Lox: Protecting the Social Graph in Bridge Distribution

by

Lindsey Tulloch

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Licence

Abstract

Access to the open Internet, free from surveillance and censorship, is an important part of fulfilling the right to privacy. Despite this, in many regions of the world, censorship of the Internet is used to limit access to information, monitor the activity of Internet users and quash dissent. Where access to the Internet is heavily censored, anti-censorship proxies, or bridges, can offer a connection to journalists, dissidents and members of oppressed groups who seek access to the Internet beyond a censor's area of influence.

Bridges are an anti-censorship tool that can provide users inside censored regions with a link to the open Internet. Using bridges as an anti-censorship tool is fraught with risks for users inside the censored region who may face persecution if they are discovered using or requesting bridges. Bridge distribution systems that are built for widespread public distribution of bridges face the inherently conflicting issues of extending bridges to unknown users when some of them may be malicious. If not designed with care, bridge distribution systems can be quickly compromised or overwhelmed by attacks from censors and their automated agents and leak user and usage data, undermining the integrity of the system and the safety of users. It is therefore crucial to prioritize protecting users when developing such systems.

In this work, we take a holistic and realistic view of the bridge distribution problem. We analyze known threats to deployed bridge distribution systems, (i.e., The Tor Project's BridgeDB), and combine insights from prior work to create a new bridge distribution system. To this end, we propose Lox, a bridge distribution system that is open to anyone while also leveraging users' trust networks to distribute bridges. Lox protects the privacy of users and their social graphs and limits the malicious behaviour of censors.

We use an updated unlinkable multi-show anonymous credential scheme, suitable for a single credential issuer and verifier, to protect bridge users and their social networks from being identified by malicious actors. We formalize a trust level scheme that is compatible with anonymous credentials and effectively limits malicious behaviour while maintaining user anonymity. Our work includes a full system design of Lox, as well as an implementation of each of Lox's protocols. We evaluate the efficiency of our Lox protocols and show that they have reasonable performance and latency for the expected user base of our system, thus demonstrating Lox as a practical bridge distribution system.

Acknowledgements

I would like to thank my supervisor Ian Goldberg for his unwavering support, understanding, and guidance in completing this thesis during the global COVID-19 pandemic. When the challenges inherent in keeping my family safe and continuing my research seemed at odds, his enthusiasm and patience reassured me that "very little is impossible"—akin to constructing zero-knowledge systems.

This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo. This research was undertaken, in part, thanks to funding from the Canada Research Chairs program, the National Science and Engineering Research Council, the Ontario Graduate Scholarship and Zonta International.

I am grateful to my thesis committee: Urs Hengartner and Diogo Barradas for offering their time, effort, and perspectives on this thesis. I would also like to thank the CrySP lab and honorary member(s) for all of the experiences and friendships made there. In particular (and in no particular order), I would like to acknowledge and thank: Cecylia, Miti, Justin, Masoumeh, Anna, Nik, Shannon, Rasoul, Navid, Matt, Simon, and Thomas, who worked with me on projects or shared knowledge, advice, joy, rejections, long-nights before deadlines, and the experience of grad school during these very unusual times. Even though the lab experience was cut far too short during my time at Waterloo, the times that we did have in the lab, and later, outside the lab are some of my favourite memories from this time period.

Thank you also to friends near and far that continue to encourage and support my endeavours and bring joy to my life.

I am incredibly grateful to my family for their love and emotional support. They have supported me endlessly in every endeavour and I am forever grateful for this blessing.

Dedication

This thesis is dedicated to Ogadinma and Amarachukwu.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Privacy underpins the essential freedoms of expression and belief, and the freedom from fear that are required for individual autonomy. Despite this, privacy is increasingly under threat, whether through mass surveillance of online activity by nation states and corporations, targeted surveillance and harassment of indivduals through spyware and stalkerware, or censorship of the free and open Internet. In a 2021 report by Freedom House [SF21], Internet freedom was found to have declined for the 11th consecutive year with large tech companies increasingly capitulating to repressive governments' requests to filter and censor content from social media platforms and the wider Internet.

Internet censorship takes many forms depending on the capabilities and motivations of the censor, from repressing self-expression deemed to be at odds with desired norms, to controlling the political landscape and information available to users within the regions they control. Recent work demonstrates the variability of Internet censorship employed around the world. In a collaborative report by OutRight Action International, the University of Toronto's Citizen Lab and the Open Observatory of Network Interfence (OONI), Dalek et al. [DDK+21] analyze the methods employed by censors across six regions (Indonesia, Malaysia, Russia, Iran, Saudi Arabia and the United Arab Emirates) to censor LGBTIQ websites and related content. In other work, Padmanabhan et al. [PFX+21] examine the evolving face of Internet shutdowns and censorship in Burma (Myanmar) in the days preceding, during and after the military coup in February 2021. A report by OONI details the anomalies they detected in Tor network connections in Russia [XF21] beginning in December 2021, which indicated a sudden shift to widespread blocking of the Tor anonymity network by many Russian ISPs.

In the face of censorship, anti-censorship tools aim to provide journalists, activists, and individuals from marginalized groups with access to the free and open Internet. However, it is

difficult to create and deploy generalized anti-censorship tools that are robust and effective against well-resourced, ubiquitous censors aiming to prevent their use. Additionally, using such tools may be dangerous and even life-threatening to users so protecting the privacy of users must be at the forefront of their design.

The Tor [DMS04] anonymity network allows users to anonymously browse the Internet and has proven to be an important tool to evade Internet censorship and surveillance. Unfortunately, Tor's widespread popularity as a censorship circumvention tool makes it an obvious target of blocking by censors [XF21, Lew10, FT16, EFW+15]. Where access to Tor and the wider Internet is heavily censored, *bridges*, or anti-censorship proxies, must be used to connect. The Tor Project's existing bridge distribution mechanism, BridgeDB [Tor22c], provides adequate support for some users; for example, use of Tor bridges in Russia rose sharply at the above December 2021 blockage, and rose further in late February 2022 [Tor22h]. However, passive and active detection techniques such as traffic flow analysis [BSR18], Deep Packet Inspection (DPI) [WL12], website fingerprinting [CZJJ12], and active probing, have been demonstrated in prior work to reveal Tor bridges [DOG18, WL12, EFW+15], making Tor inaccessible for the vast majority of users in some regions [DOG18, Lew10].

Users that need a bridge to connect to the Internet need some way of receiving that bridge that is resistant to censorship. However, when the user is not known to the bridge distributor, it is possible for censors to pose as genuine users in order to enumerate and block bridges. The bridge distribution problem presents the inherently conflicting issues of wanting to provide bridges to unknown users while protecting the network from users that may be malicious. Prior work on bridge distribution has addressed this inherent conflict from several different angles. In Proximax, McCoy et al. [MML11] strive to maximize the overall user-hours of the system's bridges by disseminating bridges to new users through trusted users that have the greatest number of user-hours, calculated collectively among all of the users they have successfully invited. While Proximax does not provide privacy or anonymity to users and relies on the exclusivity of its trusted users to protect the system from malicious users, it introduces the concepts of a behaviour-based bridge distribution system that prioritizes distribution through a user's social network.

Aiming to introduce a system that is open to all users, Douglas et al. [DRPC16] devise the Salmon scheme, which uses a trust level hierarchy to extend and limit privileges based on user behaviour within the system. Trust levels can be inherited through being invited to the system by a highly trusted user as well as accrued over time while bridges that a user knows of remain unblocked. While Salmon's trust levels are intriguing, like Proximax, the Salmon scheme fails to provide robust privacy guarantees to users. Their use of Facebook to verify users as well as their tracking of the recommendation graph makes the distribution server a prime target for an adversary who desires information about bridge users and their social network.

Wang et al.'s rBridge [WLBH13] and Lovecruft and de Valence's Hyphae [LdV17] both focus on the importance of privacy protection in bridge distribution and each present a system, differing in their anonymous credential scheme, that protects user privacy and anonymity while disseminating bridges through users with records of good behaviour. Hyphae points out that because the bridge authority can act as both the issuer and verifier of credentials, a much simpler and more efficient anonymous credential scheme, Chase et al.'s keyed algebraic MAC credentials [CMZ14], can be used in place of rBridge's k-TAA [ASM06] credentials that require elliptic curve pairings. In this work, we follow the lead of Hyphae to design our bridge distribution system with the use of keyed anonymous MAC credentials.

By combining Salmon's trust levels into a formalized privacy protective reputation scheme using Chase et al.'s keyed algebraic MAC anonymous credentials, we can combine desirable elements from prior systems to limit malicious behaviour while providing privacy protection to users and their social graphs.

In this work, we present Lox, a new reputation-based bridge distribution system that provides privacy protection to users and their social graph and is open to all users.

1.1 Thesis Statement

In this thesis, we will show that it is possible to design a bridge distribution system that is open to all users and leverages users' trust networks for bridge distribution while protecting their privacy as well as all connections in their social graphs.

1.2 Contributions

The primary contributions of this thesis are:

1. A new design for a bridge distribution system, Lox, that:
    - **leverages a user's trust network** to distribute bridges
    - provides **open access** to untrusted users as well as elevated access to users invited by a trusted user
    - provides **privacy protection for users and their social graphs** through the use of an unlinkable multi-show anonymous credential scheme

- formalizes a **trust level scheme** that is **compatible with anonymous credentials**
- **defends against malicious behaviour** by censors and automated agents of a censor

2. An **evaluation of the performance of Lox** with a Rust implementation of our Lox protocols, demonstrating that Lox achieves reasonable performance for small bridgepools ($\sim$3600 bridges, which is comparable to Tor's current bridgepool)

Chapter 2

Censorship Circumvention

Censorship of the Internet takes many forms, matching the varied motivations and goals of state-level censors around the world. Censorship resistance tools evolve rapidly to counter these goals and provide people inside censored regions with access to the open Internet. In this thesis we focus on distributing bridges; that is, proxy connections, to users in regions where other entry points to the open Internet are compromised or otherwise inaccessible. To center the bridge distribution problem and motivate our work, in this chapter, we give an overview of censorship circumvention research and discuss known attacks and techniques used by censors to discover and block connections to the open Internet as well as censorship resistance tools that have been developed in response.

We begin by defining some terms that will be used throughout the thesis.

2.1 Definitions

Bridge

A **bridge** is a server that provides a connection to the Internet for users that connect to it. In our use of the word bridge, we envision a server that is set up with the intention of being always online to both provide bandwidth to connecting users and report some statistics about usage. A Tor bridge is similar to a Tor relay (i.e., a volunteer run node in the Tor anonymity network) in terms of functionality but is not listed in the Tor consensus. Tor bridges therefore act as the first hop into the Tor network, prior to connecting to the three Tor relay nodes required for a Tor connection. By not being listed in the public directory of Tor relays, they are harder for a censor

to enumerate and block. Distributing a bridge to a user involves distributing information about the bridge such as an IP address and port, and cryptographic keys.

Social Graph

We use the term **social graph** to refer to a given bridge user's connections to real people inside and possibly outside the network. We assume that a bridge user may use invitations to invite trusted friends who they have a real connection to outside of our bridge system. When we refer to protecting a user's social graph, we mean that the bridge authority itself, or any entity that gains control of the bridge authority, will be unable to link users to one another and thus will be unable to use our system to reconstruct the real-life connections between users of our system.

Sock-puppet

We use the term **sock-puppet**, in place of the term sybil, to refer to a censor creating multiple requests to use the system in a malicious way. We assume censors have the following intention when creating sock-puppets: to prevent legitimate users from acquiring new bridges by blocking bridges immediately upon receiving them, by building trust in the system to block bridges en masse at a predetermined critical moment, such as a political event, or by acting in any other way that prevents or hinders the use of bridges by genuine users.

2.2 Internet Censorship and Resistance Techniques

State-level censors are assumed to possess unlimited time and resources to direct towards their censorship goals. This includes the time and energy to invest in reverse engineering and experimenting with censorship circumvention systems in order to undermine them. In developing censorship resistant systems, it is useful to begin from Kerckhoffs' principle; that is, to ensure the security of the system is upheld even if the entire system, except for the key, is known to the adversary. To meet Kerckhoffs' principle, censorship resistance tools are designed to meet at least one of a small set of general anti-censorship goals: to make continued censorship more time consuming and challenging for the censor, to embed their tools in essential services in order to inflict maximal collateral damage through blocking them, or to operate covertly as long as possible without being detected by a censor by blending in with or mimicking uncensored connections. The tools that have emerged to match these censorship resistance goals have different strengths and weaknesses and have had to evolve to meet the adaptive techniques of censors.

2.2.1 Circumventing IP and Keyword Blocking

Tor [DMS04] was designed to allow users to anonymously connect to the Internet. Tor connections involve three hops: an entry node into the network, followed by a middle node and finally an exit node that passes off the request to the destination as in a typical connection. When a user connects to Tor, a publicly available *consensus* of all entry, middle and exit nodes in the network is downloaded in order to map a connection and successfully navigate through the network. Tor uses onion routing, where users' requests are encapsulated in three layers of encryption as they travel, with each node successively peeling back or decrypting a single layer of encryption until it exits the network at the end node. This prevents any one node from learning both the IP address of the user and their requested traffic. Though not originally intended as a censorship circumvention tool, connections to Tor provide anonymous access to the open Internet for many users.

Since Tor was not originally developed as a censorship circumvention tool, vanilla Tor, meaning the official Tor client alone, does not incorporate any of the anti-censorship goals mentioned above. Indeed, Tor's publicly available consensus makes blocking direct access to relays within the Tor network trivial to censors. Tor bridges [DM06], which are a set of unlisted relays, were introduced to overcome this, enabling access to the Tor network through an additional hop for anyone knowing the bridge's IP address. Without further protective measures however, Tor bridges have been shown to be an insufficient circumvention technique against a censor's passive detection and active probing techniques [DOG18].

2.2.2 Censors' Passive Detection and Active Probing Techniques

Passive detection techniques are automated processes that are easily deployed by censors and can block access to blacklisted domains with minimal collateral damage. Examples of passive detection techniques employed at the network level are traffic flow analysis [BSR18], where the censor compares the expected traffic pattern from a given destination with the observed traffic to detect differences in packet sizes and timings, and Deep Packet Inspection (DPI) [WL12], where the censor looks at the contents of data packets and headers. Examples of passive detection techniques employed at the application level are website fingerprinting [CZJJ12], where the censor uses packet sizes and timings to determine the particular websites that are being viewed, and substring searches, where the censor searches for particular substrings within packets that are sniffed. Passive detection techniques have been demonstrated in prior work to reveal Tor bridges [DOG18, WL12, EFW+15].

Pluggable transports (PTs), first proposed as a defence against passive detection [KMDA10], were designed to be interchangable add-ons to a connection that obfuscates traffic at the link layer.

PTs prevent censors from identifying traffic as suspicious and blocking connections to their destinations. However, tools like ZMap, which enable fast Internet-wide network scanning [DWH13], demonstrated that censors can perform active probing with just slightly more effort by initiating censorship-resistance protocols to every IP address and probing those that respond. A more surgical approach to active probing involves sending automated probes to an IP address and port deemed suspicious, attempting to use a censored protocol to communicate with the destination server, and then blocking the address if the request is successful. This attack, used in combination with passive probing, allows the censor to be more stringent in allowing connections through the passive detection phase since active probing greatly improves the precision of determining threats, thereby reducing collateral damage. Indeed, defending against passive detection alone was proven to be insufficient in work by Ensafi et al. [EFW+15] that presented evidence of active probing behaviour used to block Tor bridges in China.

Nobori and Sinjo showed that VPNs are subject to similar obstacles when used as censorship circumvention tools [NS14]. Without further intervention to obfuscate their use, they are vulnerable to censors' passive and active probing at the network and application levels.

2.2.3 Circumventing Passive and Active Censorship

In response to this, probe-resistant bridges (or proxies) have been developed that typically involve randomization or padding of packet sizes and timings to limit the likelihood of being detected. Probe-resistant bridges necessarily include a secret that is distributed to users out of band and that the user must prove knowledge of in order to make a successful request.

Obfsproxy, ScrambleSuit, FTE, and Meek are popular and effective probe-resistant pluggable transports that are currently distributed along with the Tor browser. Obfsproxy [Din12] has had several iterations (obfs2, obfs3, and obfs4) with obfs4 being the most popular, up to date and least vulnerable to attacks by censors. Obfsproxy is based on a 'look-like-nothing' design with the idea that a censor that has successfully filtered specific protocols based on their unique features would fail to identify obfsproxy. Obfs4 requires that the user is able to prove knowledge of a secret before connecting to a bridge, making it difficult for a probing censor to confirm that an obfs4 connection is being used to connect to bridges. ScrambleSuit [WPF13] uses a similar randomization technique and also alters the timing between packets sent and the lengths of packets. Format-transforming ecnryption FTE [DCRS13] exploits modern DPI system's use of regular expressions by encoding data in strategically constructed strings specifying white-listed protocols such as HTTP and SSH that are not identified as suspicious by DPI systems. Meek [FHE+12] takes a different approach by relying on domain fronting to relay requests through large cloud providers that would be difficult for a censor to block without incurring

significant collateral damage [FLH+15]. This strategy has proven to be successful but can be expensive to maintain and requires the continued cooperation of large cloud providers that may be unreliable for censorship circumvention in the long term due to conflicting interests [Doe21].

Recent work by Frolov et al. [FWW20] showed that even some active probe resistant pluggable transports are susceptible to discovery by censors through fingerprintable patterns in when and how their connections are closed. This led to Frolov and Wustrow's proposal of HTTPT proxy [FW20] which uses HTTPS and functions similarly to a TLS server with a secret bridge, allowing users who know the secret key access to the HTTPT proxy and returning a benign response to probes. The ubiquity of HTTPS makes pluggable transports that utilize it for censorship virtually impossible to detect and block without knowing the address of the bridge.

Alternative approaches to bypassing censors' probes include multimedia based covert streaming [BSRN20, BSR17] and decoy routing [BG16, FDS+17, NZH17, FWT+19, VFW+20]. Each of these solutions involves steganographic mimicry or passing off information concealed within existing Internet infrastructure in a way that is completely undifferentiable from regular traffic and therefore less likely to be recognized as suspicious by censors. These systems increase the burden on the censor by requiring them to run clients compatible with the system and provide a secret in order to confirm the connection is succesful. In Protozoa [BSRN20], Barradas et al. use WebRTC to tunnel covert traffic in place of multimedia traffic between two users in a peer-to-peer connection. Protozoa's design, like HTTPT's, makes detection and blocking extremely difficult for censors without causing an intolerable amount of collatoral damage. However, making use of the system in its current state requires a pre-existing connection and a non-trivial amount of coordination between users in censored regions and those in uncensored regions.

2.2.4 Distribution of Probe-resistant Bridges

With several options among probe-resistant bridges and transports for effective censorship circumvention, the problem remains of how to distribute these bridges to genuine users without making them vulnerable to blocking by censors. Tor's BridgeDB [Tor22c] distributes probe-resistant Tor bridges and their secrets, through email, an HTTPS website, and as an option in the settings of Tor browser. These distribution methods make Tor bridges publicly available to users but censors can easily block the https sites that distribute Tor bridges and Tor browser itself. Some tools such as SWEET [HZCB17] and dnstt [Fif21, KJQ+21], which use common protocols such as email (SWEET) and DNS (dnstt) as cover, provide robust censorship circumvention for very low-bandwidth activities such as initial bootstrapping, secret/bridge address distribution, and sending apps and other censorship circumvention tools to users [Gal19]. However, publicly available distribution channels are suceptible to sock-puppet attacks, where censors pose as mul-

tiple genuine users in order to learn many bridge addresses and secrets in order to block them. Though thought to be a more resource intensive task for censors, particularly aggressive censors have been successful in sniffing out and blocking most reliable probe-resistant bridges, making Tor and the open Internet inaccessible for the majority of users in some regions [DOG18, Lew10].

2.3 Chapter Summary

In this chapter, we have given an overview of the evolving face of Internet censorship techniques and censorship resistance tools to counter them. We have described how bridges, VPNs and pluggable transports interact to provide censorship resistance against censors' passive and active probing techniques. While we have seen that connecting to probe-resistant bridges and VPNs using a shared secret is a successful strategy for censorship circumvention, the problem of distributing these addresses and secrets remains.

Existing public distribution systems for censorship resistant bridges are vulnerable to sock-puppet attacks, where censors pose as genuine users to enumerate bridges and VPN addresses, blocking access to them for genuine users. Distributing bridges effectively in a way that is resistant to blocking by censors and limits malicious behaviour remains an open problem. In the next chapter we take a more in-depth look at the current state of bridge distribution and several posited solutions to the bridge distribution problem.

Chapter 3

Bridge Distribution

The bridge distribution problem, in its simplest construction, focuses on distributing viable network bridges to honest users in the presence of a censor that aspires to prevent bridge use. As explored in previous work [WLBH13, LdV17, NFHG19, DRPC16], designing and implementing such a system for practical use requires an understanding of how censors operate in different regions. This includes the resources at a censor's disposal, the tools they are known to deploy and the known dangers to individuals who are discovered using censorship circumvention tools. While the capabilities, methods, and motivations of censors vary widely across the globe, in designing a system with potentially global reach, it is useful to consider the most resourced, capable and strict censor that will actively target the system and punish users who are discovered. In designing a system to meet this type of censor, we prioritize safety and accessibility for users under even the most severe conditions. This allows us to weigh the cost of such a system for users that do not require such extreme measures.

The Tor Project, having deployed a globally used bridge distribution system, offers further insight into the bridge distribution problem based on direct evidence from monitoring logs and regional users. Research in the bridge distribution space has offered a range of solutions based on various strategies used by censors.

In this chapter we discuss insights gained from the Tor Project's bridge distribution metrics as well as blog posts and research that has discussed experiences with bridge distribution at Tor. We then discuss prior research in the bridge distribution space and how these works have proposed to leverage trust networks, protect the social graph, and limit malicious behaviour when distributing bridges. Creating a system that improves on each of these overarching concepts motivates our work on Lox.

3.1 Insights from Tor Bridge Distribution

The Tor network [DMS04], originally designed to provide anonymity online, consists of over 6000 volunteer-run relays operating around the world. Each connection to Tor involves three hops through specific types of volunteer-run relays called guard, middle and exit nodes. The connection to each node is encrypted such that the first node (guard) can decrypt the first layer to learn the address of the second hop (middle) which can decrypt the second layer to learn the address of the third hop (exit). The exit node can then decrypt the final layer to learn the desired destination of the user. In this way, no one node in the connection can link the user and their desired destination. A connection to the Tor network is made through the Tor Browser by first consulting the Tor consensus to determine the specific guard, middle and exit nodes that will be used before reaching the requested destination.

The anonymity that Tor provides users has made it a popular censorship circumvention tool as well as a frequent target of blocking by censors. However, Tor was not originally designed as a censorship circumvention tool [DM06] and there are many elements of Tor's design that are not particularly suited for censorship circumvention. Aside from the unique features that distinguish Tor traffic, Tor relays are publicly listed, making it trivial for a censor to block access to the Tor network by blocking all public entry (guard) nodes.

Pluggable transports, proxies and bridges, discussed in detail in Section 2.2, have all been introduced to extend the reach of Tor to those in regions where direct access to the Tor network is blocked. We will focus on Tor bridges as they are the most relevant for our system, but note that this is not always the best choice for censorship circumvention. Where Tor guard nodes have not been blocked by a censor, pluggable transports may be sufficient for a user to safely connect to Tor. Bridges are needed most where censors have blocked all access to Tor guard nodes and the user needs an alternative stable connection to access Tor.

Tor metrics [LMD10] puts the current number of Tor bridges around 3500. Using bridgepool data collected from Tor metrics, we created Figure 3.1 which shows the rate of bridge replacement over the course of 2021, which we call *bridge churn*. From this data we can see that in the period we measured, the bridges that existed in the BridgeDB slowly decreased over time while it took approximately five months for the number of bridges to fully churn once; that is, for the number of new bridges to surpass the number of bridge fingerprints in the initial sample. We also see that bridges sometimes go offline temporarily and return, shown by slight dips in the graph that return to the linear trend. We note that a small number of daily bridgepool records (approximately 10 non-consecutive records) failed to report any bridges for an unknown reason. For these records we carried forward the bridges from the previous entry.

