ROBUST CENSORSHIP-RESISTANT COMMUNICATION

BY

QIYAN WANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

ABSTRACT

A key challenge in censorship circumvention is being able to direct legitimate users to redirection proxies while preventing censors, posing as insiders, from discovering their addresses and blocking them. In this thesis, we study how to protect and/or design censorship circumvention systems to resist the insider attacks.

Tor is one of the most popular censorship circumvention systems; it uses *bridges* run by volunteers as proxies to evade censorship. 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.

We propose a new framework for censorship-resistant web browsing called *CensorSpoofer* that addresses this challenge by exploiting the asymmetric nature of web browsing traffic and making use of IP spoofing. CensorSpoofer decouples the upstream and downstream channels, using a low-bandwidth indirect channel for delivering outbound requests (URLs) and a high-bandwidth direct channel for downloading web content. The upstream channel hides the request contents using steganographic encoding within Email or instant messages, whereas the downstream channel uses IP address spoofing so that the real address of the proxies is not revealed either to legitimate users or censors. We built a proof-of-concept prototype that uses encrypted VoIP for this downstream channel and demonstrated the feasibility of using the CensorSpoofer framework in a realistic environment.

To my parents, without whom I would never have made it this far.
To my wife, Jenny Xiaoxue Li, who helps me through the hard times.
To my son Samuel and my daughter Arya, who are the greatest gifts in my life and bring me so much joy.

ACKNOWLEDGMENTS

I want to give my sincere thanks to my advisor, Professor Nikita Borisov, who took me as his student in the middle of my Ph.D. and has always been supportive and helpful to me ever since. He taught me all sorts of skills of doing research, from how to select topics to how to collaborate with people. Without him, my Ph.D. would never be this productive.

I am truly grateful to Professor Klara Nahrstedt, who is one of the nicest professors I have ever met. She has been so supportive and thoughtful for me. She taught me not only how to do research, but also how to be a good researcher, such as how to give good presentation and socialize with people. I cannot recall a single time she turned me down when I asked her for help, which I really appreciate.

I would like to give my deep gratitude to other members on my Ph.D. committee, Professor Carl Gunter and Professor Nicholas J. Hopper, for being willing to serve on my committee and providing invaluable comments and feedbacks on my dissertation.

I want to thank Professor Hopper for his great help on the rBridge project, and Dr. George Danezis for his mentoring and great help on the work of de-anonymizing Tor users using bridge assignment information. Both projects become parts of my thesis.

In addition, I want to thank Dr. Himanshu Khurana and Professor Manoj Prabhakaran, who were great mentors of mine during my Ph.D. study and patiently guided me in doing research and provided me a lot of invaluable suggestions.

I am also indebted to many of my colleagues and friends who supported me through all these years. I would like to thank all the members in the Hatswitch group (in alphabetical order) including Anupam Das, Xun Gong, Sonia Jahid, Joshua Juen, Prateek Mittal, and Giang Nguyen, as well as my old colleagues in the MONET group such as Wenbo He, Ying Huang, Hoang

Nguyen, Long Vu, and Wanmin Wu. I am also grateful to my friends (just to mention a few), Hailong Ning, Xuejiao Li, Jian Yang, Xiaoxu Zhou, Yanen Li, Jianchao Yang, Henghua Jin, Na Cui, Xi Zhou, Hao Tang, and Yanna Wu.

Lastly, I want to give special thanks to my wife, Jenny Xiaoxue Li, who has always been supportive and helped me get through those toughest times. I cannot find words to express my gratitude to her. I feel truly lucky to have her in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Today, the Internet is playing an ever-increasing role in social and political movements around the world. Activists use it to coordinate their activities and to inform the general people of important information that is not available via traditional media channels. The role played by Twitter, Facebook, YouTube, CNN iReport and many other websites/blogs in the recent events in the Middle East is a great example of this [1, 2].

The free flow of information and exchange of ideas on the Internet has been perceived as a serious threat by repressive regimes. In response, they have imposed strong censorship on the Internet usage of their citizens. They monitor, filter, trace, and block data flows using sophisticated technologies, such as IP address blocking, DNS hijacking, and deep packet inspection [3,4]. For example, the "Great Firewall of China" blocks almost all popular social networks, such as Facebook, Twitter and Flickr, and other websites that may provide political information contrary to the state's agenda, such as Youtube, Wikipedia, BBC News, and CNN [5]. To exercise control over the Internet, the Chinese government employs an Internet police force of over 30 000 people to constantly monitor the citizens' online activities [6], and an individual who is caught violating the laws of Chinese censorship is subject to payment of fines [7].

A typical approach to skirting censorship is to deploy circumvention proxies outside the censored network, which can provide indirect access to blocked websites [8–11]. For example, Tor [11] 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.

There are four desirable properties on a censorship circumvention system. Firstly, it should require no special support from the network infrastructure. For instance, some designs [16–18] assume that certain special routers can be deployed by core ISPs on backbone Internet, which might not be realistic in practice. Secondly, it should require no special server, which could easily be blocked by censors due to their small quantity. For example, some circumvention circumvention systems rely on oversea Email servers that support encryption; however, only `Gmail` and `hotmail` meet the requirements. Thirdly, it should be able to support web browsing, while some circumvention systems [42–46] are not designed to support low latency communication. Lastly, it should provide strong resistance against the insider attack. The hypothesis statement of this thesis is as follows:

"It is possible to build a censorship circumvention system that does not require special server or special support from the network infrastructure, and that provides web browsing and strong resistance to the insider attack."

1.1 Contribution

Reputation based bridge distribution. Our first contribution is to propose *rBridge* [12]—a user reputation system for Tor bridge distribution in order to improve the robustness of Tor bridges against the insider attack. 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 repeatedly 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 [13].

Privacy-preserving bridge distribution. Our second contribution is to design a privacy-preserving user reputation system for Tor bridge distribution, i.e, privacy-preserving rBridge [12]. The key in ensuring user anonymity in Tor is to preserve the information about each user's selection of relays (including bridges) in building anonymous circuits. To achieve so, the bridge assignment information of each user must be kept secret from any parties, including the bridge distributor. Therefore, in order to ensure anonymity in rBridge, 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 thesis, 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.

Proxy distribution strategies by limiting the amount of information each user gets and trying to identify compromised insiders (such as rBridge and some existing schemes [13–15]) can partially mitigate the insider attack, because some of the proxies can still be blocked (although the proxy consumption is fairly low for rBridge) and the system operator needs to keep adding new proxies to replace blocked ones. The key issue in proxy based circumvention systems is that proxies' addresses are potentially revealed to malicious users and thus are blockable. An alternate approach is to never reveal the proxies' address to legitimate users and thus be completely immune to the insider attack. Some recent work suggests strategically placing special deflection routers at core Internet ISPs to transparently redirect users' traffic to the proxies [16–18]. Such a deployment, however requires a significant resource investment that is likely to come only from a (pro-Internet freedom) government agency, as well as cooperation of large ISPs.

A new censorship circumvention architecture. Our third contribution is to propose a new censorship circumvention architecture, *CensorSpoofer* [19], that can be deployed using minimal resources, perhaps volunteered by ordinary people interested in promoting Internet freedom. (The Tor project [11] has demonstrated the feasibility of building a successful service with contributions from such volunteers.) Our key insight is that it is possible to use IP address spoofing to send data from the proxy to a user without revealing its actual origin. Such a spoofed channel allows communication in a single direction only; however, we can exploit the asymmetric nature of web-browsing traffic, using a low-bandwidth indirect channel, such as steganographic instant messages or Email, to communicate requests from the user to the proxy. To avoid identification by the censor, CensorSpoofer mimics an encrypted VoIP session to tunnel the downstream data, since the VoIP protocol does not require endpoints to maintain close synchronization and does not reveal its contents to the censor. We also explore additional steps that need to be taken to prevent detection; namely, choosing a plausible fake IP source address. To demonstrate the feasibility of CensorSpoofer, we built a proof-of-concept prototype implementation and tested it in a real-world environment. Our experiments show that our prototype can be successfully used for browsing the web while resisting blocking efforts of the censors.

1.2 Roadmap

We present the motivation, design, and evaluation of rBridge – a user reputation based Tor bridge distribution strategy in Chapter 2, and elaborate the cryptographic construction and evaluation of a privacy-preserving version of rBridge in Chapter 3. Chapter 4 describes a new censorship circumvention architecture, CensorSpoofer, as well as its prototype implementation and evaluation. We finally summarize the thesis research and discuss learned lessons in Chapter 5.

CHAPTER 2

A USER REPUTATION BASED APPROACH TO SECURING TOR BRIDGE DISTRIBUTION

Tor [11] is a popular anonymity system aiming to provide low-latency anonymous communication for a large scale of users. In Tor, each user uses three randomly selected relays to build an onion encryption tunnel in order to communicate with a remote host (e.g., a website) anonymously. As of May 4 2012, there are about 3 000 relays in the Tor network and over 500 000 directly connecting users daily [20].

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.

So far, Tor has tried four different distribution strategies. First, each user can receive a small subset of bridges based on their IP address as well as the current time. Second, a small subset can be obtained by sending a request via GMail. These strategies fail to protect against an adversary who has access to a large number of IP addresses and GMail accounts; Chinese censors were able to enumerate all bridges in under a month [21]. (McLachlan and Hopper further showed that open proxies could be used to gain access to a large number of IP addresses [22]). The third strategy involves distributing bridge addresses to a few trusted people in censored countries in an ad hoc manner, who then disseminate this information to their social networks. Fourth, an individual can deploy a private bridge and give the bridge's address only to trusted contacts. These methods can resist bridge discovery but reach only a limited fraction of the population of potential bridge users.

We propose *rBridge* [12]—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. As we shall show that, rBridge provides much stronger protection for bridges than any existing scheme.

2.1 Related Work on Proxy Distribution

Researchers have tried to design better proxy distribution strategies [13–15, 23]. Feamster et al. [23] 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 [23] 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. [15], 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. [13] 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.

2.2 Concept

We now present the design goals, threat model, and scope of rBridge.

2.2.1 Goals

rBridge aims to achieve the following goals:

1. *Maximized user-hours of bridges*: McCoy et al. [13] 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. (This property is to be achieved in the cryptographic version of rBridge presented in Chapter 3.)

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.

2.2.2 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 [13–15, 21, 23] 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 [11]. 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.

2.2.3 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 [21]. 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 [24], BridgeSPA [25], and client password authorization [26]. 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 [27], 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.

2.3 The Basic rBridge Scheme

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 [21]. 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 2.5), the existing schemes have a difficult time protecting bridges from being blocked even for a very limited period of time.

2.3.1 Joining the System

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).

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.4 Discussion of Other Security Threats

2.4.1 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.

2.4.2 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 [28]) 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.

2.4.3 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 2.5 (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.

2.5 Evaluation and Comparison

We now analyze the robustness of rBridge against the following blocking strategies, and compare it with Proximax [13]. We discuss other potential attacks in Section 2.4.

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

- *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.

2.5.1 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).

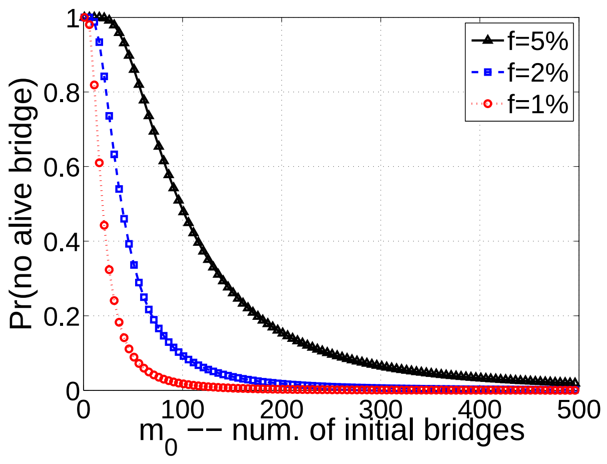

Figure 2.1: Number of initial bridges ($N = 1000$).

Each user is initially provided $k = 3$ bridges[1]. 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 2.1, 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

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

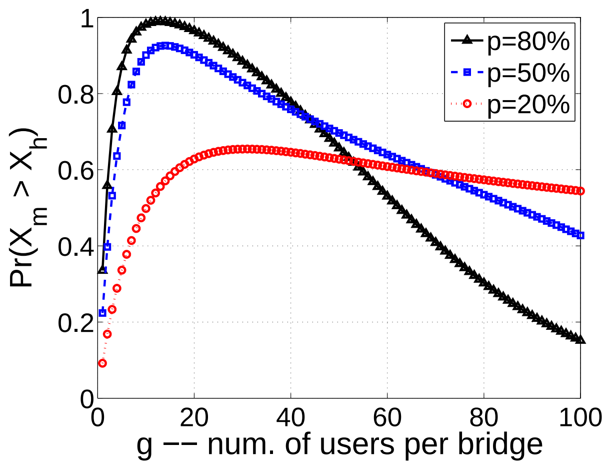

Figure 2.2: Number of users per bridge (p is attack probability, $f = 5\%$).

bridges since joining the system, we get:

$$Pr(X_h = x) = \binom{l}{x} \cdot (1-\lambda)^x \cdot \lambda^{l-x} \tag{2.1}$$

$$Pr(X_m = x) = \binom{l - p \cdot l}{x - p \cdot l} \cdot (1-\lambda)^{x-p\cdot l} \lambda^{l-x} \tag{2.2}$$

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. $Pr(X_m > X_h)$ is calculated as:

$$Pr(X_m > X_h) = \sum_{x=p\cdot l}^{l} Pr(X_m = x) \sum_{y=0}^{x-1} Pr(X_h = y) \tag{2.3}$$

Figure 2.2 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:

$$T_{lf} \cdot g \cdot s \cdot time = N \cdot k \cdot time \tag{2.4}$$

From this, we get:

$$T_0 = \frac{N \cdot k}{g \cdot s_{max}}, \quad T_1 = \frac{N \cdot k}{g \cdot s_{min}} \tag{2.5}$$

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 [20].) 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 (2.5). 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:

$$\begin{aligned} &\sum_{x=0}^{k} Pr(X_h = x) \cdot x \cdot \phi^- = \\ &\sum_{x=0}^{k} Pr(X_h = x) \cdot (k - x) \cdot (T_1 - T_0) \end{aligned} \tag{2.6}$$

From Equation (2.1) (2.6), 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:

$$\begin{aligned} \Phi_\theta \;=\; &\sum_{x=0}^{\lceil \frac{k}{2} \rceil} Pr(X_h = x | X_h \leq \lceil \frac{k}{2} \rceil) \\ &\cdot (k - x) \cdot (T_1 - T_0) \end{aligned} \tag{2.7}$$

From Equation (2.1) (2.7), we get $\Phi_\theta = 236$.

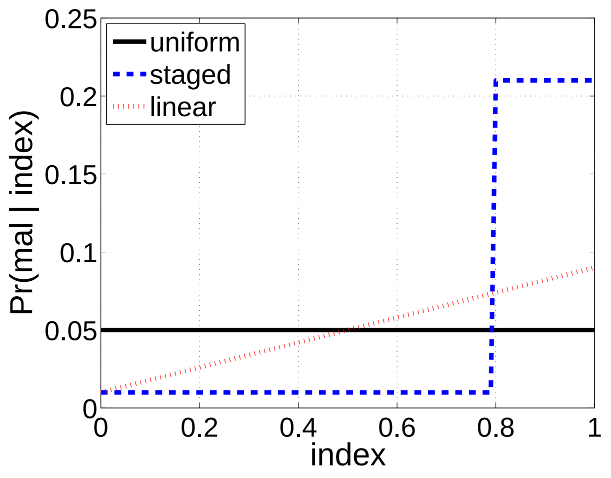

Figure 2.3: Probability distribution of malicious users ($f = 5\%$).

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, 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 0.0 to 1.0 according to the trustworthiness (e.g., the most trusted one out of 100 friends has the index 1.0/100 = 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 2.3 depicts the probability distributions of these two models.

2.5.2 Evaluation Results

Using the event-based simulator, we measured the *user-hours*, *thirsty-hours*, and *growth of user base* under different blocking strategies. We set the ratio of malicious users in the potential user group as 5%.

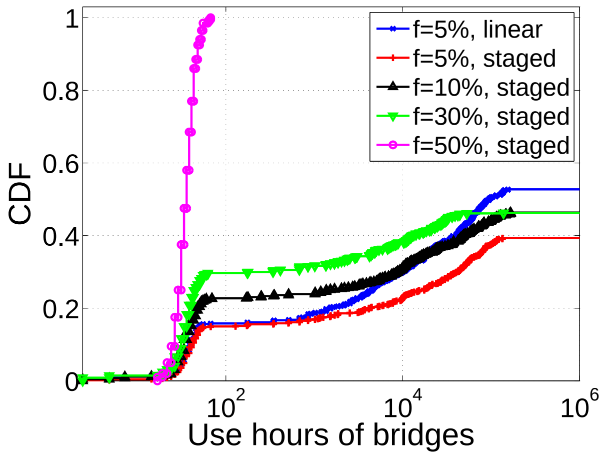

Figure 2.4: User-hours in aggressive blocking.

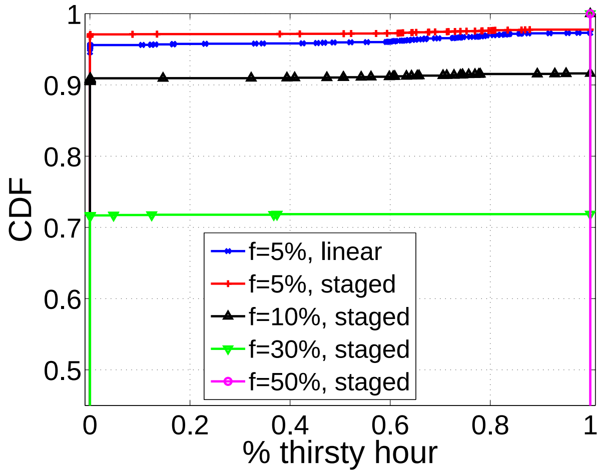

Figure 2.5: % of thirsty-hours in aggressive blocking.

Aggressive blocking. We now present the simulation results for the aggressive blocking. We can see from Figure 2.4 that when $f = 5\%$, 80%

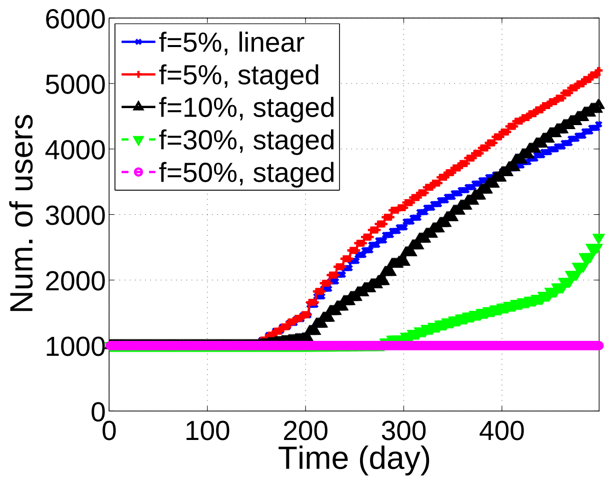

Figure 2.6: User base in aggressive blocking.

