rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation

Qiyan Wang
Department of Computer Science
University of Illinois at Urbana-Champaign
qwang26@illinois.edu

Zi Lin
Department of Computer Science
University of Minnesota
lin@cs.umn.edu

Nikita Borisov
Department of Electrical & Computer Engineering
University of Illinois at Urbana-Champaign
nikita@illinois.edu

Nicholas J. Hopper
Department of Computer Science
University of Minnesota
hopper@cs.umn.edu

Abstract

Tor is one of the most popular censorship circumvention systems; it uses bridges run by volunteers as proxies to evade censorship. A key challenge to the Tor circumvention system is to distribute bridges to a large number of users while avoiding having the bridges fall into the hands of corrupt users. We propose rBridge—a user reputation system for bridge distribution; it assigns bridges according to the past history of users to limit corrupt users from repeatedly blocking bridges, and employs an introduction-based mechanism to invite new users while resisting Sybil attacks. Our evaluation results show that rBridge provides much stronger protection for bridges than any existing scheme. We also address another important challenge to the bridge distribution—preserving the privacy of users' bridge assignment information, which can be exploited by malicious parties to degrade users' anonymity in anonymous communication.

1 Introduction

Censoring the Internet is a means adopted by many repressive regimes to control the information that their citizens can access on the Internet. Many websites that allow people to exchange political ideas (e.g., Facebook, Twitter, and Flickr) or may provide political information contrary to the state's agenda (e.g., YouTube, Wikipedia, and CNN) have been blocked by the repressive governments [23]. As of April 2012, 7 out of the top 10 non-Chinese websites[1] (70%) were blocked or partially blocked, and in total, 3716 out of 14521 websites (26%) were blocked, by the "Great Firewall of China"[2]. To further tighten the Internet censorship, the Chinese government employs an Internet police force of over 30 000 people to constantly monitor the citizens' Internet usage [13].

A typical approach to skirting censorship is to deploy circumvention proxies outside the censored network, which can provide indirect access to blocked websites. Tor [10] is one of the most popular proxy-based circumvention systems; it uses *bridges* run by volunteers as proxies to evade censorship. Censors are, however, eager to discover such bridges and block them as well. A particularly powerful approach to enumerating bridges is the *insider attack*, wherein the censor colludes with corrupt users to discover and shut down bridges; the censor can further amplify the attack by deploying a large number of *Sybils* to accelerate the discovery of bridges.

To make enumeration more difficult, the bridge distributor gives a limited number of bridges to each user. But this creates a new privacy problem, as this information could be used to fingerprint the user: an adversary who can monitor the bridges used for a set of anonymous tunnels can potentially link them back to the user. As a result, the bridge distributor becomes a fully trusted component, whereas other components of Tor must be at most honest-but-curious.

In this work, we make the following contributions to both bridge protection and user privacy preservation:

1) We propose *rBridge*—a user reputation system for bridge distribution. rBridge computes users' reputation based on the uptime of their assigned bridges, and allows a user to replace a blocked bridge by paying a certain amount of reputation credits; this prevents corrupt users from re-

[1] `http://www.alexa.com/topsites`

[2] `https://zh.greatfire.org`

peatedly blocking bridges. In addition, high-reputation users are granted opportunities to invite friends into the system. The introduction-based approach ensures the system can steadily grow the user base as recruiting new bridges, while preventing adversaries from inserting a large number of corrupt users or Sybils into the system. We performed extensive evaluation to show that rBridge provides much stronger protection for bridges than any existing scheme; for instance, the number of user-hours served by bridges in rBridge is at least one order of magnitude more than that of the state-of-the-art proxy distribution scheme [15].

2) For privacy-preserving bridge distribution, the bridge-related information on users' reputation profiles must be managed by the users themselves to avoid leaking the information to the bridge distributor. This raises the problem that malicious users could cheat the reputation system by manipulating their records. In this work, we propose a novel privacy-preserving user reputation scheme for bridge distribution, which can not only ensure the bridge distributor learns nothing about users' bridge assignment, but also prevent corrupt users from cheating. To our best knowledge, rBridge is the first scheme that is able to perfectly preserve users' privacy in bridge distribution. We implemented the privacy-preserving scheme, and experimental results show that rBridge has reasonable performance.

The rest of this paper is organized as follows. We introduce related work in Section 2. Section 3 presents the basic concepts, including design goals, threat model, and the scope of this work. In Section 4, we elaborate the basic rBridge scheme without privacy preservation and provide evaluation results. Section 5 presents the privacy-preserving scheme and performance evaluation results. We analyze potential attacks in Section 6, and conclude in Section 7.

2 Background and Related Work

2.1 Tor and Bridge Distribution

Tor [10] is primarily an anonymous communication system. A Tor user randomly selects 3 relays to build an onion encryption tunnel to communicate with a target anonymously. As of May 4 2012, there are about 3 000 relays in the Tor network [1]. The selection of relays must be kept private from any entity, because otherwise an adversary could likely link the user to his communicating target. To ensure this, each user is required to download all the relays' descriptors (from a directory authority or a Tor relay) even though he only needs 3 relays, and make his selection locally.

Recently, Tor has been increasingly used as a censorship circumvention tool. Users in a censored country can use Tor relays as proxies to access blocked sites. However, since all of the Tor relays are publicly listed, many countries (e.g., China) have blocked the public Tor relays altogether. In response, Tor turned to private relays run by volunteers, called *bridges*, to circumvent censorship. A key challenge though is to distribute the addresses of bridges to a large number of users without exposing them to the censor.

The bridge distribution strategies that have been deployed by Tor are to give a small subset of bridges to each user, as identified by a unique IP address or a Gmail account. Unfortunately, these cannot survive a powerful adversary who can access a large number of IP addresses and Gmail accounts to create a large number of Sybils; the Chinese government were able to enumerate all the bridges distributed using these strategies in under a month [2]. The alternative approaches adopted by Tor employ more stringent distribution strategies: the bridges are given to a few trusted people in censored countries in an ad hoc manner, who further disseminate the bridges to their social networks; or, individuals deploy their private bridges and give the bridges' addresses only to trusted contacts. However, the stringent bridge distribution can only reach a very limited fraction of potential bridge users and restrict the openness of the system.

2.2 Proxy Distribution Strategies

Researchers have tried to design better proxy distribution strategies [11, 14, 15, 20]. Feamster et al. [11] proposed a *keyspace-hopping* mechanism for proxy distribution, which employs computational puzzles to prevent a corrupt user from learning a large number of proxies. However, this mechanism is not likely to withstand an adversary who has strong computational power; the results of [11] show that 95% of 100 000 proxies would be discovered if the adversary can solve about 300 000 puzzles. In the scheme proposed by Sovran et al. [20], the address of a proxy is given to a few highly trusted people who play as internal proxies to relay other users' traffic to the external proxy; the addresses of these forwarders are advertised by performing random walks on social networks. However, this scheme is unable to provide users reliable circumvention service as forwarders may go offline from time to time; besides, the forwarders (residing in the censored country) could receive a severe penalty for facilitating circumvention, which may make people hesitate to serve as forwarders.

Mahdian [14] studied the proxy distribution problem from an algorithmic point of view, and theoretically analyzed the lower bound of the number of proxies required to survive a certain number of malicious insiders. Nevertheless, their scheme is not practical, as it assumes that the number of corrupt users is known in advance and there is no limit on the capacity of each proxy. Recently, McCoy et al. [15] proposed *Proximax*, which leverages social networks for proxy distribution and distributes proxies based

on the efficiency of each distribution channel to maximize the overall usage of all proxies. In this work, we explicitly compare rBridge with Proximax and show that rBridge is able to provide much stronger protection for bridges than Proximax.

We note that none of the existing proxy distribution strategies is able to preserve users' privacy. They assume the proxy distributor is fully trusted and authorized to know which proxies are given to a particular user. Applying these distribution strategies to Tor would degrade users' anonymity in anonymous communication.

2.3 Anonymous Authentication and Anonymous Reputation Systems

Researchers have put forward several designs for anonymous authentication and anonymous reputation systems [7, 8, 21, 22] that are similar to what we are seeking. Au et al. [8] proposed a k-times anonymous authentication (k-TAA) scheme that allows a user to be authenticated anonymously for a bounded number of times. The work in [21,22] extended this scheme to allow revocation without trusted third parties. Later, Au et al. [7] further extended the anonymous authentication schemes to support users' reputation management. We note that, however, none of these schemes is applicable to bridge distribution due to inability to limit misbehavior of malicious users. In bridge distribution, a user's reputation that is calculated based on the user's bridge assignment records should be managed by the user himself to avoid leaking the bridge information to the bridge distributor, which raises the risk that a malicious user could manipulate his reputation records, e.g., increasing his credit balance. Whereas, in the aforementioned schemes, users' reputation is calculated by servers that are trusted to perform the reputation calculation, and thus they do not need to consider potential cheating of malicious users.

3 Concept

In this section, we present the design goals, threat model, and scope of this work.

3.1 Goals

rBridge aims to achieve the following goals:

1. *Maximized user-hours of bridges*: McCoy et al. [15] proposed the metric *user-hours* to evaluate the robustness of a proxy distribution strategy. It represents the sum of hours that a bridge can serve for all of its users before being blocked.

2. *Minimized thirsty-hours of users*: Another important aspect, which is overlooked by prior work, is thirstiness of honest users. We use *thirsty-hours* to measure the time that an honest user has no bridge to use. We aim to minimize it to ensure high quality of service.

3. *Healthy growth of the user base*: We assume the bridge distributor can recruit new bridges from time to time, and each bridge can support up to a certain number of users due to limited capacity. The consumption of bridges is due to either new user joining or bridge blocking. By "healthy growth of the user base", we mean the user base can grow correspondingly as new bridges are added to the system, without causing thirstiness of existing users. For an ineffective bridge distribution strategy, corrupt users can drain out the bridge resource, leaving little ability to grow the user base.

4. *Privacy preservation of bridge assignment*: We aim to prevent any entity (e.g., a curious bridge distributor) from learning any information about bridge assignment of a particular user; such information can be exploited to degrade the user's anonymity.

We note that 4) distinguishes rBridge from prior work, as none of the existing approaches preserves users' bridge assignment information. For 1), 2), and 3), we shall show that rBridge can achieve much higher performance than any existing approach.

It is important to note that similar to prior work, we are not interested in ensuring a single or a few important individuals can access unblocked bridges. Instead, we aim to provide the circumvention service to the **majority of users**; in other words, it is possible that a few honest users could lose all their bridges before boosting their reputation to receive new bridges. Providing guaranteed circumvention service to a few special users can be easily achieved by deploying a few exclusive circumvention proxies; however, we believe it is more valuable to provide the circumvention service to a large number of ordinary users.

3.2 Threat Model

We consider a state-level adversary (i.e., the censor), who has access to rich human resource, i.e., controlling a substantial number of potential bridge users. In rBridge, a new user needs an *invitation ticket* (which is probabilistically distributed to high-reputation users) to register and join the system. A registered malicious user can block his assigned bridges by revealing them to the censor who can later block the bridges (referred to as the *insider attack*). Typically, the censor would like to block as many bridges as quickly as possible, but in some instances she can adopt

other strategies, such as keeping known bridges unblocked for some period of time to boost the number of insiders and later performing a massive blocking attempt in a crisis. In general, we assume the set of malicious users is a Byzantine adversary, and can deviate from the protocol in arbitrary ways to maximize their chance of blocking bridges. The adversary could also launch the *Sybil attack* by creating a large number of fake accounts in the population of potential bridge users. We note that, however, the Sybils can help the adversary discover bridges only if they can get registered. In addition, we assume that the adversary can access substantial network resources, e.g., a large number of IP addresses and Email accounts, but she has bounded computational power and is unable to subvert widely used cryptographic systems.

Unlike the existing schemes [2, 11, 14, 15, 20] that assume the bridge distributor is fully trusted, we consider an honest-but-curious model for the bridge distributor, which is within the threat model of Tor [10]. More specifically, we assume the bridge distributor honestly follows the protocol, but is interested in learning any private information about users, such as which bridges are assigned to a particular user. For ease of presentation, we assume there is a single bridge distributor, but it is straightforward to duplicate the bridge distributor by creating multiple mirrored servers.

3.3 Scope

For clarity, we do not attempt to address network-level bridge discovery. We assume the censor is able to learn bridges only from the distribution channels (i.e., based on the knowledge of registered corrupt users and Sybils). It is possible that the censor employs other techniques to discover bridges. For instance, the censor could try to probe all IP addresses on the Internet to find hosts that run Tor handshake protocols, fingerprint Tor traffic to identify bridges, or monitor the users who connect to a discovered bridge to see what other TLS connections these users establish and try to further verify whether the connected hosts are bridges [2]. We note that if the censor were able to identify bridges using such network-level bridge discovery techniques, any bridge distribution strategy would not be able to work. Defending against such attacks is an active research area; researchers have been proposing various defense mechanisms, such as obfsproxy [3], BridgeSPA [19], and client password authorization [4]. We acknowledge that effective mechanisms for resisting the network-level bridge discovery are important research problems, but they are orthogonal to this work.

In rBridge, users' reputation is calculated based on the uptime of their assigned bridges, which requires a mechanism to test reachability of bridges from censored countries. Recently, the Tor project has proposed several methods to accurately test bridges' availability [6], and we expect these mechanisms to be deployed soon. To clarify, we assume the availability information of bridges can be provided by the Tor network, and how to reliably check the bridges' reachability is out of the scope of this work.

4 The Basic rBridge Scheme

In this section, we present the basic rBridge scheme that does not provide privacy preservation, i.e., the bridge distributor knows the bridge assignment details of each user. We elaborate the privacy-preserving scheme in Section 5.