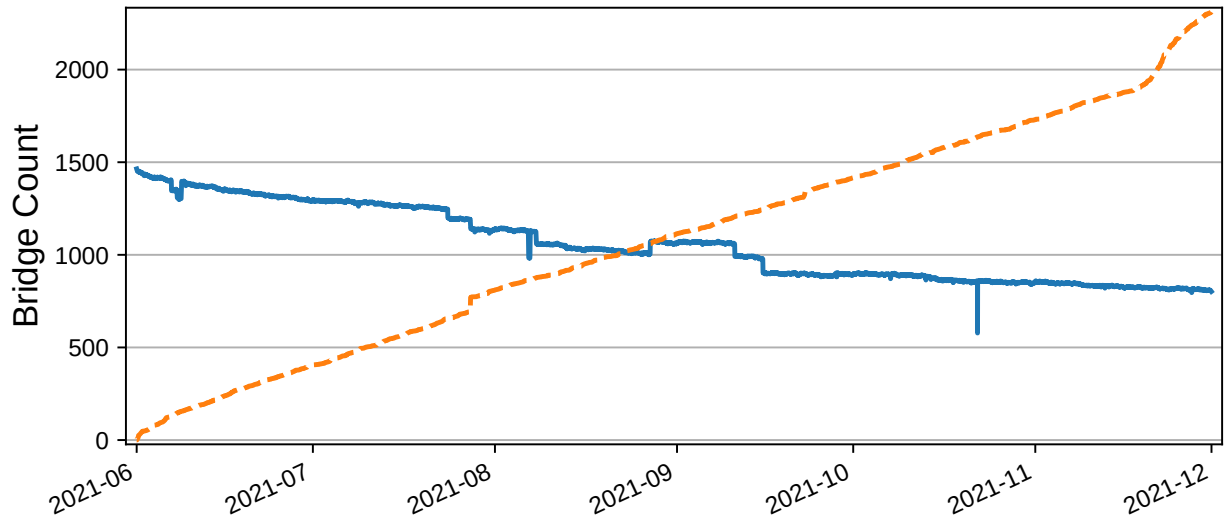


Figure 3.1: To estimate Tor bridge churn for 2021, bridge data was collected from Collector's bridgepool assignment archive [Tor22d] from June to December 2021. Collector maintains a public list of fingerprints for many of the bridges in Tor's bridgepool. These fingerprints are anonymized such that they do not identify the bridges in Tor's bridgepool but can provide statistics about these bridges. Each bridge fingerprint in the pool on June 1st was recorded and compared against bridge fingerprints in the pool for each record until December 1st 2021. The blue line represents the fingerprints that were in the archive in the initial sample and tracks the count of fingerprints that remain in the archive over time. The orange line represents fingerprints that were added to the archive since the initial sample. We see that the number of bridges in our original sample gradually decreases over the period measured and new bridges are added at a rate of approximately 500 bridges/month. After about five months, enough new bridges are added to replace the entire bridgepool.

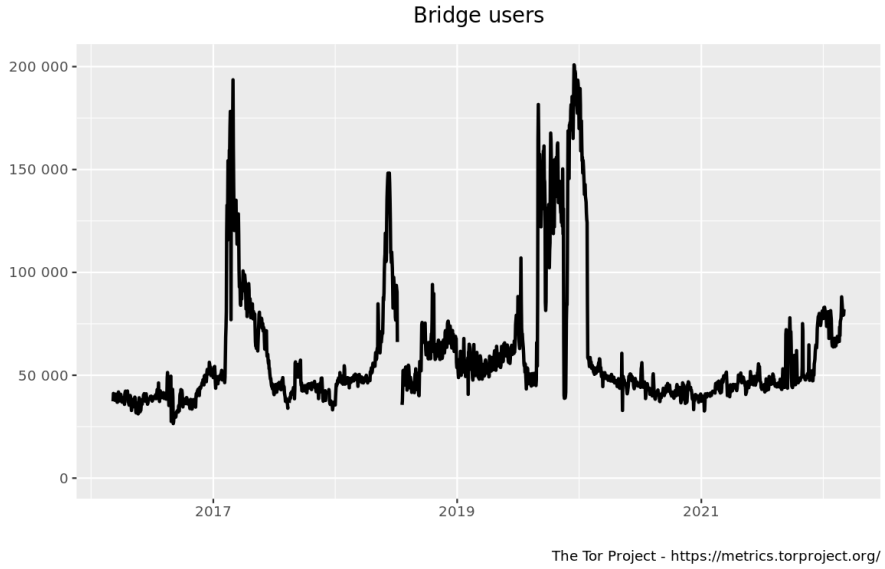


Figure 3.2: Estimates of the number of Tor bridge users from March 7th, 2016 – March 7th, 2022 [Tor22a]

Querying Tor metrics for data between 2016 and 2022 shows that the daily number of Tor bridge users can range between 30,000 and 200,000 as shown in Figure 3.2 [Tor22a]. While this gives an overall sense of bridge usage the Tor project sees, it does not give a clear understanding of how these users are distributed over bridges nor how or if this distribution changes over time. Preserving the anonymity of users makes it challenging for the Tor project to accurately report on the number of users that use relays or bridges. Some noise is applied to data collected about the number of Tor bridge users in order to uphold anonymity of users. In cases where the number of average users is very small, data may not be reported until the user base grows. Nonetheless, to get a better sense of the average number of bridge users a particular new bridge had seen over the period of one month, we queried the number of users for one month from all bridge hashes that appeared on Onionoo [Tor22e] with dates beginning after November 1st, 2021. We note that for each of these anonymized, public bridge fingerprints, Onionoo records the average number of users each day for the length of time the bridge has been online and the average has been above some threshold. Our plot in Fig 3.3 shows the average of these averages as well as the maximum of these averages for bridges that appeared after November 1st, 2021. We can see that the vast majority of bridges have relatively few users (8–15) with a small number of bridges having a larger number of users (20–60).

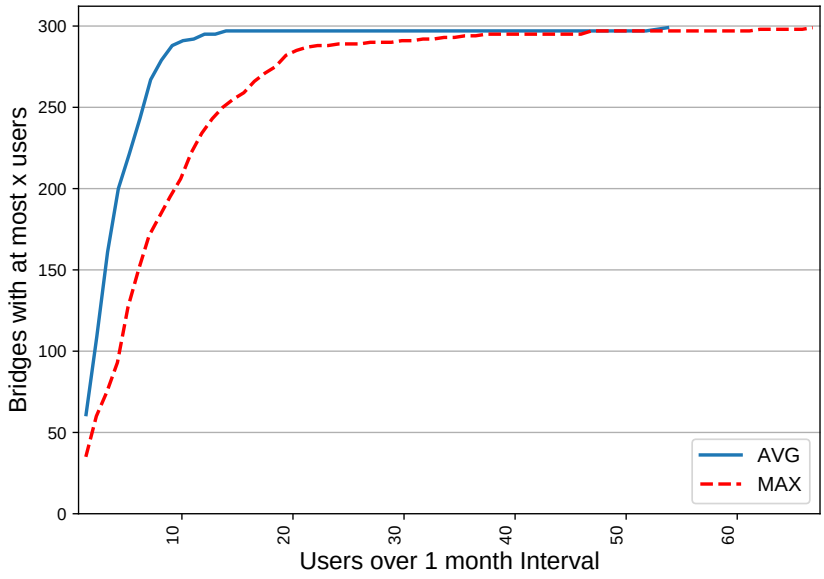
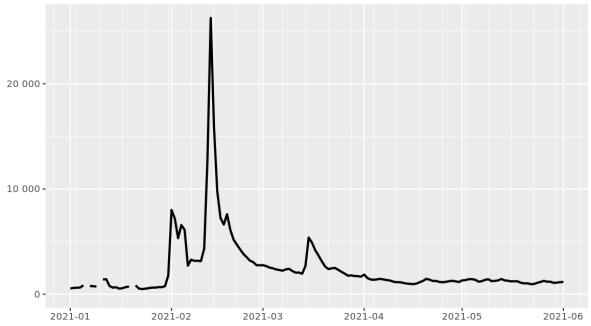


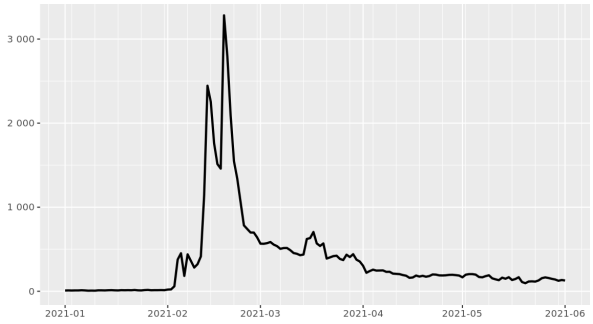
Figure 3.3: For all bridge fingerprints with a first seen date in November 2021, average user data was collected from onionoo [Tor22e]. The cumulative distribution of the average number of users is shown with the solid blue line and the same for maximum users for each fingerprint is shown with the dashed red line.

We expect that bridge requests and usage to increase during a censorship event. A recent study by Padmanabhan et al. [PFX+21] looking at Internet censorship in Myanmar (Burma) during the military coup of February 2021 gave a timeline of events from February 1st, when the coup occurred, to April 28th, when nightly Internet outages ended. When comparing these dates with relay and bridge usage data from the Tor project, we can indeed see increases in both relay and bridge use during this period in Burma as shown in Figures 3.4a and 3.4b.

Tor bridges provide an entry point to the Tor network that is not publicly listed and thus, not as easily identified by IP-blocking censors [Din11]. Tor bridges are still accessible through other public methods. Some bridges, called moat bridges, are distributed with the Tor browser and can be configured automatically or manually. Others are accessible through email by sending an email from a valid gmail, yahoo or riseup email account to `bridges@torproject.org` or by HTTPS, by visiting the Tor bridges website. Each of these methods distributes different bridges so that a censor is unable to enumerate all bridges using only one of these methods. The email method, requiring a valid yahoo, gmail or riseup account, out-sources the user validation to a third party service with the hope of limiting sock-puppet email requests. Despite these



(a) Direct connections through Tor relays in Myanmar January 1st – June 1st, 2021.



(b) Bridge connections to Tor in Myanmar January 1st – June 1st, 2021

Figure 3.4: Users from Myanmar (Burma) connecting directly to the Tor network through relays (Figure 3.4a [Tor22f]) and bridges (Figure 3.4b [Tor22b]) between January 1st and June 1st, 2021 as collected by Tor metrics. On February 1st, the day of the coup in Myanmar, there was a partial Internet shutdown. Following that from the 6th – 7th was a complete Internet shutdown. Then from February 15 – April 27 there were nightly shutdowns and blocking of wireless broadband Internet. We see an increase in both relay and bridge usage during these periods with a much higher dependency on Tor relays (over 20000) than bridges (over 3000).

measures, prior work has demonstrated that most of the bridge relay addresses are still attainable through methods that could be abused by a determined censor with minimal resources [DOG18, EFW+15,MTC17,DWH13]. Indeed, Fifield et al. [FTZ17] showed that censors in some countries are able to block bridges almost immediately after they are accessed. Further, Fifield [Fif17] demonstrated that bridge relays alone are not sufficient in their current state for providing users undetectable access to Tor. This is especially apparent when the censor is highly resourced or sophisticated in the methods they employ to prevent users from accessing censored sites and services. Obfsproxy4 [Din12] can be used to disguise a bridge connection and prevent active probing censors from confirming that an unlisted bridge is indeed a bridge. However, if another user connects to this same bridge without Obfsproxy4, a censor may block the obfs4 port as well as the port that the user connected through. In response to these findings, the Tor project has attempted to limit obfs4 bridges to users that specifically request an obfs4 bridge [BDF+19]. Figure 3.5 illustrates several scenarios of users in a censored region and their interactions with Tor, pluggable transports and bridges.

Investigating Tor's experiences with bridge distribution, particularly the number of bridges and users, the rate of bridge churn and the documented threats posed by censors, provides important insights for the design and implementation of Lox. Notably, that Tor's bridge distribution methods

are insufficient to limit a well-resourced censor that can create numerous verified email accounts or continually request new bridges through the Tor bridges website without consequence. It is admittedly difficult to both guarantee user anonymity and prevent the behaviour of malicious users, but more could be done to limit the behaviour of malicious users and upgrade users with good, trustworthy behaviour. We argue that this can be done by leveraging a user's trust network (or trusted friends) to distribute bridges and that keeping the graph of these connections private is necessary to ensure user protection. In the subsequent sections we discuss bridge distribution systems that have been proposed in the research community and analyze how they have performed on these metrics.

3.2 Leveraging Trust Networks for Bridge Distribution

Distributing bridges by leveraging a user's trust network has been proposed as a means of gaining an advantage over a sophisticated censor that has more resources than any genuine user, but does not have honest friends [WLBH13, MML11, LdV17].

McCoy et al. [MML11] introduced Proximax, one of the first schemes that looked at trust networks as a bridge distribution channel. Proximax optimizes for system uptime by monitoring the user-hours of highly trusted (registered) users and their invitees who are connected in a tree-like structure. Users estimated to have the highest additive user-hours are chosen to distribute more bridges, thus maximizing the number of users attracted to the system and the length of time a bridge remains in operation. If malicious behaviour is detected anywhere along the branch, the whole branch is considered suspicious, damaging the reputation of all users in that branch and curtailing their ability to distribute invitations.

Proximax provides a blueprint for subsequent work leveraging trust networks for bridge distribution; however its reliance on a pool of trusted registered users makes it unsuitable as a general solution open to a wider pool of users. Furthermore, Proximax relies on the tracking of users and their social graph to monitor and punish suspicious behaviour. Usage data, in particular, learning the connections between users of censorship circumvention tools, may be particularly interesting information for censors.

Wang et al. [WLBH13] built on the Proximax scheme with rBridge, which similarly leverages a user's social network for bridge distribution and as the primary contribution, adds privacy protection for users and their social graph through the use of anonymous credentials. rBridge makes several additional design changes to Proximax's trust distribution scheme to limit malicious behaviour and increase the amount of time a bridge can be used before it is blocked. Users join the rBridge system through invitations and accrue credit by keeping their bridges unblocked. When

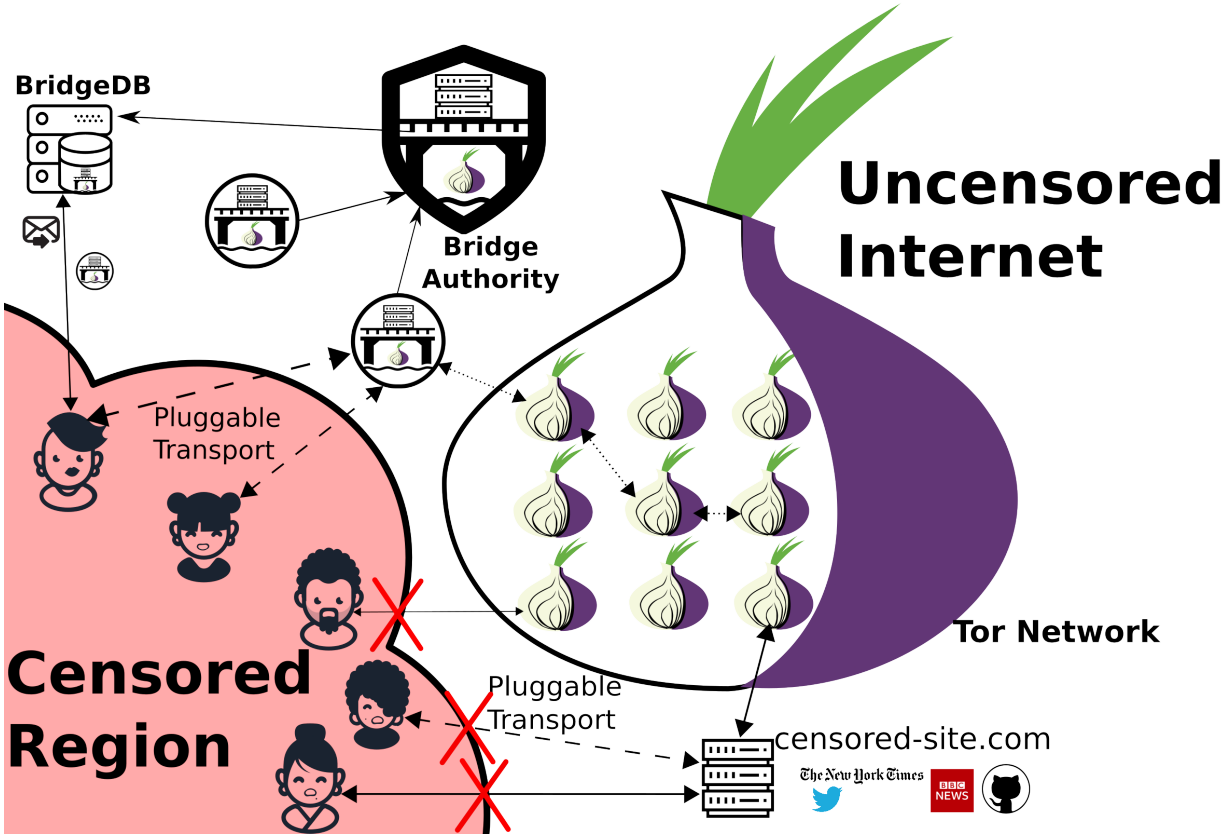


Figure 3.5: Overview of Tor bridge distribution and bridge usage among censored users. The censor has blocked access to the Tor Network as well as some specific sites outside of the censored region. Users cannot access Tor directly but are able to request bridges from the Tor BridgeDB by email and connect to an unblocked bridge using a pluggable transport. Users may have a friend running a bridge outside of the Tor network and may also be able to connect to the Internet outside of the censored region through such a bridge. In this diagram, connections made with a pluggable transport are indicated by a dashed line, encrypted connections in the Tor network are indicated by a dotted line and unobscured connections are indicated by an unbroken line. The red x's indicate blocking by the censor either by IP filtering, DNS poisoning or some other means.

presenting an invitation to the bridge authority (that acts as the issuer and verifier of credentials), users first receive k bridges through oblivious transfer with the authority so that the authority does not learn which bridges are being requested. Each is signed and tagged by the authority so that the user cannot replace them with other bridges. The user then requests their user credential, complete with commitments to their bridges (so the bridge authority does not learn the specific bridges but can still issue them in zero knowledge), and can begin accruing credit. When one of the user's bridges is blocked, they can report it and accrue unclaimed credit up until the blockage. If they have enough credit, they are able to exchange the credit for a new bridge. This allows users to build trust with good behaviour and avoid being locked out of the system permanently. Highly trusted users can request invitation tokens to invite their friends to the system but rBridge's bridge authority will only grant the request with a certain probability in order to encourage users to hand out invitations sparingly, to only their most trusted friends.

Lovecruft and de Valence's Hyphae [LdV17] leaves much of the rBridge protocol intact while replacing the anonymous credential scheme with keyed-verified algebraic MACs [CMZ14]. However, they omit the initial oblivious transfer, having the authority issue bridge tokens that users can immediately use to exchange for new bridges anonymously. They also propose reducing the amount of time required for a user to begin inviting friends to the system and allowing users to use their accrued trust credits to exchange for invitations much like they can exchange for new bridges if they are blocked.

Both rBridge and Hyphae operate under the assumption of an invite-only environment where the vast majority of users are not malicious. This makes their systems much more robust against censors with invitations less likely to be distributed to malicious users. If a censor *is* able to join the system, whether through initially being invited or else invited by a friend, a patient censor could continuously invite themself to the system at every opportunity and use these sock-puppet users to both increase the odds of further invites and enumerate more bridges. This strategy would be time-consuming and would require leaving bridges unblocked while sock-puppet users are propogated, suggesting a long period of bridge availability that may be sufficient for many systems. However, the exclusivity of the system means that it can only benefit a relatively small group that is already socially connected to one another. A system that is open to all users must go further to limit malicious behaviour of censors since the same behaviours that could eventually overwhelm rBridge and Hyphae, could happen much more quickly in an open system with a greater percentage of users acting maliciously.

Like in rBridge and Hyphae, Douglas et al.'s Salmon [DRPC16] scheme includes distributing bridges through users who have built their reputation over time by proving continued use of the system while bridges remain unblocked. The Salmon scheme additionally provides an entry point through proof of a valid account with some external entity that makes account duplication difficult (the authors explicitly use Facebook). The Salmon scheme tracks the trust level of users

as well as the quality of bridges and allows users to advance through deterministic levels of trust to unlock different privileges and access more reliable bridges. As a user increases their trust level, the bridges they know also gain in reliability and are removed from the pool of bridges distributed to untrusted users. When a user becomes eligible to invite friends, they are invited to the same bridges at an elevated trust level. This encourages users to be cautious when handing out invitations (to keep their bridges unblocked) and limits a malicious user's ability to enumerate bridges through gaining trust and inviting themselves. As long as no malicious users were added to a bridge before it stopped being distributed to untrusted users, the bridge can become immune from the malicious behaviour of a censor. Salmon also tracks a user's *suspiciousness*, which is increased each time a bridge they know about is blocked, whether they are responsible for blocking it or not. When a user's suspicion threshold has been reached, the user is banned from the system.

Salmon's use of trust levels and suspicion helps to limit malicious behaviour among a less trusted pool of users while allowing trusted users to distribute bridges and establish sequestered bridge groups that are immune to censors. While there are many appealing aspects to this design, Salmon does not specify the privacy protections provided to users while being monitored for trust and suspicion assignments. We expand on this in the following section.

Trust networks have been shown to be an effective means of distributing bridges that inherently disadvantages censors. However, systems that use this method may introduce a new target to censors if they are not designed to protect the privacy and anonymity of users and their social graph. Next we look into ways that we can protect the social graph in a system that relies on a user's trust network for bridge distribution.

3.3 Protecting the Social Graph

Douglas et al. [DRPC16] narrowly define an adversarial censor as one that simply blocks access to bridges rather than seeks to identify and persecute users and members of their social network. Using this threat model, Salmon tracks the invitees of trusted users and stores a subset of each user's social graph. The authors argue that a state-level censor would gain little information about a user's social network from Salmon server logs comparative to what they would be able to learn from their access to state-connected social media sites and other surveillance tools at their disposal.

While we concede that a sophisticated and determined censor may be able to undermine most privacy preserving mitigations of a bridge distribution system, it does not follow that privacy protection for users and their social networks should be abandoned altogether. As noted earlier,

censorship strategies vary greatly across regions and can change over time. It is impossible to know how a censor might uniquely use server logs that provide clear evidence of individuals working together to evade censorship.

With this understanding in mind, we consider a censor that may seek out and punish censorship circumvention system users and their networks if they can be identified. We thus strive to make it impractical, if not impossible for a censor to identify particular users or their friend groups by compromising the server in any way during operation or by confiscating information stored on the bridge distribution server. As such, the bridge distribution server must not learn, store or log identifiable or linkable information about users or their social graph. To achieve this, we look to prior work using anonymous reputation systems, implemented with anonymous credentials and other cryptographic primitives [WLBH13, LdV17].

As stated above, rBridge [WLBH13] proposed the first privacy preserving bridge distribution system that includes building a reputation system among users with anonymous credentials. The underlying credential scheme used by rBridge is the k times anonymous authentication (k-TAA) scheme by Au et al. [ASM06], which allows credentials to be shown k times unlinkably (but further showings are linkable). Hyphae [LdV17] leans heavily on the concepts introduced by rBridge [WLBH13] but proposes a redesign of the anonymous credential scheme, noting that the value of k chosen for rBridge is not specified. Instead, Hyphae uses the efficient keyed-verification anonymous credential scheme proposed by Chase et al. [CMZ14], which allows credentials to be shown arbitrarily many times unlinkably and is suitable for situations where access to a resource and authorization of accounts is managed by the same party, such as a bridge authority. Hyphae makes use of multiple anonymous credentials and tokens: user credentials, bridge tokens, wallet tokens and invitation tokens. Each of these credentials can support multiple attributes that can be revealed to or hidden from the issuer to maintain privacy and unlinkability while making requests, allowing a user to first procure bridges and then anonymously and unlinkably build their reputation through subsequent transactions with the bridge authority. We discuss anonymous credentials further in Chapter 5.

