 Account Creation

To create an account, a prospective user needs an *invite token*. An invite token is a credential with a single attribute, an invite code. (Creation of invite tokens is discussed in Section 6.6).

Name	Description	Issuance	Presentation
n	Invite token nonce	Hidden	Revealed

Credential 3: Invite Token

Name	Description	Issuance	Presentation
u	User ID	Revealed	Hidden

Credential 4: Account Credential

To redeem an invite token, a prospective user proceeds as follows. First, they parse their invite code to the token $(P_0, Q_0, n_{\text{invite}})$, then re-randomises the tag by computing

$$t \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z} \qquad (P, Q) \leftarrow (tP_0, tQ_0).$$

They then create an ElGamal keypair (d, D) and choose nonces $n_0, n_1, \ldots, n_\kappa \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$, where κ is the system parameter denoting the number of bridge tokens per user. Next, they create encryptions

$$e_i \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z} \qquad \text{Enc}_D(n_iB) \leftarrow (e_iB, n_iB + e_iD)$$

for each $i = 0, \ldots, \kappa$, and prove these were created correctly:

$$\pi_{\text{UserNonces}} = \text{NIPK}\{(d, e_0, \ldots, e_\kappa, n_0, \ldots, n_\kappa : \text{Enc}_D(n_iB) = (e_iB, n_iB + e_iD) \forall i \in [0, \ldots, n] \wedge D = dB\}.$$

The prospective user sends the invite token $(P, Q, n_{\text{invite}})$, along with the $\text{Enc}_D(n_iB)$ and $\pi_{\text{UserNonces}}$.

The distributor verifies the invite token and marks it as spent. If successful, the distributor choooses a random userid $u \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ and uses $\text{Enc}_D(n_0B)$ to issue an initial wallet token with userid u, nonce n_0, and balance $b = \beta\kappa + 1$. This is sufficent to purchase κ initial bridges at β credits each. Since purchasing a bridge requires a previous bridge token, the distributor uses the $\text{Enc}_D(n_iB)$, $i = 1, \ldots, n$, to create κ bridge tokens with bridge id $b = 0$. The user can present these while purchasing a new bridge (as the bridge id is hidden then), but cannot use them to claim credit (as the bridge id is revealed then, and the distributor rejects the request).

The distributor sends the user credential, wallet token, and initial bridge tokens to the new user, who uses the protocol of Section 6.4 to purchase κ bridges using their initial bridge tokens. Since these requests are indistinguishable from any other users' bridge purchases, the only way for the distributor to track a user's initial bridge assignment is by observing the timing of requests. We therefore propose that the user's client buys a single bridge immediately (and linkably), so they can connect to the network, and adds random delays to stagger the remaining $\kappa - 1$ bridge purchases over a longer time period.

6.2 Bridge Tokens

Users collect points as long as their bridges remain unblocked. For the purposes of this section, we assume that the distributor already maintains a list of blocked bridges.

A *bridge token* represents a proof that a particular user was assigned a particular bridge at a particular time. Times are measured in coarse units since an epoch t_0; the time resolution is a system parameter τ. Bridges are identified by a *Bridge ID*, a hash of some canonical representation of the bridgeline[1].

Bridge tokens are used in three ways. First, users present them to claim reputation credit, as described in Section 6.3. Second, users present them when buying a new bridge, ensuring that each user has a fixed number of valid bridge tokens, as described in Section 6.4. Third, users present them when reporting that a bridge is blocked, as described in Section 6.5.

Reporting that a bridge is blocked requires revealing the bridge b. This should be unlinkable from buying a new bridge, so that the distributor cannot track bridge assignments. We therefore use two nonces, to give the following semantics: a bridge token may either be used to claim reputation credit, or it may be used to buy a new bridge and to report that a bridge is blocked. This is summarized in Credential 5, where H denotes that the attribute is hidden and R that it is revealed.

1. Simply hashing the bridgeline is not sufficient, as the pluggable transport arguments are unordered, free-form key-value pairs.