4.1 Overview

The *openness* of a proxy-based censorship circumvention system and its *robustness* to the insider attack seem to be in conflict. On the one hand, allowing anyone to join the system and get bridges allows malicious users to quickly enumerate all of the bridges [2]. On the other hand, applying highly stringent restrictions on user registration and bridge distribution (e.g., giving bridges only to highly trusted people using social networks) enhances robustness, but makes it hard for the majority of potential bridge users to get bridges.

Our key insight is that it is possible to bridge the gap between the openness and robustness of bridge distribution by building a user reputation system. Instead of trying to keep all malicious users outside the system, we adopt a less restrictive user invitation mechanism to ensure the bridge distribution can reach a large number of potential users; in particular, we use a loosely trusted social network for user invitation, and a well-behaving user can invite his less close friends into the system. Meanwhile, we leverage a user reputation system to punish blockers and limit them from repeatedly blocking bridges; more specifically, each user earns credits based on the uptime of his bridges, needs to pay credits to get a new bridge, and is provided opportunities to invite new users only if his credit balance is above a certain threshold.

It is important to note that our goal is not to keep bridges unblocked forever; instead, we try to achieve a more practical goal—having bridges serve a sufficiently long period of time so that the overall rate of recruiting new bridges outpaces the rate of losing bridges. We also note that this is still a very challenging problem; as will be shown by the comparison results (Section 4.3.3), the existing schemes have a difficult time protecting bridges from being blocked even for a very limited period of time.

4.2 Scheme

When joining the system, a new user U receives k bridges $B_1, \cdots, B_k$ as well as a *credential*, which is used to verify

U as a legitimate registered user. The credential is signed by the bridge distributor D and includes the following information:

$$\mathtt{U} \| \Phi \| \{B_i, \tau_i, \phi_i\}_{i=1}^{k}$$

wherein Φ denotes the total credits owned by U, τ_i denotes the time when B_i is given to U, and ϕ_i denotes the credits that U has earned from B_i. (At the initialization, $\Phi = 0$, $\phi_i = 0$, and τ_i is the joining time). The selection of the bridges $B_1, \cdots, B_k$ is at random. D keeps counting the number of users assigned to each bridge and stops giving a bridge's address to new users once the number of users assigned to the bridge reaches an upper limit (denoted by g).

4.2.1 Earning Credits

U is given credits based on the uptime of his bridges. The credit assignment policy should have the following properties. First, it should provide incentives for corrupt users to keep the bridge alive for at least a certain period of time, say T_0 days, which should be long enough to make sure enough new bridges can be recruited in time to maintain the overall bridge resources in the system. Second, the total credits that a user earns from a bridge should be upper-bounded, to prevent corrupt users from keeping one bridge alive to continuously earn credits and using the earned credits to request and block other bridges.

Now, we define the credit assignment function $\mathsf{Credit}(\cdot)$. Let T_{cur} denote the current time, and β_i denote the time when B_i gets blocked (if B_i is not blocked yet, $\beta_i = \infty$). We define t as the length of the time period from the time when U knows B_i to the time when B_i gets blocked or the current time if B_i is not blocked, i.e., $t = \min\{\beta_i, T_{cur}\} - \tau_i$. We let ρ denote the rate of earning credits from a bridge (credits/day) and T_1 denote the upper-bound time by which U can earn credits from the bridge. Then, the amount of credits ϕ_i earned from B_i is defined as:

$$\phi_i = \mathsf{Credit}(t) = \begin{cases} 0 & t < T_0 \\ (t - T_0) \cdot \rho & T_0 \le t \le T_1 \\ (T_1 - T_0) \cdot \rho & t > T_1 \end{cases}$$

Without loss of generality, we define $\rho = 1$ credit/day; then, the maximum credits that a user can earn from a bridge are $(T_1 - T_0)$.

From time to time (e.g., before requesting a new bridge), U requests D to update his credit balance Φ with his recently earned credits, say, from B_i. D first validates U's credential (verifying the tagged signature), and then re-calculates the credits $\tilde{\phi}_i$ according to the uptime of B_i, adds the difference $\tilde{\phi}_i - \phi_i$ to Φ, updates ϕ_i with $\tilde{\phi}_i$, and finally re-signs the updated credential.

4.2.2 Getting a New Bridge

To limit the number of bridges that a corrupt user knows, we allow each user to have k or fewer alive bridges at any time. This is enforced by granting a new bridge to a user U only if one of his bridges (say B_b) has been blocked. In particular, upon a request for a new bridge in replace of B_b, D first verifies that B_b is in U's credential and has been blocked. D also checks whether U has enough credits to pay for a new bridge, i.e., $\Phi > \phi^-$, where ϕ^- is the price for a new bridge.

After giving out a new bridge $\tilde{B}_b$, D updates U's credential by replacing the record $\{B_b, \tau_b, \phi_b\}$ with $\{\tilde{B}_b, T_{cur}, 0\}$ and updating the total credits with $\tilde{\Phi} = \Phi - \phi^-$. To prevent a malicious user from re-using his old credentials that has more credits, D keeps a list of expired credentials (e.g., storing the hash value of the credential); once U's credential is updated, the old credential is added to the expired credential list and cannot be used again.

We note that temporarily blocking a bridge just to create an open spot for a new bridge does not help a corrupt user, because he still needs to pay the same amount of credits to get a new bridge and the availability loss of a temporarily blocked bridge is strictly smaller than that of a permanently blocked bridge.

4.2.3 Inviting New Users

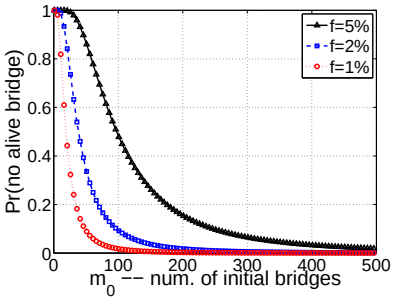
D periodically sends out *invitation tickets* to high-reputation users whose credit balances are higher than the threshold Φ_θ. Since the censor may let some corrupt users behave legitimately to simply accumulate credits and obtain invitation tickets in order to deploy more corrupt users or Sybils in the system, we let D randomly select the recipients of invitation tickets from qualified users. A user who has received an invitation ticket can give it to any of his friends, who can later use the ticket to join the system.

Note that the system needs to reserve a certain fraction (e.g., 50%) of bridge resource (i.e., the sum of the remaining capacity of unblocked bridges) for potential replacement of blocked bridges for existing users, while using the rest bridge resource to invite new users. The amount of reserved resource can be dynamically adjusted according to the amount of current bridge resource and the plans for growing the user base and recruiting new bridges.

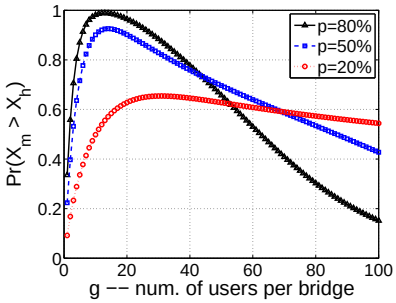
4.3 Evaluation and Comparison

We now analyze the robustness of rBridge against the following blocking strategies, and compare it with Proximax [15]. We discuss other potential attacks in Section 6.

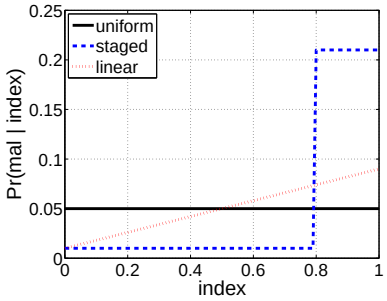
- *Aggressive blocking:* The censor is eager to block discovered bridges, i.e., shutting down the bridge once it is known to a corrupt user.



(a) Number of initial bridges ($N = 1000$)



(b) Number of users per bridge (p is attack probability, $f = 5\%$)



(c) Probability distribution of malicious users ($f = 5\%$)

Figure 1: Parameter selection

- *Conservative blocking:* A sophisticated censor may keep some bridges alive for a certain period of time to accumulate credits, and use the credits to discover new bridges and/or invite more corrupt users.
- *Event-driven blocking:* The censor may dramatically tighten the control of the Internet access when certain events (e.g., crisis) take place. We consider such attacks by assuming that malicious users do not block any bridges until a certain time, when suddenly all the discovered bridges get blocked.

To evaluate rBridge under these attacks, we implemented an event-based simulator using a timing-based priority queue, by treating each state change of the system as an event, such as inviting a new user, getting a new bridge, blocking a bridge, recruiting a new bridge, etc. Each event contains a time stamp indicating when the event occurs as well as an ID of the subject indicating who will carry out the event. We start with choosing the parameters for our simulation.

4.3.1 Parameter Selection

We employ probabilistic analysis to select appropriate parameters. To simplify the parameter calculation, we consider a static user group (i.e., no new users join the system); later, we validate our parameter selection in a dynamic setting using the event-based simulator. In practice, the bridge distributor can periodically re-calculate the parameters (e.g., every 30 days) using the current size of the user group.

Initial setup. Let f denote the fraction of malicious users among all potential bridge users (note that f is not the actual ratio of malicious users in the system). We expect a typical value of f between 1% and 5%, but we also evaluate rBridge with much higher f to see its robustness in extreme cases. The system starts with $N = 1000$ users, which are randomly selected from the pool of all potential bridge users; for instance, `D` could randomly select a number of Chinese users on Twitter (based on their profiles) as the initial bridge users, and very likely these users are willing to use the bridge based circumvention service because they already used some circumvention tools to access Twitter (which is blocked in China).

Each user is initially provided $k = 3$ bridges[3]. Suppose there are m_0 initial bridges in the system; the number of users per bridge is $g_0 = \frac{N \cdot k}{m_0}$ on average. Assuming a corrupt user blocks all of his bridges, the probability that an honest user has no alive bridge is $(1-(1-f)^{g_0})^k$. According to Figure 1a, we choose $m_0 = 200$ to make sure the majority of users can survive the initial blocking.

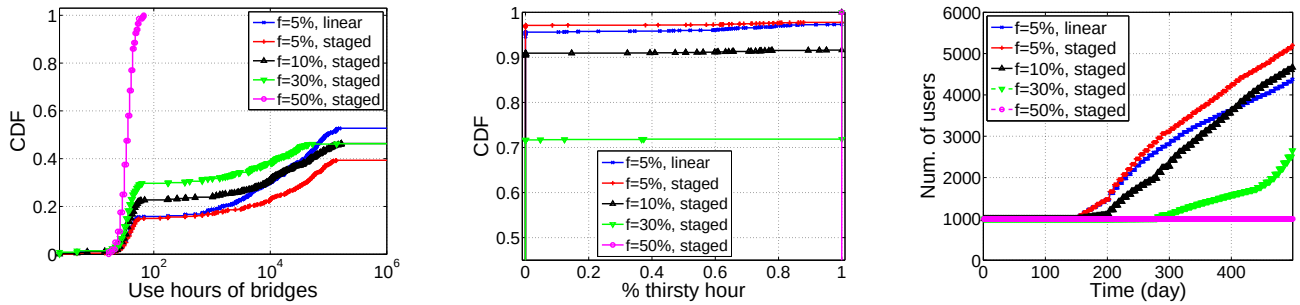
g—the maximum number of users per bridge. In rBridge, when a bridge gets blocked, all the g users sharing this bridge will be "punished" (i.e., receive no more credits from the bridge and need to pay credits to get a new bridge); intuitively, with a smaller g, it is easier to precisely punish the real blocker, as fewer honest users would be punished by mistake. On the other hand, we should make sure g is sufficiently large to avoid underusing the bridges.

Here, we calculate the probability that a user has a certain number of blocked bridges; this probability depends on g and determines the punishment on the user. Let p denote the probability that a corrupt user blocks a bridge he knows, and λ denote the probability that a bridge is blocked. Then, we have $\lambda = 1-(1-f \cdot p)^g$ (here we use f to approximate the ratio of corrupt users in the system). We define X_h (or X_m) as the number of blocked bridges of an honest (or corrupt) user. Assuming a user obtains l bridges since joining the system, we get:

$$Pr(X_h = x) = \binom{l}{x} \cdot (1-\lambda)^x \cdot \lambda^{l-x} \quad (1)$$

$$Pr(X_m = x) = \binom{l-p\cdot l}{x-p\cdot l} \cdot (1-\lambda)^{x-p\cdot l}\lambda^{l-x} \quad (2)$$

[3]In the current bridge distribution strategy deployed by Tor, each requesting user is given 3 different bridges.



(a) User-hours (b) % of thirsty-hours (c) User base

Figure 2: Aggressive blocking

We are interested in calculating $Pr(X_m > X_h)$ —the probability that a corrupt user has more blocked bridges than an honest user (i.e., the likelihood that a corrupt user receives more punishment than an honest user); ideally, this probability should be maximized.

$$Pr(X_m > X_h) = \sum_{x=p\cdot l}^{l} Pr(X_m = x) \sum_{y=0}^{x-1} Pr(X_h = y) \quad (3)$$

Figure 1b depicts $Pr(X_m > X_h)$ with $l = 10$ and $f = 5\%$. While $Pr(X_m > X_h)$ is maximal when g is small, we choose a fairly large value $g = 40$ to make sure bridges are not underutilized.