Using Chase et al.'s anonymous credential scheme, we find that we can integrate Salmon-like trust levels and suspicion as attributes of an anonymous credential scheme similar to Hyphae's, removing the need for a wallet credential. We can use the trust level as a proxy for time spent as a user in the system where bridges remain unblocked and increase a user's suspicion when a trusted user's bridges are blocked. A single credential can contain all attributes needed to maintain a user's reputation and the attributes taken together make up a user's reputation and control their access to privileged operations.

Anonymous credentials combined with other cryptographic primatives such as Pedersen commitments [Ped92], ElGamal encryption [Elg85] and zero-knowledge proofs [CS97], can

provide users of a bridge distribution system with anonymity and effectively protect their social graph with unlinkable multi-show credentials. This makes the bridge authority a less interesting target to a censor hoping to learn about bridge users and their social networks. However, malicious behaviour attributed to censors may take many forms. In the next section, we discuss further actions and mitigation measures to limit malicious behaviour in a bridge distribution system.

3.4 Limiting Malicious Behaviour

The bridge distribution problem measures wins against a censor in two distinct ways. The first is through maximizing the number of days a censor would need to keep a bridge they learn about unblocked in order to gain additional advantages within the system. The second is the degree to which a design choice benefits users and disadvantages censors. Prior work has identified and employed different techniques to limit malicious behaviour and maximize wins against censors.

Nasr et al. [NFHG19] take a game-theoretic approach to the bridge distribution problem, offering interesting insights into the optimal censor strategy against bridge distribution systems. They find that a bootstrapping period, during which only trusted users are invited to use the system, followed by an ongoing strict and sparse distribution profile paired with the regular introduction of new proxies, are essential to counter this strategy and prevent censors from overwhelming the system.

In line with these findings, the bridge churn from BridgeDB that we showed in Figure 3.1 gives clear evidence for the importance of regular new bridge introduction. Additionally, many of the systems we discussed above already involve restricting access to trusted groups of users or else verifying their accounts prior to registration [WLBH13, LdV17, DRPC16]. In its most effective simulation, Salmon [DRPC16] grouped users linked by their social graph in a single server and began their simulation with a small group (20) of users who operated above the trust hierarchy and were able to invite new highly trusted users each day. Each of these describe or resemble bootstrapping periods where trusted users are given invitations for exclusive access to the system for some time before it is opened to unknown or untrusted users. This would encourage some relatively trusted enclaves to form around the users with early access, with the ability to grow through invitations over time.

In Lox, we follow a similar bootstrapping period to ensure that trusted users have time to establish enclaves that will remain available regardless of the censor's strategy for acting maliciously toward open-entry bridges.

3.4.1 Reputation Systems

As described in the previous sections, most bridge distribution systems from prior work employ some form of reputation system to limit access to functionality that has a higher risk to the system if used maliciously [WLBH13,LdV17,DRPC16]. This forces the censor to make a tradeoff between gaining trust in order to cause greater damage (i.e., learning more trusted brides to block, inviting more censors to bridges, etc., depending on the constraints of the specific reputation system), and keeping bridges unblocked for longer periods of time, which is necessary to build that trust. Reputation systems must also be careful not to disadvantage the censor at the expense of genuine users.

rBridge and Hyphae's protocols allow users to accrue credit by keeping their bridges unblocked. These credits can be used to unlock or purchase new bridges or invitations. Salmon tracks each user's *suspicion*, *trust level* and *recommendation (social) graph* which collectively helps to ensure that trustworthy users and their friends are rewarded with reliable connections to bridge servers and censors are not elevated to trust levels where they can do significant damage to the system. Douglas et al. do not clearly define their trust levels, leaving that to be decided by the implementation, but explain that trust levels can be increased as users spend more time in the system and limited when a user's bridges are blocked. While no specific trust level scheme is fully evaluated in Salmon, extending privileges according to time spent in the system is compatible with rBridge and Hyphae's privacy preserving microcredit schemes and can be used to enhance them.

Since genuine users are much more likely than censors to be using the bridges they receive, intermittent check-ins with the bridge authority to claim credit or request invitations are not undue burdens. This adds more work and strategic planning for a censor trying to gain extra privileges in the system.

In Lox, we propose a reputation system with clearly defined trust levels that a user can incrementally ascend after proving a period of time has passed since gaining knowledge of bridges that have remained unblocked in the interim. Each trust level, to a defined limit, unlocks greater privileges for users.

3.4.2 Inheritance

We can limit the censor's advantage further by having invited users *inherit* certain properties from their inviters. While no other system uses this term, Salmon [DRPC16] already incorporates the concept of inheritance as we have defined it. When a user with a trust level of L invites a friend to the system, the friend learns of the same bridges the inviter knows and enters the system at

a trust level of $L - 1$. This describes inheritance of bridges and inheritance of trust level since both of these traits are dependent on the inviter and passed to the invitee. To limit a malicious actor from gaining an advantage we can also make suspiciousness inheritable. If a user has seen a blocking event, any user they invite can also be considered suspicious to prevent malicious users from inviting themselves with a clean slate after behaving maliciously.

For a privacy preserving anonymous credential scheme using Chase et al.'s keyed-verified anonymous credentials, some number of bridges could be grouped into buckets to indicate a specific inheritable set of bridges. In Lox, we use such a scheme, having an identifier for this inheritable bucket as a single attribute in a Lox user's credential. Trust level and level of suspicion, indicating the number of blocked bridges a user has seen, make up additional attributes. When a trusted user reaches some trust level threshold that allows them to issue an invitation, any issued invitations will also indicate the bucket of bridges the invitee can join, a predetermined trust level and the same suspicion level as the user.

3.4.3 Open While Defending Against Sock-Puppets

Sock-puppet accounts used to enumerate and block bridges are an ever-present problem in the bridge distribution space. Though any service that requires an account can be a target of sock-puppets, providing anonymity to users complicates many existing approaches to curb this behaviour. Relying on third-party email providers to verify users was demonstrated, by Ling et al. [LLY+15], to leave Tor's BridgeDB vulnerable to censors that are able to make multiple accounts. The authors additionally showed that censors can generate multiple IP addresses to connect to the https website, appearing to be different users. While there are no perfect solutions to preventing sock-puppets in bridge distribution systems, prior work offers some suggestions to dissuade or limit the number of sock-puppet accounts that are able to access their systems.

Proximax, rBridge, and Hyphae all take the approach of allowing only trusted users to join the system. This inherently makes these systems more robust against DoS attacks that flood the system with registration requests to block new, genuine users from getting bridges. However, sock-puppet accounts can still cause considerable damage if they find paths into these systems. The Proximax scheme is particularly vulnerable to such attacks, since censors that are invited to the system are able to accrue user-hours by inviting a multitude of sock puppet users and can take down large swaths of the system while poisoning their entire trust branch. rBridge and Hyphae-like reputation systems limit the amount of damage a censor can do, forcing them to make a tradeoff between disrupting a relatively small number of users immediately and keeping bridges open longer to enumerate more bridges or invite more malicious users.

Taking a different approach, the Salmon scheme recommends only allowing the registration

of users with valid Facebook accounts, created some number of years ago and with profile pictures showing the user's face. Douglas et al. argue that this is necessary to prevent censors or malicious actors from creating multiple accounts to attack the system and undermining their reputation system. Since Salmon relies on the ability to ban users permanently, being able to trust some external body to verify that a user is not a sock-puppet is critical to the functionality of the Salmon design. Leaving aside Facebook's privacy concerns [IH18] and the ease with which a state-level censor could create multiple fake accounts, relying on *any* centralized body to verify a user's identity, whether social media, government agency or otherwise, introduces challenges that can cause disproportionate harm to some users over others. While BridgeDB's current approach is imperfect, a less restrictive means for users to gain invites and join the system is desireable.

Though the Salmon scheme's reliance on Facebook is protective against the threat of sock-puppets, there are several elements of their scheme that could be adopted to provide protection. As mentioned above, inheriting bridges and suspicion, as we have incorporated into our design of Lox, can limit the damage sock-puppets could do in trusted buckets.

In Lox, we design a system that incorporates features that are protective against sock-puppet attacks such as reputation systems and bridge distribution by trusted friends. While striving to keep Lox open to all users we rely on two methods of joining Lox. The open path could use an https website or email verification to distribute untrusted bridges, similar to Tor's BridgeDB [Tor22c]. The trusted path requires an invitation from a trusted user.

Offering two streams of entry, one open to anyone and one for trusted users who are given elevated privileges, calls for a change in the number of buckets distributed to users in each stream from Tor's BridgeDB that hands out three bridges to every user.

In Lox, distributing just a single bridge initially, where the percentage of censors in the pool of untrusted users is likely to be much higher, helps to preserve the bridgepool and slow enumeration by sock-puppets at the registration point.

3.5 Chapter Summary

In this chapter we discussed the current state of the bridge distribution problem. Tor's deployed BridgeDB [Tor22c] provides a wealth of practical insights into both the triumphs and shortcomings of the current state of an internationally used bridge distribution system and acts as a baseline to compare to proposed systems. Each of the systems from prior work that we have described above [WLBH13, DRPC16, LdV17, MML11] offers unique insights towards improving the state of bridge distribution systems. However, no one system fully incorporates each of the features we have highlighted.

Table 3.1: We compare Lox to prior work on several features that we set out to provide.

Bridge Features	**BridgeDB** [Tor22c]	**Proximax** [MML11]	**rBridge** [WLBH13]	**Hyphae** [LdV17]	**Salmon** [DRPC16]	**Lox**
Open to All Users	✓				✓	✓
Distribution by Trust Network		✓	✓	✓	✓	✓
Unlinkable Transactions			k times	✓		✓
Protects Social Graph	N/A		✓	✓		✓
Reputation System		✓	✓	✓	✓	✓
Inheritance					✓	✓
Sock-puppet Immune Enclaves					✓	✓
Defined Trust Levels					✓	✓
Friends Assigned to Same Bridges					✓	✓
Blinded Migrations						✓

In the subsequent chapters, we introduce Lox, a new bridge distribution system that incorporates each of these features into a unique, privacy preserving scheme. Lox allows users to build trust over time, as in Salmon [DRPC16], towards increasing designated trust levels. Upon reaching a predetermined elevated trust level, users are able to invite some number of friends like in other schemes discussed in this section [WLBH13, MML11, LdV17]. When a user is invited by a trusted user, they join Lox at a lower trust level and inherit their inviter's bridges and the number of blocking events they have witnessed. Each of these features are encoded as attributes in a user's anonymous Lox credential, using Chase et al. [CMZ14]'s scheme, which allows the user to remain anonymous through their interactions with the system and unlinkable to any other Lox users.

In Table 3.1, we summarize the contributions from prior work on bridge distribution systems in terms of the different features we have discussed in this chapter. We include the features we include in the design and implementation of our system Lox, which we will detail in the subsequent chapters.

Chapter 4

System Design

Lox addresses the bridge distribution problem by incorporating several ideas from prior work: leveraging a user's trust network for bridge distribution, protecting the privacy of users and their social graph, and limiting malicious behaviour, into a new privacy preserving bridge distribution system. Unique to Lox are blinded migrations that allow trusted Lox users to migrate to new trusted bridges when their bridges become blocked without revealing either set of bridges. Lox aims to provide a path for all users, including those not connected to trusted networks, to attain a stable, trusted bridge for themselves and their friends.

Our design considers four interacting parties that may or may not act maliciously: a single **Lox Authority** that verifies and distributes Lox credentials, which include all the information needed for a user to attain a bridge, a **bridge operator** that registers with the Lox authority, allows users to connect, and periodically reports suspected blockages and anonymous statistics to the Lox Authority, a Lox **user** that wants to use a bridge, and **a friend** of the Lox user. We emphasize that there are many of the latter three parties and these party types are not mutually exclusive. The Lox Authority runs a server that holds the bridge database, containing hundreds to thousands of individual bridge records (i.e., IP addresses and cryptographic keys) that can be used by individuals to connect to the open Internet. A **Lox invitation token distributor** is also required for our system but could operate much like Tor's existing BridgeDB [Tor22c] to distribute invitation tokens by email, through an https website or embedded in a website or chat client. We note that the Lox invitation token distributor and Lox Authority could be the same entity.

Before discussing our system design and protocol constructions in further detail, we first present our high-level design goals and threat model.

4.1 Design Goals and Threats

Our goal in designing Lox is to provide bridges to genuine users (as opposed to censors) in censored regions while protecting both their identity and the connections in their social graph, from discovery by a censor. Inherent in providing bridges to genuine users are the additional goals of maximizing uptime of bridges while limiting the number of bridges a censor can enumerate.

In discussing our design goals, we must consider the capabilities and motivations of our adversary and strive to limit their effectiveness. With this in mind, we propose a reputation system that is open to all, leverages trust networks to distribute bridges and emphasizes protection of users and unlinkability of social graphs.

4.1.1 Threat Model

We envision our adversary to be a state-level censor with significant resources and two high-level motivations:

1. to learn of as many bridges as possible in order to block them immediately or at some later time;
2. to learn the social graph of users.

Given that a state-level censor has vastly greater resources and compute power than an individual (or non-profit organization), we accept that it is impossible to completely stop a censor from infiltrating and impacting our system, but we can prevent such a censor from learning the social graphs of Lox users, while keeping users' actions anonymous and limiting the rate at which bridges can be enumerated.

We assume that a censor may pose as many genuine users in order to cause disruptions and enumerate bridges in order to block them. We also assume that a censor may attempt to uncover the social graph by gaining access to communication logs between the Lox Authority and Lox users. We acknowledge that other tactics such as traffic fingerprinting and network scanning to discover bridges [DWH13, MTC17], registering a large number bridges to act as honeypots, and DoS attacks against bridges or the Lox Authority [JVS19] are within the capabilities of many censors and have a significant impact on bridge distribution. However, we consider these to be orthogonal problems that are out of scope for the Lox system we present.

Lox Authority

A fully compromised Lox Authority (LA), which verifies and distributes Lox credentials, could trivially block all bridges in the system, meeting its first motivation. For the purpose of protecting the list of bridges, we therefore assume that the LA is not fully compromised by an adversary.

We further assume that the LA has committed to a set of keys for issuing and verifying anonymous credentials and made this commitment publicly accessible. Davidson et al. [DGS+18] stress the importance of having the published commitments to keys in a public place, such as the Tor consensus, that is visible to all users and can be used to independently verify the correctness of all tokens.

Without such a commitment, the LA could use a different key for each new Lox user to track users' usage of Lox. With this assumption we can assure privacy protection to users (beyond the initial open-entry interaction with the LA) and their social graph against the adversary's second motivation, even if the LA is fully compromised.

To successfully distribute bridges, we assume that the LA performs the Lox protocols correctly; however, even if it does not, it will be unable to learn a user's social graph.

Having described our threat model, we describe the elements of our system that work together to preserve the privacy of and ensure unlinkability between users while limiting the ability of a malicious actor to enumerate and block bridges.

4.1.2 Openness

Anyone with access to the Internet should be able to use Lox provided they are able to either access one of the channels that distributes open invitations or else receive an invitation from a trusted user. Since censors can pose as genuine users in either of these scenarios, we limit privileges beyond the use of a single bridge to users who have some proven trust accrued over time.

Access Control: Trust levels are assigned, as in the Salmon scheme [DRPC16], and are based on the trust level encoded in a user's invitation token when they first join Lox. The user's trust level gives a loose indication of the amount of time the user has known of unblocked bridges and is used to limit the privileges accessible to users.

Incentivized System Usage: Users are awarded extra privileges as they gain trust levels as we will see in Section 4.2.2. A user that can prove, through their trust level, that they have been successfully interacting with the Lox Authority without their originally distributed bridges being blocked is given greater privileges.

4.1.3 Leveraging Trust Networks

Lox leverages users' trust networks to distribute bridges. As mentioned above, highly trusted users are able to invite some number of friends for each trust level increase as we will see in Section 4.2.2. In Lox, distributing bridges through the trust networks of users has several purposes and advantages:

Immunity for Trusted Users: Lox allows highly trusted users and their friend groups to have immunity from blocking by a censor. When an untrusted user increases their trust after a designated period where their bridges have remained unblocked, the Lox Authority removes these bridges from the pool of bridges distributed through open invitations. The only way a new user can access these bridges is through an invitation by a trusted user that is already part of this immune group. As long as a censor has not gained access to these bridges, the users with knowledge of any of these bridges will be immune from blocking by a censor.

Sequestered Friend Groups: Invitations sent by trusted users to their friends allow access to the same bridges known by the inviter only. This ensures that censors that gain access to the system can not easily work together to gain any advantage towards enumerating bridges. Further it encourages judiciousness in the invitations a trusted user distributes.

4.1.4 Privacy and Unlinkability of Users

We utilize anonymous credentials [CMZ14] to maintain anonymity and unlinkability of users as they use our system. Lox credentials are verified and issued by the Lox authority. A Lox credential held by a user consists of all of the attributes needed for building trust in the Lox system. Since trusted users can invite friends to the same bridges they have been assigned, we must be careful in our design to ensure that we do not expose the links between friends or the bridges given to each user. To ensure the Lox Authority's legitimacy and to maintain anonymity and unlinkability across transactions with the Lox Authority, Lox must provide:

Key Consistency: Users are assured that their tokens and credentials are issued by the correct Lox Authority. The Lox Authority commits to a particular set of keys that are used to sign all tokens and credentials and publishes this commit in a publicly accessible and viewable location (such as the Tor consensus) so they can be verified by users of the system.

User Anonymity: The Lox Authority learns only the protocols that are requested as well as any revealed attributes in the credential presented. Except on the initial issuance for untrusted users, the Lox Authority never learns the bridges a user knows. Before receiving bridges, a user may need to reveal their IP address to the Lox Authority, however, once the user receives their

initial bridge, their IP address will be hidden through the use of a Tor circuit extended through that bridge and no further transactions can be linked to that user. Users who join through invitations can have their inviter send a bridge line along with the invitation so they can redeem the invitation over a Tor circuit to remain anonymous to the Lox Authority. Invitees are incentivised to use Lox (as opposed to just directly using the bridge line without joining) due to the added benefits of access to more than a single bridge, the ability to invite new users and the ability to migrate to new bridges after a blocking event.

Message Unlinkability: The credential ID must be one-show unlinkable in independent protocols and linkable only between two-step protocols. The credential ID can not be used to track users across transactions and only indicates the freshness of a credential to the Lox Authority.

4.1.5 Limit Malicious Behaviour

To protect bridges from discovery by a censor, and given our assumptions of the LA, we make the following assurances:

Correctness: Users assigned a particular bridge attribute always receive the correct bridges. Invitations from users with knowledge of particular bridges only allow invitees to join the same bucket. When all bridges in a bucket are blocked, users with that bucket attribute are unable to rejoin the system at the same elevated trust level.

Request Privacy: Requests to the Lox Authority do not reveal any identifying or unneccessary information through the revealed attributes required to complete each protocol. Further, the user's attributes and identity remain anonymous and unlinkable to other users.

Bridge Database Privacy: Requests to the Lox Authority never reveal bridges in the database directly and only reveal a single bucket identifier that can be used to query and decrypt the single entry corresponding to the user's bucket attribute in a publicly available encrypted bucket list. Learning any or multiple bucket identifiers and the bridges they provide access to does not confer any additional knowledge about the system such as other bridge addresses, identifiers, or decryption keys that would allow enumeration of bridges or buckets beyond those already known to the user.

Bridge Groups and Bucket Integrity: A group of bridges contained in a single bucket are not placed into any other bucket. An exception to this assurance is untrusted buckets (containing a single bridge) and their associated trusted buckets (a superset bucket of three specific untrusted one-bridge buckets). When bridges become blocked, they are removed from the bridge database as well as their bucket. Trusted buckets with fewer than two reachable bridges are considered blocked.

Inherited Attributes: Users that are invited to the system by trusted users will *inherit* Lox credential attributes matching those of their inviter. A designated trust level will be set for any user joining Lox through a trusted user.

4.2 System Overview

In the design and implementation of our Lox system, we aim to protect the usage patterns of users as well as their social graph through unlinkable, one-show transactions with the Lox Authority. With these considerations in mind, we present our design and implementation details of Lox, using the keyed-verification anonymous credential scheme proposed by Chase et al. [CMZ14] incorporating many insights from Lovecruft and de Valence [LdV17], and adding new features of our own. Chase et al.'s anonymous credentials allow for a credential holder to verify attributes of their credentials with an authority arbitrarily many times without revealing their identity or any other unnecessary attributes. Since the LA acts as both the issuer and verifier of anonymous credentials, this is an appropriate scheme to achieve our goals.

4.2.1 Lox Authority

Lox relies on a single central Lox Authority (LA) that acts as both the issuer and verifier of anonymous credentials and tokens for all Lox protocols. We assume that the LA holds the bridge database, containing hundreds to thousands of individual bridge lines (i.e., IP addresses and cryptographic keys) that can be used by individuals to connect to the open Internet. We further assume that the LA can verify with bridge operators whether or not a bridge is blocked at any given time. A **Lox invitation token distributor** is also required for our system but could operate much like Tor's existing BridgeDB [Tor22c] to distribute invitation tokens by email, through an https website or embedded in a website or chat client. We note that the Lox invitation token distributor and Lox Authority could be the same entity.

Allocating Bridges to Buckets

The LA divides bridges into open-entry and invite-only buckets. We consider that open-entry buckets are much more likely to be distributed to malicious users whereas trusted buckets are less likely to be held by malicious users. To limit a censor's ability to enumerate bridges and increase the probability that any given user's buckets will remain unblocked, open-entry buckets contain a single bridge and invite-only buckets contain three bridges. The LA assigns an ID i

to each bucket and maintains a global list of buckets. Each bucket has an encryption key K_i that the LA derives from a hash function (i.e., $K_i = H(LA_{SK}, i)$ where LA_{SK} is a single secret stored by the LA). For each bucket i the LA encrypts the bucket with the key K_i and posts the full list of concatenated encrypted buckets to a public website accessible to users. Since this website is itself susceptible to blocking by a censor, the LA hands out a bridge line, containing the information necessary to connect to a bridge, to users presenting valid invitation requests (along with the initial Lox credential, discussed below). Users are given the bridge key (i, K_i) as an attribute of their Lox credential. This allows them to decrypt the corresponding encrypted bucket to access their bridges. Open-entry buckets can upgrade to invite-only buckets over time. To handle this, the LA pre-groups three open-entry buckets into an invite-only superset bucket with its own unique key. Untrusted users are able to access this superset bucket after successfully increasing their trust level (L) and becoming a trusted user. This also assumes that over time, new bridges will arrive into the system to serve as open-entry bridges, as we have shown to be the case for Tor's bridges in Figure 3.1. Additionally, if a bridge in a trusted bucket goes *down*, but is not *blocked*, it can be replaced in the same bucket by the LA to be retrieved by the user, as we explain in the Bucket Reachability Credentials portion of the following section.

We note that an important aspect of open-entry bridge distribution is the mechanism with which the LA distributes particular open-entry buckets to users requesting them. The LA should be implemented to do this in a way that maximizes the chances of genuine users gaining access to buckets and minimizes the likelihood of censors being given the same buckets as genuine users. As this open-entry distribution mechanism may look different for different systems, we leave this important aspect of open-entry bridge distribution to future work and discuss this issue further in Chapter 7.

4.2.2 Anonymous Credentials and Tokens

Credentials and tokens describe the same general construct in Lox but tokens are used for operations that allow access to a new bucket (open-entry, migration) that is not connected with another Lox user (i.e., through an issued invitation). Credentials and tokens are authenticated collections of attributes. Issuers create authenticated credentials for users and users show their credentials at a later time to a verifier. In the type of credentials Lox uses, authentication is done with MACs and the issuer and verifier is the same entity: the LA. The LA can verify that a shown credential was authentically issued, but cannot link it to a particular credential it issued because showings are blinded. Users can utilize the issuer's public key to verify that credentials issued to them are correctly authenticated with a zero-knowldge proof of knowledge, created at issuing time by the issuer, but third parties cannot generally verify the authenticity of a credential.

In Lox, we use several types of spend-once credentials and tokens that can be passed between users and/or to the LA. Inspired by Hyphae [LdV17], we utilizes Chase et al.'s [CMZ14] keyed-verification anonymous credentials from algebraic MACs to implement our Lox protocols.