Name	Description	Issue	Claim	Buy	Blocked
n_1	Nonce	H	R	R	H
n_2	Nonce	H	R	H	R
u	User ID	H	H	H	H
t	Timestamp	R	H	H	H
b	Bridge ID	R	R	H	R

Credential 5: Bridge Token

6.3 Earning Reputation with Bridge Tokens

A user with a bridge token $\text{tok}_{\text{bridge}}$ issued at time t can claim credit at time t' as follows. At time t', the user has earned credit $c = \rho(t' - t)$, where ρ is a system parameter representing the rate of reputation credit per bridge per time. The user presents their bridge token and wallet token to the distributor. If the bridge remains unblocked, the distributor issues a new bridge token with updated timestamp t', and a new wallet token with updated balance $w' = w + c$. If the bridge has been blocked, the distributor issues only a new wallet token, with the original balance w.

The user must reveal the bridge ID, so that the distributor can verify the bridge remains unblocked. Revealing the original timestamp t would make requests highly linkable; to hide it, we amend the protocol of Section 5.1. Instead of revealing the credit amount c, the user commits to it and to the original timestamp t; the distributor uses these commitments to verify that the claimed credit amount is correct.

As in Section 5.1, the user user re-randomises the tags on $\text{cred}_{\text{user}}$, $\text{tok}_{\text{bridge}}$, and $\text{tok}_{\text{wallet}}$ to obtain

$$P_{\text{cred}_{\text{user}}}, Q_{\text{cred}_{\text{user}}}$$
$$P_{\text{tok}_{\text{bridge}}}, Q_{\text{tok}_{\text{bridge}}}$$
$$P_{\text{tok}_{\text{wallet}}}, Q_{\text{tok}_{\text{wallet}}}.$$

Next, they prepare presentations of $\text{cred}_{\text{user}}$, $\text{tok}_{\text{bridge}}$, and $\text{tok}_{\text{wallet}}$. This produces commitments

$$C_{u,\text{cred}_{\text{user}}} = uP_{\text{cred}_{\text{user}}} + z_{u,\text{cred}_{\text{user}}}A$$
$$C_{u,\text{tok}_{\text{bridge}}} = uP_{\text{tok}_{\text{bridge}}} + z_{u,\text{tok}_{\text{bridge}}}A$$
$$C_{u,\text{tok}_{\text{wallet}}} = uP_{\text{tok}_{\text{wallet}}} + z_{u,\text{tok}_{\text{wallet}}}A$$

to the u values of the user's credentials (which are equal for honest users), a commitment

$$C_w = wP_{\text{tok}_{\text{wallet}}} + z_w A$$

to the current wallet balance w, and a commitment

$$C_t = tP_{\text{tok}_{\text{bridge}}} + z_t A$$

to the bridge token timestamp t. The user then chooses randomness $z_{w'} \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ and computes commitments

$$C_{c,\text{tok}_{\text{bridge}}} \leftarrow cP_{\text{tok}_{\text{bridge}}} - z_t A$$
$$C_{c,\text{tok}_{\text{wallet}}} \leftarrow cP_{\text{tok}_{\text{wallet}}} + (z_{w'} - z_b)A$$

to the credit amount c, and uses $C_{c,\text{tok}_{\text{wallet}}}$ to compute

$$C_{w'} \leftarrow C_w + C_{c,\text{tok}_{\text{wallet}}} = w'P_{\text{tok}_{\text{wallet}}} + z_{w'}A.$$

To request issuance of new wallet and bridge tokens, the user chooses nonces $n'_1, n'_2, n'' \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$, an ephemeral ElGamal keypair (d, D), and creates encryptions $\text{Enc}_D(uB)$, $\text{Enc}_D(n'_1 B)$, $\text{Enc}_D(n'_2 B)$, $\text{Enc}_D(n''B)$, $\text{Enc}_D(w'B)$.