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 2.5 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 2.6 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

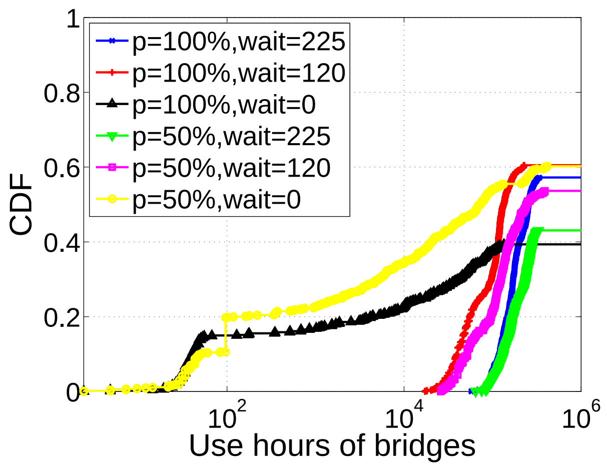

Figure 2.7: User-hours in conservative blocking.

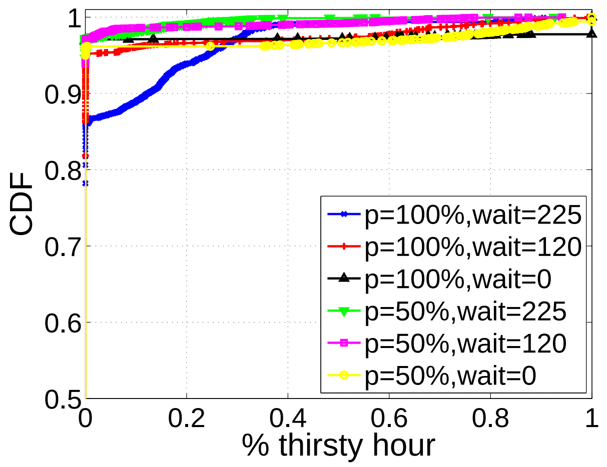

Figure 2.8: % of thirsty-hours in conservative blocking.

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 2.7, Figure 2.9, and Figure 2.8 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

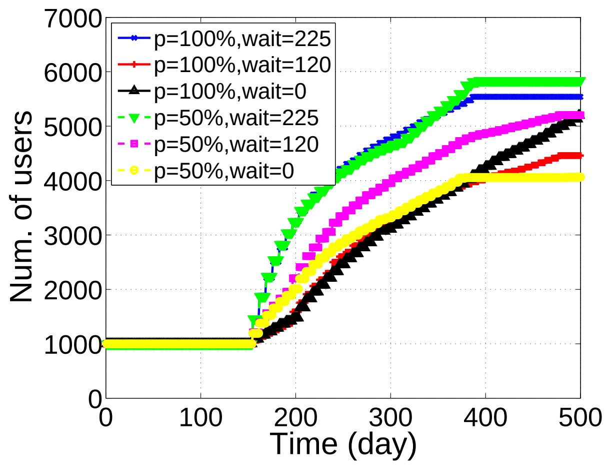

Figure 2.9: User base in conservative blocking.

can block more bridges, which causes more recently joined users to become thirsty.

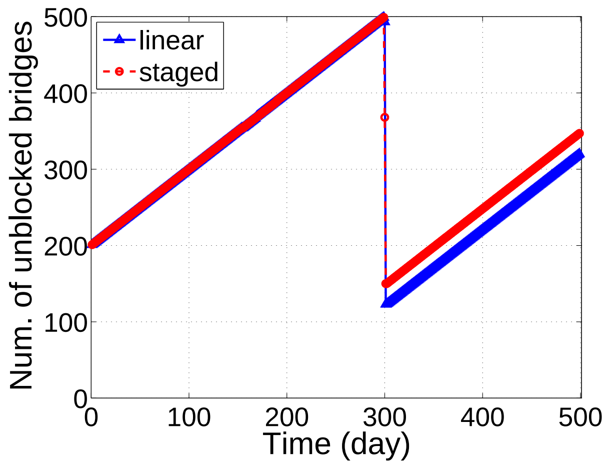

Figure 2.10: Unblocked bridges in event-driven blocking.

Event-driven blocking. For event-driven blocking, we let all of the corrupt users are synchronized to block all their bridges simultaneously on the 300-th day. Figures 2.10 and 2.11 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 2.12 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

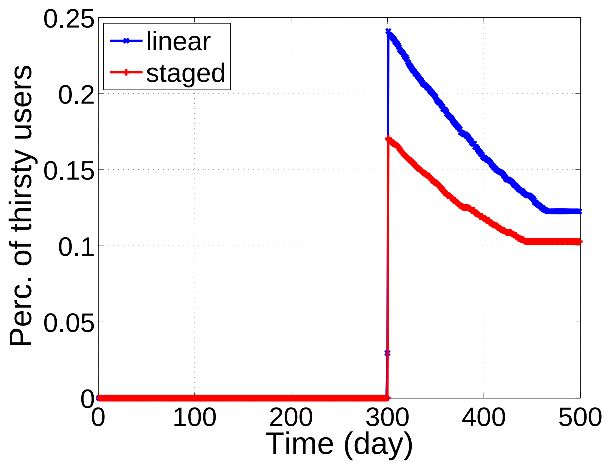

Figure 2.11: % of thirsty users in event-driven blocking.

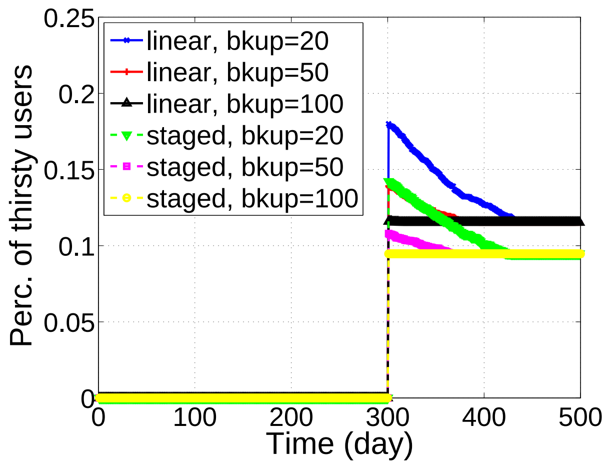

Figure 2.12: % of thirsty users with backup bridges in event-driven blocking.

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.

2.5.3 Comparison with Proximax

W now compare rBridge with Proximax [13]—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

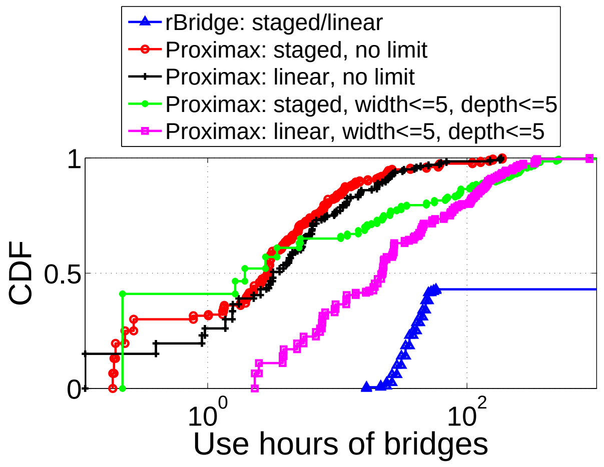

Figure 2.13: User-hours in comparison.

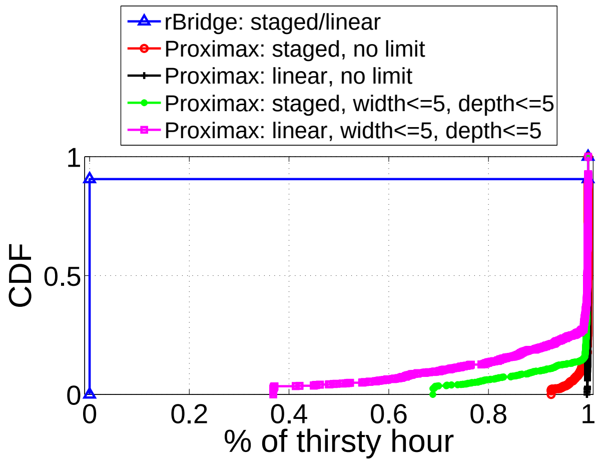

Figure 2.14: % of thirsty-hours in comparison.

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[2].

Figure 2.13 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

[2]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).

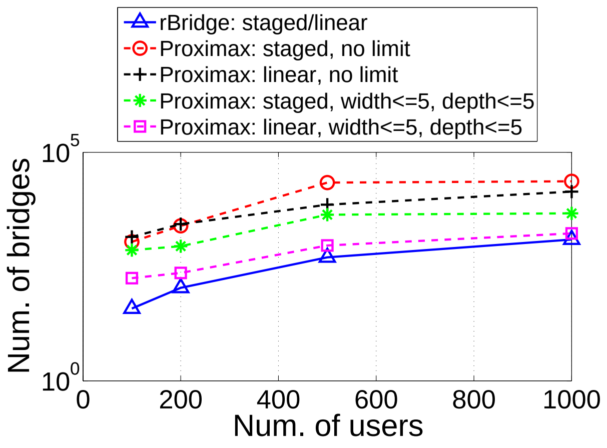

Figure 2.15: User base in comparison.

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 2.15 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 2.13, Figure 2.14, and Figure 2.15 also provide 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' 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.

CHAPTER 3

A PRIVACY-PRESERVING USER REPUTATION SYSTEM FOR TOR BRIDGE DISTRIBUTION

Tor is primarily designed to provide anonymous communication service; it uses 3 randomly selected relays to forward users' traffic. A key design philosophy of Tor is to avoid any entity knowing which relays are used by a particular user; for instance, although each user only needs 3 relays, he has to download all the relays' descriptors from one of the directory authorities to hide the selected relays from curious directory authorities.

To make it more difficult for adversaries to enumerate bridges, 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.

Nevertheless, the privacy issue in Tor bridge distribution has been unfortunately overlooked by all the existing schemes including those that have been deployed by Tor. They all assume that the bridge distributor is fully trusted and authorized to know which bridges are used by each user, which degrades the user anonymity.

In this chapter, we first present a new attack of de-anonymizing Tor users using bridge assignment information to demonstrate the importance of privacy preservation in Tor bridge distribution. We then propose a novel privacy-preserving Tor bridge distribution scheme based on rBridge [12] using several cryptographic primitives, and provide performance evaluation results as well as rigorous security proofs.

3.1 Motivation of Preserving Privacy in Tor Bridge Distribution: An Attack of De-anonymizing Tor Users using Bridge Assignment Information

3.1.1 Anonymity in Tor

Tor is a practical system aiming to provide low-latency anonymous communication for ordinary users by using onion encryption tunnels each constructed with three randomly picked relays. Tor aims to achieve a practical goal regarding protecting user anonymity. In particular, it aims to achieve "end-to-end " anonymity, i.e., preventing attackers from learning *both* the initiator (i.e., the user) and the destination (e.g., a website), as a real-world adversary is less interested in only learning whether a particular user is using Tor or whether a particular website is being viewed. This anonymity property is one of a set of more formally defined anonymity properties (e.g., sender and receiver anonymity) proposed by Ptzmann and Hansen [29]. In this work, we focus on analyzing end-to-end anonymity of Tor.

In our threat model, we assume the adversary can control a fixed set of malicious relays that comprise a ratio of f of all relays and can learn information about users' traffic only from these malicious relays (for clarity, we do not consider AS-level adversaries or malicious websites). This implies that the adversary can learn the destination of a connection of a particular user if and only if the exit relay is malicious. Likewise, the adversary can directly see the user's address if the guard (or the bridge) of the connection is malicious.

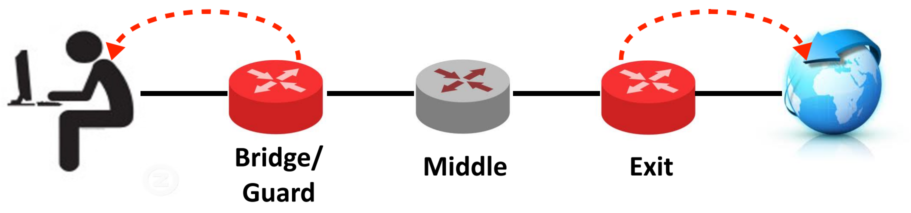

Figure 3.1: For a compromised circuit with malicious first- and last-hop relays, both the user's identity and the communication destination can be observed.

A widely used metric to evaluate Tor's anonymity is the percentage of "compromised" circuits, i.e., those having both malicious first- and last-hop relays. An illustration of compromised circuits is shown in Figure 3.1. The ratio of compromised circuits can be roughly estimated as f^2, if we do not consider relays' bandwidth in relay selection. For instance, when $f = 20\%$, about 4% of the established circuits will fail to preserve users' anonymity, i.e., the adversary will be able to link the websites a user is browsing to the user. However, as we shall show later, bridge distribution without privacy preservation opens up another way for adversaries to compromise users' anonymity, and this attack works even when none of the user's bridges/guards is malicious and thus substantially degrades the anonymity of Tor.

3.1.2 Fingerprinting Users using Bridge Assignment Information

Guards are first-hop relays of anonymous communication circuits, and bridges are a special type of guards, which are used by users in censored countries as stepping stones to circumvent censorship. In existing bridge distribution strategies adopted by Tor, the bridge distributor knows exactly what bridges are given to a particular user, and thus can use the bridge assignment information as a fingerprint to identify users. For instance, assuming that each user is given 3 bridges, if the adversary is able to learn that 3 bridges, say B_1, B_2 and B_3, belong to the same user, then the adversary who has access to the bridge assignment information (e.g., by colluding with the distributor) can easily narrow the anonymity set of the user down to a small subset of users who are given B_1, B_2 and B_3. As we can see in Figure 3.2 that when each bridge/guard is selected at uniform random, over 95% of bridge/guard sets are shared by fewer than 0.2% users, and over 70% of bridge/guard sets can be uniquely mapped to a single user. Figure 3.3 shows the results when the selection of bridge/guard is weighted using each bridge/guard's bandwidth (as the guard selection strategy currently adopted by Tor); in this case, about 60% of bridge/guard sets can uniquely identify their owners.

Therefore, besides compromising the first-hop relay of a user's anonymous circuit – which can be viewed as a direct way to learning the user's identity, the adversary can also indirectly connect an observed communication destination to the user by using bridge assignment information as a link joint. In particular, we present a long-term attack that leverages web cookie and

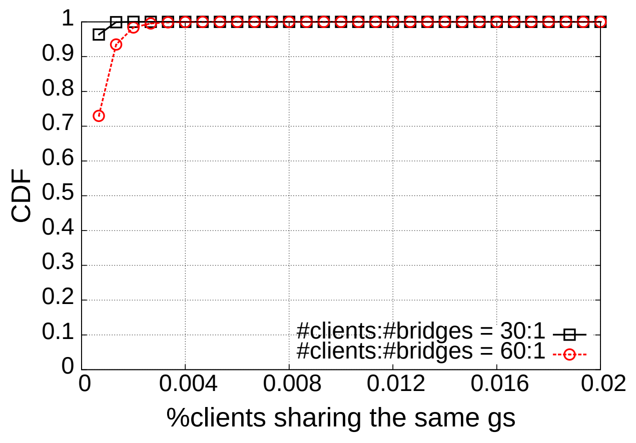

Figure 3.2: CDF of percentage of users who share a common bridge/guard set, when each bridge/guard is selected at uniform random ($\#users = 1500$).

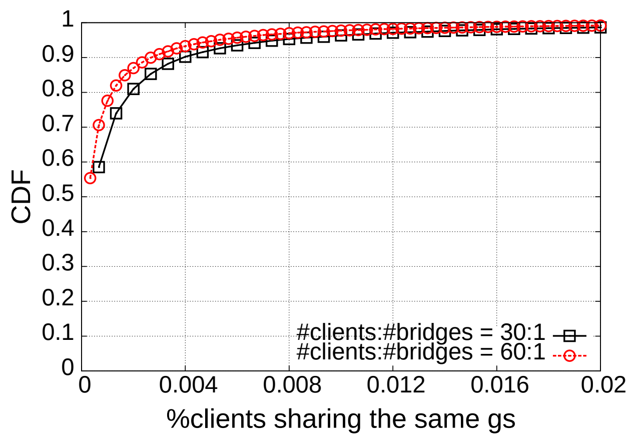

Figure 3.3: CDF of percentage of users who share a common bridge/guard set, when bridge/guard selection is weighted using bridge/guard's bandwidth ($\#users = 1500$).

bridge assignment information to de-anonymize users.

3.1.3 Grouping Bridges using Web Cookies

As we discussed above, bridge assignment information allows an adversary to link a particular bridge/guard set to its owner (i.e., the user) with high probability; thus, if the adversary is able to find out the bridge/guard set and link the user's visited websites to the bridge/guard set, she can finally link the websites to the user.

The key of this attack is to find out which bridges/guards belong to the same user (i.e., in the same bridge/guard set). As we know, when the exit relay is malicious, the adversary can see not only the communication destinations (e.g., the IP address of the website) but also all of the communication content, including web cookie, as long as the application layer protocol does not use encryption (such as HTTPS). Web cookie is a small piece of data sent from a website and stored in a user's web browser while a user is browsing a website, and when the user visits the same website again, the data stored in the cookie will be retrieved by the website to notify the website of the user's previous activity.[1] By keeping track of web cookies observed from malicious exit relays, the adversary can infer that multiple circuits that share the same cookie belong to the same user. For instance, in the example shown in Figure 3.4, the adversary can observe cookies from circuits having malicious exit relays (i.e., C_1, C_3, C_4 and C_5); out of them, C_1, C_3 and C_5 share a common cookie X, and thus they belong to the same user. Furthermore, if multiple web cookies are observed from a circuit, e.g., C_1 and C_3, and one of cookies is the target cookie (i.e., X), then we can learn that the other cookies also belong to the user and therefore can be added to the set of target cookies (e.g., the target cookie set in this example is $\{X, Y, Z\}$). As a result, we can infer that C_1, C_3, C_4, and C_5 belong to the target user, since they have at least one of the target cookies.

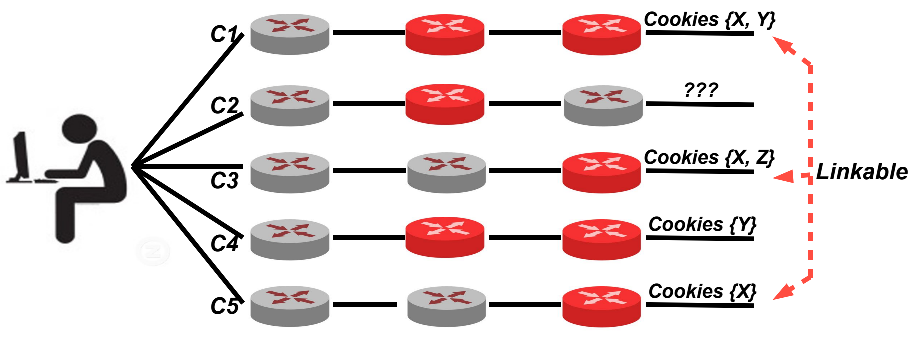

Figure 3.4: An example of linking circuits based on web cookies.

We note that this cookie-based circuit correlation can be carried out over

[1]We note that although the presented attack only works when web cookies can be observed by malicious exit relays, our attack strategy is generic and can be applied to cases when there are no direct observations of cookies, e.g., when users clean up web cookies after each use of Tor. This is because our attack only requires some application-level information that can link different web browsing transactions of the user; besides web cookies, such application-level information could be usernames, photos, affiliations, addresses, personal interests, etc.

a long period of time, since web cookies can typically live in the client's web browser for a long time. In addition, we are less interested in the case where the first-hop relay of a circuit is malicious, because in this case the adversary can always learn the user's identity due to direct connection no matter whether bridge assignment information is available. To demonstrate the importance of protecting bridge assignment information, we focus on the scenario when all of the bridges/guards of the target victim user are honest and show that even in this case an adversary with bridge assignment information is still able to de-anonymize the user.