These allow a user to show their credentials arbitrarily many times with absolute unlinkability as long as the credential issuer is honest. A dishonest issuer could compromise the integrity of the Lox authority by using their own values of a secret key and issuing unique credentials to different clients which would allow the attacker to deanonymize tokens at redemption time and track users across transactions. Davidson et al. [DGS+18] suggest countering this attack by having the issuer commit to a key and publish this commit in a public place, such as the Tor consensus, so that it is visible to all users and can be used to independently verify the correctness of all tokens. The LA will have a separate private key for each type of credential/token. In Chapter 5 we discuss the Lox anonymous credentials and protocols in greater detail.

Lox Credential ($\Psi_{\mathfrak{L}}$)

Lox users hold a Lox credential ($\Psi_{\mathfrak{L}}$) as a non-rerandomizable (one-show) anonymous credential that is presented for re-issue at every interaction between the user and the LA. It contains the following attributes:

> ***ID*** Φ : A random ID (nonce) is jointly created by the user and the LA at Lox Credential issue time such that the LA does not learn the ID of the issued credential, and the user cannot unilaterally select it. When the user presents this credential to the LA at the time of the next interaction, Φ is revealed so the LA can add it to its database of spent IDs. It is then discarded.
>
> ***Time*** t: Lox credentials have a time attribute that marks the time the user last updated their trust level. If the user has just joined the system, the time will be marked with their join date.
>
> ***Trust Level*** L: Users receive a trust level attribute that is assigned by the LA when their credential is created. All open-entry users enter Lox with $L = 0$. All invited users enter Lox with $L = 1$. Users that can prove to the LA that a sufficient time period has passed without the bridges in their bucket becoming blocked, can upgrade their trust level through the LA. If a user continues to hold a credential for an unblocked bucket, they can continue to level up to a maximum of $L = 4$ with intermittent level up requests to the LA. Table 4.1 shows the days required to increase L at each level and what each level of L allows users to do.

Table 4.1: User capabilities and invitations for different L values as well as the days required to level up. Users migrating to unblocked buckets are given $L - 2$. A user's trust level does not increase beyond $L = 4$. However, users moving up to $L = 4$ receive $a = 6$ invitations. Those that have had $L = 4$ for 84 days or more can receive $a = 8$.

L	**Status**	**Invites** a **(INVITATIONS)**	**Upgrade** t **(DAYS)**
0	Untrusted, 1-bridge bucket	0	30
1	Trusted, 3-bridge bucket	0	14
2	Trusted, Invitations	2	28
3	Trusted, Invitations, Migration Eligible	4	56
4	Trusted, Invitations, Migration with Invitations Eligible	6, 8	84

Bridge bucket β**:** Users receive (i, K_i) as an attribute in their credential where i is a bucket ID assigned by the LA and K_i is the encryption key used to encrypt the bucket. Open-entry users receive one bridge in their bucket whereas invited users receive three bridges.

Available invitations a**:** The invitation countdown counter is an attribute that allows users to issue invitations once they have advanced to $L \geq 2$. The a value is set to the correct value (see Table 4.1) when the user upgrades their trust level. The value of a is decremented for each invitation issued by the user.

Blockages d**:** The blockages attribute records the number of times the user has migrated to a new trusted bucket. Open-entry users will always begin with $d = 0$. Users who are invited will inherit the d value of their inviter. Experiencing blockages limits the trust level a user can achieve as shown in Table 4.2; 3 and 4 blockages limit the user to trust levels 3 and 2 respectively, after which they will be be ineligible to migrate.

Table 4.2: Summarizes the maximum value of L a user can achieve given the value of d in their Lox credential.

d	**Max** L
0	4
1	4
2	4
3	3
4	2

We note that in our design of Lox, we have indicated particular values for the L, t, a, and d values. The selection of these values started from L and were otherwise selected with the intention of demonstrating how a system can use trust levels to increase user privileges as they use they system. We expect the particular values chosen for t, a, and d will require further optimization and consideration depending on the specifics of the system and threats faced by Lox implementers. We discuss optimization of Lox parameters further in Chapter 7.

Invitation Token ($\Omega_{\mathfrak{I}}$)

An open-entry invitation token ($\Omega_{\mathfrak{I}}$) is issued to anyone that is able to request them through one of the methods available to new Lox users. We assume that Lox users will have the same methods available as users of Tor's BridgeDB [Tor22c]: an https website, through email to a specific email address, or to be embedded in a browser such as Tor. However, we note that other distribution methods could be appropriate depending on the deployment. Invitation tokens have their own ID attribute Φ, a time attribute so that they can expire after some period, and an optional attribute that can be used by the LA to select an open-entry bucket from a specific pool of buckets (e.g., if buckets were further divided into email, https, or browser bucket).

Lox incentivizes any new user with friends already using the system with an elevated trust level, to request bridges from these friends. Entering Lox with a trusted invitation credential automatically gives the new user more privileges. However, open-entry invitation tokens can fill the gap for users who are unconnected to trusted friends or whose friends have not yet reached a high enough trust level to issue invitations. Invitation tokens aim to ensure anyone that needs a bridge is able to get one and can start building their own trust level immediately even without friends already using the system.

Invitation Credential ($\Psi_{\mathfrak{I}}$)

Invitation credentials ($\Psi_{\mathfrak{I}}$) can be generated only by trusted users when they achieve a trust level $L \geq 2$ and have invitations remaining (i.e., $a > 0$). An invitation credential contains a hidden invitation ID (Φ), the day (t) that it was generated, and the inviter's bridge bucket key (β) and blockages (d). Proposed invitation credentials are presented by the inviter to the LA and if verified, are signed and issued back to the inviter to be given to a friend. The invite ID ensures that invitation credentials can not be reused once they are redeemed and the day attribute enforces that they expire after 15 days. Since the user requesting the invitation credential already has access to Lox bridges, they may perform the initial invitation redemption request on behalf of the friend they are generating the invitation for. Presenting and issuing an invitation credential

is discussed further in Chapter 5. If each user is using all of their available invitations each time they are eligible to issue invitations, the number of users in a particular bucket will grow exponentially. While we want to encourage users to be judicious about who they issue invitations to, we recognize that this could become an issue for the most well-established bridges with highly trusted users. We discuss some strategies for mitigating this issue in Chapter 7.

Migration Key Credential & Migration Token

A migration key credential ($\Psi_{\mathfrak{M}}$) can be requested in the first step of a migration protocol. There are two types of migration protocols in the Lox system, each with two steps:

1. **Trust Promotion:** When an open-entry user requests a trust promotion from $L = 0$ to $L = 1$ because their t attribute is set to a date more than 30 days before the current date as shown in Table 4.1, and the (single) bridge that they know has remained reachable over that time. This allows them to learn of the superset bucket that contains their current bridge and two others.

2. **Blockage Migration:** When a user with a trust level $L \geq 3$ and blockages $d < 4$ requests to be unblocked and the LA can confirm that the bridges in their bucket are blocked. This allows them to learn of a new bucket and re-enter Lox with a trust level $L - 2$ and blockages $d + 1$.

As discussed below, the LA maintains a list of buckets that are eligible for migration. This list is updated daily to include newly blocked buckets and bridges that have been newly added to the database. When a request to migrate is initiated, the LA verifies that the user meets the criteria to migrate for the indicated migration type and if this succeeds, the LA blindly issues a migration key credential to the user. The attributes of a migration key credentials are a Lox credential ID (Φ) and the bucket the Lox credential is migrating from (β_{FROM}). The LA returns this blindly issued credential (so that the LA does not learn any of the attributes, save that they match those in the blinded Lox credential whose attributes were checked to be eligible for migration as described above), along with an encrypted migration table (see Section 5.2 for details).

The user then uses a hash of Φ, β_{FROM}, and the unblinded MAC from the migration key credential to decrypt an entry from the migration table to yield a *migration token* ($\Omega_{\mathfrak{M}}$). Migration tokens have attributes of Φ, the buckets β_{FROM} and β_{TO} the user is migrating from and to, and a flag indicating whether this is a trust promotion or a blockage migration.

In the second phase, the user blindly presents their Lox credential and their migration token (with matching Φ and $\beta = \beta_{\text{FROM}}$), and the LA blindly issues in return a new Lox credential with β set to β_{TO}, and L and d modified appropriately.

Bucket Reachability Credentials

Bucket reachability credentials ($\Psi_{\mathfrak{R}}$) are created by the LA and act as a certificate of whether or not the bridges in a particular bucket are accessible to users. Bucket reachability credentials have only two attributes, the current day (t), and the bridge bucket (β) and must be presented to the LA for protocols that require proof of an unblocked bucket. Each bucket with reachable bridges will have its bucket reachability credential placed in the bucket itself (in the encrypted bucket list) each day. Bridge users would download their encrypted bucket (over Tor) every day to obtain their current bucket reachability credential. At the same time, they would learn replacements for any bridges in their bucket that had gone offline (but not blocked).

4.2.3 Ancillary Tasks Performed by the Lox Authority

Aside from issuing and verifying credentials, the LA performs a number of tasks to set up the system and provide information about the system.

Bridge Reachability Check.

The LA performs a check of bridge reachability once daily and generates a bucket reachability credential for each bucket known to it. If two or more of the three bridges in a bucket are unreachable, the bucket is considered to be blocked for the purpose of the $\Psi_{\mathfrak{R}}$ and the date attribute of the token is not updated to the current date. After checking the reachability of each bridge in a bucket, the LA places each $\Psi_{\mathfrak{R}}$ in its corresponding encrypted bucket to be posted with the full list of encrypted buckets described in Section 4.2.1. Users with credentials for the bucket are able to anonymously retrieve the $\Psi_{\mathfrak{R}}$ for their bucket through a Tor circuit each day.

Maintain a Table of Eligible Migrations.

For both trust promotion from trust levels $L = 0$ to $L = 1$ and bucket blockage migration operations after a trusted bucket's bridges are blocked, a user can request to be migrated to a new bucket. This requires that the LA maintain a table of eligible bucket migrations for each of these two operations that can be encrypted and sent to a user when a migration is requested and the presented Lox credential is verified. The trust promotion migration table holds the list of open-entry buckets that have remained unblocked since they appeared in an open-entry bucket, and their corresponding trusted superset buckets. The blockage migration table holds the list of trusted buckets that have become blocked and their corresponding trusted unblocked, replacement

buckets that have been held in a reserve of hot spare buckets that have not yet been handed out to users.

Maintain a List of Used Token Nonces.

As in Hyphae [LdV17], our system makes use of several spend-once tokens that the LA must maintain in a list of spent token ids for each type of credential and token. At issue time, users will prepare ids (random nonces) that are hidden from the LA in order to be included as attributes of a new credential. The LA adds its own nonce homomorphically to the hidden id attribute, signs the credential and issues it back to the user. At presentation time, the ID is revealed to the LA, that then adds the ID to its list of spent ID tokens for the given credential, allowing for unlinkable spend-once tokens. Periodically, the list can be reset to prevent overflow by changing the LA's public credential keys and continuing to issue credentials only with that key. For some time after, credentials issued with the current key as well as past keys will continue to be accepted. Once a past key is no longer accepted, IDs that were signed by that key can be safely forgotten.

4.2.4 Bootstrapping Period

As Nasr et al. [NFHG19] elucidate, a bootstrapping period where only trusted users are able to populate the system is required to ensure that Lox is not overwhelmed and made unusable by censor-generated sock puppet accounts when deployed. Similar bootstrapping periods, or alternatively only opening the system to trusted users, are features of all prior bridge distribution work [WLBH13, DRPC16, LdV17]. To bootstrap Lox, prior to public release of open invitations, some number of trusted users should be invited to the system. As an example, special invitations that have an elevated trust level of $L \geq 2$ and include a bridge line to the bridge operator's bridge could be given to some number of bridge operators to distribute to friends. As with any trusted invitation, this method has the added advantage of allowing trusted users to join Lox through a bridge line so they do not need to reveal their IP to the LA. Alternatively, trusted users could be given early access to Lox with a small number of open-entry invitations that are able to level up to being able to invite new users more quickly. Before Lox is open to the wider public, the time required to upgrade could be readjusted to something more suitable to untrusted users.

However it is implemented, a bootstrapping period allows a number of bridge buckets to be populated by highly trusted users and their similarly trusted invitees, limiting the number of censors that are able to gain access to trusted buckets.

4.2.5 Lox Authority Protocols

Having described the components of Lox, we now give a high-level overview of the various interactions a user of our system may have with the LA through each of the Lox protocols.

Open-entry Invitation

Any user can request a Lox credential from the LA by presenting an open-entry invitation with $\Omega_{\mathfrak{I}}.\Phi$ revealed. When a valid open-entry request is presented to the LA and the invitation token $\Omega_{\mathfrak{I}}$ is verified by the LA in zero knowledge, the initial Lox credential $\Psi_{\mathfrak{L}}$ is blindly issued.

Although the attributes in the initial Lox credential are set and issued almost unilaterally by the LA, the jointly chosen credential ID Φ ensures that subsequent transactions cannot be linked to the user. If the set of Lox users is very small, however, the adversary may be able to infer which user is making the subsequent request by timing when the recipient of an open-entry request becomes eligible for trust promotion. To mitigate this, it is important to ensure that a constant stream of users continue to join Lox during bootstrapping and afterward. As mentioned above, a bootstrapping period is required to ensure that Lox is not overwhelmed and made unusable by censor-generated sock puppet accounts when deployed. To bootstrap Lox, open invitations with elevated trust levels should be distributed to some number of highly trusted users for a limited time period prior to public release of open invitations. The buckets distributed to these users will be removed from the open entry bucket list, limiting the number of trusted buckets that a censor can gain access to.

- When a valid open-entry request is presented to the LA and the invitation token ($\Omega_{\mathfrak{I}}$) is verified by the LA, the Lox credential is issued as:

  Φ: jointly chosen by the user and LA

  β: set by the LA

  L: 0, set by the LA

  t: today's date, set by the LA

  a: 0, set by the LA

  d: 0, set by the LA

Although the initial credential is set and issued almost unilaterally by the LA, the jointly chosen credential ID (Φ) ensures that subsequent transactions can not be linked to the user. Of

course, if the set of users is so small that the subsequent trust promotion and level up protocols could only be called by a particular user, then a malicious actor watching the LA could deduce which user was calling those protocols. To mitigate this, it is important to ensure that a large group of users is able to join in quick succession when initially bootstrapping the system as described in Section 4.2.4 and that new Lox users continue to join steadily.

Trust Promotion and Trust Migration

As discussed in Section 4.2.2, trust promotion from $L = 0$ to $L = 1$ is a two-step protocol, marking the upgrading user's transition from an untrusted user to a trusted user where they gain access to a new trusted superset bucket credential with three bridges. When the LA identifies that an open-entry bucket has remained reachable for 30 days, it stops giving that bridge's bucket credential to open-entry users. This will occur just prior to the time when the first untrusted users that were given access to the bucket become eligible to upgrade to $L = 1$. Once this occurs, the only way new users will learn the newly trusted bucket credential is by invitation from someone already using that bucket.

To initiate the Trust Promotion interaction, the user must present a valid Lox credential as follows: (Phase 1)

- Client presents their Lox credential ($\Psi_{\mathfrak{L}}$):

  Φ: revealed to the LA, recorded in the list of trust promotion spent tokens

  β: hidden

  L: revealed to be 0

  t: hidden but proven in zero knowledge to be 30 days or more older than today's date

  a: revealed to be 0

  d: revealed to be 0

We note that the protocol does not require a bucket reachability credential since if an untrusted bridge becomes blocked, the LA will not place it in the table of eligible trust promotion migrations. The LA verifies the presented Lox credential and all zero knowledge proofs, then checks that Φ was not used before to request a trust promotion, and adds it to the trust promotion specific list of spent tokens but does *not* add it to the used Lox credential spent token list. The LA then returns a migration key credential to the user (described in Section 4.2.2). The user uses their migration key credential to decrypt their migration token ($\Omega_{\mathfrak{M}}$). The user is then ready to make a trust

migration request by presenting their migration token along with their current Lox credential for verification and their updated Lox credential with the hidden new bucket attribute matching the migration token's $\Psi_{\mathfrak{M}}.\beta_{\text{TO}}$ for issuing:

(Phase 2)

- Lox Credential $\Psi_{\mathfrak{L}_{\text{OLD}}}$:

  Φ: revealed, recorded in the list of Lox credential spent tokens

  β: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{M}}.\beta_{\text{FROM}}$

  L: revealed to be 0

  t: hidden

  a: revealed to be 0

  d: revealed to be 0

- Migration Token $\Omega_{\mathfrak{M}}$:

  Φ: revealed, must match $\Psi_{\mathfrak{L}_{\text{OLD}}}.\Phi$, recorded in migration token spent tokens

  β_{FROM}: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{L}_{\text{OLD}}}.\beta$

  β_{TO}: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{L}_{\text{NEW}}}.\beta$

- Lox Credential $\Psi_{\mathfrak{L}_{\text{NEW}}}$:

  Φ: jointly chosen by the user and LA

  β: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{M}}.\beta_{\text{TO}}$

  L: revealed to be 1

  t: revealed to be today's date

  a: revealed to be 0

  d: revealed to be 0

If the LA successfully verifies these credentials in zero knowledge, the LA blindly issues the new Lox credential to the user, without having learned which buckets the user had migrated from or to.

Level Up

Lox requires that users actively engage with the LA to upgrade trust levels after a pre-determined interval has passed, as described in Table 4.1. For example, even if a user has known of their buckets long enough to have achieved the highest trust level, if they have not upgraded past $L = 1$, they will only be eligible to upgrade to $L = 2$, thereby rewarding continued engagement with Lox. This makes attaining higher trust levels more challenging for users who forget to perform the upgrade or do not use Lox often, since they must reengage with the system at regular intervals to build their reputation. This differs from the Salmon scheme [DRPC16] that passively upgrades trust levels as users remain in the system. Regular engagement with the LA encourages active engagement with the system and rewards the most active Lox users. It also prevents censors from gaining trust by simply acquiring bridges and not blocking them, and of course, forces them to keep bridges unblocked in order to gain trust at all. Automating upgrades, by integrating Lox into a browser for example, could help to prevent users from forgetting to upgrade while rewarding regular engagement. We discuss this further in Chapter 7. A user with $L \geq 1$ must prove to the LA that their Lox credential attribute t is older than some pre-determined number of days (see Table 4.1) and that the bucket in their Lox credential is reachable at the time of the upgrade by blindly presenting a valid bucket reachability credential for their bucket β.

To level up, the user presents the following credentials to the LA:

- Lox Credential $\Psi_{\mathfrak{L}_{\text{OLD}}}$:

  Φ: revealed, recorded in the list of Lox credential spent tokens

  β: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{R}}.\beta$

  L: revealed, and must be at least 1

  t: hidden, but proven in zero knowledge to be at least the appropriate number of days old (See Table 4.1)

  a: hidden

  d: hidden, but proven in zero knowledge that it is below the limit for the target trust level

- a Bucket Reachability credential $\Psi_{\mathfrak{R}}$:

  t: revealed to be today

  β: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.\beta$

- Lox Credential $\Psi_{\mathfrak{L}_{\text{NEW}}}$:

Φ: jointly chosen by the user and LA

β: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.\beta$

L: revealed to be $\Psi_{\mathfrak{L}_{\text{OLD}}}.L + 1$

t: set to today's date

a: revealed to be the number of invites for the new level (note that the invites remaining from the previous credential are *not* carried over)

d: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.d$

If the credentials are successfully verified by the LA in zero knowledge, the LA issues the new Lox credential with today's date, $L \leftarrow L + 1$ and a set to the appropriate value for the new level in Table 4.1. That table also lists the advantages that accrue to users at higher trust levels. Users with $L \geq 2$ can generate invitations for their friends to join their trusted bucket at a trust level of $L = 1$. Users with $L \geq 3$ and $d < 3$ are eligible to use their trust level to gain access to a new bucket in the event that two or more of the bridges within their bucket become blocked.

Issue Invitation

Users with trust levels of $L = 2$ become eligible to issue invitations. Invitations allow untrusted users to enter into the system at an elevated trust level ($L = 1$) and gain knowledge of the same bucket (β) and blockages (d) held by their inviter. Grouping friends in the same bucket encourages the inviter to be cautious in distributing invitation tokens and prevents a malicious user who is able to gain access at a high trust level from enumerating more bridges. To issue an invitation, the user blindly presents their Lox credential with $a > 0$ and a bucket reachability credential to the LA. If the LA successfully verifies these credentials in zero knowledge, the LA will sign the new Lox credential (with a decremented a) as well as the new invitation credential, and issue them to the user.

To issue an invitation, the user presents the following credentials to the LA:

- Lox Credential $\Psi_{\mathfrak{L}_{\text{OLD}}}$:

  Φ: revealed, recorded in the list of Lox credential spent tokens

  β: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{R}}.\beta$

  L: hidden, but proven in zero knowledge that $L \geq 2$

  t: hidden

a: hidden, but proven in zero knowledge that $a > 0$

d: hidden

- a Bucket Reachability credential $\Psi_{\mathfrak{R}}$:

  t: revealed to be today

  β: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.\beta$

- Lox Credential $\Psi_{\mathfrak{L}_{\text{NEW}}}$:

  Φ: jointly chosen by the user and LA

  β: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.\beta$

  L: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.L$

  t: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.t$

  a: hidden, but proven in zero knowledge to be $\Psi_{\mathfrak{L}_{\text{OLD}}}.a - 1$

  d: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.d$

- An Invitation Credential $\Psi_{\mathfrak{I}}$:

  Φ: jointly chosen by the user and LA

  t: revealed to be today

  β: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.\beta$

  d: blinded, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}_{\text{OLD}}}.d$

If the LA successfully verifies these credentials, the LA will sign the new Lox credential as well as the invitation credential and issue them to the user.

Lox's trusted invitations give very little advantage to a censor that keeps bridges open to gain trust since gaining and distributing more invitations does not allow them to enumerate more bridges. Since blockages are inherited, they are also unable to issue an invitation to themselves to hide past malicious behaviour.

Thus, if a censor gains access to an invite-only bucket, their most likely action would be to immediately block the bridges. Still, whether they choose to block immediately or during a certain event or crisis, this behaviour only impacts a fraction of the overall system. Genuine users, conversely, may be inclined to share bridge addresses they know with friends outside of Lox, so grouping users in the same bucket matches expected user behaviour patterns and the possibility of gaining access to new bridges in the case of a blockage provides a significant advantage to users over just sharing bridges directly, incentivizing the use of Lox.

Redeem Invitation

A new, invited user can create a Lox credential with the bucket β given to them in their invitation credential by blindly presenting their invitation credential to the LA. If the LA successfully verifies their invitation token and proposed new Lox credential in zero knowledge, the LA blindly issues the new credential back to the user.

An untrusted user with an invitation can join the Lox system by presenting the following credentials to the LA:

- An invitation credential $\Psi_{\mathfrak{I}}$:

  Φ: revealed

  t: hidden, but proven in zero knowledge to be at most 15 days old

  β: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}}.\beta$

  d: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{L}}.d$

- a new Lox credential $\Psi_{\mathfrak{L}}$:

  Φ: jointly chosen by the user and LA

  β: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{I}}.\beta$

  L: revealed to be 1

  t: revealed to be today's date

  a: revealed to be 0

  d: hidden, but proven in zero knowledge to be the same as $\Psi_{\mathfrak{I}}.d$

If the new user successfully proves the validity of their invite token and new user credential in zero knowledge, the LA signs the credential and issues it back to the user.

Check Blockage and Blockage Migration

If a trusted bucket become blocked, there are limited avenues for trusted users to re-enter the system at an elevated trust level.. We assume that the majority of Lox users will need to re-enter the system as untrusted users either through open-entry invitations or else through invitations from friends with knowledge of different buckets. However, users who have achieved $L \geq 3$ are able to migrate to a new trusted bucket. When trusted users migrate, their d attribute increases

by 1. When a user reaches $d = 4$ they are no longer able to upgrade their trust level high enough to be eligible to migrate, as described in Section 4.2.2.