Finally, the user computes the proof

$$\begin{aligned}\pi_{\text{BridgeCredit}} = \text{NIPK}\{&(u, w, w', n'_1, n'_2, n'', e_u, e_w, e_{w'}, e_{n'}, e_{n''}, d,\\ &z_t, z_w, z_{w'}, z_{u,\text{cred}_{\text{user}}}, z_{u,\text{tok}_{\text{bridge}}}, z_{u,\text{tok}_{\text{wallet}}}) :\\ &\text{Enc}_D(uB) = (e_u B, uB + e_u D)\\ \wedge\ &\text{Enc}_D(wB) = (e_w W, w'W + e_{w'} D)\\ \wedge\ &\text{Enc}_D(w'B) = (e_{w'} W, w'W + e_{w'} D)\\ \wedge\ &\text{Enc}_D(n'_1 B) = (e_{n'_1} B, n'_1 B + e_{n'_1} D)\\ \wedge\ &\text{Enc}_D(n'_2 B) = (e_{n'_2} B, n'_2 B + e_{n'_2} D)\\ \wedge\ &\text{Enc}_D(n''B) = (e_{n''} B, n''B + e_{n''} D)\\ \wedge\ &D = dB\\ \wedge\ &C_{u,\text{cred}_{\text{user}}} = uP_{\text{cred}_{\text{user}}} + z_{u,\text{cred}_{\text{user}}}A\\ \wedge\ &C_{u,\text{tok}_{\text{bridge}}} = uP_{\text{tok}_{\text{bridge}}} + z_{u,\text{tok}_{\text{bridge}}}A\\ \wedge\ &C_{u,\text{tok}_{\text{wallet}}} = uP_{\text{tok}_{\text{wallet}}} + z_{u,\text{tok}_{\text{wallet}}}A\\ \wedge\ &C_{c,\text{tok}_{\text{bridge}}} = cP_{\text{tok}_{\text{bridge}}} - z_t A\\ \wedge\ &C_w = wP_{\text{tok}_{\text{wallet}}} + z_w A\\ \wedge\ &C_{w'} = w'P_{\text{tok}_{\text{wallet}}} + z_{w'}A\},\end{aligned}$$

and sends to the distributor the presentation proofs for $\text{cred}_{\text{user}}$, $\text{tok}_{\text{wallet}}$, and $\text{tok}_{\text{bridge}}$ along with $C_{c,\text{tok}_{\text{wallet}}}$, D, $\text{Enc}_D(uB)$, $\text{Enc}_D(n'_1 B)$, $\text{Enc}_D(n'_2 B)$, $\text{Enc}_D(n''B)$, $\text{Enc}_D(w'B)$, and $\pi_{\text{BridgeCredit}}$.

The distributor verifies the presentation proofs, producing authenticated commitments $C_{u,\text{cred}_{\text{user}}}$, $C_{u,\text{tok}_{\text{bridge}}}$, $C_{u,\text{tok}_{\text{wallet}}}$, and C_t. The distributor then recomputes $C_{c,\text{tok}_{\text{bridge}}}$ as

$$C_{c,\text{tok}_{\text{bridge}}} \leftarrow \rho(t' P_{\text{tok}_{\text{bridge}}} - C_t);$$

if the user is honest this is

$$C_{c,\text{tok}_{\text{bridge}}} = \rho(t' - t)P_{\text{tok}_{\text{bridge}}} - z_t A = cP_{\text{tok}_{\text{bridge}}} - z_t A.$$

Next, the distributor recomputes $C_{w'}$ as

$$C_{w'} \leftarrow C_w + C_{c,\text{tok}_{\text{wallet}}};$$

if the user is honest this is

$$\begin{aligned}C_{w'} &= (w + c)P_{\text{tok}_{\text{wallet}}} + (z_w + z_{w'} - z_w)A\\ &= w'P_{\text{tok}_{\text{wallet}}} + z_{w'}A.\end{aligned}$$

The distributor uses these commitments to verify $\pi_{\text{BridgeCredit}}$. If the proof fails, the distributor aborts. Otherwise, the distributor checks whether the user's bridge has been blocked, and marks the bridge and wallet tokens as spent. If the bridge remains unblocked, the distributor issues a new wallet token using $\text{Enc}_D(uB)$, $\text{Enc}_D(w'B)$, $\text{Enc}_D(n''B)$, and a new bridge token using $\text{Enc}_D(uB)$, $\text{Enc}_D(n'_1 B)$, $\text{Enc}_D(n'_2 B)$, t', and b. If the bridge was blocked, the distributor issues only a new wallet token using $\text{Enc}_D(uB)$, $\text{Enc}_D(wB)$, and $\text{Enc}_D(n''B)$.

This protocol requires that the user and the distributor both agree on the current timestamp t'. While agreement on time is a hard problem in general, in this setting exact consistency is not required. First, if the time resolution is sufficiently coarse-grained (e.g., on the scale of hours), the server can add a grace period for requests near a timestamp boundary. Second, since the distributor does not mark the

tokens as spent if the proof fails to verify, the user can retry their request later with the same tokens.