Credit(t)*—the credit assignment function.* Recall that T_0 and T_1 represent the expected lower and upper bounds of a bridge's life time, respectively. We let T_{lf} denote the expected life time of a bridge, $T_0 \leq T_{lf} \leq T_1$, and s denote the speed of recruiting new bridges. To maintain the overall bridge resource, we should have:

$$T_{lf} \cdot g \cdot s \cdot time = N \cdot k \cdot time \quad (4)$$

From this, we get:

$$T_0 = \frac{N \cdot k}{g \cdot s_{max}}, \quad T_1 = \frac{N \cdot k}{g \cdot s_{min}} \quad (5)$$

where s_{max} and s_{min} denote the maximum and minimum rate of recruiting new bridges, respectively. (From May 2011 to May 2012, the Tor project recruited about 400 new bridges [1].) In our evaluation, we set $s_{max} = 1$ bridge/day and $s_{min} = 0.2$ bridge/day, which implies that $70 \sim 360$ bridges need to be recruited per year. With $N = 1000$, we get $T_0 = 75$ days and $T_1 = 375$ days according to (5). Note that with a larger number of users, the overall bridge consumption will become higher and the system needs to recruit more bridges. However, T_0 and T_1 we have calculated are the worst-case expectations; as will be shown in the simulation, the lifetime of bridges is actually much longer than the worst-case T_0 and T_1, and hence the pressure of recruiting new bridges is smaller in practice.

ϕ^-*—the price for getting a new bridge.* The credits earned from unblocked bridges should be roughly equal to the credits paid to replace blocked bridges. Therefore, approximately we have:

$$\sum_{x=0}^{k} Pr(X_h = x) \cdot x \cdot \phi^- = \sum_{x=0}^{k} Pr(X_h = x) \cdot (k - x) \cdot (T_1 - T_0) \quad (6)$$

From Equation (1) (6), we get $\phi^- = 45$.

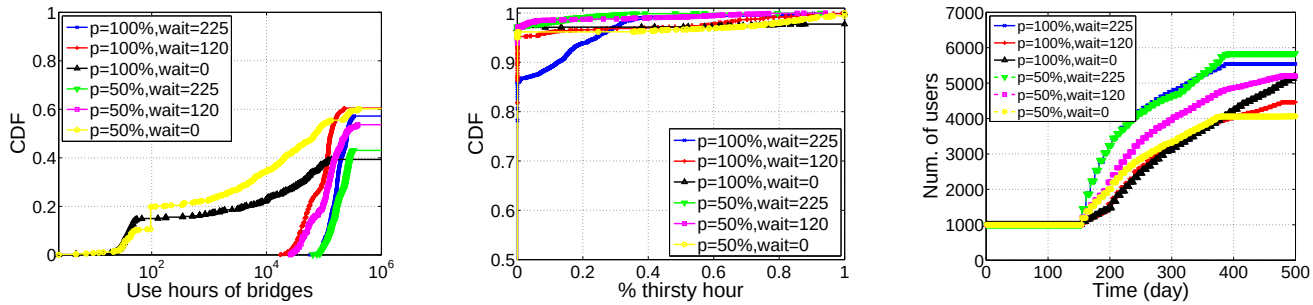
Φ_θ*—the threshold of credits for invitation.* To decide the value of Φ_θ, we assume that a user with at least half of his bridges unblocked can be considered to invite new users. Then, we have:

$$\Phi_\theta = \sum_{x=0}^{\lceil \frac{k}{2} \rceil} Pr(X_h = x | X_h \leq \lceil \frac{k}{2} \rceil) \cdot (k - x) \cdot (T_1 - T_0) \quad (7)$$

From Equation (1) (7), we get $\Phi_\theta = 236$.

User invitation. In our simulation, we set the rate of recruiting new bridges as $s = 1$ bridge/day; we reserve 50% of bridge resource for potential replacement of blocked bridges. Every 7 days, the bridge distributor calculates the number of new users to be invited based on the current bridge resource, and distributes the corresponding number of invitation tickets to randomly selected users whose credit balance is higher than Φ_θ.

Probability distribution of malicious users. Now we consider the probability that an invited user is malicious. We suppose each corrupt user always gives his invitation tickets to malicious users or Sybils. For an honest user, if he randomly selects a new user to invite, the probability that the new user is malicious is approximately f. However,



(a) User-hours (b) % of thirsty-hours (c) User base

Figure 3: Conservative blocking ($f = 5\%$, staged distribution)

in practice, a user is inclined to first invite the friends he trusts most; as receiving more and more invitation tickets, the user will start to invite less trusted friends. To model this, we assume each user ranks his friends based on trustworthiness: each friend is assigned an *index* ranging from 0 to 1 according to the trustworthiness (e.g., the most trusted one out of 100 friends has the index 1/100 = 0.01). We consider two specific models to assign probabilities of malicious users. One is *staged distribution*, wherein the friends are divided into two groups (i.e., more trusted and less trusted) and all users within a group have the same probability of being malicious. We assume 80% friends belong to the "more trusted" group and the remaining 20% are in the "less trusted" group. The other is *linear distribution*, for which the probability of being a malicious user is a linear function of the index. We suppose the probability that the most trusted friend is malicious is 1%. For both distributions, the overall ratio of malicious users is f. Figure 1c depicts the probability distributions of these two models.

4.3.2 Evaluation Results

Using the event-based simulator, we measured the *user-hours*, *thirsty-hours*, and *growth of user base* under different blocking strategies.

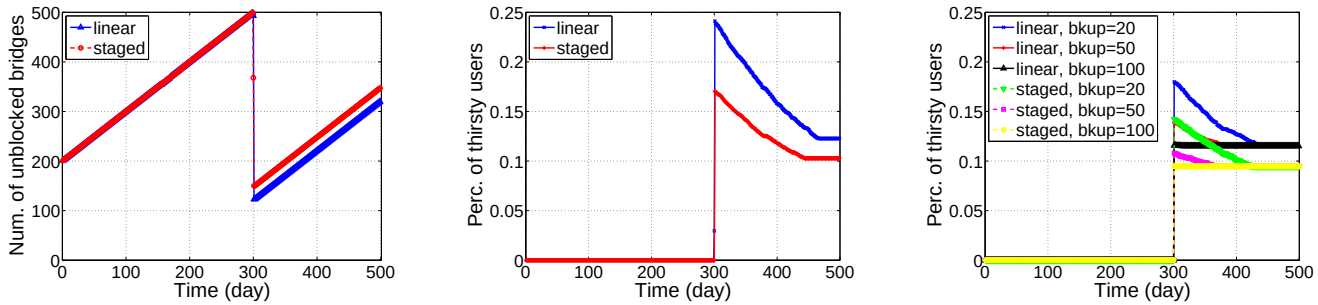
Aggressive blocking. The simulation results for the aggressive blocking are provided in Figure 2. We can see from Figure 2a that when $f = 5\%$, 80% of bridges can produce over 1000 user-hours, and 70% of bridges can serve more than 10 000 user-hours, before being blocked; about 50% of bridges are never blocked. Figure 2b shows that with $f = 5\%$, over 95% of users are never thirsty for bridges; a small fraction (about 2%) of users are unable to get new bridges, because all of their initially assigned bridges get blocked before they earn enough credits to request new bridges. Figure 2c shows that users need some time ($150 \sim 200$ days) to accumulate enough credits to become qualified for inviting friends. After the accumulation phase, the user base starts to steadily grow almost linearly with the number of newly recruited bridges. We also see that rBridge performs relatively better with the staged distribution of malicious users than with the linear distribution; this is because for the staged distribution most invited users belong to the "more trusted" group, for which the probability of being a malicious user is lower than that for the linear distribution.

In addition, we evaluate rBridge with a much higher f (using the same system configuration). We can see that rBridge can easily tolerate 10% malicious users; even when $f = 30\%$, the performance of rBridge is still acceptable; for $f \geq 50\%$, rBridge fails to provide reasonable protection for bridges.

Conservative blocking. There are two factors related to the conservative blocking: the probability of blocking a bridge (p), and the waiting time to block a bridge ($wait$). Since the time to earn credits from a bridge is upper-bounded by T_1, we assume a corrupt user always blocks his bridges by time T_1 (i.e., $wait \leq T_1$). In the simulation, we consider the following cases for the waiting time: 0 day (i.e., aggressive blocking), 120 days (by which the earned credits are sufficient to get a new bridge), and 225 days (i.e., the middle point between T_0 and T_1).

We can see from Figure 3 that compared with the aggressive blocking, the conservative blocking causes less damage to the user-hours and the growth of user base; this is because the bridges under the conservative blocking can serve a longer time and more users can accumulate enough credits to invite new users. We also notice that when $wait = 225$ days and $p = 100\%$, about 10% of users are thirsty for 15% of their time, which is worse than the aggressive blocking; the reason for this is that after waiting 225 days, malicious users earn enough credits to be considered for inviting new (malicious) users (i.e., $(225 - 75) \times 3 = 450 > 236$), and overall they can block more bridges, which causes more recently joined users to become thirsty.

Event-driven blocking. For event-driven blocking, we let



(a) Unblocked bridges (b) Thirsty users (c) Thirsty users with backup bridges

Figure 4: Event-driven blocking ($f = 5\%$)

all of the corrupt users are synchronized to block all their bridges simultaneously on the 300-th day. Figures 4a and 4b show that right after the massive blocking, the number of available bridges drops from 500 to 150, and the percentage of thirsty users rises to 25%. We note that the damage of the event-driven blocking can be effectively mitigated by keeping a small number of backup bridges (that are never seen by any user). We can see from Figure 4c that with 50 backup bridges (about 10% of deployed bridges), the number of thirsty users can be reduced by half; with 100 backup bridges, the number of thirsty users is minimized. We also notice that keeping backup bridges cannot entirely eliminate thirsty users; this is because there are a small fraction (about 10%) of users who join the system not long before the massive blocking and have not accumulated enough credits to request new bridges.

4.3.3 Comparison with Proximax

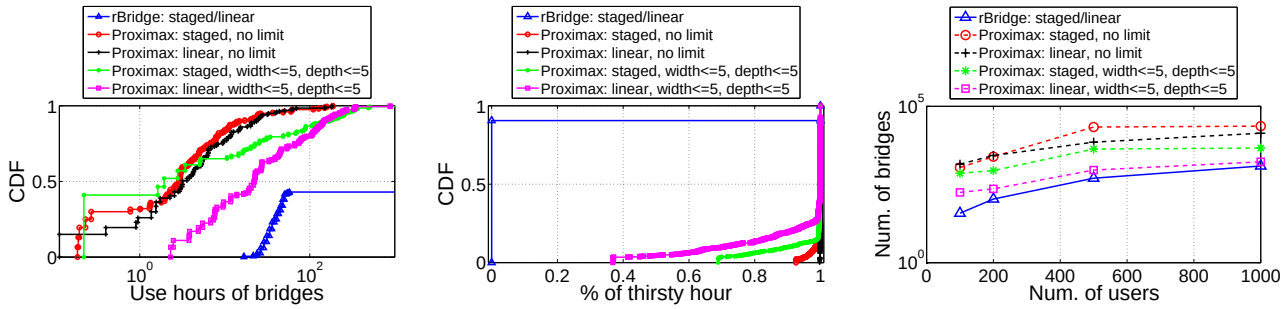
W now compare rBridge with Proximax [15]—the state-of-the-art proxy distribution scheme. Using the same methodology, we developed an event-based simulator for Proximax. Since the authors of Proximax did not provide sufficient details about how to invite new users, we only consider a static set of users for the comparison. We evaluate both rBridge and Proximax under the aggressive blocking using the same system configuration as before; for Proximax, we set the maximum delay of distributing bridges at each hop (i.e., from when a user receives a bridge to when he distributes the bridge to his friends) to 1 day[4].

Figure 5a shows that in Proximax less than 5% bridges are able to produce more than 20 user-hours, and none of the bridges can serve over 126 user-hours. In comparison, in rBridge, over 99% bridges can produce over 20 user-hours, and 57% bridges are not ever blocked and are able to continuously generate user-hours. In addition, in Proximax 99% of users are always thirsty for bridges, and this number is only 10% for rBridge. Since our simulation for the comparison only considers a static user group, we are unable to evaluate Proximax in terms of the growth of the user base over time; instead, we measure how many bridges are required to support different-sized user bases for 30 days, while making sure that each existing user has at least one unblocked bridge at any time. We can see from Figure 5c that Proximax requires a substantially larger number of bridges than rBridge; for instance, to support 200 users, Proximax requires at least 2400 bridges, while rBridge only needs 108 bridges. We note that for all these metrics (user-hours of bridges, thirsty-hours of users, and bridge consumption), the performance of rBridge is at least one order of magnitude higher than that of Proximax.

Discussion. It is possible to improve Proximax by adopting a more *restrictive* distribution strategy, e.g., limiting how many people a bridge recipient can share the bridge with (i.e., the width of the distribution tree) as well as how many hops a bridge can be distributed (i.e., the depth of the distribution tree). Figure 5 also provides the results for limiting the maximum width and depth of the distribution tree to 5. While the restrictive distribution strategy can improve the robustness of Proximax, it is still much worse than rBridge. More importantly, the restrictive approach degrades the openness of the system.

The main reason that rBridge outperforms Proximax is that Proximax calculates "reputation" at the granularity of distribution trees (or called distribution channels) rather than individual users, and the formation of each distribution tree is fixed and thus an honest user would be permanently "infected" if he resides in a tree that contains a corrupt user. Whereas, rBridge allows a user to join a different user group when receiving a new bridge, and keeps track of each individual user's records, based on which the bridge distributor can reward well-behaving users and punish blockers individually. We note that recording each individual user' bridge assignment leads to greater risks of violating users'

[4]In the simulation of Proximax, we found that higher propagation delay leads to higher user-hours; hence, we chose a fairly large value (1 day).



(a) User-hours

(b) Thirsty-hours

(c) Required bridges to support a certain # of users for 30 days

Figure 5: Comparison with Proximax ($f = 5\%$)

privacy; we describe how to perfectly protect users' bridge assignment information in the next section.