Migrating to a new unblocked bucket is a two-step protocol similar to the trust upgrade protocol from $L = 0$ to $L = 1$. A user can request migration to another trusted bucket when their bucket becomes blocked by blindly presenting their Lox credential to the LA. The LA checks that Φ does not appear in the Lox credential spent token list, but does *not* add it to the used token list since users may need to make the request multiple times if the migration table has not yet been updated or if the bridge is just temporarily unavailable.

A user can request migration to another trusted bucket when their bucket becomes blocked as follows:

(Phase 1)

- Client presents their Lox credential ($\Psi_{\mathfrak{L}}$):

  Φ: revealed to the LA, checked against the list of spent token IDs but not added to it

  β: hidden

  L: revealed to be 3 or above

  t: hidden

  a: hidden

  d: hidden

Upon verifying the credential in zero knowledge, the LA returns a migration key credential to the user (described in Section 4.2.2). The user uses their migration key credential to decrypt their migration token $\Omega_{\mathfrak{M}}$ and blindly presents this, along with their current and proposed new Lox credential, to the LA:

(Phase 2)

- Lox Credential $\Psi_{\mathfrak{L}_{\text{OLD}}}$:

  Φ: revealed, recorded in the list of Lox credential spent tokens

  β: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{M}}.\beta_{\text{FROM}}$

  L: revealed to be 3 or higher

  t: hidden

  a: hidden

d: hidden

- Migration Token $\Omega_{\mathfrak{M}}$:

  Φ: revealed, must match $\Psi_{\mathfrak{L}_{\text{OLD}}}.\Phi$, recorded in the list of migration token spent tokens

  β_{FROM}: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{L}_{\text{OLD}}}.\beta$

  β_{TO}: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{L}_{\text{NEW}}}.\beta$

- Lox Credential $\Psi_{\mathfrak{L}_{\text{NEW}}}$:

  Φ: jointly chosen by the user and LA

  β: hidden but proven in zero knowledge to equal $\Psi_{\mathfrak{M}}.\beta_{\text{TO}}$

  L: revealed to be 2 less than $\Psi_{\mathfrak{L}_{\text{OLD}}}.L$

  t: revealed to be today's date

  a: revealed to be 0

  d: hidden but proven in zero knowledge to be 1 more than $\Psi_{\mathfrak{L}_{\text{OLD}}}.d$

If the LA successfully verifies these credentials in zero knowledge, it will now add Φ to the Lox credential spent token list, sign the new Lox credential and issue it to the user, allowing them to access their new trusted bridges.

We provide the details of the attribute options for each of our Lox credentials as they are used across protocols in the Chapter 5.

4.3 Chapter Summary

Lox is designed with the overall goals of being open to all users, leveraging trust networks for bridge distribution, providing privacy to users and their social graph, and limiting malicious behaviour against an adversary that wants to learn as many bridges as possible and learn the social graph of users. Lox achieves these goals through a central Lox Authority that, even if fully malicious, guarantees protection of a user's social graph. Lox further guarantees protection of the set of bridges from the censor as long as the Lox Authority is not itself collaborating with the censor. The Lox Authority acts as the issuer and verifier of several types of anonymous credentials and tokens that are issued to and presented by Lox users. These credentials are authenticated collections of attributes that determine a user's level of access to bridges, invitations, and blockage migration. Lox protocols serve to verify and adjust Lox credentials to allow users

who have had unblocked bridges for a sufficient time period to increase their trust level towards issuing invitations to invite friends and migrate to new bridges if their bridges become blocked. Deployment of the Lox system should be prefaced by a bootstrapping period that allows trusted users early access or else access with an elevated trust level in order to increase the number of buckets that are populated only by trusted users and their friends and free of censors.

Having fully described the motivation and goals of Lox's design, in the following chapter, we discuss the specifics of the anonymous credential scheme and cryptographic primatives that give Lox its privacy preserving properties at the implementation level.

Chapter 5

Lox System Implementation Details

It is crucial to prioritize protecting users when developing censorship circumvention systems. If not designed with care, bridge distribution systems can leak information about how and when bridges are used for censorship circumvention, undermining the integrity of the system and safety of users. For bridge distribution systems that allow users to invite friends [LdV17, DRPC16, WLBH13], compromise of the bridge distribution authority can additionally expose the social graph of any targeted users, multiplying the damage. Anonymous, one-show credentials, that are designed to be unlinkable across transactions, prevent malicious actors from re-identifying users of the system beyond their initial open-entry request and completely conceal users' social graphs.

With these considerations in mind, we implement our Lox design, detailed in Chapter 4, using the keyed-verification anonymous credential scheme proposed by Chase et al. [CMZ14] and incorporating many insights from Lovecruft and de Valence [LdV17]. Chase et al.'s anonymous credentials allow for a credential holder to verify attributes of their credentials with an authority arbitrarily many times without revealing their identity or any other unnecessary attributes. Since the Lox Authority acts as both the issuer and verifier of anonymous credentials, this is an appropriate scheme to achieve our goal.

In this chapter, we detail the cryptographic primitives we use to provide anonymity to Lox users and describe how these are used to implement Lox transactions and protocols.

5.1 Anonymous Credentials for Protecting the Social Graph

Anonymous credentials are a versatile cryptographic construction with practical uses in the areas of communication, payment and credential transaction systems. Initially proposed by

Chaum [Cha85], anonymous credentials enable all sides of a transaction to protect their security and privacy interests while sharing and verifying information required for an authenticated transaction to proceed. This allows secure messages to be exchanged anonymously and unlinkably between an organization or authority and its users. As discussed in Chapter 3, rBridge [WLBH13] and Hyphae [LdV17] both propose bridge distribution systems that utilize anonymous credentials to protect the privacy of bridge users and their social graph. In these systems, credit is accumulated or lost based on the user's behaviour, as judged by anonymous and unlinkable interactions with the bridge authority.

The k-TAA scheme by Au et al. [ASM06], based on Camenisch and Lysyanskaya's BBS+ [CL04], is constructed from bilinear pairings and is secure in the random oracle model. rBridge provides a detailed and novel implementation of their bridge distribution system using the k-TAA scheme which allows credentials to be shown k times unlinkably. However, as pointed out in Hyphae, the value of k chosen for rBridge is not specified. Additionally, pairing based signature schemes are often cumbersome to implement and are not an ideal solution for those looking for a practical system to deploy and maintain. Noting that a bridge distribution system with a single authority for both issuing and verifying of credentials could benefit from simpler keyed-verification credentials, Hyphae reenvisions the rBridge design using Chase et al.'s CMZ credentials [CMZ14] which use much simpler message authentication codes. Chase et al.'s anonymous credential scheme includes two algebraic MACs: MAC_{GGM}, which is a generalization of a MAC presented by Dodis et al. [DKPW12] that they prove satisfies the standard notion of MAC unforgeability (uf-cmva security) in the generic group model (GGM), and MAC_{DDH}, which is proved uf-cmva secure under the decisional Diffie-Hellman (DDH) assumption. Though Hyphae stops short of a full implementation and evaluation of their design, their system uses the MAC_{GGM} anonymous credentials, which we expect contributes to greater protocol efficiency and significant performance improvements over rBridge's scheme.

For our Lox implementation, we also make use of Chase et al.'s credential scheme. With these insights into anonymous credentials, as well as those gained from other prior work [DRPC16, NFHG19], we provide an implementation of our Lox protocols from Chapter 4. In Chapter 6, we discuss the performance of this implementation.

5.2 Cryptographic Preliminaries and Notation

We repeat the MAC_{GGM} construction of Chase et al. [CMZ14] applied to the context of Lox with the Lox Authority acting as both the credential issuer and verifier.

Let A, B be generators of a fixed group $\mathbb{G}$ of prime order ℓ, written additively, such that

$\log_B(A)$ is unknown. For each credential with n attributes, the LA's secret key is the vector

$$(\tilde{x}_0, x_0, x_1, \ldots, x_n) \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^{n+2}$$

which we will denote as $(\vec{x_{\mathfrak{C}}})$ where $\mathfrak{C}$ specifies the credential type (Lox: $\mathfrak{L}$, Migration: $\mathfrak{M}$, Invitation: $\mathfrak{I}$, Reachability: $\mathfrak{R}$) and the public key is

$$(X_0, X_1, \ldots, X_n) = (\tilde{x}_0 A + x_0 B, x_1 A, \ldots, x_n A).$$

We can provide assurance to Lox users that all of their credentials are issued and accepted by the correct Lox LA and not a malicious decoy by having the LA publish this public key where it can be publicly viewed (e.g., in the Tor consensus) as is suggested by Davidson et al. [DGS+18].

To create the tag (P, Q) for a vector of attributes $(m_1, ..., m_n) \in (\mathbb{Z}/\ell\mathbb{Z})^n$, the issuer selects $b \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^*$ and computes

$$P \leftarrow bB$$
$$Q \leftarrow \left(x_0 + \sum_{i=1}^{n} x_i m_i\right) P.$$

To issue a credential with hidden attributes, the user picks an ElGamal public key D, encrypts the hidden attributes to that key, and creates a zero-knowledge proof of the desired properties of the hidden attributes. The LA can then homomorphically compute the encryption of Q to the key D, even though it cannot learn Q itself. The user decrypts to learn Q. The user rerandomizes the MAC by picking $t \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^*$ and setting $(P', Q') = (tP, tQ)$.

To show a credential with hidden attributes, the user presents Pedersen commitments of the hidden attributes, a zero-knowledge proof that the hidden attributes have the desired properties, and a designated-verifier proof (that requires the secret key to check) that the (hidden) MAC is correct.

Next we highlight our mechanism for encrypting the migration tables used to allow trusted users to migrate to trusted buckets which is unique to Lox and indicate the notation we will use for our zero-knowledge proofs.

Encryption of Migration Tables

Unlike previous bridge distribution proposals, Lox has the unique ability to allow trusted users to learn new bridges after a trust upgrade or blockage event without revealing to the authority which

bridges the user previously, or subsequently, knows about. As described in Section 4.2.3, in Lox, the LA maintains a table $\langle \beta_{\text{FROM}_i}, \beta_{\text{TO}_i} \rangle_{i=1}^{r}$ of eligible migrations. In fact, there is one table for trust promotion migrations and one for blockage migrations, but we will just consider one, as they work in the same way. For the first step in the two-part protocols for trust migration and blockage migration, the user requests a blinded migration key credential ($\Psi_{\mathfrak{M}}$) and an encrypted migration table of all eligible migrations. The issuing of the blinded migration key credential will give to the user a pair $(Pk, Enc_D(Qk))$ for an ElGamal key D chosen by the user, but that Qk will be unknown to the LA. The user must learn only the β_{TO_i} for the i for which the user's current $\beta = \beta_{\text{FROM}_i}$, and also receive a signed migration token ($\Omega_{\mathfrak{M}}$) containing their current Φ, and that single $(\beta_{\text{FROM}_i}, \beta_{\text{TO}_i})$ pair.

In order to do this, the LA generates a hashtable such that for each i, there will be an entry in the hashtable with key $H_1(\Phi, \beta_{\text{FROM}_i}, Qk_i)$ and value $E_{H_2(\Phi, \beta_{\text{FROM}_i}, Qk_i)}(\beta_{\text{TO}_i}, P_i, Q_i)$. Here H_1 and H_2 are hash functions, (P_i, Q_i) is a MAC on the migration token that has attributes $\Phi, \beta_{\text{FROM}_i}$, and β_{TO_i}, and (Pk, Qk_i) is a MAC on the migration key credential that has attributes Φ and β_{FROM_i}. When the user decrypts their Qk, they can compute $H_1(\Phi, \beta_{\text{FROM}}, Qk)$ and use that as the index to the hashtable to find the appropriate encrypted entry. They then compute $H_2(\Phi, \beta_{\text{FROM}}, Qk)$ to use as the decryption key for that entry, to yield the attributes and MAC for the migration token. This satisfies the requirements that only the entry corresponding to the current bucket β_{FROM} of the user with Lox credential ID Φ can be decrypted, even if the migration key credentials are shared with other users, and that the migration token can only be used once. A valid, decrypted migration token with attributes $\Phi, \beta_{\text{FROM}}, \beta_{\text{TO}}$ is required in the second step of the migration protocol when the migration to a new bucket actually occurs. The user migrating either from $L = 0$ to $L = 1$ or else from a now blocked bucket in which they had $L \geq 3$ and $d < 3$ must present their old Lox credential with Φ that must match the migration token's Φ, hidden value β that is proven in zero knowledge to match the migration token's β_{FROM}, and an updated Lox credential with a β that matches the migration token's β_{TO}.

Zero-Knowledge Proofs

Zero-knowledge proofs are used in the Lox scheme to prove knowledge of the correct true values of the presented hidden attributes and MACs for each protocol based on relations between discrete logarithms. We follow the notation of Camenisch and Stadler [CS97] in expressing our zero-knowledge proofs as:

$$\pi = \text{NIZPK}\{(w) : \text{statements about } w\}$$

where w, the witness, are the secrets known only to the prover; any symbols appearing in the statements but not in w are public and known to both the prover and the verifier.

5.3 Credentials in Lox

Recall from Section 4.2.2 that all Lox users hold a Lox credential ($\Psi_{\mathfrak{L}}$) as a non-rerandomizable (one-show) anonymous credential with attributes: ID (Φ), time (t), trust level (L), bridge bucket (β), available invitations (a) and blockages (d). In each of the credentials issued to and presented by users, attributes may be hidden, unilaterally selected by server, jointly created or revealed to the Lox Authority. The exception in our scheme are the open-entry invitations which are not anonymous credentials, but are rather digital signatures where the bucket ID and invitation ID are signed by the LA.

In this section we discuss what each of these attribute options means at a high level. Next, we give an overview of the particular attribute options used for each attribute of each type of credential when presented or issued in the Lox system. Finally, we detail algorithms for each type of presentation and issuing attribute options used by our Lox protocols and explain the process through which these protocols are performed.

5.3.1 Attribute Options

Issuing Time: We adopt the reveal and hide issuing time attributes from prior work [CMZ14, LdV17] and append two additional options (server-selected and joint), for a total of four:

- **Reveal** ($\mathcal{R}$): This option is used for attributes that are made clear to the LA to be verified and updated (such as the time) or recorded, such as spend-once token ids.
- **Hide** ($\mathcal{H}$): This option is used to hide identifying attributes of a credential from the LA, such as the specific bucket that a user knows. The LA is able to verify the validity of these attributes in zero knowledge without seeing the revealed attribute.
- **Server-selected** ($\mathcal{S}$): This option is used for attributes that are selected solely by the LA such as the bridge bucket attribute and others that are set to a particular value when an untrusted user joins the Lox system through an open invitation. When requesting an open invitation, the user leaves these server-selected attributes out of their request entirely and the server tells the user what their values are.
- **Joint** ($\mathcal{J}$): This option is used for issuing the credential ID Φ which is jointly created by the user and the LA. The user picks a random value for the ID, and encrypts the chosen value as if it were a hidden ($\mathcal{H}$) attribute. When the user requests a credential with the partially created encrypted Φ attribute, the server picks its own random value and homomorphically

adds it to the user's encrypted Φ value before computing the signature. It then tells the user the random value it added. That way, the server never learns the final credential ID before it is revealed at the subsequent presentation, and the client cannot unilaterally select it. This prevents the client from making two credentials with the same ID in hopes something will go wrong at verification time and the LA will incorrectly verify or issue an incorrect credential.

Presentation Time: All attributes, no matter how they are issued, will either be Revealed ($\mathcal{R}$) or Hidden ($\mathcal{H}$) from the LA when a credential or token is presented by the user.

5.3.2 Overview of Attribute Options for Lox Credentials and Tokens

Having described the different attribute options utilized by the Lox system, we summarize the attribute options used for each attribute in all transactions between the user and LA in Tables 5.1–5.5. These tables can be cross referenced with Figures 5.2– 5.7 which show the workflow of each of the Lox protocols as well as their attribute options at each step.

Lox Credential Attribute Options Across Protocols

Lox credentials are initially issued by the LA with either an open invitation or an invitation from a trusted user. In both cases, the user must present their blinded credential ID (Φ) to be contributed to and signed by the LA and then included as part of the issued credential. When the Lox credential is next presented to the LA, the credential ID (Φ) is revealed to the LA so that it can be added to a database of seen credential IDs, making the credential invalid for subsequent showings. The tables below show the attribute options for each protocol for issuing operations by the LA (Table 5.1) and credential presentation by the client (Table 5.2).

Table 5.1: **Lox Credential Issuing:** Lox credential attribute options in issuing operations by the LA. Note that trust promotion and check blockage protocols issue a migration key credential, not a Lox credential, and so are not included in this table.

Name	Description	Open Invitation	Trust Migration	Level Up	Issue Invitation	Redeem Invitation	Blockage Migration
$\Psi_{\mathfrak{L}}.\Phi$	ID	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$
$\Psi_{\mathfrak{L}}.t$	Time	$\mathcal{S}$	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{R}$
$\Psi_{\mathfrak{L}}.L$	Trust	0	1	$\mathcal{R}$	$\mathcal{H}$	1	$\mathcal{H}$
$\Psi_{\mathfrak{L}}.\beta$	Bucket	$\mathcal{S}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$
$\Psi_{\mathfrak{L}}.a$	Invitations Left	0	0	$\mathcal{S}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{S}$
$\Psi_{\mathfrak{L}}.d$	Blockages	0	0	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$

Table 5.2: **Lox Credential Presentation:** Lox credential attribute options in presentation transactions with the LA. Note that open-entry invitation and redeem invitation protocols are not included in this table, because they do not involve presenting Lox credential attributes. Invitation credential attribute options can be found in Table 5.3.

Name	Description	Trust Promotion	Level Up	Issue Invitation	Check Blockage	Migration
$\Psi_{\mathfrak{L}}.\Phi$	ID	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{R}$
$\Psi_{\mathfrak{L}}.t$	Time	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$
$\Psi_{\mathfrak{L}}.L$	Trust	0	$\mathcal{R}$	$\mathcal{H}$	$\mathcal{R}$	$\mathcal{R}$
$\Psi_{\mathfrak{L}}.\beta$	Bucket	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$
$\Psi_{\mathfrak{L}}.a$	Invitations Left	0	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$
$\Psi_{\mathfrak{L}}.d$	Blockages	0	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$

Invitation Credential Attribute Options

Invitation credentials $\Psi_{\mathfrak{I}}$ (Table 5.3) can be generated by users with a trust level $L \geq 2$. To ensure that invitations can both be verified by the LA as valid and remain unlinkable to the user that requests them, the invitation ID attribute $\Psi_{\mathfrak{I}}.\Phi$ is jointly issued and revealed by the invitee when requesting their Lox credential. The bucket β and blockage d attributes are hidden both at issue and claim time so the LA never learns the bucket of the inviter or invitee. The time attribute t is proved in zero knowledge to be less than 15 days old.

Table 5.3: **Credential:** Invitation Credential

Name	Description	Issue	Redeem
$\Psi_{\mathfrak{I}}.\Phi$	Invitation ID	$\mathcal{J}$	$\mathcal{R}$
$\Psi_{\mathfrak{I}}.t$	Time	$\mathcal{S}$	$\mathcal{H}$
$\Psi_{\mathfrak{I}}.\beta$	Bucket	$\mathcal{H}$	$\mathcal{H}$
$\Psi_{\mathfrak{I}}.d$	Bucket	$\mathcal{H}$	$\mathcal{H}$

Bucket Reachability Credential Attribute Options

Bucket reachability credentials $\Psi_{\mathfrak{R}}$ (Table 5.4) are made available and refreshed each day by the LA with only the date t and bucket β as attributes. Users present bucket reachability credentials along with their existing Lox credential when they engage in the level up or issue invitation protocols, which require they prove the liveness of their bucket β. Both β attributes (in the Lox credential and in the bucket reachability credential) are hidden from the LA but are able to be proved equal in zero knowledge.

Table 5.4: **Credential:** Bucket Reachability Credential

Name	Description	Presentation
$\Psi_{\mathfrak{R}}.t$	Time	$\mathcal{R}$
$\Psi_{\mathfrak{R}}.\beta$	Bucket	$\mathcal{H}$

Migration Token Attribute Options

As discussed above, the Migration Token $\Omega_{\mathfrak{M}}$ (Table 5.5) is created with the credential ID Φ and bucket β_{FROM} from the user's Lox credential and the decrypted migration table value β_{TO}. When presenting the Migration Token to the LA, Φ will be revealed with all β attributes hidden. However β_{FROM} will be proven in zero knowledge to match the user's presented $\Psi_{\mathfrak{L}}.\beta$ from their existing credential and β_{TO} will be proven in zero knowledge to match the user's prepared $\Psi_{\mathfrak{L}}.\beta$ (to be migrated to).

Table 5.5: **Credential:** Migration Token

Name	Description	Presentation
$\Omega_{\mathfrak{M}}.\Phi$	Lox credential ID	$\mathcal{R}$
$\Omega_{\mathfrak{M}}.\beta_{\text{FROM}}$	From Bucket	$\mathcal{H}$
$\Omega_{\mathfrak{M}}.\beta_{\text{TO}}$	To Bucket	$\mathcal{H}$

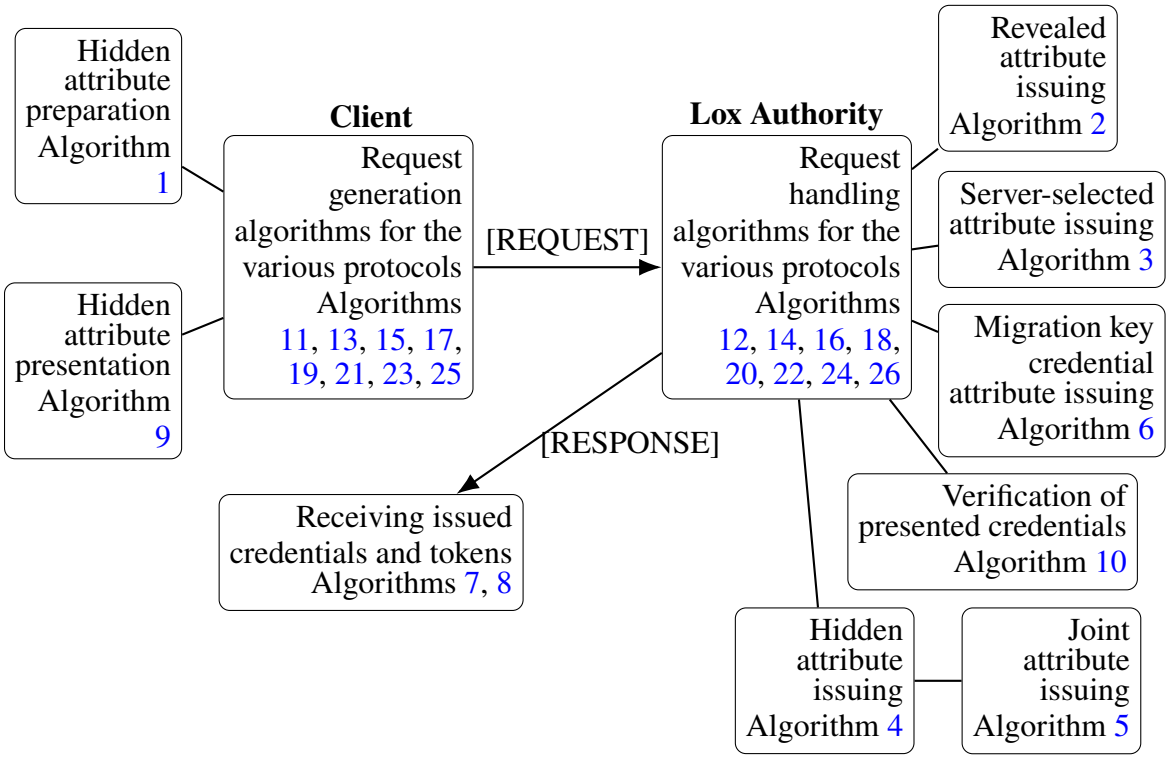


Figure 5.1: Overview of the Lox System and Protocols

5.4 Overview of our Lox System Implementation

In Figure 5.1 we provide an overview of our Lox system implementation. In the following sections we will describe each of the credential operations performed by the client and Lox Authority as well as each of the protocols that they request and handle.