6.4 Buying New Bridges

Users with sufficient reputation can use their credits to buy new bridges. To cap the number of bridges a user may use to collect points, purchasing a new bridge requires invalidating a previous bridge token.

To purchase a new bridge, a user proceeds as follows. First, the user prepares presentations of their user credential $\text{cred}_{\text{user}}$ and current bridge token $\text{tok}_{\text{bridge}}$. Here, the user hides all attributes of $\text{tok}_{\text{bridge}}$ except the nonce n_1. This prevents the distributor from partitioning users based on their bridge assignments. These presentations produce commitments

$$C_{u,\text{cred}_{\text{user}}} = uP_{\text{cred}_{\text{user}}} + z_{u,\text{cred}_{\text{user}}}A$$
$$C_{u,\text{tok}_{\text{bridge}}} = uP_{\text{tok}_{\text{bridge}}} + z_{u,\text{tok}_{\text{bridge}}}A.$$

The cost of a new bridge is a system parameter β. The user prepares a wallet transaction as in Section 5.1, debiting their wallet by β. The user then prepares a request for a new bridge token, choosing nonces $n_1', n_2' \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$, an ElGamal keypair (d, D), and creating encryptions $\text{Enc}_D(n_1'B)$, $\text{Enc}_D(n_2'B)$, $\text{Enc}_D(uB)$. The user then creates a proof

$$\begin{aligned}\pi_{\text{BuyBridge}} = \text{NIPK}\{(u, n_1', n_2', e_u, e_{n_1'}, e_{n_2'}, d, z_{u,\text{cred}_{\text{user}}}, z_{u,\text{tok}_{\text{bridge}}}) :\\ \text{Enc}_D(uB) = (e_uB, uB + e_uD)\\ \wedge\ \text{Enc}_D(n_1'B) = (e_{n_1'}B, n_1'B + e_{n_1'}D)\\ \wedge\ \text{Enc}_D(n_2'B) = (e_{n_2'}B, n_2'B + e_{n_2'}D)\\ \wedge\ C_{u,\text{cred}_{\text{user}}} = uP_{\text{cred}_{\text{user}}} + z_{u,\text{cred}_{\text{user}}}A\\ \wedge\ C_{u,\text{tok}_{\text{bridge}}} = uP_{\text{tok}_{\text{bridge}}} + z_{u,\text{tok}_{\text{bridge}}}A\},\end{aligned}$$

and sends to the distributor the presentations of $\text{cred}_{\text{user}}$ and $\text{tok}_{\text{bridge}}$, the wallet transaction, D, $\text{Enc}_D(uB)$, $\text{Enc}_D(n_1'B)$, $\text{Enc}_D(n_2'B)$, and $\pi_{\text{BridgeCredit}}$.

The distributor verifies the presentation proofs to obtain $C_{u,\text{cred}_{\text{user}}}$ and $C_{u,\text{tok}_{\text{bridge}}}$, verifies $\pi_{\text{BuyBridge}}$ to ensure $\text{Enc}_D(n_1'B)$, $\text{Enc}_D(n_2'B)$, and $\text{Enc}_D(uB)$ are well-formed, and verifies the wallet transaction. The distributor then issues a new wallet token (as described in Section 5.1), and chooses a new bridge, using $\text{Enc}_D(n_1'B)$, $\text{Enc}_D(n_2'B)$, and $\text{Enc}_D(uB)$ to issue a new bridge token with the current timestamp t, and returns the new wallet and bridge tokens to the user.

6.5 Reporting Blocked Bridges

Users who can no longer access a bridge can use their bridge token to report that a bridge is blocked. To do this, a user presents their bridge token $\text{tok}_{\text{bridge}}$, revealing the nonce n_2 and the bridge b, and hiding all other attributes. The user can then later present the same token to buy a replacement bridge unlinkably.

Because users reveal n_2 when claiming reputation (as described in Section 6.3), they cannot claim reputation on a bridge they have reported as blocked. Similarly, they cannot re-use an old bridge token to repeatedly report the same bridge as blocked. This means that a user can only report a blocked bridge if they were issued that bridge, and they may only do so once. It is therefore difficult for a malicious actor to falsify blocking reports at scale.