For a circuit having a malicious middle relay (such as C_1, C_2, and C_4 in Figure 3.4), the bridge/guard of the circuit is observable. Therefore, if both the exit relay and the middle relay are malicious, the adversary can observe both the web cookies of visited websites and the bridge/guard, and associate them together. As a result, the adversary can group the observed bridges/guards of circuits that are linkable by target cookies. For instance, bridges/guards of C_1 and C_4 can be grouped together using the target cookies X and Y.

Although the probability of having both malicious middle and exit relays in a circuit is low, over a long time the adversary can have a fairly good chance to accumulate multiple circuits that share a common cookie and have observable bridges/guards. Furthermore, we note that getting a circuit with both malicious middle and exit relays is not the only way to link an observed cookie to an observed bridge/guard. We next show how to leverage timing information in circuit scheduling to find pairs of observed cookies and bridges/guards. This approach can expedite grouping bridges/guards of the target user.

3.1.4 Linking Circuits based on Timing of Circuit Scheduling

Since Tor aims to provide low-latency anonymous communication, it adopts some mechanisms to optimize performance. For example, the client software predicts the number and types of circuits that are likely to be needed and pre-emptively builds them to avoid the delay of run-time circuit establishment. In particular, throughout the online period, the client maintains two *usable* circuits at any time for each port that was seen in the past hour. By "usable", we refer to a circuit being able to accept new connections.[2] A *clean* circuit becomes *dirty* once a connection is relayed on it. A dirty circuit stays usable only for 10 minutes, and a clean circuit can stay usable for 1 hour. All of the established circuits are closed when the client goes offline.

Because of the circuit scheduling mechanism, the timing of building/closing circuits follows specific patterns. For instance, once a circuit becomes unusable (i.e., 1 hour after being built for a clean circuit, and 10 minutes after becoming dirty for a dirty circuit), it's highly likely that the client will build a new circuit to replace the expired circuit. Such timing information can help the adversary decide whether two observed circuits belong to the same user. For example, if one circuit is built right after another circuit becomes unusable, it is very likely that they are both built by the same client.

We now define the rules of linking circuits based on the timing of circuit scheduling. We let T_{build}, T_{close} and T_{dirty} denote the times when a circuit is built, closed, and becomes dirty, respectively. We assume that the adversary can infer T_{build}, T_{close}, and T_{dirty} of a circuit, as long as one of the relays of the circuit is malicious. We define ϵ as a very short period of time, e.g., a few seconds. We call two circuits C_i and C_j are *highly likely* to be *co-resident* if the one of following conditions is satisfied:

1. C_i is clean, and $C_j.T_{build} - (C_i.T_{build} + 1h) \leq \epsilon$
2. C_i is dirty, and $C_j.T_{build} - (C_i.T_{dirty} + 10m) \leq \epsilon$
3. $C_j.T_{close} - C_i.T_{close} \leq \epsilon$

Our previous analysis on the example shown in Figure 3.4 is that the adversary is able to link C_1, C_3, C_4, and C_5 together based on the common cookie X, but only knows that the bridges/guards of C_1 and C_4 belong to the target user because the first hops of C_3 and C_5 are not observed (in other words, the cookies observed from C_3 and C_5 cannot help us infer the user's bridge/guard set). Whereas, the approach to linking circuits based on timing of circuit scheduling allows us to make use of circuits (like C_3) that do not

[2]For a connection request made by an upper-level application, the client first tries to assign it to an existing circuit that can support the connection, i.e., the state of the circuit is either clean or dirty & usable and the exit policies of its exit relay can support the port of the connection. If there exist multiple suitable existing circuits, the client prefers a dirty & usable circuit to a clean circuit; if candidate circuits are all clean, a more recently built circuit is preferred; if candidate circuits are all dirty, a circuit that became dirty more recently is preferred.

have directly observed bridges/guards. For example, if there exists a circuit that has an observable first hop (say C_2 in Figure 3.4) and is highly likely co-resident to C_3 according to the three conditions presented before, we can combine the observations from C_2 and C_3 (i.e., C_2's bridge/guard and C_3's cookie X) together and use them to infer the target user's bridge/guard set. In this case, the adversary can infer that with high probability the observed bridges/guards of C_1, C_2 and C_4 belong to the target user.

3.1.5 An Attack of De-anonymizing Users using Bridge Assignment Information

We now present the attack strategy to de-anonymize users using bridge assignment information. First of all, the adversary records information of each observed circuit, including T_{build}, T_{close}, T_{dirty}, and observed relays of the circuit; if the observed circuit has a malicious exit relay, the adversary also records the observed cookies and communication destinations. The adversary selects an interested target user, who is the owner of one of the observed cookies (say X) provided by a target website, and aims to find out the real identity of the target user. As we mentioned before, we are only interested in de-anonymizing users who have no malicious bridges/guards, which is infeasible without using bridge assignment information. Note that once the target user is de-anonymized, the adversary learns not only this user's browsing histories on the website that creates the initial target cookie X, but also his visiting records on many other websites, since the adversary can accumulate cookies of the user over time.

The adversary keeps a list of candidate bridges/guards of this target user with each entry on the list containing the ID of a bridge/guard that has been discovered at least once using the two approaches we have described, i.e., satisfying one of the two conditions below:

1. The bridge/guard is on a circuit C that has both malicious middle and exit relays, and at least one of the cookies observed from C is a target cookie (and the rest cookies are added to the target cookie set).

2. The bridge/guard is on a circuit C that has a malicious relay and is highly likely co-resident to another circuit C' that has a malicious exit relay, and at least one of the cookies observed from C' is a target cookie.

Each entry also contains a counter, which records how many times this bridge/guard has been discovered; for a candidate bridge/guard that meets Condition 1 once, its counter value is fixed as the maximum value because we can be sure that this bridge/guard belongs to the target user. The list is updated as the adversary collects more observations over time. In the end, the adversary outputs the three candidate bridges/guards with the highest counter values (assuming each user is given three bridges/guards) as the bridge/guard set of the target user, and then she can look up the mapping tables between bridge/guard sets and users to identify the target user or narrow down the anonymity set of the target user to a small number of users.

3.1.6 Evaluation

To evaluate the effectiveness of this attack, we build a simulator by simulating the circuit scheduling and path selection mechanisms of the client-side software of Tor. We run our simulation on a smaller sized Tor network, by randomly selecting a subset of all Tor relays while using their original relay descriptors (including bandwidth, tags, families, and etc.) downloaded from Tor website[3]. We randomly pick 150 relays and use 1500 users, and run the simulation for 100 days. We employ bandwidth weighted bridge/guard selection. Each user is configured to be online for T hours every day, where T is uniformly random in $[1h, 3h]$, and during each online period, each user makes connections following a poisson process with mean value 10 minutes. According to the study in [30], we model the dwell time of each web visit using Weibull distribution with the parameters suggested in [30] shape $k = 1$ and scale $\lambda = 70s$. We use the top 100 most popular websites published by Google doubleclick[4] as connection destinations, and set the initial cookie as the one created by the most popular website.

We can see from Figure 3.5 that the number of events of discovering candidate bridges/guards grows almost in linear with time, and as shown in Figure 3.6, the number target cookies obtained by the adversary also increases accordingly. Figure 3.7 shows that the number of entries on the candidate list increases with time, but its growing rate is sublinear because the same

[3]`https://metrics.torproject.org/data.html`

[4]`http://www.google.com/adplanner/static/top1000/`

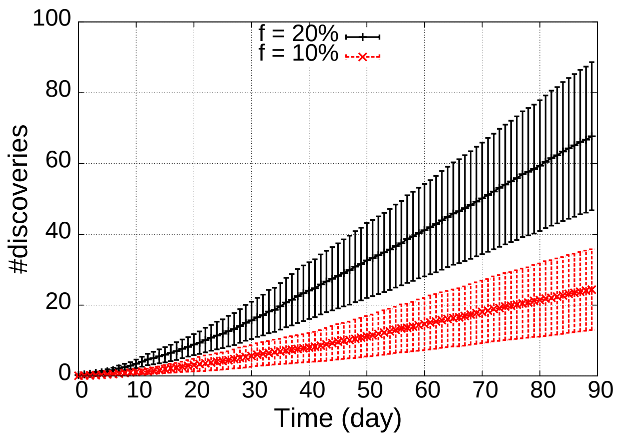

Figure 3.5: Total number of events of a candidate bridge/guard being discovered with 95% confidence interval.

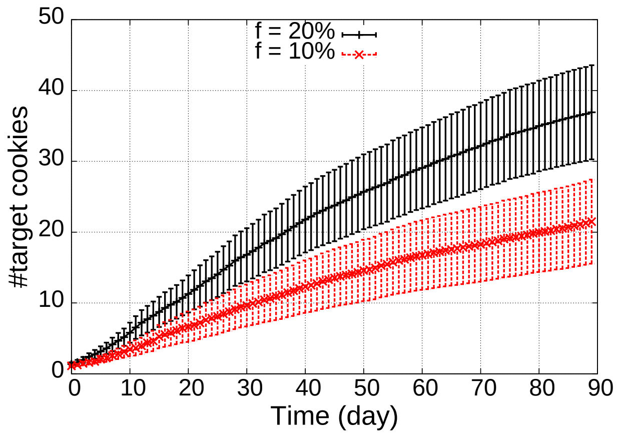

Figure 3.6: Number of target cookies with 95% confidence interval.

bridge/guard may be discovered for multiple times.

We evaluate the accuracy of this de-anonymizing attack by computing the probability that the true bridge/guard set is contained in the top few entries of the candidate list. We can see from Figure 3.8 that when 20% of relays are malicious, after 90 days of observation, the top 3 entries on the candidate lists are the true bridge/guard set with probability of over 60%; with 10% malicious relays, this probability is about 30%. Considering that each bridge/bridge can be typically mapped to a very small number of users as shown in Figure **??**, there is a fairly good chance for the adversary to de-anonymize the target user.

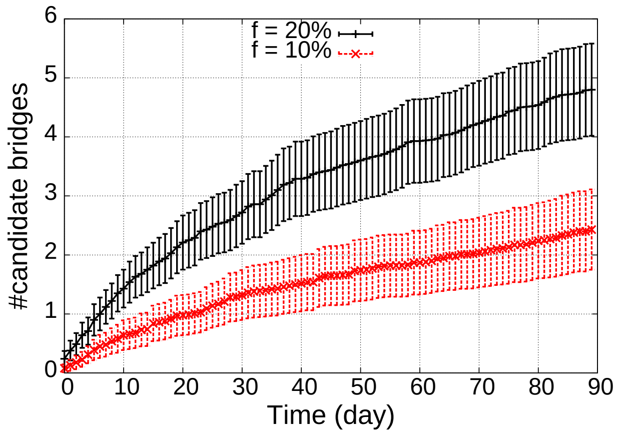

Figure 3.7: Number of discovered candidate bridges/guards with 95% confidence interval.

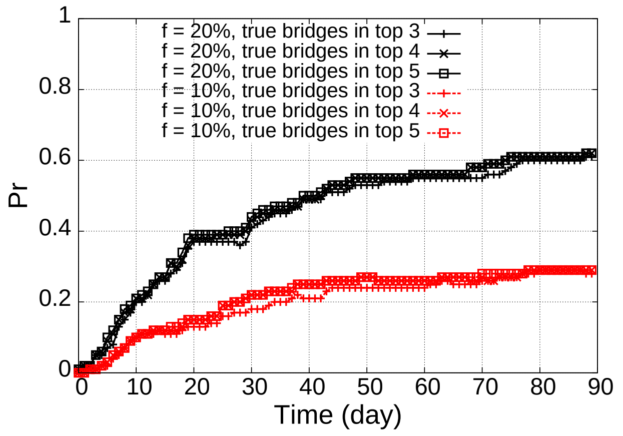

Figure 3.8: The probability that the target user's bridge/guard set is contained in the top 3, 4 or 5 entries of the candidate list.

3.2 Related Work on Anonymous Authentication and Anonymous Reputation Systems

Researchers have put forward several designs for anonymous authentication and anonymous reputation systems [31–34] that are similar to what we are seeking. Au et al. [34] 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 [31, 33] extended this scheme to allow revocation without trusted third parties. Later, Au et al. [32] 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.

3.3 Challenges and Requirements

Our goal is to provide perfect privacy preservation for Tor users in rBridge by hiding bridge assignment information from any parties including the bridge distributor.

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 $\mathtt{U}$'s credential, such as $\{B_i, \tau_i, \phi_i\}$, should be written and updated by $\mathtt{U}$, rather than by $\mathtt{D}$. 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 [31–34], 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.

3.4 Cryptographic Building Blocks

We next introduce the cryptographic primitives used in the construction of the privacy-preserving rBridge scheme.

3.4.1 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 [35] to construct rBridge due to its simplicity of implementation.

To make it hard for corrupt users to collaboratively enumerate bridges, we let $\mathtt{D}$ (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.

3.4.2 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 [36] 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, $\texttt{U}$ can compute a commitment of Φ, i.e., $(C_\Phi, O_\Phi) = \mathsf{CMT}(\Phi)$, and put C_Φ in his credential. To prevent $\texttt{U}$ from manipulating his credential (e.g., increasing Φ), we let $\texttt{D}$ sign C_Φ using his private key $SK_{\texttt{D}}$, i.e., $\sigma_\Phi = \texttt{Sign}(SK_{\texttt{D}}, C_\Phi)$ and tag the signature σ_Φ to the credential, and $\texttt{U}$ needs to prove to $\texttt{D}$ that both the commitment and the signature are valid. To prevent $\texttt{D}$ from linking $\texttt{U}$'s transactions based on the values of commitments and signatures, we need another cryptographic primitive—zero-knowledge proof.

3.4.3 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 [37]. We follow the notation introduced by Camenisch and Stadler [38], 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, $\texttt{U}$ can construct the following proof:

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

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. [34] because of its compatibility with zero-knowledge proofs.

3.5 Scheme

3.5.1 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.

3.5.2 Registration

A new user $\mathtt{U}$ first presents an invitation ticket to $\mathtt{D}$. 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 $\mathtt{D}$; the ticket is verifiable to $\mathtt{D}$, and cannot be forged by anyone else. Then $\mathtt{U}$ runs $\binom{m}{1}$-OT with $\mathtt{D}$ to get k initial bridges $B_1, \cdots, B_k$[5]. After that, $\mathtt{U}$ 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, $\mathtt{U}$ 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 $\mathtt{D}$. (wherein $\mathsf{OWF}(\cdot)$ is simply a discrete-log based one-way function.)

Note that since $\mathtt{D}$ does not know the bridges received by $\mathtt{U}$ (i.e., $\{B_i\}_{i=1}^k$), $\mathtt{U}$ couSectiold 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, $\mathtt{D}$ needs to verify that the bridges to be written in the credential are actually the bridges that $\mathtt{U}$ received in OT. To achieve this, we let $\mathtt{D}$ (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 $\mathtt{U}$, use $SK_{\mathtt{D}}^o$ to sign each available bridge B_j, and tag the signature σ_j^o to B_j. After OT, $\mathtt{U}$ 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, $\mathtt{U}$ 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 $\mathtt{U}$ provide $\mathtt{D}$ a random nonce non_j for each available bridge B_j; non_j is included in the computation of σ_j^o to prevent $\mathtt{D}$ from later finding out which bridges are retrieved by $\mathtt{U}$ using these signatures (refer to Appendix B for details).

To get the credential, $\mathtt{U}$ constructs the following proof:

[5] $\mathtt{U}$ 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 $k = 3$).

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

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 $(1 \leq i \leq k)$, 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 [26], in order to count the ever connected users.)

3.5.3 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 [34] (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:

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

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 $\tilde{\sigma}_u$.

3.5.4 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:

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

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 $\kappa_\Phi \notin \mathit{elist}_\Phi$, $\kappa_b \notin \mathit{elist}_{B_b}$, and π_3 (where elist_{B_b} denotes the list of used indicators of B_b), D adds κ_Φ to elist_Φ 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:

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

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 $\tilde{\sigma}_b$.

3.5.5 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.

3.6 Security Analysis

In this section, we discuss potential attacks on the privacy-preserving rBridge scheme. We present rigorous security proof in Section 3.8.

3.6.1 Curious Bridge Distributor

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.

3.6.2 Malicious Users

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

Table 3.1: Performance (averaged over 100 runs)

Operation	Comp. (s) U	Comp. (s) D	Comm. (KB)
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

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.

3.7 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 [35] 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

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

[7]`http://gmplib.org/`

12 GB RAM, running Ubuntu 10.04. Table 3.1 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. We believe these occasional computations can be handled by both `U` and `D` with fairly strong computational power. In addition, we note that the user does not need to idly wait for the server completing the cryptographic operations, because as long as the user has an unblocked bridge he can proceed to do web browsing after submitting the requests to the bridge distributor.

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 [39].

3.8 Cryptographic Proof

3.8.1 Security Model

The protocol of rBridge involves two parties: the bridge distributor `D`, and the user `U`. A corrupt `U` aims to deceive `D` with false credential information, such as manipulated credit balance, and a corrupt `D` aims to learn information about `U`'s assigned bridges. We consider either of `U` and `D` (not both) could be controlled by the adversary. A compromised party can perfectly collude with the adversary by sharing information and executing instructions from the adversary. For presentation simplicity, we consider the adversary and the compromised party as one joint entity referred to as the "adversary".

We use the real/ideal world paradigm to prove security. In the real world, on inputs (or instructions) from the environment $\mathcal{Z}$, the parties interact according to the protocol ψ in the presence of a real adversary $\mathcal{A}$, and the outputs of the parties are handed to $\mathcal{Z}$. In the ideal world, there is a "trusted" ideal process $\mathcal{F}_\psi$ to carry out the desired task in the presence of an ideal adversary $\mathcal{S}$, namely, the parties interact with each other through $\mathcal{F}_\psi$ that provides the desired computation results, and hand their outputs to $\mathcal{Z}$. In addition, the adversary ($\mathcal{A}$ or $\mathcal{S}$) can interact with the environment $\mathcal{Z}$ throughout the course of the protocol and influence $\mathcal{Z}$ generating inputs for the parties. The protocol ψ is called *secure* if it "emulates" the ideal functionality $\mathcal{F}_\psi$. More formally, we have:

Definition 1. *The protocol ψ is secure if for any adversary $\mathcal{A}$ there exists an ideal adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and ψ or with $\mathcal{S}$ and the ideal process $\mathcal{F}_\psi$. We call ψ securely realizes $\mathcal{F}_\psi$.*

The ideal functionality $\mathcal{F}_\psi$ of rBridge is defined as below.

- At the initialization, $\mathcal{F}_\psi$ gets a copy of the list of available bridges $\{B_j\}_{j=1}^m$ from $\mathtt{D}$.