Finally, we note that computing reputation merely based on the bridges' uptime is not the only way to design the reputation system. For instance, it is possible to extend rBridge by including the reputations of the "introducees" as a factor to calculate the reputation of the "introducer". However, the increased complexity of the reputation system makes it even harder (if possible) to design a practical privacy-preserving mechanism to perfectly protect users' bridge assignment information. Therefore, our design philosophy is to make the reputation system as simple as possible, while ensuring its robustness against various blocking strategies.

5 rBridge with Privacy Preservation

In the basic rBridge scheme, D knows all the bridge assignment details of each user. A malicious entity equipped with such information can degrade the users' anonymity in the anonymous communication. For instance, the current Tor network ensures that each user can conceal his identity as long as the entry relay (i.e., the bridge) is not compromised. Whereas, with the bridge assignment information, an adversary is able to narrow the anonymity set of the user down to a small group of people who are given this bridge, even though the bridge is not compromised. Unfortunately, this issue is overlooked by all the prior bridge distribution schemes. Our goal is to preserve the bridge information of each user.

5.1 Challenges and Requirements

In rBridge, a user (U) can get a bridge only if he can authenticate himself to D by presenting a valid credential (recall that in the basic scheme, a credential includes the following information $\mathtt{U}\|\Phi\|\{B_i, \tau_i, \phi_i\}_{i=1}^{k}$). Firstly, in order to unlink the user from his assigned bridges, we should **conceal the user's identity** by replacing U's real identity with a pseudonym on his credential and letting him build a Tor circuit to communicate with D to hide his IP address.

However, the above measures are not sufficient. Suppose U has received 10 bridges from D (who knows the bridges but not U's identity), and 2 of them are malicious and know U's identity due to direct contact and collude with D; then it is highly likely that D can link U to all of his bridges, since very few users happen to know both of the malicious bridges. A natural solution to this is using *Oblivious Transfer* (OT) for privacy-preserving bridge retrieval — preventing D from learning which bridge is retrieved when U requests a new bridge (called a *transaction*). However, since a user is very likely to request a new bridge right after one of his bridges gets blocked, D can infer the blocked bridge by checking which bridge was recently blocked. As a result, D can learn all of U's (blocked) bridges as long as D can link different transactions of U. Therefore, **unlinkability of transactions** is required to avoid such information leaks.

Thirdly, since we intend to hide the bridge assignment from D, the bridge related information in U's credential, such as $\{B_i, \tau_i, \phi_i\}$, should be written and updated by U, rather than by D. This raises the risk that a malicious user could put incorrect information on his credential, e.g., by changing the credits ϕ_i or replacing B_i with another bridge B_i' so that he can block B_i without being punished. Therefore, we also need to protect the **integrity of credentials**.

Although researchers have proposed several designs for anonymous authentication/reputation systems [7, 8, 21, 22], none of them is able to ensure integrity of credentials. In this work, we propose a novel privacy-preserving user reputation scheme that is specially designed for bridge distribution and satisfies all the three aforementioned requirements. Our design integrates OT with several other cryptographic primitives (such as commitments and zero-knowledge proofs) to both preserve users' privacy and prevent misbehavior of corrupt users. We start with introducing

the cryptographic primitives used in our construction.

5.2 Cryptographic Building Blocks

5.2.1 1-out-of-m Oblivious Transfer

1-out-of-m Oblivious Transfer (denoted by $\binom{m}{1}$-OT) is a secure two-party computation protocol, where the sender has m secrets and the chooser can get 1 and only 1 secret from the sender without revealing any information about which secret is selected. We use the two-round $\binom{m}{1}$-OT scheme proposed in [17] to construct rBridge due to its simplicity of implementation.

To make it hard for corrupt users to collaboratively enumerate bridges, we let D (i.e., the sender) randomly shuffle the list of available bridges (i.e., the secrets) before running the OT protocol with each user (i.e., the chooser), so that the user will randomly "choose" which bridge to get. Because of the randomized OT, it is possible that a user gets a bridge that is already assigned to him even though the chance is very small. We show how to deal with duplicate bridges in Appendix A.

5.2.2 Commitment

A commitment scheme enables a party to create the digital equivalent of an envelope for a secret. It supports two important properties: *hiding* protects the secrecy of the committed message, and *binding* ensures it can only be opened to the committed message. Pedersen commitments [18] are information-theoretically hiding and binding under the discrete logarithm assumption. We use $(C, O) = \mathsf{CMT}(M)$ to denote a Pedersen commitment to a secret M, where C is the commitment and O is the opening to the commitment.

In rBridge, we use commitments to conceal the content on a user's credential. For instance, to hide the amount of credits Φ, U can compute a commitment of Φ, i.e., $(C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi)$, and put C_Φ in his credential. To prevent U from manipulating his credential (e.g., increasing Φ), we let D sign C_Φ using his private key $SK_\mathtt{D}$, i.e., $\sigma_\Phi = \mathtt{Sign}(SK_\mathtt{D}, C_\Phi)$ and tag the signature σ_Φ to the credential, and U needs to prove to D that both the commitment and the signature are valid. To prevent D from linking U's transactions based on the values of commitments and signatures, we need another cryptographic primitive—zero-knowledge proof.

5.2.3 Zero-Knowledge Proof

In a zero-knowledge proof scheme, a prover convinces a verifier that some statement is true while the verifier learns nothing except the validity of the statement. A zero-knowledge proof can be converted into a corresponding non-interactive version in the random oracle model via Fiat-Shamir heuristic [12]. We follow the notation introduced by Camenisch and Stadler [9], e.g., $\mathsf{NIPK}\{(x) : y = g^x\}$ denotes a "*non-interactive zero-knowledge proof of knowledge of integer x, s.t., $y = g^x$, and g is public and x is secret.*".

Our main use of zero-knowledge proofs is to prove knowledge of commitments and possession of signatures. For instance, U can construct the following proof:

$$\pi = \mathsf{NIPK}\left\{\begin{array}{l}(\Phi, C_\Phi, O_\Phi, \sigma_\Phi) : \\ \quad \Phi > \Phi_\theta \wedge \\ \quad (C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi) \wedge \\ \quad \mathsf{Verify}(PK_\mathtt{D}, \sigma_\Phi, C_\Phi) = Accept\end{array}\right\}$$

to prove that his credit balance Φ is above the threshold Φ_θ and is not tampered (i.e., correctly signed), without revealing the credit balance Φ, the commitment C_Φ, the opening O_Φ, or the signature σ_Φ (where $PK_\mathtt{D}$ is the public key of D, and Verify is the function to verify the signature σ_Φ). In our construction, we employ the k-TAA blind signature scheme proposed by Au et al. [8] because of its compatibility with zero-knowledge proofs.

5.3 Scheme

We now present the rBridge scheme with privacy preservation. We refer interested readers to Appendix B for detailed cryptographic constructions of the scheme.

5.3.1 Anonymous Credential

A key concept in rBridge is the *anonymous credential*, which anonymously records the user's bridges and reputation and allows the user to anonymously authenticate himself to D. Each part of the credential is signed by D individually, so that they can be verified and updated separately. In particular, an anonymous credential contains the following information:

$$x \| \{\Phi, C_\Phi, O_\Phi, \sigma_\Phi\} \| \{\omega, C_\omega, O_\omega, \sigma_\omega\} \| \{B_i, \tau_i, \phi_i, C_i, O_i, \sigma_i\}_{i=1}^{k}$$

where x is the secret key that is selected by U when registering the credential, $\sigma_\clubsuit$ is the signature on $C_\clubsuit$, and ω denotes the latest time when the user requests an invitation ticket (ω is used to prevent corrupt users from repeatedly requesting tickets; we discuss this later). We note that all the information in the credential must be kept secret from D.

To prevent a corrupt user from replacing some part of his credential with that of others' credentials (e.g., a higher Φ from a well-behaving colluding user), we use x to link different parts of the credential by including x in each of their commitments. To be specific, $(C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi, x)$, $(C_\omega, O_\omega) = \mathsf{CMT}(\omega, x)$, and $(C_i, O_i) =$

$\mathsf{CMT}(B_i, \tau_i, \phi_i, x)$. To get a credential, a new user runs the following registration protocol.

5.3.2 Registration

A new user U first presents an invitation ticket to D. An invitation ticket is an one-time token formed as $tk = \{r^*, \mathsf{HMAC}_{scrt_{\mathtt{D}}}(r^*)\}$, where r^* is a random number and $scrt_{\mathtt{D}}$ is a secret only known to D; the ticket is verifiable to D, and cannot be forged by anyone else. Then U runs $\binom{m}{1}$-OT with D to get k initial bridges $B_1, \cdots, B_k$[5]. After that, U randomly picks a secret key x and performs the following computations: set $\Phi = 0$ and compute $(C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi, x)$; set $\omega = T_{cur}$ (recall that T_{cur} denotes the current time) and compute $(C_\omega, O_\omega) = \mathsf{CMT}(\omega, x)$; for each $i \in [1, k]$, set $\tau_i = T_{cur}$, $\phi_i = 0$, and compute $(C_i, O_i) = \mathsf{CMT}(B_i, \tau_i, \phi_i, x)$. To prevent multiple colluding users from using the same x to share some parts of their credentials, U is required to provide an *indicator* of his selected x, formed as $\kappa_x = \mathsf{OWF}(x)$, to prove that x has not been used by other users while hiding x from D. (wherein $\mathsf{OWF}(\cdot)$ is simply a discrete-log based one-way function.)

Note that since D does not know the bridges received by U (i.e., $\{B_i\}_{i=1}^k$), U could try to put other bridges on his credential, i.e., replacing B_i with B_i^* in $\mathsf{CMT}(B_i, \tau_i, \phi_i, x)$, so that he can block all of $\{B_i\}_{i=1}^k$ instantly without worrying about potential loss of credits. To prevent this attack, D needs to verify that the bridges to be written in the credential are actually the bridges that U received in OT. To achieve this, we let D (before running OT) generate a pair of one-time public/private keys (denoted by $PK_{\mathtt{D}}^o$, $SK_{\mathtt{D}}^o$), give $PK_{\mathtt{D}}^o$ to U, use $SK_{\mathtt{D}}^o$ to sign each available bridge B_j, and tag the signature σ_j^o to B_j. After OT, U gets $\{B_i \| \sigma_i^o\}_{i=1}^k$, and he needs to prove the possession of a valid signature σ_i^o on B_i. Intuitively, U is no longer able to replace B_i with any other bridge (B_i^*) that is not one of the k received bridges, because he does not have a signature on B_i^*. In addition, we let U provide D a random nonce non_j for each available bridge B_j; non_j is included in the computation of σ_j^o to prevent D from later finding out which bridges are retrieved by U using these signatures (refer to Appendix B for details).

To get the credential, U constructs the following proof:

$$\pi_1 = \mathsf{NIPK}\left\{ \begin{array}{l} (x, \Phi, O_\Phi, \omega, O_\omega, \{B_i, \tau_i, \phi_i, \sigma_i^o, O_i\}_{i=1}^k) : \\ \quad \bigwedge_{i=1}^k ((C_i, O_i) = \mathsf{CMT}(B_i, \tau_i, \phi_i, x) \wedge \\ \qquad \mathsf{Verify}(PK_{\mathtt{D}}^o, \sigma_i^o, B_i) = Accept \wedge \\ \qquad \tau_i = T_{cur} \wedge \phi_i = 0) \wedge \\ \quad (C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi, x) \wedge \\ \quad \kappa_x = \mathsf{OWF}(x) \wedge \\ \quad \Phi = 0 \wedge \\ \quad (C_\omega, O_\omega) = \mathsf{CMT}(\omega, x) \wedge \\ \quad \omega = T_{cur} \end{array} \right\}$$

and sends $\kappa_x \| C_\Phi \| C_\omega \| \{C_i\}_{i=1}^k \| \pi_1$ to D.

After verifying the validity of π_1 and the freshness of κ_x, D signs C_Φ, C_ω, and C_i ($1 \le i \le k$), respectively, and sends the signatures $\sigma_\Phi \| \sigma_\omega \| \{\sigma_i\}_{i=1}^k$ to U. Finally, D adds κ_x to the list of indicators of used secret keys (denoted by $elist_x$) to prevent other users from re-using it.

Note that in the privacy-preserving scheme, it is infeasible for D to count the number of users who ever connected to each bridge; instead, we let each bridge notify D once the number of users that ever connect to it exceeds the threshold g, and then D will exclude the bridge from the list when running OT. (The bridge could let each new user register himself upon the first connection, e.g., by setting up a password [4], in order to count the ever connected users.)

5.3.3 Updating Credit Balance

U can update his credit balance Φ with recently earned credits from time to time. Suppose the credits are from B_u. The new credit balance is calculated as $\tilde{\Phi} = \Phi + \tilde{\phi}_u - \phi_u$, where $\tilde{\phi}_u = \mathsf{Credit}(T_{cur} - \tau_u)$. U needs to show that B_u is not blocked. To do so, we let D compute $b_j = \mathsf{OWF}(\bar{B}_j)$ for each of the blocked bridges $\{\bar{B}_j\}_{j=1}^{\bar{m}}$ (where $\bar{m}$ is the total number of blocked bridges) and publish $\{b_j\}_{j=1}^{\bar{m}}$; U needs to prove that $\mathsf{OWF}(B_u)$ is not equal to any of $\{b_j\}_{j=1}^{\bar{m}}$.

D must record expired credentials to prevent re-use of old credentials (e.g., those with more credits). For this, we let U provide an indicator of Φ to show that Φ is up-to-date. Note that we cannot use $\kappa_\Phi = \mathsf{OWF}(\sigma_\Phi)$ as the indicator, since D could try all of the signatures he has generated to find a match between κ_Φ and σ_Φ to link U's transactions. To address this, we craft a special indicator function $\kappa_\Phi = \mathsf{Indic}(\sigma_\Phi)$ based on the feature of k-TAA blind signature [8] (Essentially, this indicator function first converts σ_Φ into another form σ'_Φ using a random factor and then applies an one-way function to σ'_Φ to get κ_Φ. See Appendix B for more details.)