In Section 6.2, we assume that the distributor maintains a list of blocked bridges. Instead of relying on a seperate measurement infrastructure, the distributor can use the authenticated reports of blocked bridges it receives to decide when a bridge has been blocked.

6.6 Inviting New Users

Users with sufficient reputation can also use their credits to buy invite tokens to give to their friends. An invite token is a credential containing a single nonce and no other information. With our parameter choices, the entire token (P, Q, n) is only 96 bytes, so it is easy to share.

The cost of a new account is α, a system parameter. To purchase an invite token, an existing user with wallet balance w prepares a transaction as in Section 5.1, requesting a new wallet with balance $w' = w - \alpha$, and chooses a random nonce $n \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ and prepares a blind issuance request for an invite token.

The user sends these requests to the distributor, who verifies them and issues a new wallet token with an updated balance and an invite token. The existing user can then send the invite token to a friend using another channel, such as a QR code, `base64`-encoded string, etc.

6.7 Parameter Choices

The system parameters for Hyphae are α, the cost of a new account; β, the cost of a new bridge; τ, the timestamp resolution; ρ, the credit rate; κ, the number of bridge tokens; and $\Omega = 3^\omega$, the maximum wallet balance.

The parameter choices in the original rBridge design are not directly comparable, for two reasons. First, they are derived from estimates of the number of Tor bridges and bridge users circa 2011–2012, which are no longer current. Second, our scheme has slightly different invitation behaviour than rBridge: in rBridge, users are assigned invite tokens, while in Hyphae they purchase them. This change is discussed in Section 7.

XXX Choose updated parameter choices based on simulator, rescale credit units so that the maximum balance is a power of 3.

7 Analysis and Comparison to rBridge

Hyphae retains the rBridge concept and leaves the incentive structure relatively unchanged, but entirely replaces the suggested privacy-preserving protocol. In this section, we describe differences between Hyphae and rBridge.

7.1 Differences between Hyphae and the rBridge protocol

In the rBridge protocol, users use oblivious transfer (OT) [NP01] to prevent the distributor from learning their initial bridge assignments. We avoid OT entirely by allowing users to purchase their initial bridges unlinkably from their account creation. The rBridge protocol also allows a user to request additional bridges in the event they recieve a duplicate bridge during OT. We do not handle this case explicitly; in the

unlikely event that a user gets assigned duplicate bridges, the protocol is unaffected – they simply have two tokens for the same bridge – and they will continue to earn reputation at the same rate as any other user.

The rBridge protocol uses k-times anonymous authentication (k-TAA) signatures [ASM06], a modification of the BBS+ signature scheme [CL04], which require pairings. These credentials may be shown k times unlinkably, but the value of k chosen for rBridge is unspecified. Instead, we use the anonymous credentials of [CMZ14], as described in Section 4, which may be shown arbitrarily many times unlinkably. They are also much smaller, faster, simpler, and easier to implement.

7.2 Differences between Hyphae and the rBridge concept

In addition to the cryptographic redesign, Hyphae also makes a few comparatively minor changes to the incentive structures of rBridge.

In rBridge, the total amount of reputation a user can earn from a single bridge is capped at 300 credits. The stated rationale is to prevent a malicious user from holding one bridge open to gain reputation, and using it to buy and report other bridges to a censor. However, the cost of a new bridge is only 45 credits, meaning that a malicious user just needs to hold one in six bridges open and can still carry out the attack. We could amend Hyphae to cap the reputation earned per bridge, by adding a "credit earned" attribute to a bridge token, incrementing it in zero-knowledge, and proving that the earned reputation was below some bound using a rangeproof. But this seems like a significant increase in complexity for relatively small decrease in the censor's capability.

rBridge also requires a user's bridge to stay online for a certain time period (75 days) before a user is able to earn any credit from it. The stated rationale is to provide incentive for a censor not to block bridges immediately. However, this requirement affects both censors and legitimate users equally; legitimate users seeking to invite their friends must wait an additional two months before they can start earning reputation. Because this requirement does not differentially affect censors, we drop it.

Finally, in rBridge, users with sufficient credit are eligible to request an invite token. The distributor randomly decides, according to a probability based on the available bridge resources, to grant or deny the request. In order for the request to be granted, the user must additionally supply proof that sufficient time has passed since their last request, in order to prevent a malicious user from repeatedly requesting invite tokens to increase the odds of receiving one.