- On input $(\mathsf{register}, tk, \mathtt{U})$ from $\mathtt{U}$, where tk is an invitation ticket, $\mathcal{F}_\psi$ forwards it to $\mathtt{D}$. $\mathtt{D}$ as in the protocol ψ checks the freshness and validity of tk. If the ticket is verified, $\mathtt{D}$ sets $b = 1$ and adds tk to $elist_{tk}$; otherwise, b is set as 0. $\mathtt{D}$ then sends b to $\mathcal{F}_\psi$. If $b = 1$, $\mathcal{F}_\psi$ randomly picks k initial bridges $\{B_i\}_{i=1}^k$ from the available bridges, creates an entry $\langle \mathtt{U}, \Phi, \{B_i, \tau_i, \phi_i\}_{i=1}^k \rangle$ in the user database, where $\Phi = 0$, $\tau_i = T_{cur}$, and $\phi_i = 0$, increments the counter $count(B_i)$, $i \in [1, k]$, which records the number of users assigned to B_i, and finally sends $\{B_i\}_{i=1}^k$ to $\mathtt{U}$. Otherwise, $\mathcal{F}_\psi$ sends $\perp$ to $\mathtt{U}$.[8]

- On input $(\mathsf{update}, B_u, \mathtt{U})$ from $\mathtt{U}$, $\mathcal{F}_\psi$ first checks whether $\mathtt{U}$ has a record in the user database, B_u was assigned to $\mathtt{U}$, and B_u is not on the list of blocked bridges. If not, $\mathcal{F}_\psi$ aborts by returning $\perp$ to $\mathtt{U}$. Otherwise, $\mathcal{F}_\psi$ computes $\tilde{\phi}_u = \mathsf{Credit}(T_{cur} - \tau_u)$, $\tilde{\Phi} = \Phi + \tilde{\phi}_u - \phi_u$, and updates ϕ_u, Φ with $\tilde{\phi}_u$, $\tilde{\Phi}$ in $\mathtt{U}$'s record, and finally sends $\top$ to $\mathtt{U}$ and (update) to $\mathtt{D}$, respectively.

[8]Once $count(B_i)$ reaches the capacity upper bound g, $\mathcal{F}_\psi$ removes B_i from the list of bridges for distribution. $\mathcal{F}_\psi$ periodically informs $\mathtt{D}$ what bridges are fully occupied so that $\mathtt{D}$ can make plans for new bridge recruitment.

- On input (query, B_b), $\mathcal{F}_\psi$ checks whether B_b is on the list of blocked bridges. If so, $\mathcal{F}_\psi$ returns B_b's blocking time β_b; otherwise, $\mathcal{F}_\psi$ returns $\perp$.

- On input $(\mathsf{newbridge}, B_b, \mathtt{U})$ from $\mathtt{U}$, $\mathcal{F}_\psi$ first checks whether $\mathtt{U}$ has a record in the user database, B_b was assigned to $\mathtt{U}$ and is on the list of blocked bridges, and $\Phi + \tilde{\phi}_b - \phi_b > \phi^-$, where $\tilde{\phi}_b = \mathsf{Credit}(\beta_b - \tau_b)$, β_b is the time when B_b got blocked, and ϕ^- is the amount of credits to pay for a new bridge. If not, $\mathcal{F}_\psi$ aborts by returning $\perp$ to $\mathtt{U}$. Otherwise, $\mathcal{F}_\psi$ randomly picks an available bridge $\tilde{B}_b$, updates $\mathtt{U}$'s record with $\tilde{\Phi}, \tilde{B}_b, \tilde{\tau}_b, \tilde{\phi}_b$, where $\tilde{\Phi} = \Phi + \tilde{\phi}_b - \phi_b - \phi^-$, $\tilde{\tau}_b = T_{cur}$, $\tilde{\phi}_b = 0$, and finally sends $\tilde{B}_b$ to $\mathtt{U}$ and $(\mathsf{newbridge}, B_b)$ to $\mathtt{D}$, respectively.

- On input $(\mathsf{ticket}, \mathtt{U})$ from $\mathtt{U}$, $\mathcal{F}_\psi$ first checks whether $\mathtt{U}$ has a record in the user database, his credit balance Φ satisfies $\Phi > \Phi_\theta$, and his last time applying for a ticket is at least ω days ago, i.e., $T_{cur} - \omega > \omega_\theta$. If not, $\mathcal{F}_\psi$ aborts by returning $\perp$ to $\mathtt{U}$. Otherwise, $\mathcal{F}_\psi$ updates ω with T_{cur}, and then flips a biased coin b to decide whether to grant a ticket to $\mathtt{U}$: if $b = 1$, $\mathcal{F}_\psi$ generates a ticket tk and sends it to $\mathtt{U}$ and sends (ticket) to $\mathtt{D}$; otherwise, $\mathcal{F}_\psi$ sends $\perp$ to $\mathtt{U}$.

3.8.2 Proof

Theorem 1. *The protocol ψ of rBridge securely realizes the ideal functionality $\mathcal{F}_\psi$ of rBridge.*

The security proof is based on indistinguishability of the environment $\mathcal{Z}$'s views on the real adversary $\mathcal{A}$ and the ideal adversary $\mathcal{S}$. In order to prove the indistinguishability, we construct an ideal adversary $\mathcal{S}$, which invokes a copy of the real adversary $\mathcal{A}$ that interacts with a "dummy" party and interacts with $\mathcal{F}_\psi$ and the environment $\mathcal{Z}$ in such a way that the probability distributions of $\mathcal{S}$'s outputs and $\mathcal{A}$'s outputs are computationally indistinguishable.

We define a series of games between the real world and the ideal world, $Game_0, \cdots, Game_n$, where $Game_0$ corresponds to the real world and $Game_n$ corresponds to the ideal world. We define $Pr(Game_i)$ as the probability that $\mathcal{Z}$ distinguishes between the distribution of outputs of $Game_i$ and that of the real world, and $|Pr(Game_{i+1}) - Pr(Game_i)|$ denotes the probability that $\mathcal{Z}$ distinguishes between the distribution of outputs in two consecutive games. By definition, $Pr(Game_0) = 0$. We need prove $Pr(Game_n)$ is negligibly small. Since

$$\begin{aligned} Pr(Game_n) &= \sum_{i=0}^{n-1} Pr(Game_{i+1}) - Pr(Game_i) \\ &\leq \sum_{i=0}^{n-1} |Pr(Game_{i+1}) - Pr(Game_i)| \end{aligned}$$

it is sufficient to prove that each of $|Pr(Game_{i+1}) - Pr(Game_i)|$, $i \in [0, n-1]$, is negligibly small.

To prove the theorem, we divide it into several claims according to different scenarios and prove them separately.

When Bridge Distributor Is Corrupt

Claim 1. *rBridge is secure against corrupt users under the receiver's privacy property of the* OT *protocol, the hiding property of the commitment scheme, the privacy property of the blind signature scheme, and the zero-knowledgeness property of the zero-knowledge proof scheme.*

Proof. We show by constructing a series of games in such a way that the environment $\mathcal{Z}$ cannot distinguish two consecutive games with non-negligible probability.

$Game_0$: This game corresponds to the execution of the protocol ψ in the real world with a corrupt $\mathtt{D}$ and a honest $\mathtt{U}$. Thus, $Pr(Game_0) = 0$.

$Game_1$: This game proceeds as $Game_0$, except that it aborts if $\mathcal{A}$ breaks the OT protocol and learns the bridge chosen by $\mathtt{U}$ in OT. Under the receiver's privacy property of OT, $|Pr(Game_1) - Pr(Game_0)| = \nu_1(l)$, where l is the security parameter, and ν_1 is a function whose values are negligibly small.

$Game_2$: This game proceeds as $Game_1$, except that in the execution of *Updating credit balance*, κ_Φ is replaced with a random value κ^*_Φ, $\tilde{C}_\Phi$ is replaced with a commitment to $(\tilde{\Phi}^*, x^*)$, $\tilde{C}_u$ is replaced with a commitment to $(B^*_u, \tau^*_u, \tilde{\phi}^*_u, x^*)$, and π_2 is replaced with a simulated proof [9]. Since κ_Φ is calcu-

[9]Since the proof follows the standard discrete-logarithm-based Σ-protocol, $\mathcal{S}$ can sim-

lated as z^{s_Φ}, where $s_\Phi = s'_\Phi + s''_\Phi$ and s'_Φ is randomly picked by $\mathtt{U}$, the distribution of κ^*_Φ is indistinguishable to $\mathtt{D}$ (i.e., $\mathcal{A}$) from that of κ_Φ. Under the hiding property of the commitment protocol and the zero-knowledgeness property of the zero-knowledge proof scheme, $|Pr(Game_2) - Pr(Game_1) = \nu_2(l)$.

$Game_3$: This game proceeds as $Game_2$, except that in the execution of *Getting a new bridge*, κ_Φ, κ_b are replaced with random values, $\tilde{C}_\Phi$ is replaced with a commitment to $(\tilde{\Phi}^*, x^*)$, $\tilde{C}_b$ is replaced with a commitment to $(\tilde{B}^*_b, \tilde{\tau}^*_b, \tilde{\phi}^*_b, x^*)$, and π_3, π_4 are replaced with simulated proofs. Under the hiding property of the commitment protocol and the zero-knowledgeness of the zero-knowledge proof, $|Pr(Game_3) - Pr(Game_2)| = \nu_3(l)$.

$Game_4$: This game proceeds as $Game_3$, except that in the execution of *Inviting new users*, $\kappa_\Phi, \kappa_\omega$, are replaced with random values, $\tilde{C}_\Phi$ is replaced with a commitment to (Φ^*, x^*), $\tilde{C}_\omega$ is replaced with a commitment to (ω^*, x^*), and π_5 is replaced with a simulated proof. Under the hiding property of the commitment protocol and the zero-knowledgeness of the zero-knowledge proof, $|Pr(Game_4) - Pr(Game_3)| = \nu_4(l)$.

$Game_5$ (the ideal world): This game proceeds as $Game_4$ except that the actual participants are $\mathtt{U}$ and $\mathcal{S}$, who interact with each other through $\mathcal{F}_\psi$, and $\mathcal{S}$ has black-box access to $\mathcal{A}$ by simulating a dummy user following the protocol ψ. In more details, $Game_5$ works as follows.

- *Registration*: Upon receiving $(\mathsf{register}, tk, \mathtt{U})$ from $\mathtt{U}$, $\mathcal{F}_\psi$ forwards it to $\mathcal{S}$, who further hands it to $\mathcal{A}$. $\mathcal{S}$ receives the verification result b from $\mathcal{A}$ and forwards it to $\mathcal{F}_\psi$. In addition, $\mathcal{S}$ simulates a dummy user to perform OT with $\mathcal{A}$, receiving $\{B^*_i\}^k_{i=1}$ and a credential with $\{B^*_i\}^k_{i=1}$ and a randomly picked secret x^* on it.

- *Updating credit balance*: Upon receiving (update) from $\mathcal{F}_\psi$, $\mathcal{S}$ simulates a dummy user as described in $Game_2$ to interact with $\mathcal{Z}$.

- *Getting a new bridge*: Upon receiving $(\mathsf{newbridge}, B_b)$ from $\mathcal{F}_\psi$, $\mathcal{S}$ first hands B_b to $\mathcal{A}$ who replies with β_b. $\mathcal{S}$ then simulates a dummy user as described in $Game_3$ to interact with $\mathcal{Z}$.

ulate this Σ-protocol in the two standard ways: (1) rewind the adversary (interactive proof) or (2) use its control over the random oracle (non-interactive proof). To prevent any rewinding difficulties, this protocol should be executed sequentially.

- *Inviting new users*: Upon receiving $(\textsf{ticket})$ from $\mathcal{F}_\psi$, $\mathcal{S}$ simulates a dummy user as described in $Game_4$ to interact with $\mathcal{Z}$.

The distribution produced in $Game_5$ is identical to that of $Game_4$, i.e., $|Pr(Game_5) - Pr(Game_4)| = 0$. By summation, we have $Pr(Game_5) \leq \nu(l)$, where $\nu(l) = \nu_1(l) + \nu_2(l) + \nu_3(l) + \nu_4(l)$.

□

When User Is Corrupt

Claim 2. *rBridge is secure against a corrupt bridge distributor under the collision-resistant property of the hash function, the sender's privacy property of the* $\textsf{OT}$ *protocol, the unforgeability property of the blind signature scheme, and the binding property of the commitment scheme.*

Proof. We construct a series of games between the real world and the ideal world.

$Game_0$: This game corresponds to the execution of the protocol ψ in the real world with a corrupt $\texttt{U}$ and a honest $\texttt{D}$. Thus, $Pr(Game_0) = 0$.

$Game_1$: This game proceeds as $Game_0$, except that the public key $PK_{\texttt{D}}$ of $\texttt{D}$ is replaced with $PK^*_{\texttt{D}}$, which is obtained using the same key generation algorithm. Since $PK^*_{\texttt{D}}$ has the same distribution as $PK_{\texttt{D}}$, $|Pr(Game_1) - Pr(Game_0)| = 0$.

$Game_2$: This game proceeds as $Game_1$, except that it aborts if $\mathcal{A}$ generates a valid ticket that has not been given to $\mathcal{A}$ without knowing the secret $scrt_{\texttt{D}}$ of $\texttt{D}$. The probability that $Game_2$ aborts is bounded by $\nu_1(l)$ due to the collision-resistant property of the employed hash function. Therefore, $|Pr(Game_2) - Pr(Game_1)| = \nu_1(l)$.

$Game_3$: This game proceeds as $Game_2$, except that it aborts if $\mathcal{A}$ obtains more than one bridge in the $\binom{m}{1}$-$\textsf{OT}$ protocol. Under the sender's privacy property of $\textsf{OT}$, $|Pr(Game_3) - Pr(Game_2)| = \nu_2(l)$.

$Game_4$: This game proceeds as $Game_3$, except that it aborts if $\mathcal{A}$ generates a valid one-time blind signature on a bridge that is not the one received in $\textsf{OT}$ (e.g., σ_i^{o*} on B_i^*). Under the unforgeability property of the blind signature, $|Pr(Game_4) - Pr(Game_3)| = \nu_3(l)$.

$Game_5$: This game proceeds as $Game_4$, except that it aborts if $\mathcal{A}$ finds another indicator κ_x^*, s.t., $\kappa_x^* = \mathsf{OWF}(x)$ and $\kappa_x^* \neq \kappa_x$. Since κ_x is uniquely calculated as z^x, $|Pr(Game_5) - Pr(Game_4)| = 0$.

$Game_6$: This game proceeds as $Game_5$, except that it aborts if any of the following cases takes place in *Registration*: (i) $(O_i, B_i, \tau_i, \phi_i, x)$ is a correct opening of C_i, but $\tau_i \neq T_{cur}$ or $\phi_i \neq 0$; (ii) (O_Φ, Φ, x) is a correct opening of C_Φ, but $\Phi \neq 0$; (iii) (O_ω, ω, x) is a correct opening of C_ω, but $\omega \neq T_{cur}$. Under the binding property of the commitment protocol, $|Pr(Game_6) - Pr(Game_5)| = \nu_4(l)$.

$Game_7$: This game proceeds as $Game_6$, except that it aborts if $\mathcal{A}$ generates a valid blind signature on a commitment (e.g., σ_Φ^* on C_Φ^* or σ_u^* on C_u^*) independently of $\mathtt{D}$. Under the unforgeability property of the blind signature, $|Pr(Game_7) - Pr(Game_6)| = \nu_5(l)$.

$Game_8$: This game proceeds as $Game_7$, except that it aborts if $\mathcal{A}$ finds another indicator $\kappa_\Diamond^*$, $\Diamond \in \{\Phi, b, \omega\}$, s.t., $\kappa_\Diamond^* = \mathsf{Indic}(\sigma_\Diamond)$ and $\kappa_\Diamond^* \neq \kappa_\Diamond$. Since $\kappa_\Diamond$ is uniquely calculated as $z^{s_\Diamond}$, where $\sigma_\Diamond = (A_\Diamond, e_\Diamond, s_\Diamond)$, $|Pr(Game_8) - Pr(Game_7)| = 0$.

$Game_9$: This game proceeds as $Game_8$, except that it aborts if in *Updating credit balance* when $(O_u, B_u, \tau_u, \phi_u, x)$ is a correct opening of C_u, (O_Φ, Φ, x) is a correct opening of C_Φ, $(\tilde{O}_u, B_u, \tau_u, \tilde{\phi}_u, x)$ is a correct opening of $\tilde{C}_u$, and $(\tilde{O}_\Phi, \tilde{\Phi}, x)$ is a correct opening of $\tilde{C}_\Phi$, we have $\tilde{\phi}_u \neq \mathsf{Credit}(T_{cur} - \tau_u)$ or $\tilde{\Phi} \neq \Phi + \tilde{\phi}_u - \phi_u$. Under the binding property of the commitment protocol, $|Pr(Game_9) - Pr(Game_8)| = \nu_6(l)$.

$Game_{10}$: This game proceeds as $Game_9$, except that (x, B_u) are extracted from the zero-knowledge proof π_2 [10]. This fails with negligible probability, $|Pr(Game_{10}) - Pr(Game_9)| = \nu_7(l)$.

$Game_{11}$: This game proceeds as $Game_{10}$, except that it aborts if the following condition happens for the proof π_3 in *Getting a new bridge*: $(O_b, B_b, \tau_b, \phi_b, x)$ is a correct opening of C_b, (O_Φ, Φ, x) is a correct opening of C_Φ, and $(\tilde{O}_\Phi, \tilde{\Phi}, x)$ is a correct opening of $\tilde{C}_\Phi$, but $\tilde{\phi}_b \neq \mathsf{Credit}(T_{cur} - \tau_b)$, $\tilde{\Phi} \neq \Phi + \tilde{\phi}_b - \phi_b - \phi^-$, or $\tilde{\Phi} \leq 0$. Under the binding property of the commitment protocol,

[10]A simulator with black-box access to the adversary can extract the values of hidden variables in a zero-knowledge proof by rewinding the adversary, as long as the the protocol is executed sequentially.

$|Pr(Game_{11}) - Pr(Game_{10})| = \nu_8(l)$.

$Game_{12}$: This game proceeds as $Game_{11}$, except that it aborts if the following condition happens for the proof π_4 in *Getting a new bridge*: $(\tilde{O}_\Phi, \tilde{\Phi}, x)$ is a correct opening of $\tilde{C}_\Phi$ and $(\tilde{O}_b, \tilde{B}_b, \tilde{\tau}_b, \tilde{\phi}_b, x)$ is a correct opening of $\tilde{C}_b$, but $\tilde{\tau}_b \neq T_{cur}$ or $\tilde{\phi}_b \neq 0$. Under the binding property of the commitment protocol, $|Pr(Game_{12}) - Pr(Game_{11})| = \nu_9(l)$.

$Game_{13}$: This game proceeds as $Game_{12}$, except that x is extracted from the zero-knowledge proof π_3. This fails with negligible probability, $|Pr(Game_{13}) - Pr(Game_{12})| = \nu_{10}(l)$.

$Game_{14}$: This game proceeds as $Game_{13}$, except that it aborts if in *Inviting new users* when (O_Φ, Φ, x) is a correct opening of C_Φ, (O_ω, ω, x) is a correct opening of C_ω, $(\tilde{O}_\Phi, \tilde{\Phi}, x)$ is a correct opening of $\tilde{C}_\Phi$, and $(\tilde{O}_\omega, \tilde{\omega}, x)$ is a correct opening of $\tilde{C}_\omega$, we have $\Phi \leq \Phi_\theta$, $T_{cur} - \omega \leq \omega_\theta$ or $\tilde{\omega} \neq T_{cur}$. Under the binding property of the commitment protocol, $|Pr(Game_{14}) - Pr(Game_{13})| = \nu_{11}(l)$.

$Game_{15}$: This game proceeds as $Game_{14}$, except that x is extracted from the zero-knowledge proof π_5. This fails with negligible probability, $|Pr(Game_{15}) - Pr(Game_{14})| = \nu_{12}(l)$.

$Game_{16}$ (the ideal world): This game proceeds as $Game_{15}$ except the actual participants are $\mathtt{D}$ and $\mathcal{S}$, who interact with each other through $\mathcal{F}_\psi$, and $\mathcal{S}$ has black-box access to $\mathcal{A}$ by simulating a dummy bridge distributor following the protocol ψ. In more details, $Game_{16}$ works as follows.