In particular, U constructs the following proof:

[5] U needs to run k rounds of $\binom{m}{1}$-OT to get k bridges. While it is possible to invoke one round of $\binom{m}{k}$-OT, but its complexity is higher when k is much smaller than m (typically $k = 3$).

$$\pi_2 = \mathsf{NIPK}\left\{\begin{array}{l}(x, \Phi, C_\Phi, O_\Phi, \sigma_\Phi, B_u, \tau_u, \phi_u, C_u, O_u, \sigma_u,\\ \tilde{\phi}_u, \tilde{O}_u, \tilde{\Phi}, \tilde{O}_\Phi) :\\ \quad \bigwedge_{j=1}^{\bar{m}} (b_j \neq \mathsf{OWF}(B_u)) \wedge\\ \quad (C_u, O_u) = \mathsf{CMT}(B_u, \tau_u, \phi_u, x) \wedge\\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_u, C_u) = Accept \wedge\\ \quad (C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi, x) \wedge\\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_\Phi, C_\Phi) = Accept \wedge\\ \quad \kappa_\Phi = \mathsf{Indic}(\sigma_\Phi) \wedge\\ \quad \tilde{\phi}_u = \mathsf{Credit}(T_{cur} - \tau_u) \wedge\\ \quad \tilde{\Phi} = \Phi + \tilde{\phi}_u - \phi_u \wedge\\ \quad (\tilde{C}_u, \tilde{O}_u) = \mathsf{CMT}(B_u, \tau_u, \tilde{\phi}_u, x) \wedge\\ \quad (\tilde{C}_\Phi, \tilde{O}_\Phi) = \mathsf{CMT}(\tilde{\Phi}, x) \wedge\end{array}\right\}$$

U builds a Tor circuit (using one of his bridges as the entry relay) to send $\kappa_\Phi \| \tilde{C}_\Phi \| \tilde{C}_u \| \pi_2$ to D.

D verifies π_2 and checks that κ_Φ is not on the list of seen indicators (denoted by $elist_\Phi$); then, D signs $\tilde{C}_\Phi$ and $\tilde{C}_u$, sends the signatures $\tilde{\sigma}_\Phi$ and $\tilde{\sigma}_u$ to U, and adds κ_Φ to $elist_\Phi$. Finally, U updates his credential with $\tilde{\Phi}, \tilde{C}_\Phi, \tilde{O}_\Phi, \tilde{\sigma}_\Phi, \tilde{\phi}_u, \tilde{C}_u, \tilde{O}_u$, and $\tilde{\sigma}_u$.

5.3.4 Getting a New Bridge

To get a new bridge, U first needs to prove that one of his bridges (say B_b) on his credential has been blocked and his credit balance is higher than ϕ^-. Since a user usually requests a new bridge right after one of his bridges got blocked which allows D to figure out what B_b is by checking which bridge was recently blocked, we do not intend to hide B_b from U. We note that revealing B_b will not degrade the anonymity, as long as D is unable to link the transaction of replacing B_b with other transactions of U.

U first sends B_b to D through a Tor circuit. After verifying B_b is blocked, D replies with β_b (i.e., the time when B_b got blocked). The new credit balance of U is $\tilde{\Phi} = \Phi + (\tilde{\phi}_b - \phi_b) - \phi^-$, where $\tilde{\phi}_b = \mathsf{Credit}(\beta_b - \tau_b)$, by considering the credits earned from B_b and the cost for getting a new bridge. U constructs the following proof:

$$\pi_3 = \mathsf{NIPK}\left\{\begin{array}{l}(x, \Phi, C_\Phi, O_\Phi, \sigma_\Phi, \tau_b, \phi_b, C_b, O_b, \sigma_b,\\ \tilde{\phi}_b, \tilde{\Phi}, \tilde{O}_\Phi) :\\ \quad (C_b, O_b) = \mathsf{CMT}(B_b, \tau_b, \phi_b, x) \wedge\\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_b, C_b) = Accept \wedge\\ \quad \kappa_b = \mathsf{Indic}(\sigma_b) \wedge\\ \quad (C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi, x) \wedge\\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_\Phi, C_\Phi) = Accept \wedge\\ \quad \kappa_\Phi = \mathsf{Indic}(\sigma_\Phi) \wedge\\ \quad \tilde{\phi}_b = \mathsf{Credit}(\beta_b - \tau_b) \wedge\\ \quad \tilde{\Phi} = \Phi + \tilde{\phi}_b - \phi_b - \phi^- \wedge\\ \quad \tilde{\Phi} > 0 \wedge\\ \quad (\tilde{C}_\Phi, \tilde{O}_\Phi) = \mathsf{CMT}(\tilde{\Phi}, x)\end{array}\right\}$$

and sends $\kappa_\Phi \| \kappa_b \| \tilde{C}_\Phi \| \pi_3$ to D. Note that we use κ_b to make sure each blocked bridge can be used only once to request a new bridge.

After verifying $\kappa_\Phi \notin elist_\Phi$, $\kappa_b \notin elist_{B_b}$, and π_3 (where $elist_{B_b}$ denotes the list of used indicators of B_b), D adds κ_Φ to $elist_\Phi$ and κ_b to $elist_{B_b}$. Similar to the registration, U runs $\binom{m}{1}$-OT with D to obtain a new bridge $\tilde{B}_b$ with a tagged signature $\tilde{\sigma}^o_b$. Then, U sets $\tilde{\tau}_b = T_{cur}$ and $\tilde{\phi}_b = 0$, computes $(\tilde{C}_b, \tilde{O}_b) = \mathsf{CMT}(\tilde{B}_b, \tilde{\tau}_b, \tilde{\phi}_b, x)$, constructs the following proof:

$$\pi_4 = \mathsf{NIPK}\left\{\begin{array}{l}(x, \tilde{\Phi}, \tilde{O}_\Phi, \tilde{B}_b, \tilde{\tau}_b, \tilde{\phi}_b, \tilde{\sigma}^o_b, \tilde{O}_b) :\\ \quad (\tilde{C}_\Phi, \tilde{O}_\Phi) = \mathsf{CMT}(\tilde{\Phi}, x) \wedge\\ \quad \tilde{\tau}_b = T_{cur} \wedge \tilde{\phi}_b = 0 \wedge\\ \quad (\tilde{C}_b, \tilde{O}_b) = \mathsf{CMT}(\tilde{B}_b, \tilde{\tau}_b, \tilde{\phi}_b, x) \wedge\\ \quad \mathsf{Verify}(PK^o_{\mathtt{D}}, \tilde{\sigma}^o_b, \tilde{B}_b) = Accept\end{array}\right\}$$

and sends $\tilde{C}_b \| \pi_4$ to D.

D verifies π_4, signs $\tilde{C}_\Phi$ and $\tilde{C}_b$, and sends $\tilde{\sigma}_\Phi$ and $\tilde{\sigma}_b$ to U. Finally, U updates his credential with $\tilde{\Phi}, \tilde{C}_\Phi, \tilde{O}_\Phi, \tilde{\sigma}_\Phi, \tilde{B}_b, \tilde{\tau}_b, \tilde{\phi}_b, \tilde{C}_b, \tilde{O}_b$, and $\tilde{\sigma}_b$.

5.3.5 Inviting New Users

U can request D for an invitation ticket as long as his credit balance is higher than Φ_θ. D grants the request with certain probability. Recall that ω in the credential represents the latest time when U requested an invitation ticket. To prevent a corrupt user from repeatedly requesting invitation tickets to increase his chance of getting one, we let each requesting user prove that his last time requesting a ticket is at least ω_θ days ago, i.e., $T_{cur} - \omega > \omega_\theta$. In particular, to request a ticket, U constructs the proof:

$$\pi_5 = \mathsf{NIPK}\left\{\begin{array}{l}(x, \Phi, C_\Phi, O_\Phi, \sigma_\Phi, \omega, C_\omega, O_\omega, \sigma_\omega, \tilde{\omega},\\ \tilde{O}_\omega, \tilde{O}_\Phi) :\\ \quad (C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi, x) \wedge\\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_\Phi, C_\Phi) = Accept \wedge\\ \quad \kappa_\Phi = \mathsf{Indic}(\sigma_\Phi) \wedge\\ \quad (C_\omega, O_\omega) = \mathsf{CMT}(\omega, x) \wedge\\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_\omega, C_\omega) = Accept \wedge\\ \quad \kappa_\omega = \mathsf{Indic}(\sigma_\omega) \wedge\\ \quad \Phi > \Phi_\theta \wedge\\ \quad T_{cur} - \omega > \omega_\theta \wedge\\ \quad \tilde{\omega} = T_{cur} \wedge\\ \quad (\tilde{C}_\omega, \tilde{O}_\omega) = \mathsf{CMT}(\tilde{\omega}, x) \wedge\\ \quad (\tilde{C}_\Phi, \tilde{O}_\Phi) = \mathsf{CMT}(\Phi, x)\end{array}\right\}$$

and sends $\kappa_\Phi \| \kappa_\omega \| \tilde{C}_\Phi \| \tilde{C}_\omega \| \pi_5$ to D through a Tor circuit.

After verifying $\kappa_\Phi \notin elist_\Phi$, $\kappa_\omega \notin elist_\omega$ and π_5, D signs $\tilde{C}_\Phi$ and $\tilde{C}_\omega$, sends $\tilde{\sigma}_\Phi$ and $\tilde{\sigma}_\omega$ to U, and adds κ_Φ to $elist_\Phi$ and κ_ω to $elist_\omega$. Then, D flips a coin to decide whether to grant the request: if yes, D generates a ticket for U; otherwise, U needs to wait at least ω_θ days to try again.

Table 1: Performance (averaged over 100 runs)

Operation	Comp. (s)		Comm. (KB)
	U	D	
Registration	5.15	17.44	388.1
Updating credit balance	0.51	0.47	34.7
Getting a new bridge	5.35	17.62	340.1
Inviting new users	0.27	0.16	2.0

5.4 Performance Evaluation

We implemented rBridge using Paring-Based Crytography (PBC) Library[6] and GNU Multiple Precision library[7] in C++, and used OpenSSL for hash related routines. The credential system, built upon k-TAA signature, was implemented with Type-A curves, which is defined in the PBC Library in the form of $E : y^2 = x^3 + x$ over the field $\mathcal{F}_q$ for some prime q. Bilinear groups G_1 and G_2, both formed by points over $E(\mathcal{F}_q)$, are of order p for some prime p, such that p is a factor of $q + 1$. Using the default setting, p and q are 160-bit and 512-bit in length respectively. We implemented the $\binom{m}{1}$-OT protocol in [17] using G_1 as the underlying group. We consider there are 1000 available bridges and 100 blocked bridges, and set the size of each bridge descriptor the same as that of the current bridge descriptor (208 bits including 32-bit IP, 16-bit port, and 160-bit fingerprint).

We measured the computational times for U and D on a Dell Precision T3500 workstation with a quad-core 2.67 GHz Intel Xeon W3550 CPU and 12 GB RAM, running Ubuntu 10.04. Table 1 shows that it takes only less than 0.5 seconds for U and D to update credit balance or process an invitation ticket request. It takes longer to perform the initial registration or request a new bridge, which are about 5 seconds and 17 seconds for U and D, respectively. The majority of computational times are spent on the retrieval of new bridges. We note that these operations are quite infrequent; each user only needs to register once, and according to our simulation results, the averaged time interval to request new bridges is 133 days. Hence, we believe these occasional computations can be handled by both U and D with fairly strong computational power. We also measured the communication costs. The operations of registration and getting a new bridge incur 388 KB and 340 KB communication overheads respectively, and the data transmission for any other operation is less than 35 KB. In comparison, with 2000 Tor relays, each Tor client needs to download a 120 KB "network status document" every 3 hours, and 1.25 MB of relay "descriptors" spread over 18 hours [16].

[6] http://crypto.stanford.edu/pbc/

[7] http://gmplib.org/

6 Security Analysis

In this section, we discuss the potential attacks that are not covered by the evaluation (Section 4.3).

6.1 Curious Bridge Distributor in Privacy Preservation

One of the major security requirements in the privacy-preserving scheme is to ensure different transactions of a particular user are unlinkable. A typical way to link a user's transactions is based on the credential content. For example, suppose in the x-th transaction U received a new bridge B_X and updated his credential with some signed information of B_X (e.g., B_X, τ_X, and ϕ_X); later, when B_X gets blocked and U requests a new bridge to replace it in the y-th transaction, U needs to use the signed information to prove that, e.g., B_X was assigned to him and a certain amount of credits should be earned from B_X; however, such information can be utilized by D to link the two transactions. (Similar attacks are applicable in the cases when U updates his credit balance or requests invitation tickets.) To ensure the unlinkability of transactions, rBridge uses commitments and zero-knowledge proofs to conceal the content of the credential, so that D only knows the validity of the credential but learns nothing about the content of the credential.

Although B_X is revealed to D in the y-th transaction (i.e., after B_X gets blocked), D is unable to link B_X to U due to the use of a Tor circuit; more importantly, since B_X is perfectly hidden in any other transactions, D cannot use B_X to link any two transactions of U.

Similarly, D can learn U's identity (i.e., his IP address, but nothing else) in the registration, since at that moment U has no bridge to build a Tor circuit to hide his IP address (if he does not use any other circumvention tool or ask an existing user to perform the registration on his behalf). Nevertheless, D is unable to learn which bridges are retrieved because of U in OT; moreover, since U's IP address will be hidden in all the later transactions of U, D cannot use U's IP address to link his transactions.