However, this design also fails to differentially affect censors versus legitimate users. Instead, we allow Hyphae users to purchase their invite tokens in the same manner as they would purchase a bridge. This change simplifies the protocol, and allows implementers to re-use the same procedures for purchasing bridges and purchasing invite tokens.

8 Implementation

This document is currently in draft state, and the following section is to be completed as Hyphae is implemented. Currently, this section provides merely notes on the implementation progress and ideas. Minor changes may be made to the protocol in the process of implementation.

We plan to implement Hyphae in Rust using Decaf for Curve25519 [Ham15] to provide a model for a prime-order group.

9 Acknowledgements

The authors are very grateful to Open Technology Fund for both their financial support of this work and the community of excellent people and projects they have assembled. We also thank Ian Goldberg for his insights, valuable discussions, and shared ideas; George Danezis for originally suggesting we look at [CMZ14]; and Tony Arcieri, Oleg Andreev, and George Tankersley for helpful discussions.

References

[Ang14] Yawning Angel. *obfs4: the obfourscator*, May 2014. (Specification).

[APKD10] Anonymous, Mike Perry, George Kadianakis, and Roger Dingledine. Tor trac ticket #4185: Bridge easily detected by gfw. October 2010. (Ticket #4185).

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In *International Conference on Security and Cryptography for Networks*, pages 111–125, Berlin Heidelberg, 2006. Springer. (PDF).

[BHKL13] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 967–980. ACM, 2013. (PDF).

[BK16] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. *Proceedings of NDSS'16, 21–24 February 2016, San Diego, CA, USA. ISBN 1-891562-41-X*, 2016.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Annual International Cryptology Conference*, pages 56–72, Berlin Heidelberg, 2004. Springer. (PDF).

[CMZ14] Melissa Chase, Sarah Meiklejon, and Greg Zavuchera. Algebraic macs and keyed-verification anonymous credentials. ACM CCS, 2014. (PDF).

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in CryptologyCRYPTO'97*, pages 410–424. Springer, 1997. (PDF).

[Din08] Roger Dingledine. Tor changelog: Changes in version 0.2.1.1-alpha - 2008-06-13. Technical report, The Tor Project, June 2008. (Changelog Entry).

[Din11a] Roger Dingledine. Tor changelog: Changes in version 0.2.2.22-alpha - 2011-01-25. Technical report, The Tor Project, February 2011. (Changelog Entry).

[Din11b] Roger Dingledine. Tor changelog: Changes in version 0.2.3.4-alpha - 2011-09-13. Technical report, The Tor Project, September 2011. (Changelog Entry).

[Din11c] Roger Dingledine. Tor trac ticket #4014: Iran filters tor by ssl handshake, sept 2011, September 2011. (Ticket #4014).

[Din12] Roger Dingledine. Tor changelog: Changes in version 0.2.3.17-beta - 2012-06-15. Technical report, The Tor Project, June 2012. (Changelog Entry).

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 355–374. Springer, 2012. (PDF).

[DM06] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system. Technical Report 2006-1, The Tor Project, November 2006. (PDF).

[FHE+12] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Roger Dingledine, Phil Porras, and Dan Boneh. Evading censorship with browser-based proxies. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*, pages 239–258. Springer, July 2012. (PDF).

[Fif17a] David Fifield. *Flash Proxies*, 2013–2017. (Webpage).

[Fif17b] David Fifield. Meek documentation. 2014–2017. (Webpage).

[FLH+15] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015. (PDF).

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 253–280. Springer, 2015.

[GSU12] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, pages 1–25, 2012. (PDF).

[Ham15] Mike Hamburg. Decaf: Eliminating cofactors through point compression. In *Annual Cryptology Conference*, pages 705–723. Springer, 2015. (PDF).

[Han16] Serene Han. *Introducing Snowflake (webrtc pt)*. The Tor Project, January 2016. (Archived Email).

[Han17] Serene Han. *Snowflake*. The Tor Project, April 2017. (Webpage).

[HG11] Ryan Henry and Ian Goldberg. Formalizing anonymous blacklisting systems. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, May 2011. (PDF).

[KM11a] George Kadianakis and Nick Mathewson. obfs2.c. Technical report, The Tor Project, 2011. (Sourcecode).