5.4.1 Lox Credential Issuance Operations

A user prepares the revealed and hidden attributes of their credential for issuing by the LA and sends it. Upon receipt of the request, the LA computes the partial MACs on the revealed, server-selected, hidden and/or joint attributes and returns the MAC over all attributes as well as any server generated values. Finally, the response is used, along with the user's stored state, to generate the updated credential. In this section we detail the subroutines used for issuing a credential to a Lox user. We note that revealed attributes do not require additional processes for

preparation, so we omit this subroutine in favour of specifying revealed attributes and their values when preparing a request in Section 5.5.

Preparation of Hidden Attributes

The user prepares attributes to be issued in an updated credential by encrypting any attributes that must be kept secret (i.e., those that are not issued directly by the LA or else are not set to a system-defined value, such as 0, for the protocol) when shown prior to being verified and issued by the LA. The user prepares their credentials for presentation as demonstrated in Algorithm 1.

Algorithm 1 Prepare attributes that must be hidden from the Lox Authority when being verified for issuance of a new or updated Lox credential, adapted from Chase et al. [CMZ14]

Input

d, D	ElGamal key pair
$m_1, ..., m_n$	Lox credential attributes

Output To Present Request

$\pi_{\text{USER\_HIDDEN}}$	Proof that hidden attributes were correctly computed
$E_{m_1}, ..., E_{m_n}$	Hidden credential attributes to be issued

function PREPARE_HIDDEN$(d, D, m_1, ..m_n)$

for all $i \in 1, ..., n$ **do**

Generate $e_{m_i} \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $E_{m_i}[0] \leftarrow (e_{m_i} \cdot B,)$

Compute $E_{m_i}[1] \leftarrow (m_i \cdot B + e_{m_i} \cdot D)$ ▷ ElGamal Encryption of m_i

end for

Generate $\pi_{\text{USER\_HIDDEN}} \leftarrow NIZPK\{d, (m_i, e_i)_{i=1}^{n} :$

$E_{m_i} = e_i \cdot B, m_i \cdot B + e_i \cdot D \quad \forall i \in 1, ..., n$

$\wedge\ D = dB$

$\}$

return $\{E_{m_1}, ..., E_{m_n}, \pi_{\text{USER\_HIDDEN}}\}$

end function

Issuance of Revealed Attributes

The LA issues a Lox credential with revealed attributes $(m_i, ..., m_n) \in (\mathbb{Z}/\ell\mathbb{Z})^n$ by creating a tag (P, Q) on those attributes. The tag is then returned, together with a partial MAC on the revealed attributes as described in Algorithm 2.

Algorithm 2 Issue a Credential with Attributes revealed, adapted from Chase et al. [CMZ14]

Input Global LA State
$(\vec{x_{\mathfrak{L}}})$ Lox credential issuer private key
Input from Algorithms 14, 16, 18, 20, 22, 24, 26
P P component of MAC tag
$m_1, ..., m_n$ Revealed attributes to be issued
Output
$E_{Q_{\mathcal{R}}}$ Encrypted partial MAC on revealed attributes
s Random nonce used for ElGamal encryption of Q_R
function ISSUE_REVEALED$(P, m_1, ..., m_n)$
Compute $Q_{\mathcal{R}} = (\sum_{i=1}^{n} x_{\mathfrak{L},i} m_i)P$ ▷ Partial MAC on received m_i
Generate $s \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ ▷ Random nonce
Compute $E_{Q_{\mathcal{R}}}[0] \leftarrow sB$ ▷ Encrypt $Q_{\mathcal{R}}$
Compute $E_{Q_{\mathcal{R}}}[1] \leftarrow Q_{\mathcal{R}} + sD$
return $s, E_{Q_{\mathcal{R}}}$
end function

Issuance of Server-Selected Attributes

Aside from the credential ID, when a user first joins the Lox system with an open invitation, the LA issues server-selected attributes for every attribute. The user's initial trust level (L), available invitations (a) and blockages (d) are set to 0. The date a user joined is set to the current date. This is shown in Algorithm 3. Server-selected attribute issuance allows the LA to know most of the attributes in a user's initially issued credential. However, the credential ID (Φ) and ability to hide attributes in subsequent showings ensure that the LA is unable to identify or track users after this initial credential is issued.

Algorithm 3 Server-selected attribute issuance occurs when a user presents an untrusted invitation token, $\Omega_{\mathfrak{I}}$. Each value is assigned by the server, where BUCKET_SELECT is an auxiliary, customizable function that we envision (but do not detail) selecting a bucket based on some encoded indication of a user's geographical location or method of acquiring $\Omega_{\mathfrak{I}}$.

Input Global LA State

$(\vec{x_{\mathfrak{L}}})$	Lox credential issuer private key

Input from Algorithm 12

P	P component of MAC tag
$\Omega_{\mathfrak{I}}$	Optional invitation token

Output

$E_{Q_{\mathcal{S}}}$	Encrypted partial MAC on the server-selected attributes
$\Psi_{\mathfrak{L}}.\beta$	Bucket attribute
$\Psi_{\mathfrak{L}}.t$	Today's date for time attribute
s	Random nonce

function ISSUE_SERVER($P, \Omega_{\mathfrak{I}}$)

- Generate $\Psi_{\mathfrak{L}}.\beta \leftarrow$ BUCKET_SELECT($\Omega_{\mathfrak{I}}$) ▷ $\Omega_{\mathfrak{I}}$ encodes a suggested bucket
- Generate $\Psi_{\mathfrak{L}}.t \leftarrow \delta$ ▷ Set to today's date
- Generate $\Psi_{\mathfrak{L}}.\{L, a, d\} \leftarrow 0$
- Compute $Q_{\mathcal{S}} \leftarrow (\Psi_{\mathfrak{L}}.\beta \cdot x_{\mathfrak{L},\beta} + \Psi_{\mathfrak{L}}.t \cdot x_{\mathfrak{L},t})\, P$ ▷ Partial MAC on $\mathcal{S}$ attributes
- Generate $s \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ ▷ Random nonce
- Compute $E_{Q_{\mathcal{S}}}[0] \leftarrow sB$ ▷ Encrypt $Q_{\mathcal{S}}$
- Compute $E_{Q_{\mathcal{S}}}[1] \leftarrow Q_{\mathcal{S}} + sD$
- **return** $E_{Q_{\mathcal{S}}}, \Psi_{\mathfrak{L}}.\beta, \Psi_{\mathfrak{L}}.t, s$ ▷ Return to Algorithm 12

end function

Proof on Revealed Attributes

For both revealed and server-selected attribute issuing, the proof that values $E_{Q_{\mathcal{R},\mathcal{S}}}$ were created correctly follows:

$$
\begin{aligned}
\pi_{\text{ISSUE\_REVEALED}} = NIZPK\{(\tilde{x_{\mathfrak{L},0}}, x_{\mathfrak{L},0}, x_{\mathfrak{L},1}, ..., x_{\mathfrak{L},n}, s) : \\
&\wedge X_i = x_{\mathfrak{L},i}A && \forall i \in 1, ..., n \\
&\wedge X_0 = x_{\mathfrak{L},0}B + \tilde{x_{\mathfrak{L},0}}A \\
&\wedge E_{Q_{\mathcal{R},\mathcal{S}}}[0] = s \cdot B \\
&\wedge E_{Q_{\mathcal{R},\mathcal{S}}}[1] = s \cdot D + \sum_{i=1}^{n} x_{\mathfrak{L},i}(m_i P) && \forall i \in 1, ..., n \\
&\}
\end{aligned}
$$

We note that $\pi_{\text{ISSUE\_REVEALED}}$ is not sufficient as a proof for any of the Lox protocols we detail in Section 5.5 as every protocol includes at least one hidden attribute. However, to improve readability of the eventual π_{ISSUE} we detail the aspects of the proof that can be created for revealed and server-selected attributes alone.

Issuance of Hidden Attributes

All issuing operations involve some hidden attributes. Since the joint issuing process has some commonality with the hidden attribute issuing, we call the joint issuing subroutine from within the hidden attribute issuing algorithm. The LA computes the partial MAC on hidden attributes homomorphically as shown in Algorithm 4.

Algorithm 4 To issue hidden attributes, the LA first homomorphically encrypts each hidden attribute with a corresponding re-randomized secret key. When this has been done for each hidden attribute, the homomorphically encrypted attributes are then added together to create the partial MAC on hidden attributes. We note that $(\vec{x_{\mathfrak{C}}})$ describes a generic issuer private key that will be specified to match the protocol calling this Algorithm. Adapted from Chase et al. [CMZ14]

Input Global LA State
- $(\vec{x_{\mathfrak{C}}})$ Issuer private key for credential type

Input from Algorithms 12, 14, 16, 18, 20, 22, 24, 26
- b Random nonce
- $E_{m_1}, ..., E_{m_k}$ User encrypted hidden attributes to be issued

Output
- $E_{Q_{\mathcal{H}}}$ Partial MAC on hidden attributes
- $T_1, ..., T_k$ Auxiliary variables used to verify proofs
- j_Φ Random server selected ID

function ISSUE_HIDDEN$_{\mathfrak{C}}(b, E_{m_1}..., E_{m_k})$
- **for all** $i \in \{1, ...k\}$ **do**
  - **if** $i == 1$ **then** ▷ $\vec{x_{\mathfrak{C},1}}$ is always Φ
    - $E_{j_\Phi}, j_\Phi \leftarrow$ JOINT_ISSUE(E_{m_Φ}) ▷ Joint Issue E_{m_Φ} with Algorithm 5
    - $E_{m_i} \leftarrow E_{j_\Phi}$
  - **end if**
  - Compute $t_i \leftarrow x_i b$
  - Compute $T_i \leftarrow t_i A$
  - Compute $E_{T_{m_i}}[0] \leftarrow t_i \cdot E_{m_i}[0]$ ▷ Partial MAC on received E_{m_i}
  - Compute $E_{T_{m_i}}[1] \leftarrow t_i \cdot E_{m_i}[1]$
- **end for**
- Compute $E_{Q_{\mathcal{H}}}[0] \leftarrow (\sum_{i=1}^{k} E_{T_{m_i}}[0])$ ▷ Partial MAC on hidden attributes
- Compute $E_{Q_{\mathcal{H}}}[1] \leftarrow (\sum_{i=1}^{k} E_{T_{m_i}}[1]) + x_{\mathfrak{C},0} bB$
- **return** $E_{Q_{\mathcal{H}}}, T_1, ..., T_k, j_\Phi$ ▷ Return to calling Protocol

end function

Issuance of Joint Attributes

The LA issues joint attributes on the credential ID attribute (Φ), prepared for issue as the hidden attribute E_{m_Φ} for most protocols. To issue a joint attribute, the LA first selects a random value j_Φ as shown in Algorithm 5.

Algorithm 5 The Lox Authority issues a credential with joint attributes for each E_{m_Φ} presented by the client. This algorithm is called from Algorithm 4 when the ID attribute is seen by the LA. This algorithm shows only how the server generated random nonce is added to the encrypted Lox ID attribute passed by the user for issuing. Once the nonce is added to E_{j_Φ}, it is handled the same way as hidden attributes passed to Algorithm 4. This ensures both that the user cannot independently select or bias the ID while still keeping the ID hidden from the LA until it is presented.

Input from Algorithm 4

E_{m_Φ}	User encrypted credential ID

Output

E_{j_Φ}	Auxiliary variable to prove correctness of encryptions
j_Φ	Random server selected ID

upon receipt of: E_{m_Φ}

function ISSUE_JOINT(E_{m_Φ})

Generate $j_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$ ▷ Server generated random nonce

Compute $E_{j_\Phi}[0] \leftarrow (E_{m_\Phi}[0])$

Compute $E_{j_\Phi}[0] \leftarrow (E_{m_\Phi}[1] + j_\Phi \cdot B)$

return E_{j_Φ}, j_Φ ▷ Return to Algorithm 4

end function

The resulting E_{j_Φ} is then hidden by returning it to Algorithm 4 with the other attributes prepared by the client for issuing.

Proof on Issued Attributes

With both hidden and revealed attributes, the LA homomorphically adds the encrypted attributes together to get:

$$E_Q \leftarrow E_{Q_{\mathcal{H}}} + E_{Q_{\mathcal{R}}} + E_{Q_{\mathcal{S}}}$$

which is proved correct with the combined zero-knowledge proof:

$$\begin{aligned}
\pi_{\text{ISSUE}} \leftarrow NIZPK\{ & \tilde{x_0}, x_0, x_1, ..., x_n, b, t_1, ..., t_n, s) : \\
& \wedge X_i = x_i A \qquad \forall i = 1, ..., n \\
& \wedge X_0 = x_0 B + \tilde{x_0} A \\
& \wedge P = bB \\
& \wedge T_i = bX_i \wedge T_i = t_i A \qquad \forall_i \in 1, .., n \\
& \wedge E_Q[0] = (sB + \sum_{i=1}^{n} t_i E_i[0]) \\
& \wedge E_Q[1] = (sD + \sum_{i=1}^{n} t_i E_i[1]) + \\
& \quad x_0 P + \sum_{i=1}^{n} x_i (m_i P) \\
& \}
\end{aligned}$$

Issuance of Migration Key Credentials

Migration key credentials are issued as a result of making either a trust promotion or check blockage request to the LA that is successful. We detail the migration credential issuing in Algorithm 6.

Algorithm 6 Issue a migration key credential $(Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}})$ of the specified type with $E_{\text{TABLE}_{\mathfrak{M}}}$ being a hashtable with encrypted entries that the client can decrypt using their $\Psi_{\mathfrak{L}}.\Phi$, $\Psi_{\mathfrak{L}}.\beta$ attributes and the decrypted MAC Qk on the migration key credential .

Input Server global state

$\vec{x_{\mathfrak{M}}}$	Migration secret key
$\vec{x_{\mathfrak{K}}}$	Migration key secret key
bridge_table	Table of Lox bridges

Input User

D	ElGamal key
$\Psi_{\mathfrak{L}}.\Phi$	Credential ID
$E_{\Psi_{\mathfrak{L}}.\beta}$	Hidden bucket attribute
TYPE	Type of migration: trust promotion or blockage

Output

Pk	Migration key credential tag
E_{Qk}	Encrypted migration key credential MAC
$E_{\text{TABLE}_{\mathfrak{M}}}$	Hashtable of eligible bucket migrations, whose entries are encrypted with $\Psi_{\mathfrak{L}}.\Phi$, $\Psi_{\mathfrak{L}}.\beta$, Qk as described in Section 5.2

function ISSUE_MIGRATION(D, $\Psi_{\mathfrak{L}}.\Phi$, $E_{\Psi_{\mathfrak{L}}.\beta}$, TYPE)

Select $b, s \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$
Generate $Pk \leftarrow bB$
Compute $Q_{k\Phi} \leftarrow (x_{\tilde{\mathfrak{M}},0} + x_{\mathfrak{M},\Phi}\Psi_{\mathfrak{L}}.\Phi)Pk$
Compute $E_{Q_{k\Phi}}[0] \leftarrow sB$ ▷ Partial MAC on Φ
Compute $E_{Q_{k\Phi}}[1] \leftarrow Q_{k\Phi} + sD$
Compute $t_\beta \leftarrow x_{\mathfrak{M},\beta} \cdot b$
Compute $E_{Q_{k\beta}}[0] \leftarrow t_\beta \cdot E_{\Psi_{\mathfrak{L}}.\beta}[0]$ ▷ Partial MAC on β
Compute $E_{Q_{k\beta}}[1] \leftarrow t_\beta \cdot E_{\Psi_{\mathfrak{L}}.\beta}[1]$
Compute $E_{Qk} = (E_{Qk_\Phi}[0] + E_{Qk_\beta}[0], E_{Qk_\Phi}[1] + E_{Qk_\beta}[1])$
Generate $\text{E}_{\text{TABLE}_{\mathfrak{M}}} \leftarrow$ ENCRYPT_TABLE($\Psi_{\mathfrak{L}}.\Phi$, bridge_table, P_k, $\vec{x_{\mathfrak{M}}}$, $\vec{x_{\mathfrak{K}}}$, TYPE)
▷ See Section 5.2

return $Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}}$ to the user

end function

Receiving Issued Credentials

Upon receipt of the encrypted issued credential from the LA, the client decrypts it to get their updated credential as described in Algorithm 7.

Algorithm 7 The user accepts the credential received by the LA if π_{ISSUE} verifies as adapted from Chase et al. [CMZ14].

Input Server
- $\Psi_{\mathfrak{C}}.P$ — P component of MAC tag corresponding to $\Psi_{\mathfrak{C}}$
- $\Psi_{\mathfrak{C}}.E_Q$ — Encrypted MAC Q
- $\Psi_{\mathfrak{C}}.t_\Phi, \Psi_{\mathfrak{C}}.t_1, ..., \Psi_{\mathfrak{C}}.t_n$ — Server-selected issued attributes
- π_{ISSUE}

Input User
- d, D — ElGamal key pair from client state
- $m_1, ..., m_n$ — Plaintext attributes from client state

Output
- $\Psi_{\mathfrak{C}}$ — Generic credential, typically a Lox credential

upon receipt of: $P, E_Q, t_\Phi, t_1..., t_n, \pi_{\text{ISSUE}}$ where the request is for any Lox protocol

function HANDLE($\Psi_{\mathfrak{C}}.P, \Psi_{\mathfrak{C}}.E_Q, \Psi_{\mathfrak{C}}.t_\Phi, \Psi_{\mathfrak{C}}.t_1..., \Psi_{\mathfrak{C}}.t_n, \pi_{\text{ISSUE}}$)
  if VERIFY(P, π_{ISSUE})) **then** ▷ Check: P$\neq$ Identity, ZKP verifies
    Generate $\Psi_{\mathfrak{C}}\{$
      $\Psi_{\mathfrak{C}}.P = P$
      $\Psi_{\mathfrak{C}}.Q = \Psi_{\mathfrak{C}}.E_Q[1] - (d \cdot \Psi_{\mathfrak{C}}.E_Q[0])$ ▷ Decrypt $\Psi_{\mathfrak{C}}.E_Q$
      $\Psi_{\mathfrak{C}}.\Phi = m_\Phi + t_\Phi$ ▷ Add received t_Φ to client state m_Φ
    $\}$
    for all $i \in 1, ..., n$ **do**
      if $\Psi_{\mathfrak{C}}.t_i \in \Psi_{\mathfrak{C}}.t_1..., \Psi_{\mathfrak{C}}.t_n$ **then**
        $\Psi_{\mathfrak{C}}.i = \Psi_{\mathfrak{C}}.t_i$
      else$\Psi_{\mathfrak{C}}.i = \Psi_{\mathfrak{L}}.m_i$
      end if
    end for
    return $\Psi_{\mathfrak{C}}$
  end if
  return Proof Failure
end function

Retrieving the Migration Token

After making a trust promotion or check blockage request, the user decrypts the received encrypted hashtable in the LA's response to create a migration token as described in Algorithm 8.

Algorithm 8 The user accepts the migration key credential received from the LA and uses their $\Psi_{\mathfrak{L}}.\Phi$, $\Psi_{\mathfrak{L}}.\beta$ attributes, the decrypted MAC on the migration key credential Qk, and the migration public keys to form a migration token to migrate to a new bucket

Input Server
  Pk — Migration key credential key tag
  E_{Qk} — Encrypted migration key credential MAC
  $E_{\text{TABLE}_{\mathfrak{M}}}$ — Hashtable of bucket migrations, with entries encrypted with $\Psi_{\mathfrak{L}}.\Phi$, $\Psi_{\mathfrak{L}}.\beta$, Qk as described in Section 5.2
Input Client State
  d — ElGamal private key from client state
  $\Psi_{\mathfrak{L}}$ — Lox credential
Output
  $\Omega_{\mathfrak{M}}$ — Migration token
function HANDLE_MIGRATION(Pk, E_{Qk}, $E_{\text{TABLE}_{\mathfrak{M}}}$)
  if $Pk \neq$ Identity **then**
    Generate $Q_k \leftarrow E_{Q_k}[1] - (d \cdot E_{Q_k}[0])$
    $\Omega_{\mathfrak{M}} \leftarrow$ DECRYPT_TABLE(Q_k, $\Psi_{\mathfrak{L}}.\Phi$, $\Psi_{\mathfrak{L}}.\beta$, TYPE, $E_{\text{TABLE}_{\mathfrak{M}}}$) ▷ See Section 5.2
    return $\Omega_{\mathfrak{M}}$
  end if
end function

The migration token $\Omega_{\mathfrak{M}}$ is made up of the following attributes:

$$\Omega_{\mathfrak{M}} : \{ \quad Pk : \Omega_{\mathfrak{M}}.Pk \\ Q_k : \Omega_{\mathfrak{M}}. \\ \Phi : \Psi_{\mathfrak{L}}.\Phi \\ \beta_{\text{FROM}} : \Psi_{\mathfrak{L}}.\beta \\ \beta_{\text{TO}} : \Omega_{\mathfrak{M}}.\beta \quad \}$$

where the indicated $\Psi_{\mathfrak{L}}$ attributes are the same as those in the client's existing Lox credential and the $\Omega_{\mathfrak{M}}.\beta_{\text{TO}}$ attribute indicates the bucket that was revealed by decrypting the entry in the migration table to find the bucket the client is eligible to migrate to.

5.4.2 Lox Credential Presentation

A user presents their revealed and hidden attributes to the LA for a given protocol in order to prove something about the credential they have. Upon receipt of the request, the LA verifies that the required conditions for the protocol are met. If the verification is successful, the LA accepts the attributes and continues to issue the prepared attributes the user has sent (as described in Section 5.4.1) otherwise, the request fails. In this section we detail the subroutines used for presenting and verifying a Lox credential. We omit the revealed attributes subroutine noting that these do not require additional processes for presentation. Instead, we specify which attributes are revealed when preparing a request in Section 5.5.

Presentation of Hidden Attributes

When a user interacts with the LA, they must present some combination of revealed and hidden attribues from the credentials they hold. For those attributes that are hidden, the user must be able to prove that they are correct in order to be verified by the LA. For a credential $(P, Q, m_1..., m_n)$ where (P, Q) have already been rerandomized by multiplying both components by a random t as shown in Section 5.2 and where some subset of attributes will be hidden, the user will proceed to present their credentials as demonstrated in Algorithm 9, which was first described by Chase et al. [CMZ14].

Algorithm 9 Present attributes that must be hidden from the LA, adapted from Chase et al. [CMZ14]. For each attribute, including the Q component of the MAC, a random value is selected and a Pedersen commitment to the attribute is formed.

Input Client Global State

$\vec{X}$	Lox credential issuer public key, type must be specified ($\mathfrak{L}, \mathfrak{R}, \mathfrak{M}, \mathfrak{I}$)

Input

P, Q	MAC Tag
$m_1, ..., m_n$	Lox credential attributes

Output To Present Request

$C_{h_1}, ..., C_{h_n}$	Pedersen commitments to hidden attributes
C_Q	Pedersen commitment to MAC Q
$\pi_{\text{CRED\_SHOW}}$	Proof that hidden values were generated correctly

function PRESENT_HIDDEN($P, Q, m_1, ..., m_n$)

for all $i \in 1, .., n$ **do**

Generate $z_i \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $C_{h_i} \leftarrow m_i \cdot P + z_i \cdot A$ ▷ Form Pedersen commitment on m_i

end for

Generate $z_Q \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $C_Q \leftarrow Q + z_Q \cdot A$ ▷ Form Pedersen commitments to the MAC

Compute $V \leftarrow (\sum_{i=1}^{n} z_i \cdot X_i) - z_Q \cdot A$ ▷ Error Factor

Generate $\pi_{\text{CRED\_SHOW}} \leftarrow NIZPK\{((m_i, z_i)_{i=1}^{n}, z_Q) :$

$C_{h_i} = m_i \cdot P + z_i \cdot A \quad \forall i \in 1, ..., n$

$\wedge V \leftarrow (\sum_{i=1}^{n} z_i \cdot X_i) - z_Q \cdot A$

$\}$

return $C_{h_i}, C_Q, \pi_{\text{CRED\_SHOW}}$

end function

Along with the revealed attributes and the Pedersen commitments to hidden attributes, the user also generates a zero-knowledge proof of knowledge (π_{USER}) with elements of both the hidden attributes to be presented and the ElGamal encrypted attributes to be issued. This must be verified by the LA.