6.2 Users' Misbehaviors in Privacy Preservation

In the privacy-preserving scheme, a corrupt user could try to manipulate the information on his credential, e.g., increasing the credit balance. rBridge uses zero-knowledge proofs with the help of blind signatures to verify the correctness of the credential without revealing any information in the credential.

In addition, since D does not know what bridge is retrieved by U in OT, U could try to replace the received bridge with another one when updating his credential so that he can block the assigned bridge instantly without worrying about

potential loss of credits. To address this, we let D employ one-time signatures to make sure the bridge written in the credential is indeed the bridge that the user received in OT.

Furthermore, malicious users could try to re-use old credentials that have more credits or use a blocked bridge to request more than one new bridges. To prevent these, we let D record all the used credentials as well as the claimed blocked bridges, and ask U to provide indicators to prove that the presented credential or the claimed blocked bridge has not been used before.

6.3 Sybil Attacks

The adversary could launch Sybil attacks by creating a large number of Sybils in the population of potential bridge users, so that the ratio f of compromised entities in the potential users can be dramatically increased. However, we note that deploying a large number of Sybils does not necessarily lead to increase in corrupt users in the system. For honest users, their invitation tickets are given to people they know. While corrupt users give all their received tickets to colluding entities, the number of malicious entities they can invite is bottlenecked by the number of invitation tickets they have received, rather than by the number of malicious entities the adversary can create in the population of potential users.

Alternatively, the adversary could try to deploy Sybils directly in the system, which requires the adversary to provide each Sybil with a valid credential; however, this is infeasible without knowing the bridge distributor's private key. We also note that it is infeasible to let corrupt users share their credentials with the Sybils either, because the bridge distributor recycles used credentials and the total number of bridges that can be learnt by the adversary does not increase.

6.4 Blocking the Bridge Distributor

We suppose the IP address of the bridge distributor is publicly known. Hence, the censor could simply block the bridge distributor to either prevent new users from joining the system or stop existing users from receiving new bridges. For an existing user who has at least one unblocked bridge, he can use the bridge to build a Tor circuit to access the bridge distributor. For a user without any usable bridge (e.g., a new user), he can use a high-latency but more robust circumvention tool (e.g., Email based circumvention [5]) to communicate with the bridge distributor to get the initial bridges or a replacement bridge. Besides, a new user could ask his inviter (i.e., the existing user who gave him the invitation ticket) to perform the initial bootstrapping on his behalf to get the initial bridges.

6.5 Well Behaving of Corrupt Users

In order to increase the number of corrupt users in the system, the adversary could let the corrupt users behave legitimately (i.e., keeping their bridges alive) for a certain period of time to accumulate credits in order to receive invitation tickets. However, we note that since the invitation tickets are randomly distributed to qualified users, corrupt users may not necessarily receive invitation tickets even if they have saved up sufficient credits. In addition, keeping bridges alive also allows honest users to accumulate enough credits to become qualified to receive invitation tickets; therefore, overall, the chance of receiving invitation tickets by corrupt users is no better than that of honest users. Our simulation results in Section 4.3 (where the corrupt users do not block bridges until the 225-th day) show that this attack strategy cannot help the adversary increase of ratio of corrupt users in the system. In addition, rBridge does not allow users to transfer credits to others, and hence it is infeasible to deploy a few well-behaving corrupt users to help other corrupt users by sharing their credits.

7 Conclusion

We proposed rBridge, a user reputation system for Tor bridge distribution. rBridge addresses two key challenges to bridge distribution: protecting bridges from being blocked by corrupt users, and preserving bridge assignment information of each user. rBridge makes use of users' reputation to punish blockers and limit them from repeatedly blocking bridges, and adopts an introduction-based mechanism to invite new users while resisting Sybil attacks. Our simulation results show that rBridge is able to provide much stronger protection for bridges than any existing scheme. In addition, we addressed privacy preservation in rBridge by concealing users' bridge assignment information. Such information can be explored to degrade users' anonymity in anonymous communication. We designed a novel privacy-preserving reputation system for bridge distribution using several cryptographic primitives. To our best knowledge, rBridge is the first scheme that is able to perfectly preserve users' privacy in bridge distribution. We implemented a prototype of rBridge, and the experiments showed that rBridge has reasonable performance.

8 Acknowledgement

We thank anonymous reviewers for invaluable comments on the paper. Qiyan Wang was supported in part by NSF CNS 09-53655 and Zi Lin was supported in part by NSF CNS 09-17154.

References

[1] https://metrics.torproject.org/network.html#networksize.

[2] Ten ways to discover Tor bridges. https://blog.torproject.org/blog/research-problems-ten-ways-discover-tor-bridges.

[3] https://www.torproject.org/projects/obfsproxy.html.en.

[4] Proposal 190: Password-based Bridge Client Authorization. https://lists.torproject.org/pipermail/tor-dev/2011-November/003042.html.

[5] Feed Over Email (F.O.E). http://code.google.com/p/foe-project/.

[6] Research problem: Five ways to test bridge reachability, Dec, 1, 2011. https://blog.torproject.org/blog/research-problem-five-ways-test-bridge-reachability.

[7] M. H. Au, A. Kapadia, and W. Susilo. Blacr: Ttp-free blacklistable anonymous credentials with reputation. In *NDSS'12*, 2012.

[8] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-taa. *SCN, Lecture Notes in Computer Science*, 4116:111–125, 2006.

[9] J. Camenisch and M. Stadler. Proof system for general statements about discrete logarithms. In *Technical Report TR 260, Institute for Theoretical Computer Science*, 1997.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, August 2004.

[11] N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting web censorship with untrusted messenger discovery. In *Privacy Enhancing Technologies (PETS)*, 2003.

[12] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[13] J. Jacob. How internet censorship works in china, Feb. 17, 2011. http://www.ibtimes.com/articles/113590/20110217/china-internet-censorship -great-firewall-us-hillary-clinton-communist.htm .

[14] M. Mahdian. Fighting censorship with algorithms. In *Proceedings of FUN'10*, 2010.

[15] D. McCoy, J. A. Morales, and K. Levchenko. Proximax: A measurement based system for proxies dissemination. In *FC'11*, Feb 2011.

[16] J. McLachlan, A. Tran, N. Hopper, and Y. Kim. Scalable onion routing with torsk. In *ACM CCS'09*, 2009.

[17] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA'01*, 2001.

[18] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology*, 1992.

[19] R. Smits, D. Jain, S. Pidcock, I. Goldberg, and U. Hengartner. Bridgespa: Improving tor bridges with single packet authorization. In *WPES'11*, 2011.

[20] Y. Sovran, A. Libonati, and J. Li. Pass it on: Social networks stymie censors. In *IPTPS'08*, 2008.

[21] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable anonymous credentials: Blocking misbehaving users without ttps. In *CCS'07*, 2007.

[22] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Perea: Towards practical ttp-free revocation in anonymous authentication. In *CCS'08*, 2008.

[23] J. Zittrain and B. Edelman. Internet Filtering in China. *IEEE Internet Computing*, 7(2):70–77, 2003. http://csdl.computer.org/comp/mags/ic/2003/02/w2070abs.htm.

A Dealing with Duplicate Bridges

Suppose U receives a duplicate bridge B_d in the randomized $\binom{m}{1}$-OT, which is identical to one of his existing bridges B_e. We allow U to get a new bridge (by running $\binom{m}{1}$-OT again) to replace B_d as long as he can prove that he has valid signatures for both B_d and B_e.

We note that, however, a sophisticated D may try to infer U's bridges by constructing the list of available bridges used in OT with a single bridge (say B^*), and see if later U requests replacement of a duplicate bridge; if yes, D can be sure that B^* is one of U's existing bridges. To prevent this, we let D compute $(C_j, O_j) = \mathsf{CMT}(B_j)$ for each available bridge B_j, and publish all the C_j's; before running $\binom{m}{1}$-OT, U randomly picks C_p, C_q, $p \neq q$, and asks D to prove that B_p and B_q are different. D constructs the following proof:

$$\pi_6 = \mathsf{NIPK}\left\{ \begin{array}{l} (B_p, O_p, B_q, O_q) : \\ \quad (C_p, O_p) = \mathsf{CMT}(B_p) \wedge \\ \quad (C_q, O_q) = \mathsf{CMT}(B_q) \wedge \\ \quad B_p \neq B_q \end{array} \right\}$$

Then, U runs $\binom{m}{1}$-OT to get O_d; using O_d, U is able to open B_d from C_d.

If B_d is duplicate, U constructs the following proof:

$$\pi_7 = \mathsf{NIPK}\left\{ \begin{array}{l} (x, B_d, \tau_d, \phi_d, C_d, O_d, \sigma_d, B_e, \tau_e, \phi_e, \\ C_e, O_e, \sigma_e) : \\ \quad (C_d, O_d) = \mathsf{CMT}(B_d, \tau_d, \phi_d, x) \wedge \\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_d, C_d) = Accept \wedge \\ \quad \kappa_d = \mathsf{Indic}(\sigma_d) \wedge \\ \quad (C_e, O_e) = \mathsf{CMT}(B_e, \tau_e, \phi_e, x) \wedge \\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_e, C_e) = Accept \wedge \\ \quad B_d = B_e \end{array} \right\}$$

and sends $\kappa_d \| \pi_7$ to D though an established Tor tunnel.

D verifies $\kappa_d \notin elist_{B_d}$ and π_7, runs $\binom{m}{1}$-OT to provide a new bridge $\tilde{B}_d$, and adds κ_d to $elist_{B_d}$. After receiving B_d, U constructs the proof:

$$\pi_8 = \mathsf{NIPK}\left\{ \begin{array}{l} (x, B_d, \tau_d, \phi_d, C_d, O_d, \sigma_d, \tilde{B}_d, \tilde{\tau}_d, \tilde{\phi}_d, \tilde{O}_d) : \\ \quad (C_d, O_d) = \mathsf{CMT}(B_d, \tau_d, \phi_d, x) \wedge \\ \quad \mathsf{Verify}(PK_{\mathtt{D}}, \sigma_d, C_d) = Accept \wedge \\ \quad \kappa_d = \mathsf{Indic}(\sigma_d) \wedge \\ \quad \tilde{\tau}_d = T_{cur} \wedge \tilde{\phi}_d = 0 \wedge \\ \quad (\tilde{C}_d, \tilde{O}_d) = \mathsf{CMT}(\tilde{B}_d, \tilde{\tau}_d, \tilde{\phi}_d, x) \end{array} \right\}$$

and sends $\tilde{C}_d \| \pi_8$ to D. Note that we include the commitment and signature of the duplicate bridge B_d in π_8 to prevent U from giving this opportunity of receiving a replacement bridge to another (colluding) user. Finally, D verifies π_8, signs $\tilde{C}_d$, and sends $\tilde{\sigma}_d$ to U.

B Construction Details

Let (G_1, G_2) be a bilinear group pair and G_p be a group of order p where DDH is intractable with $\hat{e} : G_1 \times G_2 \to G_p$, s.t., $\hat{e}(P^a, Q^b) = \hat{e}(P, Q)^{ab}$, for all $P \in G_1$, $Q \in G_2$, $a, b \in Z_p$.

B.1 Key Generation

Let $g_0, g_1, g_2, g_3, g_4, g_5$ be generators of G_1, and h be a generator of G_2. D chooses $sk \xleftarrow{R} Z_p^*$ as the private key, and computes $pk = h^{sk}$. Let z denote a random element in G_1. The public key is $(g_0, g_1, g_2, g_3, g_4, g_5, z, h, G_1, G_2, G_p, \hat{e}, pk)$. In addition, D chooses a random secret $scrt_{\mathtt{D}}$ from Z_p^* to generate invitation tickets.

B.2 Registration

U first picks a list of nonces $y'_j \xleftarrow{R} Z_p^*$, $1 \le j \le m$, computes $Y'_j = g_1^{y'_j}$, constructs the following proof:

$$\pi_0 = \mathsf{NIPK}\left\{ \begin{array}{l} (\{y'_j\}_{j=1}^m) : \\ \quad \bigwedge_{j=1}^m \left[Y'_j = g_1^{y'_j} \right] \end{array} \right\}$$

and sends $\{Y'_j\}_{j=1}^m \| \pi_0$ to D.

D verifies π_0, and then chooses a pair of one-time keys $sk^o \xleftarrow{R} Z_p^*$, $pk^o = h^{sk^o}$. For each available bridge B_j, D randomly selects $e_j^o, y''_j \xleftarrow{R} Z_p^*$, computes $A_j^o = (g_0 g_1^{y''_j} Y_j g_3^{B_j})^{\frac{1}{e_j^o + sk^o}}$, and tags (A_j^o, e_j^o, y''_j) to B_j.