- *Registration*: Upon receiving tk from $\mathcal{A}$, $\mathcal{S}$ sends $(\mathsf{register}, tk, \mathtt{U})$ to $\mathcal{F}_\psi$, and obtains $\{B_i\}_{i=1}^k$. $\mathcal{S}$ then invokes a simulator of OT to interact with $\mathcal{A}$ using $\{B_i\}_{i=1}^k$ and $m - k$ random values for the rest bridges (such a simulator always exists for a sender privacy preserving OT scheme). After that, $\mathcal{S}$ simulates a dummy bridge distributor to verify π_1 and provides $\mathcal{A}$ a simulated credential. Finally, $\mathcal{S}$ stores $\langle \mathtt{U}, x \rangle$.

- *Updating credit balance*: Upon receiving $\kappa_\Phi \| \tilde{C}_\Phi \| \tilde{C}_u \| \pi_2$ from $\mathcal{A}$, $\mathcal{S}$ first extracts (x, B_u) from π_2, and locates the owner of this credential (i.e., $\mathtt{U}$) by looking up the stored records using x (note that $\mathcal{A}$ could use the credentials of other corrupt users). $\mathcal{S}$ then sends $(\mathtt{update}, B_u, \mathtt{U})$ to $\mathcal{F}_\psi$. After that, $\mathcal{S}$ simulates a dummy distributor with $\mathcal{A}$ following the protocol ψ.

- *Getting a new bridge*: Upon receiving B_b from $\mathcal{A}$, $\mathcal{S}$ sends a query (query, B_b) to $\mathcal{F}_\psi$, and forwards the query result (either β_b or $\perp$) to $\mathcal{A}$. If $\mathcal{A}$ decides to proceed by sending $\kappa_\Phi \| \kappa_b \| \tilde{C}_\Phi \| \pi_3$, $\mathcal{S}$ extracts x from π_3 and uses it to locate $\mathtt{U}$, and then sends $(\mathsf{newbridge}, B_b, \mathtt{U})$ to $\mathcal{F}_\psi$ and receives $\tilde{B}_b$. After that, $\mathcal{S}$ runs a OT simulator to give $\tilde{B}_b$ to $\mathcal{A}$, and finally uses her private key $SK^*_{\mathtt{D}}$ to generate signatures $\tilde{\sigma}_b$ and $\tilde{\sigma}_\Phi$.

- *Inviting new users*: Upon receiving $\kappa_\Phi \| \kappa_\omega \| \tilde{C}_\Phi \| \; \tilde{C}_\omega \| \pi_5$ from $\mathcal{A}$, $\mathcal{S}$ finds $\mathtt{U}$ using x that is extracted from π_5, and sends $(\mathsf{ticket}, \mathtt{U})$ to $\mathcal{F}_\psi$, and finally sends the output received from $\mathcal{F}_\psi$ (either tk or $\perp$) to $\mathcal{A}$.

The distribution produced in $Game_{16}$ is identical to that of $Game_{15}$, i.e., $|Pr(Game_{16}) - Pr(Game_{15})| = 0$. By summation, we have $Pr(Game_{16}) \leq \nu(l)$, where $\nu(l) = \sum_{i=1}^{12} \nu_i(l)$.

$\square$

CHAPTER 4

A NEW CENSORSHIP CIRCUMVENTION ARCHITECTURE USING ASYMMETRIC COMMUNICATION AND IP SPOOFING

In this chapter, we present a new censorship circumvention architecture – CensorSpoofer [19], which hides the redirection proxy's address from any user, thus fundamentally resisting to the insider attack, while providing low-latency censorship circumvention service. Our key insight is that it is feasible to apply IP spoofing to send downstream traffic from the proxy to users while concealing the proxy's IP address, and due to the asymmetric nature of web browsing traffic, we can use a separate low-bandwidth indirect channel (such as Email or Instant Messaging) with steganography to communicate the user's messages (e.g., URLs) to the proxy. This asymmetric communication architecture allows the censorship circumvention system deviate from need of specially support from the network infrastructure as many existing systems [16–18] require, and allows ordinary people to set up their circumvention systems to help people in censored countries.

We start with introducing the related work on censorship circumvention systems, and present our system models and design goals. We then describe the CensorSpoofer framework and give a concrete example of designing a circumvention system based on this framework. We lastly show the prototype implementation and evaluation results.

4.1 Related Work on Censorship Circumvention Systems

In response to Internet censorship, many pragmatic systems such as Dynaweb/freegate [8], Ultrasurf [9], Psiphon [10], and Tor [11] have been developed to help people bypass censorship. All these systems are based on a simple idea: let the user connect to one of the redirection proxies deployed outside the censor's network, which can fetch blocked webpages for the user.

Table 4.1: Comparison of censorship circumvention systems

	P1	**P2**	**P3**	**P4**
Redirection proxy based [8–11, 13–15, 23, 40]	✓	✓	✓	✗
Infrastructure assisted [16–18]	✓	✗	✓	✓
Email based [28, 41]	✗	✓	✓	✓
Anti-censorship content sharing [42–46]	✓	✓	✗	✓
CensorSpoofer [19], rBridge [12]	✓	✓	✓	✓

P1: Require no special server (By "special server", we mean the number of such servers on the Internet is much smaller than that of ordinary servers. An example of special servers is external Email servers that provides encrypted communication; right now, only Gmail and Hotmail satisfy this requirement to Chinese users.)
P2: Require no special support from core ISPs
P3: Support low-latency communication, e.g., web browsing
P4: Strong resistance against the insider attack

To hide the nature of the traffic, the communications with the proxy are encrypted. Infranet [47] takes things a step further, embedding the real communication inside a cover web session, using covert channels to communicate the request and image steganography to return the data. However, while escaping detection by outsiders, these designs are vulnerable to the insider attack, where the censor pretends to be an ordinary user to learn the location of the proxies and then block them.

Several researchers have tried to design better relay distribution strategies [13–15, 23] that aim to identify users who are likely to lead to a relay being blocked using past history and directing new relay information towards other users. However, these designs are not likely to withstand a censor who controls a large number of corrupt users.

TriangleBoy [40] is a circumvention system that is similar in spirit to Tor-bridge and also uses IP spoofing. In TriangleBoy, a user connects to one of the TriangleBoy proxies run by volunteers, and the proxy forwards the user's URLs to a SafeWeb server, which fetches the web pages and sends them back to the user by spoofing the source IP address with the proxy's IP. The only difference between Tor-bridge and TriangleBoy is that in Tor the downstream traffic takes the same route (through several relays) as the upstream traffic, while in TriangleBoy the server sends the downstream traffic to the user directly using IP spoofing to improve efficiency. Note that, in spite of using IP spoofing, the TriangleBoy proxies are still exposed to users, which renders the same problem to the insider attack as Tor bridges; whereas, CensorSpoofer adopts a different architecture by using IP spoofing to conceal the proxy's IP address, and hence perfectly resists the insider attack.

Similar to CensorSpoofer, another school of prior research tries to fundamentally resist the insider attack, i.e., tolerating any fraction of corrupted users, by hiding the relay's IP from any user and therefore the censors. One way to achieve that is to utilize indirect channels, i.e., relaying the traffic sent to/by the relay through one or more intermediate nodes. For example, MailMyWeb [41] and FOE [28] utilize Email as the indirect channel. For these systems, users are required to be able to access foreign servers that support encryption (e.g., Gmail), in order to avoid being detected by the censor. Nevertheless, considering the Chinese government once temporarily blocked Gmail [48], we can envision the censor would again block the few special Email providers, once finding out they are popularly used to bypass censorship.

It is important to note that, while an indirect channel is also used in CensorSpoofer, we only use it for sending outbound messages (e.g., URLs), which are usually very small (especially after encoding URLs into small numbers) and easy to hide into any indirect channel using steganography. This allows us to obviate the need for special servers (e.g., external Email providers supporting encryption) to provide a secured and high-bandwidth indirect channel. Consequently, the cost of blocking the outbound channel of CensorSpoofer is significantly higher: the censor has to block all overseas indirect communication (e.g., overseas Email and IM) even though the users only use the local Email and IM providers controlled by the censor.

More recently, researchers proposed several infrastructure-assisted circumvention systems, including Telex [17], Decoy routing [16], and Cirripede [18]. Although these systems can support low-latency communication and perfectly resist the insider attack, they require a significant investment of effort by core Internet ISPs. By contrast, CensorSpoofer is an infrastructure-independent circumvention system, allowing individuals to deploy their own anti-censorship systems without requiring any additional support from network infrastructure.

Instead of aiming to provide low-latency communication service, some anti-censorship systems are designed to achieve censorship-resistant content sharing and/or distribution. For example, some works leverage peer-to-peer (P2P) networks to provide privacy-preserving file sharing, e.g., Freenet [43], membership concealing overlay network [44], and darknet [45, 46]. Collage [42] let users stealthily exchange censored information with an external relay via a website that can host user-generated content (e.g., Flickr) using steganography.

Table 4.1 summarizes the requirements and capabilities of the representative censorship-circumvention systems.

4.2 Concept

4.2.1 Threat Model

We consider a state-level adversary (i.e., the censor), who controls the network infrastructure under its jurisdiction. The censor has sophisticated capabilities of IP filtering, deep packet inspection, and DNS hijacking, and can potentially monitor, block, alter, and inject traffic anywhere within or on the boarder of its network. However, the censor is motivated to allow citizens to *normally* access basic Internet services, such as IM, Email and VoIP, as blocking such services would lead to economic losses and political pressure. More specifically, we assume the censor is unwilling to interfere with the Internet connections of a user, e.g., an ongoing VoIP conversation, unless it has evidence that a particular connection is being used for bypassing censorship.

Furthermore, we assume the censor generally allows people to use common encryption protocols to protect their online privacy, e.g., SRTP [49] or ZRTP [50] for secure VoIP communication.[1] Thus far, this assumption has held true for most existing cases of Internet censorship, and the use of encrypted protocols such as SSL/TLS have formed the foundation of most existing anti-censorship systems [8–11, 16–18, 28, 41]. Once again, blocking encrypted traffic reduces the security of normal citizens using the Internet for personal or business reasons, and thus censors are motivated to allow

[1]Although much of VoIP traffic is currently unencrypted, the trend is towards more widespread use of secure VoIP protocols; for example, a number of VoIP software clients [51–57] and VoIP phones [58, 59] have encryption functionality.

such traffic through. There have been important exceptions to this, including Iran's blocking of all encrypted traffic prior to the 33rd anniversary of the Islamic Revolution [60] and Egypt's complete disconnection of the Internet in response to nationwide protests [61]. Such drastic censorship requires fundamentally different circumvention approaches that are out of scope of our work.

We assume the censor can utilize its governmental power to force local IM, Email, and VoIP providers to censor their users' communication. We also assume that the censor can block *any* foreign Internet website or service, such as an Email or instant messaging provider, if it has reason to believe that it is being used to circumvent censorship. The censor can rent hosts outside of its own network, but otherwise has no power to monitor or control traffic outside its borders. Finally, we assume that the censor has sufficient resources to launch successful insider attacks, and thus is aware of the same details of the circumvention system as are known to ordinary users.

Similar to many existing systems [11, 16–18, 42, 47], our approach requires that users run specialized circumvention software on their computers. We assume that users are able to obtain authentic copies of the software without alerting the government to this fact through some form of out-of-band communication. (We acknowledge, however, that secure and reliable mechanisms for distributing such software are an important area of future research.)

4.2.2 System Goals

CensorSpoofer aims to achieve the following goals:

Unblockability: The censor should not be able to block CensorSpoofer without incurring unacceptable costs.

Unobservability: The censor should not be able to tell whether a user is using CensorSpoofer without incurring unacceptable costs.

Perfect resistance to insider attack: The censor should not be able to break unblockability or unobservability of CensorSpoofer even if nearly all users are corrupted.

Low latency: CensorSpoofer should be able to fetch and delivery the web pages for users with low latency. For clarity, CensorSpoofer does not aim to support highly interactive web applications, such as Javascript.

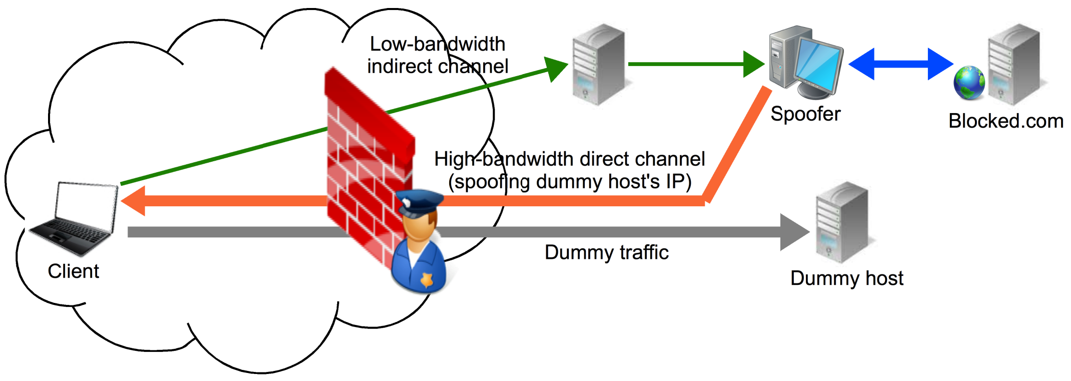
Low-bandwidth indirect channel
Spoofer
Blocked.com
High-bandwidth direct channel (spoofing dummy host's IP)
Client
Dummy traffic
Dummy host

Figure 4.1: The CensorSpoofer framework. The user pretends to communicate with an external dummy host legitimately, and sends URLs to the spoofer via a low-bandwidth indirect channel (e.g., steganographic IM/Email). The spoofer fetches blocked webpages according to the received URLs, and injects censored data into the downstream flow towards the user by spoofing the dummy host's IP.

Deployability: CensorSpoofer should be deployable by people with limited resources, without requiring any support from network infrastructure.

4.3 CensorSpoofer Framework

4.3.1 Overview

In censored countries, users cannot visit blocked websites directly and have to connect to some external relays to access these websites. These relays' IP addresses are exposed to users who connect to them, and therefore can be easily blocked by the censor who colludes with corrupt users. A natural solution to this is to employ indirect channels to hide the relay' IP. For example, MailMyWeb [41] and FOE [28] use Email as the indirect channel for which the intermediate nodes are Email servers.

To carry voluminous downstream traffic (e.g., web content), the indirect channel must have *high bandwidth*. This requirement excludes steganographic indirect channels, such as steganographic IM/Email. As a result, the circumvention system has to rely on an encrypted indirect channel so as to utilize full capacity of the indirect channel while ensuring unobservability of the transmission of censored data. This requires the intermediate nodes of the indirect channel to support encryption (e.g., TLS/SSL) and reside outside the censor's network (to avoid eavesdropping by corrupt intermediate nodes). Currently, only a few Email providers can meet this requirement: Gmail, Hotmail, and Yahoo! Mail. However, due to their limited user base in the censored country, the censor could simply block them altogether, as witness when Gmail was blocked in China in 2011 [48].

Our insights. We notice that for web browsing, the outbound traffic (e.g., URLs) is much lighter-weight than the inbound traffic. If an indirect channel is only used to send outbound messages, high bandwidth is no longer required for the indirect channel. This allows us to use *any* indirect channel with steganography to transmit outbound data. Besides, by using steganography, users can even use local IM or Email providers that potentially collude with the censor to access our circumvention system without being detected. The elimination of requiring special servers to construct the indirect channel makes it substantially harder for the censor to block our circumvention system as all overseas Email and IM communication has to be prohibited.

As for the inbound channel, since the relay's IP (i.e., source IP) is not used in packet routing, we can adopt IP spoofing to conceal the relay's IP address. This eliminates the need for an indirect channel to hide the relay's IP, allowing us to use direct channels, which are more common and higher-bandwidth, to send voluminous inbound traffic.

Our design. Based on these insights, we design a new circumvention framework for web browsing, which uses asymmetric communication with separate inbound/outbound channels. In particular, a user who requires circumvention service first starts or pretends to start a legitimate communication session (e.g., a VoIP call) with a *dummy host* residing outside the censor's network, and the relay (called *spoofer*) injects censored data into the downstream flow sent to the user by spoofing the dummy host's IP, so that the censor believes the user is legitimately communicating with the dummy host *only*. The dummy host does not need to actively cooperate with the user or the spoofer, but should look legitimate to the censor, e.g., its port for VoIP should "seem open" if the cover session is a VoIP call. Meanwhile, the user sends outbound messages containing URLs to the spoofer through a low-bandwidth indirect channel, such as steganographic IM/Email. An illustration of the framework is provided in Figure 4.1.

Next, we discuss the upstream and downstream channels in more details.

4.3.2 Downstream Channel

1) To conceal the spoofer's IP address, we apply IP spoofing in the downstream flow. Then, the first question is *what kind of traffic (TCP or UDP) is suitable for IP spoofing?*

Generally, hijacking TCP with IP spoofing is difficult. In TCP, end hosts maintain connection state and acknowledge received data. Suppose the client has established a TCP connection with the dummy host, and the spoofer knows the dummy host's IP address and sequence number and tries to inject packets containing censored data into the downstream flow. First of all, the TCP connection with the dummy host must be kept alive; otherwise, the dummy host will send RST packets in response to the client's packets, which can be easily detected by the censor. In addition, if the spoofer sends more data to the client than the dummy host (i.e., the sequence number of the spoofer is higher than that of the dummy host), the censor can detect the inconsistency of the sequence numbers as long as the dummy host sends any packet to the client[2]. Thus, the spoofer has to use the sequence numbers that have already been used by the dummy host (i.e., injecting packets as "resent packets"). However, in this case a censor with packet-recording capability can detect the injected packets by comparing the contents of packets with the same sequence number.

In contrast, UDP is a connectionless protocol and easier to hijack. Unlike TCP, end hosts of UDP do not maintain any connection state or acknowledge received data. Hence, if the dummy host remains "quiet" and the client and the spoofer cooperate closely by sharing initial information and following a proper traffic pattern, it is feasible to deceive a smart censor into believing that the client is legitimately communicating with the dummy host over a duplex UDP channel. In this work, we focus on UDP traffic for IP spoofing. We present a concrete example of hijacking UDP in Section 4.4.

2) To ensure unobservability, the communication between the client and the spoofer (and the dummy host) should look like a normal UDP session of a legitimate Internet application. So, the second question is *what carrier applications should be used?*

UDP is mainly used for time-sensitive applications, such as VoIP, video

[2]An active censor can check the dummy host's current sequence number by replaying a client's packet that is outside the dummy host's receiving window; in this case the dummy host will reply an ACK packet containing its current sequence number.

conferencing, multi-player online games, webcam chat, online TV, etc. These applications usually have high-bandwidth channels. Some other UDP applications, such as DNS and SNMP, have very limited bandwidth and thus are not suitable to carry voluminous inbound traffic.

We can further divide these applications into two classes based on their communication manner: (i) client-to-server communication, e.g., multi-players online games and online TV, and (ii) client-to-client communication, e.g., VoIP and video chat. To achieve better robustness to blocking, we prefer the applications in the second class, since for these applications the pool of dummy hosts is significantly larger (e.g., the dummy hosts could be any VoIP client on the Internet), making it much harder to block them altogether.

3) In CensorSpoofer, we use a dummy host as a cover to stealthily transmit censored data. The third question is *how to select plausible dummy hosts?*

The selection of dummy hosts is decided by the carrier application. For example, if the carrier application is VoIP, then each dummy host should be a potential VoIP client. Note that an active censor can use port scanning (e.g., using `nmap` [62]) to check if a dummy host is actually running the application, i.e., listening on a particular port (e.g., port 5060 for SIP-based VoIP). In response, we can use port scanning as well to obtain the list of dummy hosts. According to our experience, a dummy host is "quiet" (i.e., not sending any reply packet) to incoming UDP packets sent to a specific port, as long as this port is not "closed" on the dummy host. In many cases, port scanning is unable to determine whether a particular application is running on a target machine, since the target machine could be behind a firewall that is configured to filter probe packets. For example, `nmap` returns "open|filtered" or "closed|filtered" when it cannot tell whether the port is open/closed or the probe is filtered. This ambiguity plays in our favor as it makes a larger number of hosts appear to be plausible VoIP endpoints.