Verification of Presented Credential

Algorithm 10 When a Lox credential is presented by the user, the LA first validates each revealed attribute then recomputes the error factor using the secret key. If the protocol requested requires that certain hidden values are between a range, the LA verifies the range proof on the hidden value. All of these are used to verify π_{USER}. If the user and LA are honest, and all values were computed correctly, the proof verifies and the LA records the revealed Lox ID attribute $\Psi_{\mathfrak{L}}.\Phi$ as having been seen. Adapted from Chase et al. [CMZ14]

Input Server Global State

$\vec{x_{\mathfrak{L}}}$	Lox credential issuer private key
$\vec{x_{\mathfrak{M}}}$	Migration credential issuer private key
$\vec{x_{\mathfrak{R}}}$	Reachability credential issuer private key
$\vec{x_{\mathfrak{I}}}$	Invitation credential issuer private key

Input From Client

P	P component of MAC tag
$m_1, ..., m_n$	Revealed Lox credential attributes
$C_{h_1}, ..., C_{h_n}$	Pedersen commitments to hidden attributes
C_Q	Pedersen commitment to MAC Q
π_{USER}	Proof that attribute values were generated correctly
TYPE	Indicator of Migration type or false

upon receipt of: $P, (m_1, ..., m_n), (C_{h_1}, ..., C_{h_n}), C_Q$, and π_{USER} from issue function
  function VERIFY($P, (m_1, ..., m_n), (C_{h_1}, ..., C_{h_n}), C_Q, \pi_{\text{USER}}$, TYPE)
    if P$\neq$ Identity **then**
      Check m_i validity $\forall i \in 1, ..., n$
      for Each Ψ of types $\{\mathfrak{L}, \mathfrak{M}, \mathfrak{R}, \mathfrak{I}\}$ in INPUT **do**
        Using corresponding issuer private key $\vec{x}$:
        Compute: $V' \leftarrow (x_0 + \sum_{i=1}^{n} x_i m_i)\, P + \sum_{i=1}^{n} x_i C_{h_i} - C_Q$ ▷ Recompute error
      end for
      if Verify π_{USER} using V' for V **then**
        if TYPE == Trust_Promotion $\wedge$ $\Psi_{\mathfrak{L}}.\Phi$ not seen before **then**
          Add $\Psi_{\mathfrak{L}}.\Phi$ to seen $\Omega_{\mathfrak{M}}.\Phi$ list ▷ Add to Trust Promotion Seen list
          return ACCEPT
        else if TYPE == Blockage_Migration $\wedge$ $\Psi_{\mathfrak{L}}.\Phi$ not seen before **then**
          return ACCEPT ▷ Do not add to list, allow multiple attempts
        else if TYPE == Invitation $\wedge$ $\Psi_{\mathfrak{I}}.\Phi$ not seen before **then**
          Add $\Psi_{\mathfrak{I}}.\Phi$ to seen $\Psi_{\mathfrak{I}}.\Phi$ list

```
                return ACCEPT                                  ▷ Add to invitation credential seen list
            else if Ψ_ℒ.Φ not seen before then
                Add Ψ_ℒ.Φ to seen Ψ_ℒ.Φ list
                return ACCEPT                                  ▷ Add to Lox credential seen list
            end if
        end if
    end if
    return FAIL
end function
```

5.5 Lox Credentials Protocols

In this section we describe the implementation of each of the Lox protocols. There are several credentials and tokens used for each of the different protocols, all issued by the LA. We describe the Lox protocols beginning with an untrusted user joining the system by acquiring an open-entry invitation. We then continue as the user works their way into a more and more trusted position, levelling up to have buckets with more bridges, issue invitations and finally be eligible to migrate to a new bucket if their bridges are blocked. The Lox design is discussed in greater detail in Chapter 4.

We provide an overview of each Lox protocol as well as a sequence diagram with each of the presentation, preparation and issuing attribute options distinguished visually. To this end:
Present Hidden attributes will be represented by a light grey background and dashed blue frame.
Prepare Hidden attributes will be represented by a light grey background and solid red frame.
Joint Issue attributes will be represented by a darker grey background and dashed black frame.
Hidden Issue attributes will be represented by a darker grey background and solid black frame.
All revealed and server selected attributes will appear without any shaded background.

The sequence diagrams are meant to provide greater insight, at a glance, into the general purpose of each protocol, focusing on the attributes that are distinct across protocols with the background shading scheme simplifying, in some cases, actual output of calling one of the functions listed in Sections 5.4.1 and 5.4.2. For all protcols and sequence diagrams, δ is used to signify today's date. When it is ambiguous as to which credential is an old credential being presented and which is a new credential being issued, $*$ is used to indicate a new credential.

5.5.1 Open Entry Invitation

As detailed in Chapter 4, the open entry invitation is the entry to the Lox system for untrusted users.

To claim an open entry invitation and receive a Lox credential, $\Psi_{\mathfrak{L}}$, the user first passes their received invitation token, $\Omega_{\mathfrak{I}}$ to the $\text{PRESENT}_{\text{OPEN\_INVITATION}}$ function in Algorithm 11. This prepares the request for the initial Lox credential issuance to be sent to the LA. The user sends the output of Algorithm 11, that is, $\Omega_{\mathfrak{I}}, D, E_{m_\Phi}, \pi_{\text{USER\_HIDDEN}}$, to the LA to be verified for a new Lox credential.

The LA's $\text{ISSUE}_{\text{OPEN\_INVITATION}}$ function in Algorithm 12 verifies the request by first checking the signature on the open invitation is valid and checks the invitation ID against the list of used

invitation nonces. If the invitation is unspent, the invitation ID is added to this list. The LA computes its part of the joint credential ID by selecting its own random value j_Φ and adding it homomorphically to the user's encrypted ID component E_{m_Φ}. The remaining attributes in $\Psi^*_\mathfrak{L}$ are either generated as in Algorithm 3, using server-selected issuance (β, t) or are given values of 0 (L, a, d).

Finally, the LA sends the open entry issue response to the user and the user handles the response as in Algorithm 7. This protocol is expressed in the Sequence Diagram Figure 5.2 below with the resulting Lox credential:

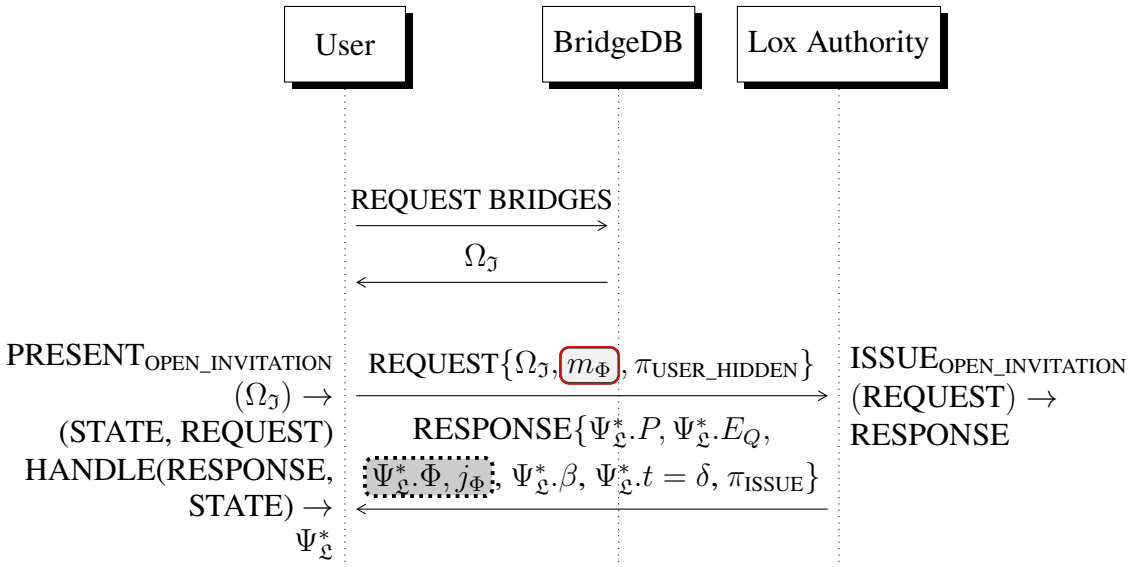


Figure 5.2: Overview of the Open Entry Invitation Protocol

$$
\begin{aligned}
\Psi^*_\mathfrak{L}\{ \quad & P : \Psi^*_\mathfrak{L}.P \\
& Q : \Psi^*_\mathfrak{L}.E_Q[1] - d \cdot \Psi^*_\mathfrak{L}.E_Q[0] \\
& \Phi : m_\Phi + j_\Phi \\
& \beta : \Psi^*_\mathfrak{L}.\beta \\
& L : 0 \\
& t : \delta \\
& a : 0 \\
& d : 0 \qquad \}.
\end{aligned}
$$

Algorithm 11 Open Entry Client Request: A user presents their invitation token to the LA along with a prepared encrypted credential ID that will be used by the LA to issue a new Lox credential.

Input Client
  $\Omega_{\mathfrak{I}}$ Invitation token
Output To Server
  REQUEST A request containing: $\Omega_{\mathfrak{I}}, D, E_{m_\Phi}, \pi_{\text{USER}}$
Output To Client State
  STATE A state containing: d, D, E_{m_Φ}, m_Φ
function PRESENT$_{\text{OPEN_INVITATION}}(\Omega_{\mathfrak{I}})$
  Generate $d, m_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$
  Compute $D \leftarrow dB$
  Compute $(E_\Phi, \pi_{\text{USER\_HIDDEN}}) \leftarrow$ PREPARE_HIDDEN(D, m_Φ) ▷ See Algorithm 1
  Set REQUEST $\leftarrow (\Omega_{\mathfrak{I}}, D, E_{m_\Phi}, \pi_{\text{USER\_HIDDEN}})$
  Set STATE $\leftarrow (d, D, E_{m_\Phi}, m_\Phi)$
  Send to Issue$_{\text{OPEN_INVITATION}}$(REQUEST) ▷ Algorithm 12
  Store STATE
end function

Algorithm 12 Open Entry Server Response: The LA receives an invitation token along with a prepared encrypted credential ID and issues a new credential to the user.

Input Client

$\Omega_{\mathfrak{I}}$	Invitation token
D	ElGamal public key
E_{m_Φ}	Client encrypted Lox ID
$\pi_{\text{USER\_HIDDEN}}$	Proof of correct user blinding

Output To Client

RESPONSE	A response to the client: $P, E_Q, j_\Phi, T_\Phi, \Psi^*_{\mathfrak{L}}.\beta, \Psi^*_{\mathfrak{L}}.t, \pi_{\text{ISSUE}}$

function $\text{ISSUE}_{\text{OPEN\_INVITATION}}(\Omega_{\mathfrak{I}}, D, E_{m_\Phi}, \pi_{\text{USER\_HIDDEN}})$

- Check validity of $\Omega_{\mathfrak{I}}$ ▷ $\Omega_{\mathfrak{I}}$ is fresh
- Verify $\pi_{\text{USER\_HIDDEN}}$ ▷ Check proof is correct
- Generate $b \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$
- Compute $P \leftarrow b \cdot B$ ▷ Generate P
- $E_{Q_\mathcal{H}}, T_\Phi, j_\Phi \leftarrow \text{ISSUE\_HIDDEN}_{\mathfrak{L}}(E_{m_\Phi})$ ▷ Algorithm 4
- $\Psi^*_{\mathfrak{L}}.\beta, \Psi^*_{\mathfrak{L}}.t, E_{Q_\mathcal{S}}, s \leftarrow \text{ISSUE\_SERVER}(P, \Omega_{\mathfrak{I}})$ ▷ Algorithm 3
- Compute $E_Q \leftarrow E_{Q_\mathcal{S}} + E_{Q_\mathcal{H}}$ ▷ Homomorphically compute MAC
- Generate π_{ISSUE} ▷ Generate proof as in Section 5.4.1
- Set RESPONSE $\leftarrow (P, E_Q, j_\Phi, T_\Phi, \Psi^*_{\mathfrak{L}}.\beta, \Psi^*_{\mathfrak{L}}.t, \pi_{\text{ISSUE}})$
- Send RESPONSE to the Client: $\text{HANDLE}(\text{RESPONSE}_{\mathfrak{L}})$ ▷ Algorithm 7

end function

5.5.2 Trust Promotion and Trust Migration

As described in Chapter 4, the trust promotion protocol is used only to upgrade from a trust level $L = 0$ to a trust level $L = 1$ and marks the transition from an untrusted user to a trusted user.

To initiate the Trust Promotion interaction, the user must present a valid Lox credential, $\Psi_{\mathfrak{L}}$ where the value for $\Psi_{\mathfrak{L}}.t$ can be proved to be at least 30 days old and not more than 541 (511+30) days old, and the bridge associated with $\Psi_{\mathfrak{L}}.\beta$ remains unblocked. We choose 511 as the expiry date for Lox credentials because values that are $2^k - 1$ make range proofs more efficient. Note that L, a, d attributes are all 0 so do not need to be sent. To reveal the plain value for t and β would deanonymize the user and make their transactions linkable. Thus, a user must hide these values and prove in zero knowledge that their credential is eligible to be upgraded. The user calls $\text{PRESENT}_{\text{Trust\_Promotion}}$, Algorithm 13, with $\Psi_{\mathfrak{L}}.\Phi$ revealed, and $\Psi_{\mathfrak{L}}.\beta$, and $\Psi_{\mathfrak{L}}.t$ hidden. This is depicted in the first step of the Sequence Diagram in Figure 5.3.

The user sends their request to the LA to issue a migration key credential. If the request verifies, the user is eligible for a trust promotion and the LA prepares a migration key credential as described in Section 4.2.2. To do this, the LA first verifies that the revealed $\Psi_{\mathfrak{L}}.\Phi$ has not been seen before, either in the general ID filter or the filter specifically for trust promotion. If this check succeeds, $\Psi_{\mathfrak{L}}.\Phi$ is added to the trust promotion filter *only* in order to keep the $\Psi_{\mathfrak{L}}$ fresh for the second step of the protocol, when the migration actually occurs. This ensures that the migration credential is only issued to a given user once and is tied to a particular Lox credential that has been confirmed as being eligible for migration.

The migration key credential gives the user all of the information they require to start constructing both their migration token, $\Omega_{\mathfrak{M}}$ and updated Lox credential, $\Psi^*_{\mathfrak{L}}$, for the trust migration. The Trust Promotion presentation (Algorithm 13) and issuing (Algorithm 14) Algorithms as well as a sequence diagram (Figure 5.3) of the entire two step protocol are detailed below.

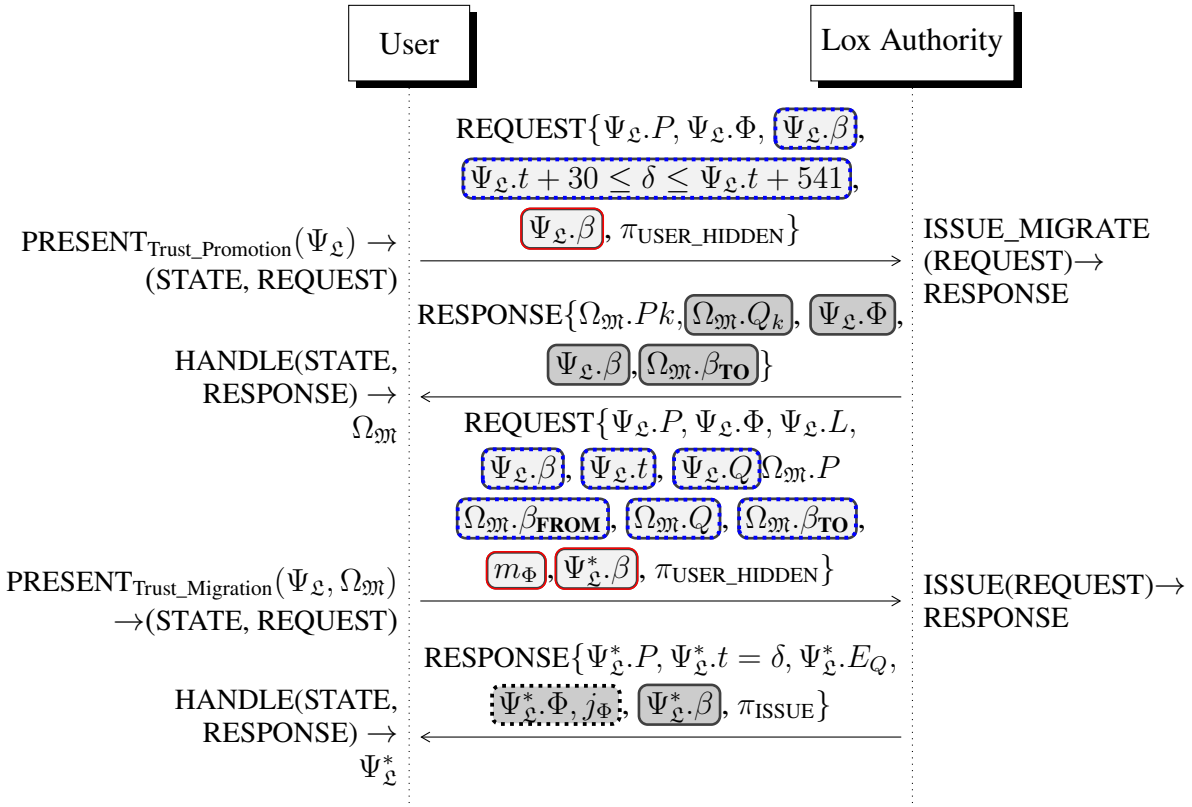


Figure 5.3: Overview of the Trust Promotion and Trust Migration Protocols.

$$
\begin{aligned}
\Omega_{\mathfrak{M}} : \{ \quad & Pk : \Omega_{\mathfrak{M}}.Pk \\
& Q_k : E_{\Omega_{\mathfrak{M}}.Q_k}[0] - d - E_{\Omega_{\mathfrak{M}}.Q_k}[1] \\
& \Phi : \Psi_{\mathfrak{L}}.\Phi \\
& \beta_{\text{FROM}} : \Psi_{\mathfrak{L}}.\beta \\
& \beta_{\text{TO}} : \Omega_{\mathfrak{M}}.\beta_{\text{TO}} \\
& \text{Migration Type: Trust Promotion} \quad \}
\end{aligned}
$$

$$
\begin{aligned}
\Psi^*_{\mathfrak{L}}\{ \quad & P : \Psi^*_{\mathfrak{L}}.P \\
& Q : \Psi^*_{\mathfrak{L}}.E_Q[1] - d \cdot \Psi^*_{\mathfrak{L}}.E_Q[0] \\
& \Phi : m_{\Phi} + j_{\Phi} \\
& \beta : \Omega_{\mathfrak{M}}.\beta_{\text{TO}} \\
& L : 1 \\
& t : \delta \\
& a : 0 \\
& d : 0 \quad \}.
\end{aligned}
$$

Algorithm 13 Trust Promotion Presentation: A user presents their hidden bucket and the time since they got their open invitation credential to the Lox Authority. The user shows using a range proof that the hidden time since receiving the attribute, t, has a value that is at least 30 days old (i.e., $\delta - 30$) and is not more than 511 + 30 days old. This makes them eligible for a trust promotion. The user prepares their revealed credential ID and hidden bucket attribute to be issued by the Lox Authority as part of the migration credential. This can then be used towards a trust level upgrade. Hidden attributes presented for verification and prepared for issuing are proved to be computed correctly with π_{USER}

Input Client
  $\Psi_{\mathfrak{L}}$ Lox credential
Output To Server
  REQUEST Contains: $P, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\beta, C_{\Psi_{\mathfrak{L}}}.t, C_Q, D, E_{\Psi_{\mathfrak{L}}}.\beta, \pi_{\text{USER}}$
Output To Client State
  STATE Contains: $d, D, E_{\Psi_{\mathfrak{L}}}.\beta, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.\beta$
function PRESENT$_{\text{TRUST_PROMOTION}}(\Psi_{\mathfrak{L}})$
  Generate $d, w \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$
  Compute $D \leftarrow dB$
  Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags
  $(C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}, C_Q, \pi_{\text{CRED\_SHOW}}) \leftarrow$ PRESENT_HIDDEN$(P, Q, \Psi_{\mathfrak{L}}.\{\beta, t\})$ ▷ Algorithm 9
  $(E_{\Psi_{\mathfrak{L}}}.\beta, \pi_{\text{USER\_HIDDEN}}) \leftarrow$ PREPARE_HIDDEN$(D, \Psi_{\mathfrak{L}}.\beta)$ ▷ Algorithm 1
  Generate $\pi_{\text{USER}} \leftarrow \{\pi_{\text{CRED\_SHOW}} \wedge \pi_{\text{USER\_HIDDEN}}$
    $\wedge$ Prove $\Psi_{\mathfrak{L}}.t + 30 \leq \delta \leq \Psi_{\mathfrak{L}}.t + 541$
    $\}$
  Set REQUEST $\leftarrow (P, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}, C_Q, D, E_{\Psi_{\mathfrak{L}}}.\beta, \pi_{\text{USER}})$
  Set STATE $\leftarrow (d, D, E_{\Psi_{\mathfrak{L}}}.\beta, \Psi_{\mathfrak{L}}.\{\Phi, \beta\})$
  Send to ISSUE_MIGRATION(REQUEST) ▷ Algorithm 14
  Store STATE
end function

Algorithm 14 Trust Promotion Server Response: The LA receives the user's request, verifies the input from $\Psi_{\mathfrak{L}}$ and in particular that $\Psi_{\mathfrak{L}}.\Phi$ has not been used to request a trust migration before and that $\Psi_{\mathfrak{L}}.t$ is at least 30 days old but at most 511+30 days old. If successful, the LA issues a migration key credential which can be decrypted to create $\Omega_{\mathfrak{M}}$ used for the next step

Input Client

P, C_Q	Hidden MAC on Lox attributes
$\Psi_{\mathfrak{L}}.\Phi$	Lox credential ID
$C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}$	Hidden Lox attributes β and t
$E_{\Psi_{\mathfrak{L}}}.\beta$	Client encrypted β attribute
D	ElGamal public key
π_{USER}	Proof of correct user blinding

Output To Client

RESPONSE	A response to the client: $Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}}$

function ISSUE$_{\text{TRUST_PROMOTION}}$($P, C_Q, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}, E_{\Psi_{\mathfrak{L}}}.\beta, D, \pi_{\text{USER}}$)
  if VERIFY($P, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}, \pi_{\text{USER}}$) **then** ▷ Algorithm 10
    $Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}} \leftarrow$ISSUE_MIGRATION($D, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.E\beta$, Trust_Promotion) ▷ Algorithm 6
    Set RESPONSE $\leftarrow (Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}})$
    Send to HANDLE_MIGRATION(RESPONSE) ▷ Algorithm 8
  end if
end function

Trust Migration

Algorithm 15 Trust Migration Presentation: A user presents hidden attributes from their Lox credential and their migration credential to the Lox Authority, proving that the hidden bucket from their Lox credential and the from bucket in the migration credential are equivalent. The user prepares an updated Lox credential with encrypted credential ID and bucket, where the encrypted bucket matches the migration credential's to bucket. Attributes presented for verification and prepared for issuing are proved to be computed correctly with π_{USER}.