After OT, U receives $\{B_i \| (A_i^o, e_i^o, y''_i)\}_{i=1}^k$. For each $i \in [1, k]$, U computes $s_i^o = y'_i + y''_i$, and sets $\sigma_i^o = (A_i^o, e_i^o, s_i^o)$. To prove possession of these signatures, U picks $r_i^{(1)}, r_i^{(2)} \xleftarrow{R} Z_p^*$, and computes $A_i^{(1)} = g_1^{r_i^{(1)}} g_2^{r_i^{(2)}}$, $A_i^{(2)} = A_i^o g_2^{r_i^{(1)}}$, $\delta_i^{(1)} = r_i^{(1)} e_i^o$, $\delta_i^{(2)} = r_i^{(2)} e_i^o$. To get the initial credential, U sets $\Phi = 0$ and $\omega = T_{cur}$, picks $s'_\Phi, s'_\omega \xleftarrow{R} Z_p^*$, and computes $C_\Phi = g_1^{s'_\Phi} g_2^x g_3^\Phi$, $C_\omega = g_1^{s'_\omega} g_2^x g_3^\omega$; for each $i \in [1, k]$, U sets $\tau_i = T_{cur}, \phi_i = 0$, picks $s'_i \xleftarrow{R} Z_p^*$, and computes $C_i = g_1^{s'_i} g_2^x g_3^{B_i} g_4^{\tau_i} g_5^{\phi_i}$ (where $s'_\spadesuit$ is the opening to $C_\spadesuit$). Then, U computes the proof:

$$\pi_1 = \mathsf{NIPK}\left\{ \begin{array}{l} (x, \Phi, s'_\Phi, \omega, s'_\omega, \{B_i, \tau_i, \phi_i, e_i^o, s_i^o, s'_i, \\ r_i^{(1)}, r_i^{(2)}, \delta_i^{(1)}, \delta_i^{(2)}\}_{i=1}^k) : \\ \quad \bigwedge_{i=1}^k \Big[A_i^{(1)} = g_1^{r_i^{(1)}} g_2^{r_i^{(2)}} \wedge \\ \qquad (A_i^{(1)})^{e_i^o} = g_1^{\delta_i^{(1)}} g_2^{\delta_i^{(2)}} \wedge \\ \qquad \frac{\hat{e}(A_i^{(2)}, pk^o)}{\hat{e}(g_0, h)} = \hat{e}(A_i^{(2)}, h)^{-e_i^o} \hat{e}(g_2, y)^{r_i^{(1)}} \\ \qquad \hat{e}(g_2, h)^{\delta_i^{(1)}} \hat{e}(g_1, h)^{s_i^o} \hat{e}(g_3, h)^{B_i} \wedge \\ \qquad \tau_i = T_{cur} \wedge \phi_i = 0 \wedge \\ \qquad C_i = g_1^{s'_i} g_2^x g_3^{B_i} g_4^{\tau_i} g_5^{\phi_i} \Big] \wedge \\ \quad C_\Phi = g_1^{s'_\Phi} g_2^x g_3^\Phi \wedge \\ \quad \kappa_x = z^x \wedge \\ \quad \Phi = 0 \wedge \\ \quad C_\omega = g_1^{s'_\omega} g_2^x g_3^\omega \wedge \\ \quad \omega = T_{cur} \end{array} \right\}$$

and sends $\kappa_x \| C_\Phi \| C_\omega \| \{A_i^{(1)}, A_i^{(2)}, C_i\}_{i=1}^k \| \pi_1$ to D.

After verifying π_1 and κ_x, D picks $e_\Phi, s''_\Phi, e_\omega, s''_\omega \xleftarrow{R} Z_p^*$, and computes $A_\Phi = (g_0 g_1^{s''_\Phi} C_\Phi)^{\frac{1}{e_\Phi + sk}}$, $A_\omega = (g_0 g_1^{s''_\omega} C_\omega)^{\frac{1}{e_\omega + sk}}$. For each $i \in [1, k]$, D picks $e_i, s''_i \xleftarrow{R} Z_p^*$, and computes $A_i = (g_0 g_1^{s''_i} C_i)^{\frac{1}{e_i + sk}}$. Then D sends $(A_\Phi, e_\Phi, s''_\Phi) \| (A_\omega, e_\omega, s''_\omega) \| \{(A_i, e_i, s''_i)\}_{i=1}^k$ to U.

U computes $s_\Phi = s'_\Phi + s''_\Phi$, $s_\omega = s'_\omega + s''_\omega$, and $s_i = s'_i + s''_i$, $1 \le i \le k$, and sets $\sigma_\Phi = (A_\Phi, e_\Phi, s_\Phi)$, $\sigma_\omega = (A_\omega, e_\omega, s_\omega)$, and $\sigma_i = (A_i, e_i, s_i)$, $1 \le i \le k$.

B.3 Updating Credit Balance

Suppose U wants to update his credit balance with the credits earned from B_u. U needs to prove possession of σ_u and σ_Φ. For that, U picks $r_u^{(1)}, r_u^{(2)}, r_\Phi^{(1)}, r_\Phi^{(2)} \xleftarrow{R} Z_p^*$, and computes $A_u^{(1)} = g_1^{r_u^{(1)}} g_2^{r_u^{(2)}}$, $A_u^{(2)} = A_u g_2^{r_u^{(1)}}$, $\delta_u^{(1)} = r_u^{(1)} e_u$, $\delta_u^{(2)} = r_u^{(2)} e_u$, $A_\Phi^{(1)} = g_1^{r_\Phi^{(1)}} g_2^{r_\Phi^{(2)}}$, $A_\Phi^{(2)} = A_\Phi g_2^{r_\Phi^{(1)}}$, $\delta_\Phi^{(1)} = r_\Phi^{(1)} e_\Phi$, $\delta_\Phi^{(2)} = r_\Phi^{(2)} e_\Phi$. In addition, U needs to show that B_u is not blocked by proving that $b_j \neq z^{B_u}$ for each $b_j = z^{\bar{B}_j}$, where $\{\bar{B}_j\}_{j=1}^{\bar{m}}$ is the list of blocked bridges.

To update the credential, U calculates $\tilde{\phi}_u = \mathsf{Credit}(T_{cur} - \tau_u)$ and $\tilde{\Phi} = \Phi + \tilde{\phi}_u - \phi_u$; then, he picks $\tilde{s}'_u, \tilde{s}'_\Phi \xleftarrow{R} Z_p^*$, and computes $\tilde{C}_u = g_1^{\tilde{s}'_u} g_2^x g_3^{B_u} g_4^{\tau_u} g_5^{\tilde{\phi}_u}$, $\tilde{C}_\Phi = g_1^{\tilde{s}'_\Phi} g_2^x g_3^{\tilde{\Phi}}$. After that, U constructs the following proof:

$$\pi_2 = \mathsf{NIPK}\left\{\begin{array}{l}(x,\Phi,C_\Phi,e_\Phi,s_\Phi,s'_\Phi,r_\Phi^{(1)},r_\Phi^{(2)},\delta_\Phi^{(1)},\delta_\Phi^{(2)},\\ B_u,\tau_u,\phi_u,C_u,e_u,s_u,s'_u,r_u^{(1)},r_u^{(2)},\delta_u^{(1)},\delta_u^{(2)},\\ \tilde{\Phi},\tilde{s}'_\Phi,\tilde{\phi}_u,\tilde{s}'_u):\\ \quad \bigwedge_{j=1}^{\bar{m}}\left[b_j \neq z^{B_u}\right]\wedge\\ \quad C_u = g_1^{s'_u} g_2^{x} g_3^{B_u} g_4^{\tau_u} g_5^{\phi_u}\wedge\\ \quad A_u^{(1)} = g_1^{r_u^{(1)}} g_2^{r_u^{(2)}}\wedge\\ \quad (A_u^{(1)})^{e_u} = g_1^{\delta_u^{(1)}} g_2^{\delta_u^{(2)}}\wedge\\ \quad \frac{\hat{e}(A_u^{(2)},pk)}{\hat{e}(g_0,h)} = \hat{e}(A_u^{(2)},h)^{-e_u}\hat{e}(g_2,y)^{r_u^{(1)}}\\ \qquad \hat{e}(g_2,h)^{\delta_u^{(1)}}\hat{e}(g_1,h)^{s_u}\hat{e}(g_2,h)^{x}\\ \qquad \hat{e}(g_3,h)^{B_u}\hat{e}(g_4,h)^{\tau_u}\hat{e}(g_5,h)^{\phi_u}\wedge\\ \quad C_\Phi = g_1^{s'_\Phi} g_2^{x} g_3^{\Phi}\wedge\\ \quad A_\Phi^{(1)} = g_1^{r_\Phi^{(1)}} g_2^{r_\Phi^{(2)}}\wedge\\ \quad (A_\Phi^{(1)})^{e_\Phi} = g_1^{\delta_\Phi^{(1)}} g_2^{\delta_\Phi^{(2)}}\wedge\\ \quad \frac{\hat{e}(A_\Phi^{(2)},pk)}{\hat{e}(g_0,h)} = \hat{e}(A_\Phi^{(2)},h)^{-e_\Phi}\hat{e}(g_2,y)^{r_\Phi^{(1)}}\\ \qquad \hat{e}(g_2,h)^{\delta_\Phi^{(1)}}\hat{e}(g_1,h)^{s_\Phi}\hat{e}(g_2,h)^{x}\hat{e}(g_3,h)^{\Phi}\wedge\\ \quad \kappa_\Phi = z^{s_\Phi}\wedge\\ \quad t_u = T_{cur} - \tau_u\wedge\\ \quad \Big[\Big(t_u < T_0 \wedge \tilde{\phi}_u = 0\Big)\vee\\ \qquad \Big(t_u \geq T_0 \wedge t_u \leq T_1 \wedge \tilde{\phi}_u = \rho(t - T_0)\Big)\vee\\ \qquad \Big(t_u > T_1 \wedge \tilde{\phi}_u = \rho(T_1 - T_0)\Big)\Big]\wedge\\ \quad \tilde{\Phi} = \Phi + \tilde{\phi}_u - \phi_u\wedge\\ \quad \tilde{C}_u = g_1^{\tilde{s}'_u} g_2^{x} g_3^{B_u} g_4^{\tau_u} g_5^{\tilde{\phi}_u}\wedge\\ \quad \tilde{C}_\Phi = g_1^{\tilde{s}'_\Phi} g_2^{x} g_3^{\tilde{\Phi}}\wedge\end{array}\right\}$$

and sends $\kappa_\Phi \| A_\Phi^{(1)} \| A_\Phi^{(2)} \| A_u^{(1)} \| A_u^{(2)} \| \tilde{C}_\Phi \| \tilde{C}_u \| \pi_2$ to D.

D verifies π_2 and κ_Φ, picks $\tilde{e}_\Phi, \tilde{s}''_\Phi, \tilde{e}_u, \tilde{s}''_u \xleftarrow{R} Z_p^*$, and computes $\tilde{A}_\Phi = (g_0 g_1^{\tilde{s}''_\Phi}\tilde{C}_\Phi)^{\frac{1}{\tilde{e}_\Phi + sk}}$, $\tilde{A}_u = (g_0 g_1^{\tilde{s}''_u}\tilde{C}_u)^{\frac{1}{\tilde{e}_u + sk}}$. Then D sends $(\tilde{A}_\Phi, \tilde{e}_\Phi, \tilde{s}''_\Phi) \| (\tilde{A}_u, \tilde{e}_u, \tilde{s}''_u)$ to U.

U computes $\tilde{s}_\Phi = \tilde{s}'_\Phi + \tilde{s}''_\Phi$, $\tilde{s}_u = \tilde{s}'_u + \tilde{s}''_u$, sets $\tilde{\sigma}_\Phi = (\tilde{A}_\Phi, \tilde{e}_\Phi, \tilde{s}_\Phi)$, $\tilde{\sigma}_u = (\tilde{A}_u, \tilde{e}_u, \tilde{s}_u)$, and updates his credential with $\tilde{\Phi}, \tilde{C}_\Phi, \tilde{s}'_\Phi, \tilde{\sigma}_\Phi, \tilde{\phi}_u, \tilde{C}_u, \tilde{s}'_u$, and $\tilde{\sigma}_u$.

B.3.1 Getting a New Bridge

Suppose U wants to replace a blocked bridge B_b with a new bridge. U sends B_b to D through an established Tor tunnel, and D verifies B_b is indeed blocked and then replies with the blocking time β_b of B_b.

U needs to prove possession of the signatures σ_b and σ_Φ. For this, U picks $r_b^{(1)}, r_b^{(2)}, r_\Phi^{(1)}, r_\Phi^{(2)} \xleftarrow{R} Z_p^*$, and computes $A_b^{(1)} = g_1^{r_b^{(1)}} g_2^{r_b^{(2)}}$, $A_b^{(2)} = A_b g_2^{r_b^{(1)}}$, $\delta_b^{(1)} = r_b^{(1)} e_b$, $\delta_b^{(2)} = r_u^{(2)} e_b$, $A_\Phi^{(1)} = g_1^{r_\Phi^{(1)}} g_2^{r_\Phi^{(2)}}$, $A_\Phi^{(2)} = A_\Phi g_2^{r_\Phi^{(1)}}$, $\delta_\Phi^{(1)} = r_\Phi^{(1)} e_\Phi$, $\delta_\Phi^{(2)} = r_\Phi^{(2)} e_\Phi$.