4) Finally, we note that not all Internet hosts can launch IP spoofing. Some ASes apply ingress and/or egress filtering to limit IP spoofing. The MIT ANA Spoofer project [63] has collected a wide range of IP spoofing test results, showing that over 400 ASes (22%) and 88.7M IPs (15.7%) can be used to launch IP spoofing. Therefore, we need to deploy our spoofer in the ASes where IP spoofing is not prohibited. We can utilize some tools, such as `nmap` and the spoofing tester developed by the Spoofer project [63], to test whether a host can perform IP spoofing.

4.3.3 Upstream Channel

To send outbound requests, we use a steganographic channel embedded in communications such as IM or Email. Note that URLs are typically quite short and can be easily embedded into a small number of messages. Communication requirements can be further reduced by using a pre-agreed list of censored URLs and sending just the index of the desired site. Likewise, navigation within a site can use relative link numbering, requesting, e.g., the 3rd link from the front page of `www.cnn.com`. Note that steganography requires the use of a secret encoding key to remain invisible; this process can be made resilient to insider attacks by having each user register a separate pairwise key when joining the system. Specific steganographic constructions and their security are beyond the scope of this work. An important challenge that we must address, however, is the possibility that the censor will perform blocking based on the recipient's IM identifier or Email address; we discuss a solution in Section 4.4.

4.4 A Design of CensorSpoofer

The CensorSpoofer framework can be instantiated using a number of protocol choices. In this section, we present a concrete design based on VoIP. We start with some background about VoIP systems.

4.4.1 Background of SIP-based VoIP

VoIP is an Internet service that transmits Voice over IP-based networks. It employs session control protocols, such as SIP, MGCP, and H.323, to setup and tear down calls. SIP is one of the most widely-used VoIP signal protocols, because of its light weight. In this work, we focus on SIP-based VoIP systems.

SIP is an application layer protocol. It can run on either UDP or TCP. There are three main elements in SIP systems: user agents, location services, and servers.

- *User agents* are the end devices in a SIP network. They originate SIP requests to establish media session, and send and receive media. A user agent can be a physical SIP phone or SIP client software running on a computer (also called *softphone*). A user agent needs a SIP ID, which is signed up at a SIP provider, in order to make and receive SIP calls.

- *Location service* is a database that contains information about users, such as SIP IDs, the latest login IP addresses, preferences, etc. Location services generally do not interact directly with user agents.
- *Servers* are intermediary devices that are located within the SIP network and assist user agents in session establishment. There are two main types of SIP servers: *registrar* and *proxy*. A registrar receives SIP registration requests and updates the user agent's information (such as the login IP address) into the location service. A SIP proxy receives SIP requests from a user agent or another proxy and forwards the request to another location.

Here is an example to show how a user (Alice) calls another user (Bob). Suppose Alice has signed up a SIP ID `alice@atlanta.com` at the SIP provider `atlanta.com`, and Bob got his SIP ID `bob@biloxi.com` from `biloxi.com`, and Alice knows Bob's SIP ID.

When Bob comes online, he first sends a registration request to the registrar of `biloxi.com` with its current IP address. So does Alice to register herself at the registrar of `atlanta.com`.

The SIP call initialization process is shown in Figure 4.2. First, Alice sends an INVITE message (M1), which contains her SIP ID and IP address, Bob's SIP ID, her supported media codecs, etc., to the proxy of `atlanta.com` (note that at this point Alice does not know Bob's IP address). The local proxy performs a DNS lookup to find the IP address of the proxy serving Bob's domain, i.e., `biloxi.com`, and then forwards the INVITE message (M2) to the remote proxy. At the meantime, the local proxy sends a Trying response (M3) back to Alice, indicating that the INVITE has been received and is being routed to the destination. Upon receiving the INVITE message, the proxy of `biloxi.com` sends a query to its location service to look up the registered IP address of Bob, and then it forwards the INVITE message (M4) to Bob. The user agent of Bob sends a Ringing response (M6) to the proxy indicating that Bob's phone is ringed. If Bob decides to answer the phone, an OK message containing Bob's current IP (M9) is sent towards Alice; otherwise, a Reject response is returned (not shown in the figure). From the received OK message, Alice learns Bob's IP address, and sends

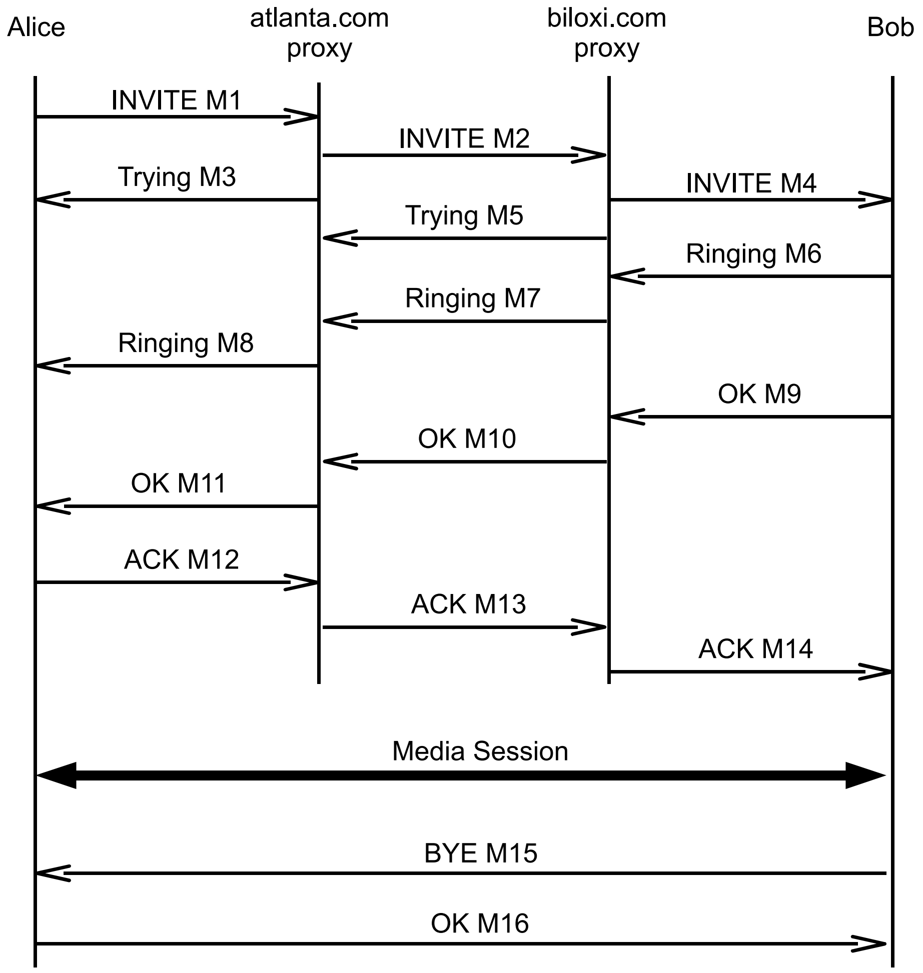

Figure 4.2: An example of a SIP session (registrars and location services are not shown).

an ACK message towards Bob (M12, M13, M14). At this point, the SIP initialization session is done, and Alice and Bob start the media session by sending each other audio data directly. At the end of the media session, either party can send a BYE message (M15) to close the call.

The media session uses Real-time Transport Protocol (RTP) to transmit audio data, and Real-time Transport Control Protocol (RTCP) to provide out-of-band statistic and control information for the RTP flow. Both RTP and RTCP run on top of UDP. VoIP clients can use SRTP/SRTCP [49]—an encrypted version of RTP/RTCP—to encrypt their voice communication. SRTP/SRTCP only requires the user to install a user agent that has encryption features, and does not require VoIP servers to support encryption. This implies that the user can use any VoIP provider, including local providers that collude with the censor, to access our circumvention system. The encryption key for SRTP/SRTCP can be either established beforehand, e.g., via MIKEY [64], or negotiated on the fly using ZRTP [50]. In this work, we consider using pre-established keys for SRTP/ SRTCP.

4.4.2 Censorship Circumvention

A sketch of the circumvention procedure is as follows. The client first initializes a SIP session with the spoofer by sending out a normal INVITE message. Upon receiving the INVITE message, the spoofer randomly selects a dummy host and replies with a manipulated OK message that looks like originating from the dummy host. When the OK message arrives, the client starts to send encrypted RTP/RTCP packets with random content to the dummy host, and the spoofer starts to send encrypted RTP/RTCP packets to the client by spoofing the dummy host's IP address. Meanwhile, the client sends URLs through a steganographic IM/Email channel to the spoofer. The spoofer fetches the webpages, puts them into RTP packet payloads and sends them to the client. To terminate the circumvention session, the client sends a termination signal to the spoofer over the outbound channel, and then the spoofer sends a BYE message (with IP spoofing) to the client to close the call.

Invitation-based Bootstrapping

Since the censor can learn the callee SIP ID from the INVITE message, the user cannot use a common callee SIP ID to call the spoofer (otherwise, he/she will be detected once the censor learns the spoofer's SIP ID from corrupt users). There is a similar issue for the steganographic IM/Email channel: the censor can detect users sending IMs or Emails to the spoofer based on the recipient's IM ID or Email address (generally referred to as *upstream ID*).

To address this, we let the spoofer use a *unique* callee SIP ID and a *unique* upstream ID to communicate with each client. Hence, the SIP IDs and upstream IDs of the spoofer learned by corrupt users cannot be used to detect honest users. To avoid the bottleneck of having the spoofer create a large number of SIP and upstream IDs by itself, we let each client sign up a callee SIP ID and an upstream ID on behalf of the spoofer, and give them to the spoofer when joining the system. We achieve this by introducing an invitation-based bootstrapping process.

In particular, if a user Alice wants to join the circumvention system, she needs an invitation and help from an existing CensorSpoofer user (say Bob). Alice must trust Bob (e.g., Bob is a friend of Alice); otherwise, Bob could simply report Alice to the censor for attempting to access circumvention service. (We note that similar invitation-based bootstrapping strategies have already been adopted by some real-world circumvention systems, e.g., Psiphon [10].) First, Alice needs to sign up two SIP IDs and two upstream IDs. One pair of SIP ID and upstream ID is for herself, and can be obtained from her local SIP and IM/Email providers (which potentially collude with the censor). The other pair is for the spoofer, and must be signed up at abroad SIP and IM/Email providers (not necessarily supporting encryption). If all external SIP, IM, or Email providers are blocked by the censor, Alice can ask Bob to use his already-established circumvention channels to sign up these IDs for her. Then, Alice encrypts the following registration information with the spoofer's public key:

caller SIP ID | *master key* |
callee SIP ID | *passwd for callee SIP ID* |
upstream ID | *passwd for upstream ID*

The master secret is used to derive SRTP/SRTCP session keys (and the key for the steganographic outbound channel if necessary), and the passwords are for the spoofer to login the callee SIP ID and the upstream ID.

To complete the bootstrapping, Alice needs to deliver the encrypted registration information to the spoofer. Alice could ask Bob to forward the whole registration information to the spoofer through his outbound channel. To reduce the bandwidth consumption of Bob's outbound channel, Alice could let Bob only forward the encrypted upstream SIP ID and password to the spoofer; once her outbound channel is established, she can send the rest registration information to the spoofer by herself.

Note that our unique-ID-assignment strategy cannot be applied to existing proxy-based circumvention systems, such as Tor, to improve the robustness against the insider attack. This is because the "ID" in CensorSpoofer is an application-level ID, and it is fairly easy to get a large number of them;

whereas, in Tor, the "ID" is the relay's IP address, and IP addresses are commonly viewed as a scarce resource and it is hard to get a large number of spare IP addresses.

For the spoofer, it needs to run multiple SIP IDs and multiple upstream IDs at the same time (possibly with a common service provider). In general, IM/Email servers and SIP registrars do not limit the number of accounts registered from a common IP address, because it is possible that multiple legitimate clients are behind a NAT sharing the same IP address. We did some tests on two real-world VoIP providers `ekiga.net` and `mixvoip.com` with 100 different SIP IDs running on one of our lab machines, respectively. It turned out for both providers, all these SIP IDs can be registered and receive calls successfully. We also did tests on Gtalk with 10 different accounts on the same machine and all of them worked properly.

Manipulating the OK Message

Once the bootstrapping is done, the client can initialize a circumvention session by calling the spoofer using the previously registered callee SIP ID. In the SIP protocol, the callee's IP address is written into the OK message (more specifically, the enclosed SDP message [65], which is used to negotiate the session format, such as codecs, ports, IP, etc.), and later is used by the caller to send RTP/RTCP packets to the callee. Since the OK message can be eavesdropped by the censor, the spoofer cannot put its real IP into the OK message.

For this, we use a trick to hide the spoofer's IP address. According to the IETF standards [65, 66], the SDP messages are not checked by SIP proxies. This means the spoofer can put the dummy host's IP, instead of its own IP, into the OK message, without influencing the OK message being forwarded back to the client. Since the registered IP of the callee SIP ID (kept by the location service of the spoofer's VoIP provider) is unknown to the censor, the manipulated OK message is still plausible to the censor. To verify the feasibility of replacing the spoofer's IP address in the OK message in practice, we utilized `netfilter_queue` [67] to modify the OK message on the fly, and tested it with two VoIP providers `ekiga.net` and `mixvoip.com` and an unmodified VoIP softphone PJSUA [54]. We found all manipulated OK messages were successfully delivered to the client and the client-side softphone started to send RTP/RTCP packets to the replaced IP address after receiving the OK message.

Selection of Dummy Hosts

A SIP client listens on TCP and/or UDP port 5060 for SIP signalling, and the ports for RTP/RTCP are selected randomly on the fly (usually RTP uses an even port and RTCP uses the next higher odd port). To check the legitimacy of a dummy host, the censor could apply port scanning to test if the ports used by VoIP are open on the dummy host. In response, we can also use port scanning to get the list of dummy hosts. As we mentioned before, in many cases, port scanning can only return an ambiguous result. For `nmap` [62] (the state-of-the-art port scanning tool), the possible probing results include "open", "closed", "filtered", "unfiltered", "open|filtered", "closed|filtered", and "host seems down". Only "closed" can clearly tell the censor that a particular application is not running on the target machine. When the status is "host seems down", it is very likely that the target host is offline. For safety, we also exclude "host seems down" from the acceptable probing states. Therefore, we let the spoofer periodically run port scanning with randomly selected IPs outside the censor's network to get a list of acceptable $\langle ip, rtp\_port \rangle$ (see Algorithm 1).

Another strategy for the censor to check legitimacy of the dummy host is to compute the AS path of the spoofing traffic and compare it against the observed entry point of the inbound traffic (i.e., where it enters the censor's network). If the dummy host is located far from the spoofer, it is likely that the entry point of the spoofing traffic is inconsistent with its claimed AS path. To deal with this, we first use `traceroute` to compute the AS path from the spoofer to the client (called *reference AS path*), and then choose a dummy host whose predicted AS path to the client is consistent with the reference AS path with respect to their entry points. Researchers have proposed several AS-path inference algorithms with high predication accuracy (such as [68]).

In addition, since the port status on a probed host may change over time, we let the spoofer keep track of the previously found dummy hosts and maintain a list of alive dummy hosts. When a circumvention request arrives, the spoofer picks a dummy host from the alive-host list, and keeps checking the VoIP ports of this dummy host during the circumvention session. If the

```
Input: $IP\_range$ // outside censored networks
Output: $dum\_hosts$
$dum\_hosts \leftarrow \{\}$ ;
$unaccepted \leftarrow \{closed, host\_seems\_down\}$ ;
foreach $ip \in IP\_range$ do
    if port_scan($ip, sip\_port$) $\notin unaccepted$ then
        $rtp\_port \leftarrow$ rand_even_port() ;
        $rtcp\_port \leftarrow rtp\_port + 1$ ;
        if port_scan($ip, rtp\_port$) $\notin unaccepted$ and
        port_scan($ip, rtcp\_port$) $\notin unaccepted$ then
            add $\langle ip, rtp\_port \rangle$ to $dum\_hosts$ ;
        end
    end
end
```

Algorithm 1: Port scanning algorithm to find a list of candidate dummy hosts

spoofer detects any port of SIP, RTP and RTCP on the dummy host is closed before the circumvention session ends, it sends a BYE message to the client immediately to terminate the SIP session. If the client wants to presume the circumvention session, it needs to initialize another SIP session with the spoofer.

Traffic Pattern and Bandwidth

To resist traffic-pattern-analysis attack, the client and the spoofer should follow certain patterns of legitimate VoIP traffic when sending RTP/RTCP packets. For VoIP, both RTP and RTCP packets are of the same size and sent periodically[3]. The packet size and sending frequency are defined by the audio codec, which is negotiated during the SIP initialization session. The codec determines the bandwidth of the inbound channel ($\sim pkt\_size \times freq$). Some codecs that are used to achieve better voice quality can provide higher bandwidth (e.g., 64 Kbps with G.711), while others provide lower bandwidth (e.g., 16 Kbps with iLBC). Note that the same bandwidth is consumed at the dummy host, due to the dummy traffic sent by the client. We can use some bandwidth estimation tools (e.g., `packet-trains` [69]) to figure out

[3]Some softphones have the option of Voice Activity Detection (VAD), which can avoid unnecessary coding and transmission of silence voice data. With VAD, the RTP packet size and sending interval may variate. In this work, we assume no VAD is used at the spoofer or the client for simplicity.

how much available bandwidth the dummy host has, and based on that, we choose an appropriate codec to avoid consuming too much bandwidth of the dummy host.

Packet Loss

UDP does not provide reliable transmission. A RTP packet containing data of a blocked webpage could be lost during transmission, causing failure of reconstructing the webpage at the client. To tolerate packet loss, we can use Forward Error Correction (FEC) codes (e.g., Reed-Solomon code [70]) inside the inbound channel, so that the client can recover the webpage as long as a certain number of packets are received.

4.5 Prototype Implementation

4.5.1 The Spoofer

Our spoofer prototype is mainly composed of the following components: a SIP message handler, a RTP/RTCP transmitter, an outbound message receiver, and a prefetching proxy.

SIP Message Handler

We use PJSUA v1.12 [54] as an out-of-box tool to register the callee SIP IDs. We choose PJSUA because we can easily register multiple SIP IDs using the `--config-file` option with different configuration files. To prevent the user-agent fingerprinting attack, we use tcpdump to pre-record the OK response messages generated by different softphones, and use them as templates to generate corresponding OK messages to response to different INVITE messages. In our implementation, we create a profile based on the softphone of Ekiga [71].

When starting the spoofer, the SIP message handler first launches PJSUA to register callee SIP IDs, so that the SIP proxies can forward INVITE messages related with these SIP IDs to the spoofer. We use `netfilter_queue` [67] to capture incoming INVITE messages. (Since PJSUA requires to bind port

5060, we do not create a socket bound to port 5060 to receive INVITE messages.) For each received INVITE message, the SIP message handler generates a corresponding OK message, by extracting the session related information (such as the caller's SIP ID, IP address, tags, etc.) from the INVITE message and putting them and the IP address of a dummy host into the pre-recorded OK message. Once the OK message is sent out, the spoofer creates a thread for the RTP/RTCP transmitter for this client.