[KM11b] George Kadianakis and Nick Mathewson. Obfsproxy architecture. Technical report, The Tor Project, December 2011. (Specification).

[KMD+11] George Kadianakis, Nick Mathewson, Roger Dingledine, Philipp Winter, Arturo Filastó, Tom Ritter, and Runa Sandvik. Tor trac ticket #4744: Gfw probes based on tor's ssl cipher list, December 2011. (Ticket #4744).

[KMDA16] George Kadianakis, Nick Mathewson, Roger Dingledine, and Yawning Angel. Pluggable transport specification, version 1. Technical report, The Tor Project, 2010-2016. (Specification).

[KSWF12] George Kadianakis, Runa Sandvik, Philipp Winter, and David Fifield. Tor Trac Ticket #6045: Ethiopia blocks Tor based on ServerHello. June 2012. (Ticket #6045).

[LLKW12] Isis Lovecruft, Karsten Loesing, George Kadianakis, and Philipp Winter. Tor Trac Ticket #6414: Automating Bridge Reachability Testing. June 2012. (Ticket #6414).

[LLY+12] Zhen Ling, Junzhou Luo, Wei Yu, Ming Yang, and Xinwen Fu. Extensive analysis and large-scale empirical evaluation of tor bridge discovery. In *INFOCOM, 2012 Proceedings IEEE*, pages 2381–2389. IEEE, 2012. (PDF).

[LMFL13] Karsten Loesing, Nick Mathewson, Matthew Finkel, and Isis Lovecruft. *BridgeDB Specification*. The Tor Project, 2013. (Specification).

[Loe11] Karsten Loesing. Case study: Learning whether a Tor bridge is blocked by looking at its aggregate usage statistics. Technical report, The Tor Project, 2011. (PDF).

[Mat12] Nick Mathewson. Tor tls history. 2012. (Archived Wiki Page).

[MF17] Nick Mathewson and Arturo Filastò. get_mozilla_ciphers.py. Tor Project Gitweb, 2011-2017. (Sourcecode).

[ML15] Nick Mathewson and Isis Lovecruft. *Bridge Guards and other anti-enumeration defenses*. The Tor Project, 2011-2015. (Specification).

[MML11] Damon McCoy, Jose Andre Morales, and Kirill Levchenko. Proximax: A measurement based system for proxies dissemination. *Financial Cryptography and Data Security*, 5(9):10, 2011. (PDF).

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2001. (PDF).

[PBF+17] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *4th Workshop on Bitcoin and Blockchain Research*. Blockstream, 2017. (PDF).

[Ped01] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Lecture Notes in Computer Science*, volume 576, pages 129 – 140, 2001.

[Pri16] Matthew Prince. The Trouble with Tor. Cloudflare Blog, 2016. (Blogpost).

[SKWF12] Runa Sandvik, George Kadianakis, Philipp Winter, and David Fifield. Tor Trac Ticket #6140: Kazakhstan uses DPI to block Tor. June 2012. (Ticket #6140).

[SSG97] Paul Syverson, Stuart Stubblebine, and David Goldschlag. Unlinkable serial transactions. In *Financial Cryptography*, pages 39–55. Springer, 1997. (PDF).

[SSG99] Stuart G Stubblebine, Paul F Syverson, and David M Goldschlag. Unlinkable serial transactions: protocols and applications. *ACM Transactions on Information and System Security (TISSEC)*, 2(4):354–389, 1999. (PDF).

[VAMM+08] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008. (PDF).

[WC12] Philipp Winter and Jedidiah R . Crandall. The Great Firewall of China: How it blocks Tor and why it is hard to pinpoint. *USENIX ;login:*, 37(6), 2012. (PDF).

[Wil11] Tim Wilde. Tor Bug 4185 Testing and Report. Technical report, Team Cymru, December 2011. (Webpage).

[Win12a] Philipp Winter. brdgrd (Bridge Guard). Github, 2012. (Sourcecode).

[Win12b] Philipp Winter. How the Great Firewall of China is Blocking Tor. 2012. (Webpage).

[WL12] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is blocking Tor. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI 2012)*, August 2012. (PDF).