U can earn $\tilde{\phi}_b = \mathsf{Credit}(\beta_b - \tau_b)$ credits in total from B_b, and the resulting credit balance after paying for the new bridge is $\tilde{\Phi} = \Phi + \tilde{\phi}_b - \phi_b - \phi^-$. U picks $\tilde{s}'_\Phi \xleftarrow{R} Z_p^*$, computes $\tilde{C}_\Phi = g_1^{\tilde{s}'_\Phi} g_2^{x} g_3^{\tilde{\Phi}}$, constructs the following proof:

$$\pi_3 = \mathsf{NIPK}\left\{\begin{array}{l}(x,\Phi,C_\Phi,e_\Phi,s_\Phi,s'_\Phi,r_\Phi^{(1)},r_\Phi^{(2)},\delta_\Phi^{(1)},\delta_\Phi^{(2)},\tau_b,\\ \phi_b,C_b,e_b,s_b,s'_b,r_b^{(1)},r_b^{(2)},\delta_b^{(1)},\delta_b^{(2)},\tilde{\Phi},\tilde{s}'_\Phi):\\ \quad C_b = g_1^{s'_b} g_2^{x} g_3^{B_b} g_4^{\tau_b} g_5^{\phi_b}\wedge\\ \quad A_b^{(1)} = g_1^{r_b^{(1)}} g_2^{r_b^{(2)}}\wedge\\ \quad (A_b^{(1)})^{e_b} = g_1^{\delta_b^{(1)}} g_2^{\delta_b^{(2)}}\wedge\\ \quad \frac{\hat{e}(A_b^{(2)},pk)}{\hat{e}(g_0,h)} = \hat{e}(A_b^{(2)},h)^{-e_b}\hat{e}(g_2,y)^{r_b^{(1)}}\\ \qquad \hat{e}(g_2,h)^{\delta_b^{(1)}}\hat{e}(g_1,h)^{s_b}\hat{e}(g_2,h)^{x}\\ \qquad \hat{e}(g_3,h)^{B_b}\hat{e}(g_4,h)^{\tau_b}\hat{e}(g_5,h)^{\phi_b}\wedge\\ \quad \kappa_b = z^{s_b}\wedge\\ \quad C_\Phi = g_1^{s'_\Phi} g_2^{x} g_3^{\Phi}\wedge\\ \quad A_\Phi^{(1)} = g_1^{r_\Phi^{(1)}} g_2^{r_\Phi^{(2)}}\wedge\\ \quad (A_\Phi^{(1)})^{e_\Phi} = g_1^{\delta_\Phi^{(1)}} g_2^{\delta_\Phi^{(2)}}\wedge\\ \quad \frac{\hat{e}(A_\Phi^{(2)},pk)}{\hat{e}(g_0,h)} = \hat{e}(A_\Phi^{(2)},h)^{-e_\Phi}\hat{e}(g_2,y)^{r_\Phi^{(1)}}\\ \qquad \hat{e}(g_2,h)^{\delta_\Phi^{(1)}}\hat{e}(g_1,h)^{s_\Phi}\hat{e}(g_2,h)^{x}\hat{e}(g_3,h)^{\Phi}\wedge\\ \quad \kappa_\Phi = z^{s_\Phi}\wedge\\ \quad t_b = \beta_b - \tau_b\wedge\\ \quad \Big[\Big(t_b < T_0 \wedge \tilde{\phi}_b = 0\Big)\vee\\ \qquad \Big(t_b \geq T_0 \wedge t_b \leq T_1 \wedge \tilde{\phi}_b = \rho(t_b - T_0)\Big)\vee\\ \qquad \Big(t_b > T_1 \wedge \tilde{\phi}_b = \rho(T_1 - T_0)\Big)\Big]\wedge\\ \quad \tilde{\Phi} = \Phi + \tilde{\phi}_b - \phi_b - \phi^-\wedge\\ \quad \tilde{\Phi} > 0\\ \quad \tilde{C}_\Phi = g_1^{\tilde{s}'_\Phi} g_2^{x} g_3^{\tilde{\Phi}}\wedge\end{array}\right\}$$

and sends $\kappa_\Phi \| \kappa_b \| A_\Phi^{(1)} \| A_\Phi^{(2)} \| A_b^{(1)} \| A_b^{(2)} \| \tilde{C}_\Phi \| \pi_3$ to D.

D verifies π_3, κ_b, and κ_Φ. Similar to the OT in the registration, U sends D a list of nonces, and D chooses a pair of one-time keys to sign each available bridge using the corresponding nonce. Running OT, U obtains $\tilde{B}_b \| \tilde{\sigma}_b^o$, where $\tilde{\sigma}_b^o = (\tilde{A}_b^o, \tilde{e}_b^o, \tilde{s}_b^o)$.

To update the credential with the new bridge $\tilde{B}_b$, U sets $\tilde{\tau}_b = T_{cur}$ and $\tilde{\phi}_b = 0$, picks $\tilde{s}'_b \xleftarrow{R} Z_p^*$, and computes $\tilde{C}_b = g_1^{\tilde{s}'_b} g_2^{x} g_3^{\tilde{B}_b} g_4^{\tilde{\tau}_b} g_5^{\tilde{\phi}_b}$. To prove possession of $\tilde{\sigma}_b^o$, U picks $\tilde{r}_b^{(1)}, \tilde{r}_b^{(2)} \xleftarrow{R} Z_p^*$, and computes $\tilde{A}_b^{(1)} = g_1^{\tilde{r}_b^{(1)}} g_2^{\tilde{r}_b^{(2)}}$, $\tilde{A}_b^{(2)} = \tilde{A}_b^o g_2^{\tilde{r}_b^{(1)}}$, $\tilde{\delta}_b^{(1)} = \tilde{r}_b^{(1)} \tilde{e}_b^o$, $\tilde{\delta}_b^{(2)} = \tilde{r}_b^{(2)} \tilde{e}_b^o$. Then, U constructs the following proof:

$$\pi_4 = \mathsf{NIPK}\left\{\begin{array}{l}(x, \tilde{\Phi}, \tilde{s}'_\Phi, \tilde{B}_b, \tilde{\tau}_b, \tilde{\phi}_b, \tilde{e}^o_b, \tilde{s}^o_b, \tilde{s}'_b, \tilde{r}^{(1)}_b, \tilde{r}^{(2)}_b, \\ \tilde{\delta}^{(1)}_b, \tilde{\delta}^{(2)}_b) : \\ \quad \tilde{C}_\Phi = g_1^{\tilde{s}'_\Phi} g_2^x g_3^{\tilde{\Phi}} \\ \quad \tilde{\tau}_b = T_{cur} \wedge \tilde{\phi}_b = 0 \wedge \\ \quad \tilde{C}_b = g_1^{\tilde{s}'_b} g_2^x g_3^{\tilde{B}_b} g_4^{\tilde{\tau}_b} g_5^{\tilde{\phi}_b} \wedge \\ \quad \tilde{A}^{(1)}_b = g_1^{\tilde{r}^{(1)}_b} g_2^{\tilde{r}^{(2)}_b} \wedge \\ \quad (\tilde{A}^{(1)}_b)^{\tilde{e}_b} = g_1^{\tilde{\delta}^{(1)}_b} g_2^{\tilde{\delta}^{(2)}_b} \wedge \\ \quad \frac{\hat{e}(\tilde{A}^{(2)}_b, pk)}{\hat{e}(g_0, h)} = \hat{e}(\tilde{A}^{(2)}_b, h)^{-\tilde{e}^o_b} \hat{e}(g_2, h)^{\tilde{r}^{(1)}_b} \\ \quad\quad \hat{e}(g_2, h)^{\tilde{\delta}^{(1)}_b} \hat{e}(g_1, h)^{\tilde{s}^o_b} \hat{e}(g_3, h)^{\tilde{B}_b}\end{array}\right\}$$

and sends $\tilde{A}^{(1)}_b \| \tilde{A}^{(2)}_b \| \tilde{C}_b \| \pi_4$ to D.

D verifies π_4, picks $\tilde{e}_\Phi, \tilde{s}''_\Phi, \tilde{e}_b, \tilde{s}''_b \xleftarrow{R} Z^*_p$, and computes $\tilde{A}_\Phi = (g_0 g_1^{\tilde{s}''_\Phi} \tilde{C}_\Phi)^{\frac{1}{\tilde{e}_\Phi + sk}}$, $\tilde{A}_b = (g_0 g_1^{\tilde{s}''_b} \tilde{C}_b)^{\frac{1}{\tilde{e}_b + sk}}$. Then D sends $(\tilde{A}_\Phi, \tilde{e}_\Phi, \tilde{s}''_\Phi) \| (\tilde{A}_b, \tilde{e}_b, \tilde{s}''_b)$ to U.

U computes $\tilde{s}_\Phi = \tilde{s}'_\Phi + \tilde{s}''_\Phi$, $\tilde{s}_b = \tilde{s}'_b + \tilde{s}''_b$, sets $\tilde{\sigma}_\Phi = (\tilde{A}_\Phi, \tilde{e}_\Phi, \tilde{s}_\Phi)$, $\tilde{\sigma}_b = (\tilde{A}_b, \tilde{e}_b, \tilde{s}_b)$, and updates his credential with $\tilde{\Phi}, \tilde{C}_\Phi, \tilde{s}'_\Phi, \tilde{\sigma}_\Phi, \tilde{\phi}_b, \tilde{C}_b, \tilde{s}'_b$, and $\tilde{\sigma}_b$.

B.4 Inviting New Users

A user U who requests an invitation ticket needs to prove that his credit balance is higher than the threshold, i.e., $\Phi > \Phi_\theta$, and the last time he applied for an invitation ticket is at least ω_θ days ago, i.e., $T_{cur} - \omega \geq \omega_\theta$. In addition, U needs to prove possession of the signatures σ_Φ and σ_ω. Hence, U constructs the following proof:

$$\pi_5 = \mathsf{NIPK}\left\{\begin{array}{l}(x, \Phi, C_\Phi, e_\Phi, s_\Phi, s'_\Phi, r^{(1)}_\Phi, r^{(2)}_\Phi, \delta^{(1)}_\Phi, \delta^{(2)}_\Phi, \omega, \\ C_\omega, e_\omega, s_\omega, s'_\omega, r^{(1)}_\omega, r^{(2)}_\omega, \delta^{(1)}_\omega, \delta^{(2)}_\omega, \tilde{\omega}, \tilde{s}'_\omega, \tilde{s}'_\Phi) : \\ \quad C_\Phi = g_1^{s'_\Phi} g_2^x g_3^\Phi \wedge \\ \quad A^{(1)}_\Phi = g_1^{r^{(1)}_\Phi} g_2^{r^{(2)}_\Phi} \wedge \\ \quad (A^{(1)}_\Phi)^{e_\Phi} = g_1^{\delta^{(1)}_\Phi} g_2^{\delta^{(2)}_\Phi} \wedge \\ \quad \frac{\hat{e}(A^{(2)}_\Phi, pk)}{\hat{e}(g_0, h)} = \hat{e}(A^{(2)}_\Phi, h)^{-e_\Phi} \hat{e}(g_2, y)^{r^{(1)}_\Phi} \\ \quad\quad \hat{e}(g_2, h)^{\delta^{(1)}_\Phi} \hat{e}(g_1, h)^{s_\Phi} \hat{e}(g_2, h)^x \hat{e}(g_3, h)^\Phi \wedge \\ \quad \kappa_\Phi = z^{s_\Phi} \wedge \\ \quad C_\omega = g_1^{s'_\omega} g_2^x g_3^\omega \wedge \\ \quad A^{(1)}_\omega = g_1^{r^{(1)}_\omega} g_2^{r^{(2)}_\omega} \wedge \\ \quad (A^{(1)}_\omega)^{e_\omega} = g_1^{\delta^{(1)}_\omega} g_2^{\delta^{(2)}_\omega} \wedge \\ \quad \frac{\hat{e}(A^{(2)}_\omega, pk)}{\hat{e}(g_0, h)} = \hat{e}(A^{(2)}_\omega, h)^{-e_\omega} \hat{e}(g_2, y)^{r^{(1)}_\omega} \\ \quad\quad \hat{e}(g_2, h)^{\delta^{(1)}_\omega} \hat{e}(g_1, h)^{s_\omega} \hat{e}(g_2, h)^x \hat{e}(g_3, h)^\omega \wedge \\ \quad \kappa_\omega = z^{s_\omega} \wedge \\ \quad \Phi > \Phi_\theta \wedge \\ \quad T_{cur} - \omega > \omega_\theta \wedge \\ \quad \tilde{\omega} = T_{cur} \wedge \\ \quad \tilde{C}_\omega = g_1^{\tilde{s}'_\omega} g_2^x g_3^{\tilde{\omega}} \wedge \\ \quad \tilde{C}_\Phi = g_1^{\tilde{s}'_\Phi} g_2^x g_3^\Phi\end{array}\right\}$$

and sends $kappa_\Phi \| \kappa_\omega \| A^{(1)}_\Phi \| A^{(2)}_\Phi \| A^{(1)}_\omega \| A^{(2)}_\omega \| \tilde{C}_\Phi \| \tilde{C}_\omega \| \pi_5$ to D.

After verifying π_5, κ_Φ and κ_ω, D picks $\tilde{e}_\Phi, \tilde{s}''_\Phi, \tilde{e}_\omega, \tilde{s}''_\omega \xleftarrow{R} Z^*_p$, computes $\tilde{A}_\Phi = (g_0 g_1^{\tilde{s}''_\Phi} \tilde{C}_\Phi)^{\frac{1}{\tilde{e}_\Phi + sk}}$, $\tilde{A}_\omega = (g_0 g_1^{\tilde{s}''_\omega} \tilde{C}_\omega)^{\frac{1}{\tilde{e}_\omega + sk}}$, and sends $(\tilde{A}_\Phi, \tilde{e}_\Phi, \tilde{s}''_\Phi) \| (\tilde{A}_\omega, \tilde{e}_\omega, \tilde{s}''_\omega)$ to U. . Then, D flips a biased coin to decide whether to give an invitation ticket to U; if so, D generates an one-time ticket $tk = \{r^*, \mathsf{HMAC}_{scrt_\mathrm{D}}(r^*)\}$, where $r^* \xleftarrow{R} Z^*_p$, and sends it to U.

Regardless of receiving an invitation ticket, U computes $\tilde{s}_\Phi = \tilde{s}'_\Phi + \tilde{s}''_\Phi$, $\tilde{s}_\omega = \tilde{s}'_\omega + \tilde{s}''_\omega$, sets $\tilde{\sigma}_\Phi = (\tilde{A}_\Phi, \tilde{e}_\Phi, \tilde{s}_\Phi)$, $\tilde{\sigma}_\omega = (\tilde{A}_\omega, \tilde{e}_\omega, \tilde{s}_\omega)$, and updates his credential with $\tilde{C}_\Phi, \tilde{s}'_\Phi, \tilde{\sigma}_\Phi, \tilde{\phi}_\omega, \tilde{C}_\omega, \tilde{s}'_\omega$, and $\tilde{\sigma}_\omega$.