RTP/RTCP Transmitter

The RTP/RTCP transmitter needs to send RTP and RTCP packets periodically with IP spoofing. For this, we use a UDP raw socket, which allows us to put an arbitrary IP into the source IP field in the IP header. To encrypt RTP/RTCP packets, we use AES-128 of OpenSSL v1.0.0 [72] with a pre-shared key. Since the sending frequency of RTCP packets is much lower than that of RTP packets, we only use RTP packets to carry censored data and send RTCP packets with randomly generated payloads.

To handle packet loss, we implemented a simple XOR-based encoder and decoder. The RTP/RTCP transmitter partitions the flow of each task (i.e., downloading a particular webpage) into fixed-sized data blocks (smaller than the RTP payload), and multiplex the blocks of different tasks of the same client into one stream, which is further divided into groups of size λ (e.g., $\lambda = 10$ blocks). For each group, the transmitter generates a redundant block by XORing all λ blocks in the group, so that any λ out of the $\lambda+1$ blocks are sufficient to recover the whole group. Whenever a RTP packet needs to be sent, the transmitter checks if there are any available blocks (including XOR blocks) in the buffer for this client. If so, it writes one block into the RTP payload and sends it out; otherwise, the RTP packet is stuffed with random data.

Note that some blocks may contain data less than their capacity (e.g., the last block of a task), and blocks may arrive at the client in different order than being sent out; besides, the client should be able to differentiate blocks for different tasks. To handle these, we use the first 4 bytes of the RTP payload to carry a block sequence number (2 bytes), a task number (1 byte), and block size (1 byte). These fields are encrypted together with the rest RTP payload.

Outbound Message Receiver

For this prototype, we use Gtalk as the outbound channel, although our system in no way depends on encrypted indirect channels like Gtalk. Gtalk employs `XMPP` [73] as the transmission protocol. We implemented a simple Gtalk client using a python API xmpppy [74] to send and receive Gtalk messages. The Gtalk ID of the spoofer is pre-given to the user. Each Gtalk message contains a URL, the user's IP address, and a task number (also contained in the RTP payload). The outbound message receiver forwards the received Gtalk message to the prefetching proxy by sending a UDP packet, and then the prefetching proxy will start downloading the webpage according to the URL.

Prefetching Proxy

For normal web browsing, a user inputs a URL in its web browser, and the browser will then fetch the html file of the webpage as well as the objects used by the webpage, such as figures and video clips. The browser downloads each object by sending a separate HTTP request.

Since each CensorSpoofer client only sends one URL (instead of separate HTTP requests) to the spoofer, the spoofer needs to prefetch the whole webpage on the behalf of the client. This means that the spoofer needs to first download the html file of the webpage, parse the html file to figure out the missing objects, and then send separate HTTP requests to fetch these objects, and finally send all the downloaded data to the client over the RTP channel.

We built a prefetching proxy (PFP) for this purpose. Instead of implementing a html parser and fetching embedded objects (which are essentially the operations of a web browser) from scratch, we use an open-source layout engine `QtWebKit` [75], which is a port of the popular `WebKit`[4] layout engine into the `Qt` application development framework. We choose `QtWebKit` because it provides a simple `QtWebPage` type that significantly reduces our development effort. Given a URL to load, a `QtWebPage` performs all the necessary network operations, including parsing, Javascript execution, etc., in order to render the webpage. The PFP obtains all the raw HTTP responses

[4]http://www.webkit.org/

for HTTP requests that the `QtWebPage` makes. As soon as PFP receives a full HTTP response, it sends the request-response pair to the client over the RTP channel. When the `QtWebPage` finishes loading the entire webpage, the PFP sends an "End-of-Page" marker to the client, to inform that there will be no more request-response pair for this webpage.

There are some limitations with our current PFP implementation. The `QtWebPage` on the PFP is a distinct browser instance from the client's browser, so the HTTP requests it generates are likely different from what the client's browser generates. This is a certainty in the presence of cookies because the cookies of the client's browser and all HTTP request headers are not forwarded to the PFP. Another limitation is that the current PFP disables Javascript on the `QtWebPage` because Javascript execution might generate additional HTTP requests after the page has "finished" loading (as notified by the `QtWebPage`), making it hard for the PFP to determine when to send the "End-of-Page" marker.

4.5.2 The Client

To avoid the censor detecting CensorSpoofer users based on the fingerprint of their softphones, we do not implement our own softphone for the clients; instead, we let the client use any existing softphone to access CensorSpoofer (i.e., for registration and sending SIP messages). Again, we use PJSUA for the client prototype without special reasons.

When running the client, PJSUA is first launched to register the user's SIP ID. Note that most softphones (including PJSUA) do not support making calls outside the user interfaces. In order to call the spoofer automatically inside our client program, we use tcpdump to pre-record the INVITE and ACK messages, and send them during the ongoing SIP initialization session with the spoofer (the ACK message needs to be updated according to the OK message before being sent out).

Once the SIP initialization is done, the client creates a UDP socket to receive RTP/RTCP packets from the spoofer and send RTP/RTCP packets to the dummy host. The client uses the pre-shared key to decrypt received packets and stores the decrypted blocks into a buffer. Once a sufficient number of blocks in a group are received, the client uses the XOR-based decoder to recover the original group.

We implemented a client-side HTTP proxy (CSP) to handle the HTTP requests made by the user's browser and the HTTP responses received from the RTP channel. When the CSP receives the first HTTP request for a page, it forwards the URL of the page to the spoofer via the Gtalk channel, but will not forward subsequent requests for other objects of the page. Instead, the CSP will "collect" in memory the HTTP request-response pairs received from the spoofer, and will serve to the client's browser the appropriate HTTP responses from its memory when the browser makes a HTTP request.

We note that any web browser supporting HTTP proxies, such as Mozilla Firefox[5], can use the CSP because the CSP provides an HTTP proxy compliant interface. Therefore, we do not have to modify existing web browsers or implement a new one. However, for ease of automating experiments, we implement a minimal browser application (totalling 150 lines of code) that is simply a wrapper around `QtWebPage` to load the webpages. This browser application also outputs various statistics useful for our evaluation.

4.6 Evaluation

We evaluate the performance of CensorSpoofer in a realistic environment, and compare it with other circumvention systems. Then, we measure the selection of dummy hosts.

4.6.1 Performance Evaluation

The spoofer was deployed on an `Emulab` machine (located in Utah, U.S.), which has 3.0 GHz 64-bit Duel Core CPU with 1 GB cache and 2 GB of RAM and runs Ubuntu 11. We deployed 8 clients on `Planetlab`, which are all located in China. Since we aim to evaluate the performance of our system, we let the clients share the same dummy host, which was randomly selected and located in Illinois, U.S.

To handle packet loss, we made the spoofer add a redundant XOR packet for every 10 packets. We chose the most commonly used VoIP codecs G.726-40, G.722-64, G.711, and iLBC, and set the corresponding RTP packet size

[5]http://www.mozilla.com/firefox

Table 4.2: Bandwidths for different VoIP codecs.

Codec	BW of inbound channel (Kbps)	Consumed BW of dummy host (Kbps)
G.711	64	87.2
G.722-64	64	87.2
G.726-40	40	54.7
iLBC	15.6	26.6

and sending interval according to the standard specifications in [76]. The bandwidth provided by each codec and the consumed bandwidth of the dummy host are provided in Table 4.2.

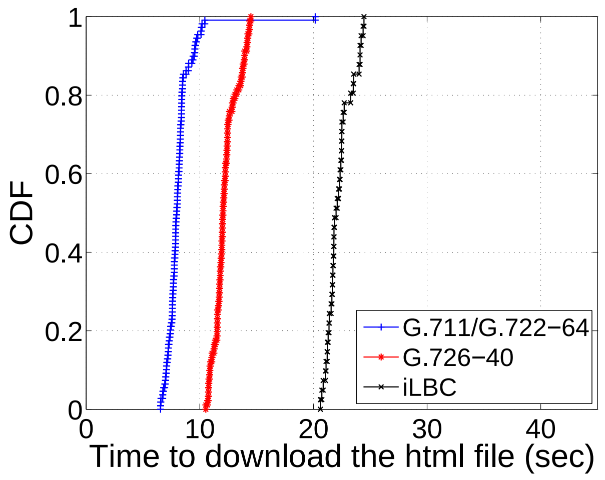

Figure 4.3: Time to download the HTML file only.

Each client was configured to repeated download the webpage of `wikipedia.org` (which is about 160 KB) for 20 times. For each download, we measured the downloading time for the entire webpage and the HTML file of the webpage. (Note that once the HTML file is downloaded, the user's web browser will display the basic frame and the text of the webpage, and the user can start reading the text-based content.) We found that the clients were able to successfully download the page of `wikipedia.org` (which was blocked in China) using CensorSpoofer. The results of downloading times are provided in Figure 4.3 and Figure 4.4. We can see that with the codec G.711 or G.722-64, the downloading time for the whole page was 27 seconds, but it only took about 6 seconds to load the HTML file.

In addition, we compared the performance of CensorSpoofer with that of existing circumvention systems. We installed a Tor client on one of the

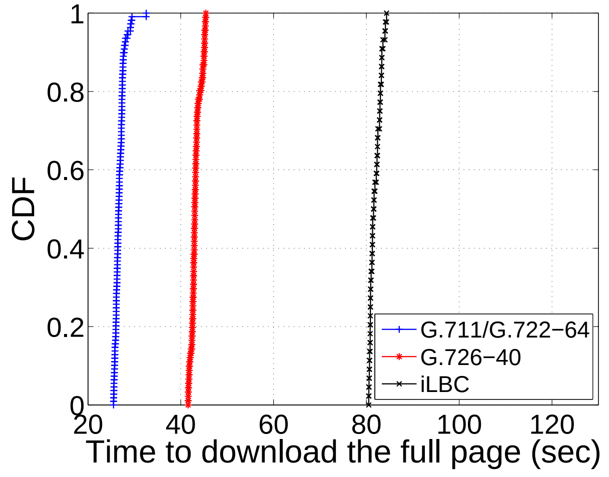

Figure 4.4: Time to download the full webpage.

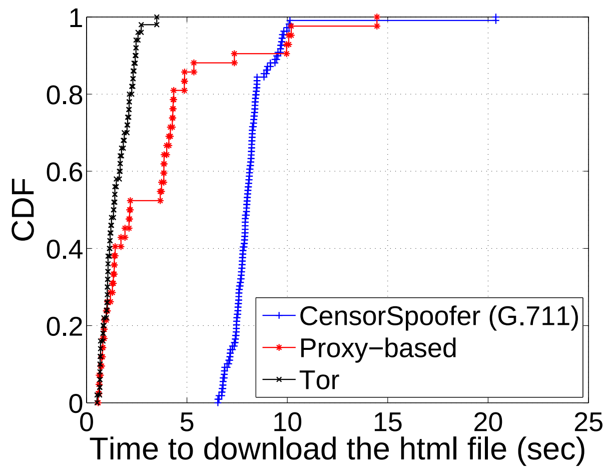

Figure 4.5: Comparison of downloading time for the HTML file only.

`Planetlab` nodes, and made it connect to a bridge in U.S. to download the webpage of `wikipedia.org` for 50 times. Additionally, we ran the same experiment by making the client connect to a public proxy of `NetShade`[6] (a proxy-based circumvention & anonymity system), which is located in U.S. Figure 4.5 and Figure 4.6 show that it did take longer time for CensorSpoofer to download the pages than the other two circumvention systems, but the downloading time for small web contents, such as HTML files, for CensorSpoofer is still acceptable.

We note that the performance of CensorSpoofer can be improved by fixing some limitations of our current implementation. For example, our current

[6]`http://www.raynersoftware.com/netshade/`

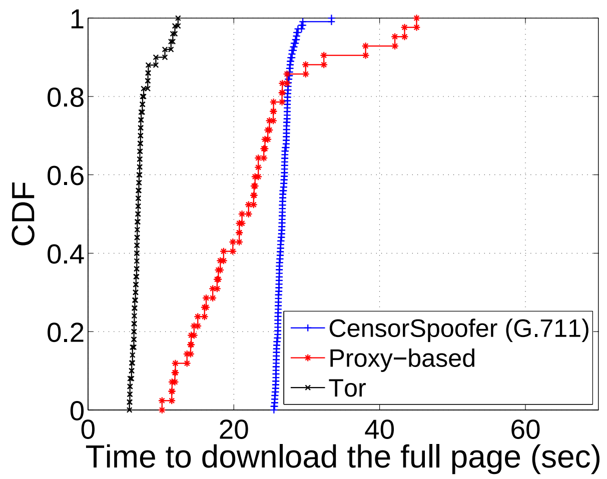

Figure 4.6: Comparison of downloading time for the full webpage.

prototype of the spoofer does not start sending any packet to the client until it has fully received a HTML file or an object. We believe removing these limitations can reduce the downloading time. Similarly, the current prototype of the client-side proxy does not deliver HTTP data to the client's web browser until the full HTML file or object is downloaded. The can be provided by pushing received data to the browser instantly.

In addition, we notice that the main performance bottleneck of CensorSpoofer is the RTP channel that carries the voice data. We believe by using a higher-bandwidth downstream channel, such as video streaming, the performance of CensorSpoofer can be much improved.

4.6.2 Measurement of Dummy-Host Selection

To evaluate the easiness of finding dummy hosts, we implemented the port scanning algorithm (i.e., Algorithm 1 in Section 4.4) using `nmap` [62]. We considered China as the censored country. We randomly selected 10 000 IPs from the entire IP space, which are located outside China, according to an IP-geolocation database [77]. We finally found 1213 IPs that can meet our requirements, and the percentage of satisfactory IPs is 12.1%. This indicates that there are a potentially large number of usable dummy hosts on the Internet.

Furthermore, we computed the percentage of appropriate dummy hosts for a specific client based on their predicted AS paths to the client. We

Table 4.3: Usable dummy hosts based on AS paths (Spoofer-ASN = 38).

DST-ASN	% of direct IPs	Entry-ASN	# of usable dummy hosts	% of usable dummy hosts
4134	39.4%	4134	225	100%
4837	19.8%	4839	225	100%
9394	8.3%	9394	217	96.4%
4538	7.1%	23911	41	18.2%

implemented a widely used AS path inference algorithm [68] that is based on AS relationships [78]. We considered the top four ASes in China in terms of the number of covered direct IPs (according to [79]), and selected a random IP (i.e., the client) from each of the ASes. We randomly picked 225 dummy hosts out of the 1213 candidate dummy hosts, and computed the AS paths between them and the four clients. Then, we compared the output paths with the AS paths from the spoofer to the clients (computed using `traceroute`), and filtered the dummy hosts with inconsistent entry points. The results are shown in Table 4.3. We can see that for a specific client, there are enough dummy hosts to use, especially for the clients located in large ASes.

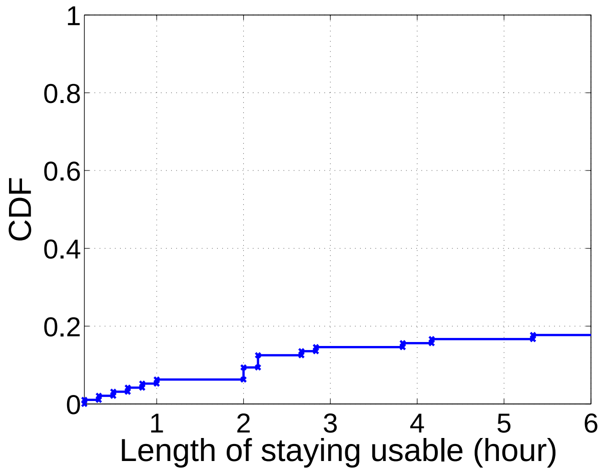

Figure 4.7: CDF (Note that the CDF plot is truncated to max_len_stay_usable = 6 hours, since many dummy hosts stay usable for a very long time).

In addition, we measured the stability of dummy hosts over time. Ideally, the dummy host should stay "usable" (i.e., none of its VoIP ports becomes "closed" or "host seems down") during the circumvention session, so that the user does not need to re-initialize the SIP session to change dummy hosts. To

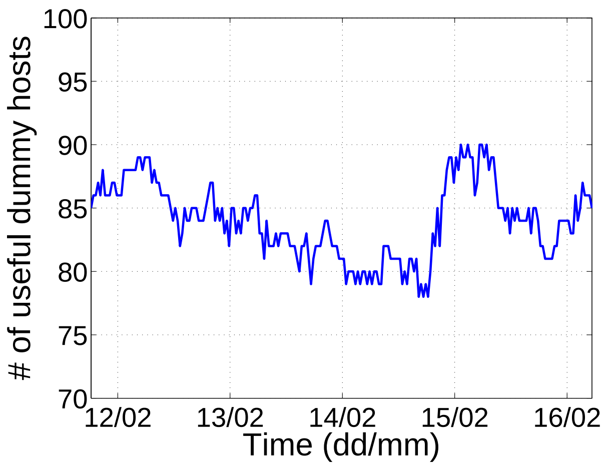

Figure 4.8: Stability of dummy hosts over a long period of time.

justify this, we randomly selected 100 dummy hosts out of the 1213 candidate dummy hosts, kept sending RTP packets to each of them and checking the states of their VoIP ports. Figure 4.7 depicts the CDF of length of staying usable for a dummy host. We can see that over 90% dummy hosts can stay usable for more than 2 hours, and over 80% can stay usable for longer than 6 hours. This means in most cases, the users only need to establish one SIP session throughout their web browsing.

We also measured the stability of dummy hosts over a longer period of time. We kept track of the states of 100 randomly selected dummy hosts from Feb. 9th 2012 to Feb. 16th 2012. To simulate the practical scenario when the dummy hosts are used by our system to receive VoIP traffic, we kept sending RTP packets to each dummy host periodically, with 1-hour sending period and 1-hour sleeping period. Figure 4.8 depicts the number of usable dummy hosts along the time. We can see that the total number of dummy hosts is almost stable, indicating that the overall pool of candidate dummy nodes does not shrink over time.

4.7 Security Analysis

We next discuss the security properties of CensorSpoofer against potential passive and active attacks.

4.7.1 Geolocation Analysis

Since the callee's SIP ID and IP address contained in the OK message are transmitted in plaintext, a sophisticated censor could record all the IP addresses that have been bound to a particular callee SIP ID over time, and try to discover abnormality based on the geolocations of these IPs. For instance, a SIP ID would look suspicious if its registered IPs for two closely conducted SIP sessions are geographically far from each other (e.g., the SIP ID is first registered with an IP in U.S. and 1 hour later it is registered again with another IP in Europe).

To deal with this, instead of picking dummy hosts randomly, the spoofer can choose a set of dummy hosts, which are geographically close, for a particular callee SIP ID, according to an IP-geolocation database (such as [77]). In particular, for the first-time use of a callee SIP ID, the spoofer randomly selects a *primary dummy host* for it, and keeps this information in the user database. For subsequent SIP sessions calling this SIP ID, the spoofer preferentially assigns its primary dummy host for it. If the port status of the primary dummy host becomes "closed", the spoofer then preferentially chooses a dummy host from those that have been assigned to this SIP ID (which are also stored in the user database). If none of them is available, the spoofer selects a new dummy host that is geographically close to the primary dummy host for this SIP ID. (Note that the spoofer should make sure that a particular dummy host is not being used by two or more callee SIP IDs at the same time.)

Furthermore, each user can create multiple callee SIP IDs. When a circumvention session is carried out very close to the previous one, or when the spoofer cannot find a suitable dummy host for a callee SIP ID, the user can choose another callee SIP ID instead.

4.7.2 User Agent & Operating System (OS) Fingerprinting

The SIP protocol defines the basic formats of SIP messages, but allows user agents (i.e., softphones or SIP phones) to add optional information into the SIP messages, such as the user's display name, timestamps, and the software/hardware information of the user agent. In addition, SIP messages (e.g., INVITE and OK) contain some random identifiers, such as "To tag" and "From tag", which are generated by the user agent with self-defined length. Additionally, the SIP messages also contain the codecs that are supported by the user agent.