[WLBH13] Qiyan Wang, Zi Lin, Nikita Borisov, and Nicholas J. Hopper. rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation. In *Proceedings of the Network and Distributed System Security Symposium - NDSS'13*. Internet Society, February 2013. (PDF).

APPENDIX A
EXPLICIT DESCRIPTION OF HYPHAE

This section will contain explicit pseudocode for the Hyphae protocol.

A.1 Account Creation

A.2 Earning Reputation with Bridge Tokens

A.3 Buying New Bridges

A.4 Reporting Blocked Bridges

A.5 Inviting New Users

APPENDIX B
INTERFACING WITH HYPHAE

Here we sketch a high-level outline of a Tor Browser user flow and present a compatible RESTful API design.

For the following, we assume that the distributor's domain is `bridges.torproject.org`and that the distributor is also running a meek reflector on some popular CDN services, which we will say is at `bridgedistributor.majorcloudprovider.com`. We assume, for all communications with the distributor, that the user is building a new Tor circuit for each API request,

and speaking over this Tor circuit to the distributor's meek reflector. [2]

B.1 Account Creation

Provided a user has an invite token, which is 96 bytes in length (or 128 bytes, if base64-encoded), they build a Tor circuit to the bridge distributor's meek reflector (at `bridgedistributor.majorcloudprovider.com`), and construct an account creation request, as specified in Section 6.6, and send a JSON-RPC request to `/account/create`. If the invite token is valid and previously unspent, the distributor responds with HTTP status code `201 CREATED` and a body which contains the newly-created account credential, a wallet token containing sufficient initial balance for k bridges, and k (blank) bridge tokens, as in Section 6.1. Otherwise, if there is some error, the bridge distributor responds with `403 FORBIDDEN`, which optionally may contain some explanation of the error.

B.2 Buying New Bridges

Afterwards, the user may purchase their first bridge. To do so, the user constructs the proofs and other supplementary data described in Section 6.4, and send it to `/wallet/debit`. If the proofs are valid, the distributor responds with `200 OK` and a response body containing the new bridge token and the user's updated wallet. Otherwise, if the proofs regarding correct payment and wallet formation do not verify, the server responds with `402 PAYMENT REQUIRED`. [3] For other errors, the bridge distributor responds with `403 FORBIDDEN`, which optionally may contain some explanation of the error.

As described in Section 6.6, the user can then stagger their other bridge purchases to acchieve absolute unlinkability.

B.3 Earning Reputation with Bridge Tokens

When requesting that their wallet be credited for some bridge token, the user constructs proofs as outlined in Section 6.2 and sends these to `/wallet/credit`. If the distributor agrees to credit the wallet, it responds `200 OK` with a body containing the new wallet. Otherwise, it responds `403 FORBIDDEN` with no new wallet, and the user is expected to simply use the old one again later.

B.4 Reporting Blocked Bridges

When a user wishes to report a bridge is blocked, it prepares the request according to Section 6.5, and sends it to `/token/blocked`. The distributor always responds to any blocking report—valid, invalid, duplicate, or otherwise—with status code `451 UNAVAILABLE FOR LEGAL REASONS`, regardless of what it thinks or had previously thought about the bridge being blocked. That is, the `451` code is essentially just an acknowledgement of receipt. The distributor may then optionally evaluate the likelihood of the bridge being blocked according to some heuristics and, optionally, add it to the database of blocked bridges.

B.5 Inviting New Users

Finally, to invite a friend to the system, the user prepares proofs and an invite request, as in Section 6.6, and sends this to `/account/invite`. If the proofs and the request are valid, the distributor responds with `201 CREATED` and the invite token, which the user can then give to their friend. If the proofs regarding correct payment and wallet formation do not verify, the server responds with `402 PAYMENT REQUIRED`. Otherwise, if some other problem occurs, the distributor responds with `403 FORBIDDEN` and no new wallet (as above, for buying a new bridge).

2. If the user's Tor is not working, for example, for users in censored regions, we assume that the user falls back to using only meek, and we further assume that the meek reflector is honest, and does not forward or log the user's IP address or other identifying information. However, if the meek reflector does do so, the only information learned is "a user at this IP address requested a new account and their first bridge at this time" but that information is still absolutely unlinkable to all future transactions, including ones regarding the first bridge.

3. This is, as far as we know, the first use of the HTTP 402 status code for a micropayments scheme, as was originally intended.