The above information allows a sophisticated censor to fingerprint a particular user agent. As a result, the censor may detect users communicating with the spoofer based on the user-agent fingerprint of the spoofer. To address this, the spoofer can create a number of user-agent profiles based on the popular SIP phones and softphones, and assign one of them to each callee SIP ID. For a SIP session calling a particular SIP ID, the spoofer generates corresponding SIP messages based on the user-agent profile of the SIP ID.

Note that some softphones are only available for certain OSes. For example, SFLphone [52] can only be used on Linux, and Blink [51] is only available for Windows and Mac users. Hence, a sophisticated censor can use OS fingerprinting tools (e.g., the OS detection of `nmap` [62]) to check if the dummy host's OS is consistent with its user agent (learnt from the user-agent fingerprint). To handle this, the spoofer can also use the OS fingerprinting tool to detect the dummy host's OS and assign an appropriate user-agent profile.

4.7.3 Traffic Manipulation

The censor can also try to manipulate traffic flows in order to detect users accessing our circumvention system.

In anonymous communication systems (e.g., Tor [11]), an attacker could use traffic analysis to detect if two relays are on the same path of a flow, by injecting a specific traffic pattern at one relay (e.g., by delaying certain packets) and detecting the same pattern at the other relay [80]. If applying the same attacking philosophy to CensorSpoofer, the censor could delay the packets sent by the user, and detect if there are any changes of the traffic pattern in the downstream flow. However, this attack is based on the precondition that the flows sent and received by the remote host are correlated, and this is not true for VoIP, since each VoIP client sends RTP/RTCP packets periodically, independent of the incoming flow.

Another way to manipulate traffic is to drop packets. Since the spoofer does not actually receive any RTP/RTCP packets from the user, the censor can drop the user's packets without even being noticed by either the spoofer or the user. The VoIP phones can tolerate a small number of random packet loss; but if there are no RTP/RTCP packets received for a certain period of time (e.g., 30 seconds), they will drop the call automatically. Hence, a censor can adopt the following strategy to detect a CensorSpoofer user: it blocks all the RTP/RTCP packets sent to the callee, and checks if the callee still sends packets to the client after a certain period of time. However, the price of mounting this attack is very high. Since the censor is unable to tell which flow carries censored data, it has to drop all VoIP flows unselectively, causing normal VoIP conversations being interrupted.

The censor can also alter, reorder, inject or replay RTP/ RTCP packets sent to the callee (i.e., the dummy host). However, since a normal VoIP client running the SRTP protocol can simply filter the invalid packets, such attacks cannot help the censor detect if the callee is a real SIP client or a dummy host.

4.7.4 SIP Message Manipulation

The censor can attempt to manipulate SIP messages. For instance, the censor can manipulate the IP of the callee (i.e., the dummy host) in the OK message, and then check if there are any RTP/RTCP packets sent to the user. Similar to the packet-dropping attack, this attack will make legitimate users unable to make and receive VoIP calls. As we explained in our threat model, we believe censors are unwilling to mount such intrusive attacks by interrupting ordinary users' communication.

4.7.5 SIP Probing Attack

Houmansadr et al. [81] recently proposed a SIP probing attack to detect dummy hosts used in CensorSpoofer from genuine SIP VoIP users. In our design, we consider that sophisticated censors could use port scanning tools to check whether the SIP/VoIP ports on a suspected dummy host have legitimate status, and we employ port scanning as well to find legitimately looking dummy hosts. Houmansadr et al. found that, besides port scanning, a censor could apply SIP probing to check the status of dummy hosts, since a genuine SIP client should reply with a SIP message in response to a SIP probe message. Because of the SIP probing attack, the percentage of candidate dummy hosts (which have legitimate status to both port scanning and SIP probing) in all of the Internet hosts would be much smaller, which implies that we need to spend more time on probing and testing in order to find a usable dummy host. We note that the SIP/VoIP based CensorSpoofer scheme is just one design instance of the CensorSpoofer architecture. It is interesting to explore other applications, such as video call, web chat, and etc., to design censorship circumvention systems based on this asymmetric communication architecture, and multiple circumvention systems can be potentially used jointly to increase the set of candidate dummy hosts.

CHAPTER 5

CONCLUSIONS

5.1 Summary of Research

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 [5].

A typical approach to skirting censorship is to deploy circumvention proxies outside the censored network, which can provide indirect access to blocked websites. However, the redirection proxies are potentially exposed to malicious users and subject to blocking by the censor. In this thesis, we studied how to protect and/or design censorship circumvention systems against such insider attacks. We tackled this problem by two approaches: one is to design security mechanisms to improve robustness of existing censorship circumvention systems against the insider attack, and the other is to design new censorship circumvention systems that can fundamentally resist the insider attacks.

In particular, we proposed rBridge [12], 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.

In addition, we proposed a new censorship circumvention framework – CensorSpoofer [19], by exploiting the asymmetric nature of web browsing. CensorSpoofer decouples the upstream and downstream channels, using a low-bandwidth indirect channel for delivering URLs and a high-bandwidth direct channel for downloading web content. The upstream channel hides the requests using steganography within Email/IM, whereas the downstream channel uses IP spoofing to conceal the proxy's real address. Unlike some existing circumvention systems, CensorSpoofer does not require any additional support from network infrastructure. We implemented a proof-of-concept prototype for CensorSpoofer, and the experimental results showed that CensorSpoofer has reasonable performance for real-world usage.

5.2 Lessons Learned

The first lesson we learned from the thesis study is that nowadays censors are not simply relying on cyber techniques to achieve censorship, but are able to deploy a significant amount of human resources to help enforcing the censorship. For instance, according to the study by Zhu et al. [82], on `weibo.com`, a Chinese version of `twitter.com`, keywords blocking is employed as the basic tool to screen users' posts that contain politics sensitive content, and besides, there are over 4000 human censors, who manually check the posts that cannot be unambiguously flagged by automatic censorship mechanisms. In addition, human censors also follow threads on major forum websites to learn about available censorship tools or redirection proxies in order to block them. The capability of deploying a large force of human censors raise a big challenge to designing robust censorship circumvention systems, because we need to fight against not only censors who have full control of the local network infrastructure, but also potentially malicious users who are indistinguishable from genuine users.

In addition, we learnt that it is not rare to see massive blocking driven by events, such as crisis. For example, Iran blocked all encrypted traffic prior to the 33rd anniversary of the Islamic Revolution [60] and Egypt completely disconnected the Internet in response to nationwide protests [61]. It is a challenging problem to design censorship circumvention systems that can survive such drastic censorship, and when we evaluate security of certain anti-censorship systems, it is desirable to measure their robustness under such event-driven blocking.

Lastly, we have seen a long term arm race between censors and people in support of anti-censorship. As new censorship circumvention systems are built and become popular, censors eagerly study them and try to figure out how to block them. The censors have proved that they are capable of quickly evolving their censorship weapons using newly developed cyber techniques. It can be envisioned that such an arm race between censorship and anti-censorship will continue, and we should be prepared to face more advanced censorship techniques in the future.

REFERENCES

[1] J. Jarvis, "Facebook, twitter, and the egyptian revolution," Feb. 13. 2011, http://thefastertimes.com/mediaandtech/2011/02/13/facebook-twitter-and-the-egyptian-revolution/.

[2] "New blocking activity from iran," June, 16, 2011, https://blog.torproject.org/blog/new-blocking-activity-iran.

[3] "Defeat internet censorship: Overview of advanced technologies and products," nov 2007, http://www.internetfreedom.org/archive/Defeat_Internet_Censorship_White_Paper.pdf.

[4] C. S. Leberknight, M. Chiang, H. V. Poor, and F. Wong, "A taxonomy of Internet censorship and anti-censorship," http://www.princeton.edu/~chiangm/anticensorship.pdf.

[5] J. Zittrain and B. Edelman, "Internet Filtering in China," *IEEE Internet Computing*, vol. 7, no. 2, pp. 70–77, 2003.

[6] J. Jacob, "How internet censorship works in china," 2011, http://www.ibtimes.com/articles/113590/20110217/.

[7] "How censorship works in china: A brief overview," http://www.hrw.org/reports/2006/china0806/3.htm#_Toc142395821.

[8] "Dynaweb," dynaweb. http://www.dongtaiwang.com/home_en.php.

[9] Ultrasurf. http://www.ultrareach.com.

[10] J. Jia and P. Smith, "Psiphon: Analysis and estimation," 2004, http://www.cdf.toronto.edu/~csc494h/reports/2004-fall/psiphon_ae.html.

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

[12] Q. Wang, Z. Lin, N. Borisov, and N. J. Hopper, "rbridge: User reputation based tor bridge distribution with privacy preservation," in *NDSS'13*, 2013.

[13] D. McCoy, J. A. Morales, and K. Levchenko, “Proximax: A measurement based system for proxies dissemination,” in *Financial Cryptography and Data Security (FC’11)*, 2011.

[14] M. Mahdian, “Fighting censorship with algorithms,” in *Proceedings of FUN 2010*, 2010.

[15] Y. Sovran, A. Libonati, and J. Li, “Pass it on: Social networks stymie censors,” in *IPTPS’08*, Feb 2008.

[16] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer, “Decoy Routing : Toward Unblockable Internet Communication,” in *USENIX FOCI*, 2011.

[17] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, “Telex: Anticensorship in the Network Infrastructure,” in *20th USENIX Security Symposium*, Aug. 2011.

[18] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov, “Cirripede : Circumvention infrastructure using router redirection with plausible deniability categories and subject descriptors,” in *ACM CCS’11*, 2011.

[19] Q. Wang, X. Gong, G. Nguyen, A. Houmansadr, and N. Borisov, “Censorspoofer: Asymmetric communication using ip spoofing for censorship-resistant web browsing,” in *ACM CCS’12*, 2012.

[20] https://metrics.torproject.org/graphs.html.

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

[22] J. McLachlan and N. Hopper, “On the risks of serving whenever you surf: Vulnerability of tor’s blocking resistance design,” in *WPES’09*, 2009.

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

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

[25] R. Smits, D. Jain, S. Pidcock, I. Goldberg, and U. Hengartner, “Bridgespa: Improving tor bridges with single packet authorization,” in *WPES’11*, 2011.

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

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

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

[29] A. Pfitzmann and M. Hansen, "A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management," Aug. 2010, v0.34.

[30] C. Liu, R. W. White, and S. Dumais, "Understanding web browsing behaviors through weibull analysis of dwell time," in *SIGIR'10*, 2010.

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

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

[33] 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.

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

[35] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *SODA'01*, 2001.

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

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

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

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

[40] TriangleBoy Whitepaper. http://www.webrant.com/safeweb_site/html/www/tboy_whitepaper.html.

[41] MailMyWeb. http://www.mailmyweb.com/.

[42] S. Burnett, N. Feamster, and S. Vempal, "Chipping away at censorship with user-generated content," in *USENIX Security*, 2010.

[43] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley, "Protecting Free Expression Online with {Freenet}," *IEEE Internet Computing*, vol. 6, no. 1, pp. 40–49, 2002.

[44] E. Y. Vasserman, R. Jansen, J. Tyra, N. Hopper, and Y. Kim, "Membership-concealing overlay networks," in *ACM CCS'09*, Nov. 2009.

[45] WASTE. http://waste.sourceforge.net/.

[46] B. Popescu, B. Crispo, and A. S. Tanenbaum, "Safe and private data sharing with turtle: Friends team-up and beat the system," in *The 12th Cambridge International Workshop on Security Protocols*, April 2004.

[47] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger, "Infranet: Circumventing Web Censorship and Surveillance," in *USENIX Security*, Aug. 2002.

[48] D. Barboza and C. C. Miller, "Google accuses chinese of blocking gmail service," http://www.nytimes.com/2011/03/21/technology/21google.html?_r=2.

[49] "The secure real-time transport protocol (srtp)," http://www.ietf.org/rfc/rfc3711.txt.

[50] "Zrtp: Media path key agreement for unicast secure rtp," http://www.ietf.org/rfc/rfc6189.txt.

[51] Blink. http://icanblink.com/.

[52] SFLphone. http://sflphone.org/.

[53] Zfone. http://zfoneproject.com/.

[54] Pjsua. http://www.pjsip.org/.

[55] PhonerLite. http://www.phonerlite.de/index_en.htm.

[56] Microsoft Lync. http://technet.microsoft.com/en-us/library/gg195673.aspx.

[57] CounterPath. http://www.counterpath.com/softphone-products.html.

[58] Cisco IP phones. http://www.cisco.com/en/US/docs/voice_ip_comm/cucm/security/4_0_1/secuview.html.

[59] Grandstream. http://www.grandstream.com/products/ip-voice-telephony/enterprise-ip-phones/gxp1100.

[60] "Iran reportedly blocking encrypted internet traffic," 2012, http://arstechnica.com/tech-policy/news/2012/02/iran-reportedly-blocking-encrypted-internet-traffic.ars.

[61] M. Crete-Nishihata and J. C. York, "Egypt's internet blackout: Extreme example of just-in-time blocking," 2011, http://opennet.net/blog/2011/01/egypt%E2%80%99s-internet-blackout-extreme-example-just-time-blocking.

[62] Nmap. http://nmap.org/.

[63] The MIT ANA Spoofer project. http://spoofer.csail.mit.edu/.

[64] "Mikey: Multimedia internet keying," http://www.ietf.org/rfc/rfc3830.txt.

[65] "Sdp: Session description protocol," http://www.ietf.org/rfc/rfc4566.txt.

[66] "Sip: Session initiation protocol," http://www.ietf.org/rfc/rfc3261.txt.

[67] Netfilter-queue. http://www.netfilter.org/projects/libnetfilter_queue/.

[68] J. Qiu and L. Gao, "Cam04-4: As path inference by exploiting known as paths," in *GLOBECOM '06*, 2006.

[69] R. Jain, S. Member, Shawn, and A. Routhier, "Packet trainsmeasurements and a new model for computer network traffic," *IEEE Journal on Selected Areas in Communications*, vol. 4, pp. 986–995, 1986.

[70] "Reed-solomon forward error correction (fec) schemes," http://www.ietf.org/rfc/rfc5510.txt.

[71] Ekiga. http://ekiga.org/.

[72] OpenSSL. www.openssl.org.

[73] "Extensible messaging and presence protocol (xmpp): Core," http://www.ietf.org/rfc/rfc3920.txt.

[74] XMPPPY. http://xmpppy.sourceforge.net/.

[75] QtWebKit. http://trac.webkit.org/wiki/QtWebKit.

[76] Cisco, "Voice over ip – per call bandwidth consumption," http://www.cisco.com/application/pdf/paws/7934/bwidth_consume.pdf.

[77] IP geolocation database. http://ipinfodb.com/.

[78] L. Gao, “On inferring autonomous system relationships in the internet,” *IEEE/ACM Trans. Netw.*, vol. 9, pp. 733–745, December 2001. [Online]. Available: http://dx.doi.org/10.1109/90.974527

[79] “Top 50 autonomous systems,” http://cyber.law.harvard.edu/netmaps/country_detail.php/?cc=CN.

[80] X. Wang, S. Chen, and S. Jajodia, “Network flow watermarking attack on low-latency anonymous communication systems,” in *IEEE Oakland*, 2007.

[81] A. Houmansadr, C. Brubaker, and V. Shmatikov, “The parrot is dead: Observing unobservable network communications,” in *IEEE Security and Privacy (Oakland)*, 2013.

[82] T. Zhu, D. Phipps, A. Pridgen, J. R. Crandall, and D. S. Wallach, “The velocity of censorship: High-fidelity detection of microblog post deletions,” in *arXiv:1303.0597*, 2013.

APPENDIX A

DEALING WITH DUPLICATE BRIDGES IN THE PRIVACY-PRESERVING RBRIDGE SCHEME

Suppose $\mathtt{U}$ 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 $\mathtt{U}$ 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 $\mathtt{D}$ may try to infer $\mathtt{U}$'s bridges by constructing the list of available bridges used in OT with a single bridge (say B^*), and see if later $\mathtt{U}$ requests replacement of a duplicate bridge; if yes, $\mathtt{D}$ can be sure that B^* is one of $\mathtt{U}$'s existing bridges. To prevent this, we let $\mathtt{D}$ 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, $\mathtt{U}$ randomly picks C_p, C_q, $p \neq q$, and asks $\mathtt{D}$ to prove that B_p and B_q are different. $\mathtt{D}$ 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, $\mathtt{U}$ runs $\binom{m}{1}$-OT to get O_d; using O_d, $\mathtt{U}$ is able to open B_d from C_d.

If B_d is duplicate, $\mathtt{U}$ constructs the following proof:

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

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

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

APPENDIX B

CRYPTOGRAPHIC CONSTRUCTION DETAILS OF THE PRIVACY-PRESERVING RBRIDGE SCHEME

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. $\mathtt{D}$ chooses $sk \overset{R}{\leftarrow} Z_p^*$ 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, $\mathtt{D}$ chooses a random secret $scrt_{\mathtt{D}}$ from Z_p^* to generate invitation tickets.

B.2 Registration

$\mathtt{U}$ first picks a list of nonces $y'_j \overset{R}{\leftarrow} Z_p^*$, $1 \le j \le m$, 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 $\mathtt{D}$.

$\mathtt{D}$ verifies π_0, and then chooses a pair of one-time keys $sk^o \overset{R}{\leftarrow} Z_p^*$, $pk^o = h^{sk^o}$. For each available bridge B_j, $\mathtt{D}$ randomly selects $e_j^o, y''_j \overset{R}{\leftarrow} Z_p^*$, 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, $\mathtt{U}$ receives $\{B_i \| (A_i^o, e_i^o, y''_i)\}_{i=1}^k$. For each $i \in [1, k]$, $\mathtt{U}$ 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, $\mathtt{U}$ picks $r_i^{(1)}, r_i^{(2)} \overset{R}{\leftarrow} Z_p^*$, 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, $\mathtt{U}$ 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 $i \in [1, k]$, $\mathtt{U}$ 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, $\mathtt{U}$ computes the proof:

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

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

After verifying π_1 and κ_x, $\mathtt{D}$ 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 $i \in [1, k]$, $\mathtt{D}$ 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 $\mathtt{D}$ sends $(A_\Phi, e_\Phi, s''_\Phi) \| (A_\omega, e_\omega, s''_\omega) \| \{(A_i, e_i, s''_i)\}_{i=1}^k$ to $\mathtt{U}$.

$\mathtt{U}$ 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 $\mathtt{U}$ wants to update his credit balance with the credits earned from B_u. $\mathtt{U}$ needs to prove possession of σ_u and σ_Φ. For that, $\mathtt{U}$ 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:

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

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.4 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.5 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:

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

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 $\mathtt{D}$.

After verifying π_5, κ_Φ and κ_ω, $\mathtt{D}$ picks $\tilde{e}_\Phi, \tilde{s}''_\Phi, \tilde{e}_\omega, \tilde{s}''_\omega \overset{R}{\leftarrow} Z_p^*$, 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 $\mathtt{U}$. . Then, $\mathtt{D}$ flips a biased coin to decide whether to give an invitation ticket to $\mathtt{U}$; if so, $\mathtt{D}$ generates an one-time ticket $tk = \{r^*, \mathsf{HMAC}_{scrt_{\mathtt{D}}}(r^*)\}$, where $r^* \overset{R}{\leftarrow} Z_p^*$, and sends it to $\mathtt{U}$.

Regardless of receiving an invitation ticket, $\mathtt{U}$ 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 $\tilde{\sigma}_\omega$.