Input Client

$\Psi_{\mathfrak{L}}$	Lox credential
$\Omega_{\mathfrak{M}}$	Migration Token

Output To Server

REQUEST	A Request containing: $P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\beta, C_{\Psi_{\mathfrak{L}}}.t, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\beta_{\text{FROM}} C_{\Omega_{\mathfrak{M}}}.\beta_{\text{TO}}, C_{Q_{\mathfrak{M}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta\}, \pi_{\text{USER}}$

Output To Client State

STATE	A State containing: $d, D, E^*_{\Psi_{\mathfrak{L}}}.\Phi, E^*_{\Psi_{\mathfrak{L}}}.\beta, m_\Phi, {\Psi_{\mathfrak{L}}}^*.\beta^*$

function $\text{PRESENT}_{\text{TRUST\_MIGRATION}}(\Psi_{\mathfrak{L}}, \Omega_{\mathfrak{M}})$

Generate $d, w, s, m_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $\quad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

Compute $P_{\mathfrak{M}} \leftarrow s \cdot \Omega_{\mathfrak{M}}.P$ $\quad Q_{\mathfrak{M}} \leftarrow s \cdot \Omega_{\mathfrak{M}}.Q$

Compute $(C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}, C_Q, \pi_{\text{CRED\_SHOW}_{\mathfrak{L}}}) \leftarrow$
$\hookrightarrow$ PRESENT_HIDDEN$(P, Q, \Psi_{\mathfrak{L}}.\beta, \Psi_{\mathfrak{L}}.t)$ ▷ Algorithm 9

Compute $(C_{\Omega_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}\}, C_{Q_{\mathfrak{M}}}, \pi_{\text{CRED\_SHOW}_{\mathfrak{M}}}) \leftarrow$
$\hookrightarrow$ PRESENT_HIDDEN$(P_{\mathfrak{M}}, Q_{\mathfrak{M}}, \Omega_{\mathfrak{M}}.\beta_{\text{FROM}}, \Omega_{\mathfrak{M}}.\beta_{\text{TO}})$ ▷ Algorithm 9

Compute $(E^*_{\Psi_{\mathfrak{L}}}.\beta, E^*_{\Psi_{\mathfrak{L}}}.\Phi, \pi_{\text{USER\_HIDDEN}}) \leftarrow$ PREPARE_HIDDEN$(D, \Omega_{\mathfrak{M}}.\beta^*, m_\Phi)$ ▷ Algorithm 1

Generate $\pi_{\text{USER}} \leftarrow \{\pi_{\text{CRED\_SHOW}_{\mathfrak{L}}} \wedge \pi_{\text{USER\_HIDDEN}} \wedge \pi_{\text{CRED\_SHOW}_{\mathfrak{M}}}$
$\wedge\, \Omega_{\mathfrak{M}}.\beta_{\text{FROM}} = \Psi_{\mathfrak{L}}.\beta$
$\wedge\, \Omega_{\mathfrak{M}}.\beta_{\text{TO}} = \Psi^*_{\mathfrak{L}}.\beta\}$

Set REQUEST$\leftarrow (P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\beta, C_{\Psi_{\mathfrak{L}}}.t, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}\}, C_{Q_{\mathfrak{M}}}, D$
$E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta\}, \pi_{\text{USER}})$

Set STATE $\leftarrow (d, D, E^*_{\Psi_{\mathfrak{L}}}.\Phi, E^*_{\Psi_{\mathfrak{L}}}.\beta, m_\Phi, {\Psi_{\mathfrak{L}}}^*.\beta$

Store STATE

Send REQUEST to the LA Issue$_{\text{TRUST_MIGRATION}}$(REQUEST) ▷ Algorithm 16

end function

Algorithm 16 Trust Migration Server Issuing: Upon receiving the TRUST_MIGRATION request from a user, the LA verifies the input for both the Lox and Migration credentials using Algorithm 10. If this is successful, the LA issues the new trusted Lox credential with $\Psi^*_{\mathfrak{L}}.t$ set to today's date (δ) and $\Psi^*_{\mathfrak{L}}.L$ set to 1. The hidden Lox credential attribute $\Psi^*_{\mathfrak{L}}.\Phi$ is jointly created with the LA and new trusted bucket $\Psi^*_{\mathfrak{L}}.\beta$ that matches the value for $\Omega_{\mathfrak{M}}.\beta_{\text{TO}}$ remains hidden during issuing. All calculations are proved to be computed correctly with π_{ISSUE} and the response is returned to the user to be handled.

Input From Client

P	P component of MAC tag on Lox attribute
$\Psi_{\mathfrak{L}}.\{\Phi, L\}$	Revealed attributes of $\Psi_{\mathfrak{L}}$
$C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}$	Hidden attributes of $\Psi_{\mathfrak{L}}$
C_Q	MAC for $\Psi_{\mathfrak{L}}$
$P_{\mathfrak{M}}$	P component of MAC tag for migration token
$C_{\Omega_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}\}$	Hidden attributes of $\Omega_{\mathfrak{M}}$
$C_{Q_{\mathfrak{M}}}$	MAC on migration token attributes
D	ElGamal public key
$E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta\}$	Client encrypted attributes
π_{USER}	Proof of correct user blinding

Output To Client HANDLE_RESPONSE

RESPONSE	A Response containing: $(P, \Psi_{\mathfrak{L}}.t, E_Q, j_\Phi, T_\Phi, T_\beta, \pi_{\text{ISSUE}})$

function ISSUE$_{\text{TRUST_MIGRATION}}$($P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}\}, C_{Q_{\mathfrak{M}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta\}, \pi_{\text{USER}}$)

if VERIFY$(P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t\}, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{TO}\}, C_{Q_{\mathfrak{M}}}, \pi_{\text{USER}})$ **then** ▷ Algorithm 10

Generate $d, w, j_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $\quad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

$s, E_{Q_{\mathcal{R}}} \leftarrow$ ISSUE_REVEALED$(P, t = \delta, L = 1)$ ▷ Algorithm 2

$E_{Q_{\mathcal{H}}}, T_\Phi, T_\beta, j_\Phi \leftarrow$ ISSUE_HIDDEN$_{\mathfrak{L}}(E_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta\})$ ▷ Algorithm 4

Compute $E_Q \leftarrow E_{Q_{\mathcal{R}}} + E_{Q_{\mathcal{H}}}$ ▷ Homomorphically compute MAC

Generate π_{ISSUE} ▷ Generate proof as in Section 5.4.1

Set RESPONSE $\leftarrow (\Psi^*_{\mathfrak{L}}.P, \Psi_{\mathfrak{L}}.t = \delta, E_Q, j_\Phi, T_\Phi, T_\beta, \pi_{\text{ISSUE}})$

Send RESPONSE to the Client: HANDLE(RESPONSE$_{\mathfrak{L}}$) ▷ Algorithm 7

end if

end function

5.5.3 Level Up

As described in Chapter 4, the level up protocol is used to incrementally increase a user's trust level beyond $L = 1$ up to $L = 4$. To do this, the user presents a valid Lox credential, $\Psi_{\mathfrak{L}}$ and reachability credential $\Psi_{\mathfrak{R}}$ to the LA as depicted in Figure 5.4. The user proves in zero knowledge that the required number of days for the requested L have elapsed since $\Psi_{\mathfrak{L}}.t$ was last updated (see Table 4.1 for DAYS and INVITATIONS arrays) but not beyond some expiry (we use 511 days + required number of days), that $\Psi_{\mathfrak{L}}.d$ has not surpassed the threshold for the requested L (see Table 4.2 for MAX_L array) and that $\Psi_{\mathfrak{L}}.\beta$ and $\Psi_{\mathfrak{R}}.\beta$ match. This is detailed in the $\text{PRESENT}_{\text{Level\_Up}}$ function in Algorithm 17.

The user sends their request to the LA with $\text{ISSUE}_{\text{Level\_Up}}$, which verifies and then issues the updated Lox credential as detailed in Algorithm 18. Note that the updated $\Psi^*_{\mathfrak{L}}.L$ and $\Psi^*_{\mathfrak{L}}.a$ are computed by the user, stored in their state and used to generate their updated Lox credential. They therefore must be computed correctly for $\Psi^*_{\mathfrak{L}}$ to be valid but are not issued directly by the LA.

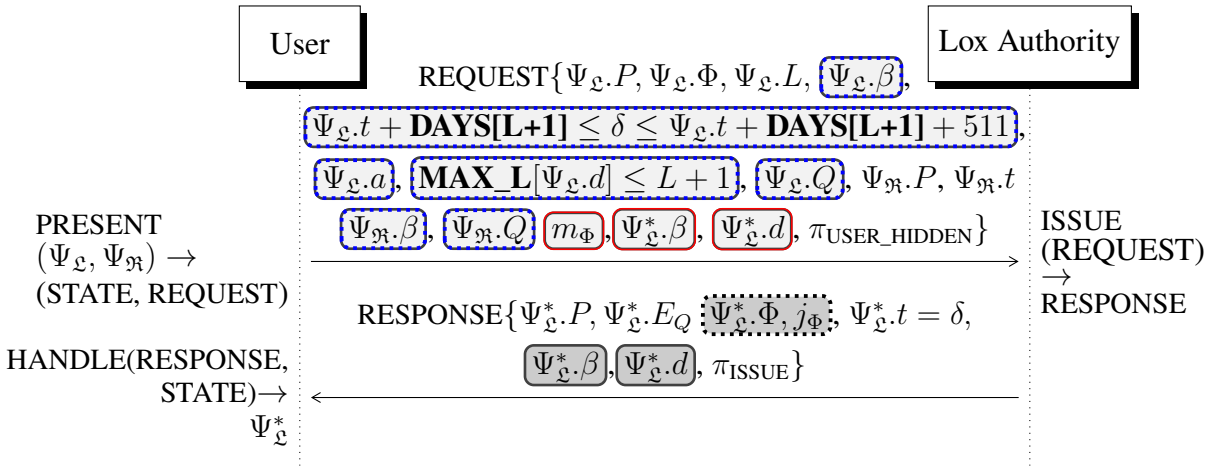


Figure 5.4: Overview of the Level Up Protocol

Where: $\Psi^*_{\mathfrak{L}}\{$ $P : \Psi^*_{\mathfrak{L}}.P$

$Q : \Psi_{\mathfrak{L}}.{E_Q}^*[1] - d \cdot \Psi_{\mathfrak{L}}.{E_Q}^*[0]$

$\Phi : m_\Phi + j_\Phi$

$\beta : \Psi_{\mathfrak{L}}.\beta$

$L : \Psi_{\mathfrak{L}}.L + 1$

$t : \delta$

$a :$ INVITATIONS$[L + 1]$ See Table 4.1

$d : \Psi_{\mathfrak{L}}.d$ $\}$.

Algorithm 17 Level Up Presentation: A user presents $\Psi_{\mathfrak{L}}.\Phi$ and hidden attributes from both their Lox credential, $\Psi_{\mathfrak{L}}$ and a fresh reachability credential, $\Psi_{\mathfrak{R}}$, to the Lox Authority. The LA checks that $\Psi_{\mathfrak{L}}.\Phi$ has not been seen before, that $\Psi_{\mathfrak{L}}.\beta$ and $\Psi_{\mathfrak{R}}.\beta$ match, that $\Psi_{\mathfrak{R}}.t$ is today's date, that $\Psi_{\mathfrak{L}}.t$ is equal to or greater than the predetermined interval for upgrading levels and finally that $\Psi_{\mathfrak{L}}.d$ does not exceed the maximum for the requested level upgrade. The user prepares an updated Lox credential with encrypted $\Psi_{\mathfrak{L}}.\{\Phi, \beta, d\}$ to be issued by the Lox Authority. Attributes presented for verification and prepared for issuing are proved to be computed correctly with π_{USER}.

Input Client

$\Psi_{\mathfrak{L}}$	Lox credential
$\Psi_{\mathfrak{R}}$	Reachability credential

Output To Server

REQUEST	A Request containing: $P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}.\beta}, C_{Q_{\mathfrak{R}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \pi_{\text{USER}}$

Output To Client State

STATE	A State containing: $d, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, m_\Phi, \Psi^*_{\mathfrak{L}}.\{\beta, L, a, d\}$

function $\text{PRESENT}_{\text{LEVEL\_UP}}(\Psi_{\mathfrak{L}}, \Psi_{\mathfrak{R}})$

Generate $d, w, s, m_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $\quad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

Compute $P_{\mathfrak{R}} \leftarrow s \cdot \Psi_{\mathfrak{R}}.P$ $\quad Q_{\mathfrak{R}} \leftarrow s \cdot \Psi_{\mathfrak{R}}.Q$

Compute $(C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, \pi_{\text{CRED\_SHOW}_{\mathfrak{L}}}) \leftarrow \text{PRESENT\_HIDDEN}(P, Q, \Psi_{\mathfrak{L}}.\{\beta, t, a, d\})$

Compute $(C_{\Psi_{\mathfrak{R}}.\beta}, C_{Q_{\mathfrak{R}}}, \pi_{\text{CRED\_SHOW}_{\mathfrak{R}}}) \leftarrow \text{PRESENT\_HIDDEN}(P_{\mathfrak{R}}, Q_{\mathfrak{R}}, \Psi_{\mathfrak{R}}.\beta)$ ▷ Algorithm 9

Compute $(E_{\Psi^*_{\mathfrak{L}}}.\beta, E_{\Psi^*_{\mathfrak{L}}}.d, E_{\Psi^*_{\mathfrak{L}}}.\Phi, \pi_{\text{USER\_HIDDEN}}) \leftarrow \text{PREPARE\_HIDDEN}(D, \Psi^*_{\mathfrak{L}}.\{\beta, d\}, m_\Phi)$ ▷ Algorithm 1

Generate $\pi_{\text{USER}} = \{\pi_{\text{CRED\_SHOW}_{\mathfrak{L}}} \wedge \pi_{\text{CRED\_SHOW}_{\mathfrak{R}}} \wedge \pi_{\text{USER\_HIDDEN}}$
$\wedge\, \Psi_{\mathfrak{R}}.\beta = \Psi_{\mathfrak{L}}.\beta \wedge \Psi_{\mathfrak{L}}.\beta = \Psi^*_{\mathfrak{L}}.\beta$
$\wedge\, \Psi_{\mathfrak{L}}.t + \text{DAYS[L+1]} \leq \delta \leq \Psi_{\mathfrak{L}}.t + \text{DAYS[L+1]} + 511$
$\wedge\, \text{MAX\_L}[\Psi_{\mathfrak{L}}.d] \leq L + 1 \wedge \Psi_{\mathfrak{L}}.d = \Psi^*_{\mathfrak{L}}.d\}$

Set REQUEST $\leftarrow (P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}}.\beta, C_{Q_{\mathfrak{R}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \pi_{\text{USER}})$

Set STATE $\leftarrow (d, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, m_\Phi, \Psi^*_{\mathfrak{L}}.\{\beta, L, a, d\})$

Store STATE

Send REQUEST to the LA $\text{ISSUE}_{\text{LEVEL\_UP}}$(REQUEST) ▷ Algorithm 18

end function

Algorithm 18 Level Up Server Issuing: Upon receiving the LEVEL_UP request from a user, the LA verifies the input for both the Lox and Reachability credentials using Algorithm 10. If this is successful, the LA issues the new trusted Lox credential with $\Psi_{\mathfrak{L}}.t$ set to today's date (δ) and $\Psi_{\mathfrak{L}}.L$ set to $L+1$ provided $L < 4$. The hidden Lox credential attribute $\Psi_{\mathfrak{L}}.\Phi$ is jointly created with the LA and the bucket $\Psi_{\mathfrak{L}}.\beta$ that matches the value for $\Psi_{\mathfrak{R}}.\beta$ as well as the number of blockages the user has seen d remain hidden during issuing. All calculations are proved to be computed correctly with π_{ISSUE} and the response is returned to the user to be handled.

Input Global LA State

INVITATIONS Array of invitation values for each trust level from Table 4.1

Input From Client

P	P component of MAC tag on Lox attribute
$\Psi_{\mathfrak{L}}.\{\Phi, L\}$	Revealed attributes of $\Psi_{\mathfrak{L}}$
$C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}$	Hidden attributes of $\Psi_{\mathfrak{L}}$
C_Q	MAC for $\Psi_{\mathfrak{L}}$
$P_{\mathfrak{R}}$	P component of MAC tag for migration credential
$C_{\Psi_{\mathfrak{R}}}.\beta$	Hidden bucket attribute of $\Psi_{\mathfrak{R}}$
$C_{Q_{\mathfrak{R}}}$	MAC on migration credential attributes
D	ElGamal Public Key
$E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}$	Client encrypted attributes
π_{USER}	Proof of correct user blinding

Output To Client HANDLE

RESPONSE A Response containing: $(P, E_Q, j_\Phi, \Psi^*_{\mathfrak{L}}.t, T_\Phi, T_\beta, T_d, \pi_{\text{ISSUE}})$

function $\text{ISSUE}_{\text{LEVEL\_UP}}(P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}}.\beta, C_{Q_{\mathfrak{R}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \pi_{\text{USER}})$

if $\text{VERIFY}(P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}}.\beta, C_{Q_{\mathfrak{R}}}, \pi_{\text{USER}})$ **then** ▷ Algorithm 10

Generate $d, w, j_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

if $\Psi_{\mathfrak{L}}.L < 4$ **then** $\Psi_{\mathfrak{L}}.L = \Psi_{\mathfrak{L}}.L + 1$

else $\Psi_{\mathfrak{L}}.L = 4$

end if

Compute $\Psi_{\mathfrak{L}}.a \leftarrow \text{INVITATIONS}[\Psi_{\mathfrak{L}}.L]$ ▷ See Table 4.1

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P \qquad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

$s, E_{Q_{\mathcal{R}}} \leftarrow \text{ISSUE\_REVEALED}(P, t = \delta, \Psi_{\mathfrak{L}}.\{L, a\})$ ▷ Algorithm 2

$E_{Q_{\mathcal{H}}}, T_\Phi, T_\beta, T_d, j_\Phi \leftarrow \text{ISSUE\_HIDDEN}_{\mathfrak{L}}(E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\})$ ▷ Algorithm 4

Compute $E_Q \leftarrow E_{Q_{\mathcal{R}}} + E_{Q_{\mathcal{H}}}$ ▷ Homomorphically compute MAC

Generate π_{ISSUE} ▷ Generate proof as in Section 5.4.1
Set RESPONSE $\leftarrow (\Psi^*_{\mathfrak{L}}.P, \Psi^*_{\mathfrak{L}}.t = \delta, E_Q, j_\Phi, T_\Phi, T_\beta, \pi_{\text{ISSUE}})$
Send RESPONSE to the Client: HANDLE(RESPONSE$_{\mathfrak{L}}$) ▷ Algorithm 7
end if
end function

5.5.4 Issue Invitation

As described in Chapter 4, the issue invitation protocol is used by trusted users with $L \geq 2$ to issue trusted invitations that can be used by new users to join the Lox system with an elevated trust level. To do this, the user presents a valid Lox credential, $\Psi_{\mathfrak{L}}$ and reachability credential $\Psi_{\mathfrak{R}}$ to the LA as depicted in Figure 5.5. The user proves in zero knowledge that $\Psi_{\mathfrak{L}}.a$ is not 0 and that $\Psi_{\mathfrak{L}}.\beta$ and $\Psi_{\mathfrak{R}}.\beta$ match. This is detailed in the PRESENT$_{\text{INVITATION}}$ function in Algorithm 19.

The user sends their request to the LA with ISSUE$_{\text{INVITATION}}$ which verifies and then issues the invitation credential $\Psi_{\mathfrak{I}}$ with attributes $\Psi_{\mathfrak{I}}.\{\beta, d\}$ matching the user's $\Psi^*_{\mathfrak{L}}.\{\beta, d\}$ as well as the updated Lox credential with $\Psi^*_{\mathfrak{L}}.a$ equal to $\Psi_{\mathfrak{L}}.a - 1$. This is detailed in Algorithm 20.

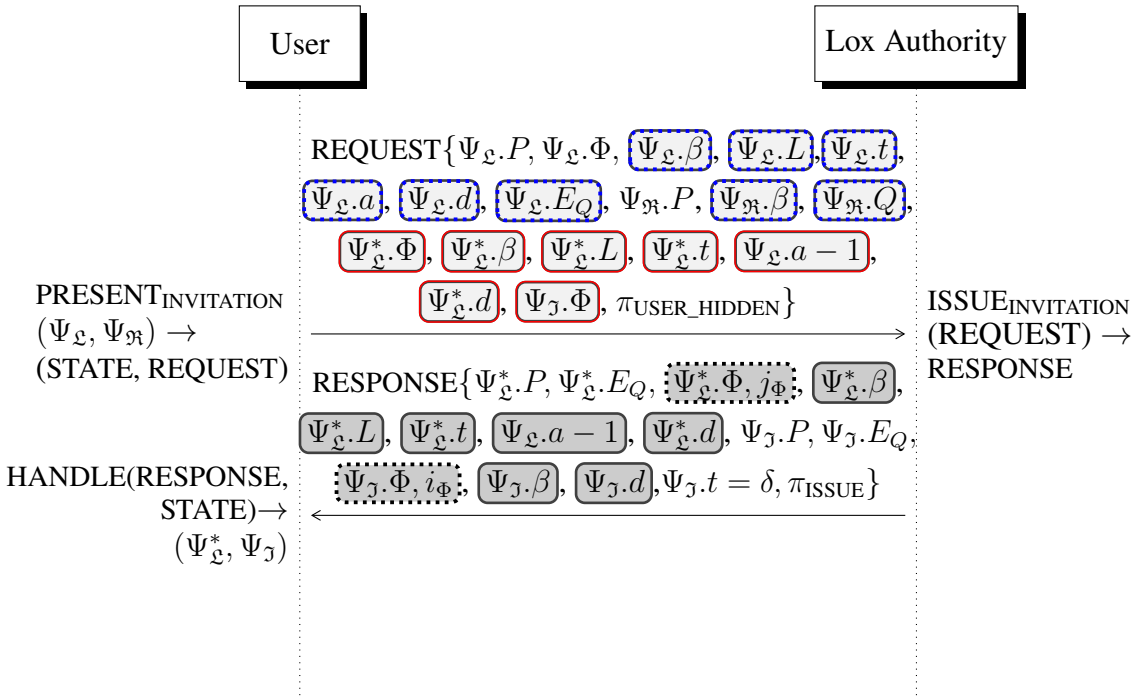


Figure 5.5: Issue Invitation Protocol

Where:$\Psi^*_{\mathfrak{L}}\{$ $P : \Psi^*_{\mathfrak{L}}.P$
$Q : \Psi^*_{\mathfrak{L}}.E_Q[1] - d \cdot \Psi^*_{\mathfrak{L}}.E_Q[0]$
$\Phi : m_\Phi + j_\Phi$
$\beta : \Psi_{\mathfrak{L}}.\beta$
$L : \Psi_{\mathfrak{L}}.L$
$t : \Psi_{\mathfrak{L}}.t$
$a : \Psi_{\mathfrak{L}}.a - 1$
$d : \Psi_{\mathfrak{L}}.d \quad \}.$

and:$\Psi_{\mathfrak{I}}\{$ $P : \Psi_{\mathfrak{I}}.P$
$Q : \Psi_{\mathfrak{I}}.E_Q[1] - d \cdot \Psi_{\mathfrak{I}}.E_Q[0]$
$\Phi : \Psi_{\mathfrak{I}}.\Phi + i_\Phi$
$\beta : \Psi_{\mathfrak{L}}.\beta$
$t : \delta$
$d : \Psi_{\mathfrak{L}}.d \quad \}.$

Algorithm 19 Issue Invitation Presentation: To issue an invitation, a user with trust level $L \geq 2$ presents their $\Psi_{\mathfrak{L}}$ and a fresh $\Psi_{\mathfrak{R}}$ with matching bucket attributes, to the LA. The presented credentials must prove that $\Psi_{\mathfrak{L}}.\Phi$ has not been seen before, that the buckets match, that $\Psi_{\mathfrak{R}}.t$ has today's date, and that $\Psi_{\mathfrak{L}}.a$, is not 0. The user prepares an updated Lox credential with encrypted attributes, including $\Psi_{\mathfrak{L}}.\Phi$ to be jointly created and the $\Psi_{\mathfrak{L}}.a - 1$. The $\Psi_{\mathfrak{L}}.\beta$ and $\Psi_{\mathfrak{L}}.d$ will be used to isse $\Psi_{\mathfrak{J}}$ along with an encrypted $\Psi_{\mathfrak{J}}.\Phi$ prepared by the user. Attributes presented for verification and prepared for issuing are proved to be computed correctly with π_{USER}.

Input Client

$\Psi_{\mathfrak{L}}$	Lox credential
$\Psi_{\mathfrak{R}}$	Reachability Credential

Output To Server

REQUEST	A Request containing: $P, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, L, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}}.\beta,$ $C_{Q_{\mathfrak{R}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, L, t, a-1, d\}, E_{\Psi_{\mathfrak{J}}}.\Phi,$ π_{USER}

Output To Client State

STATE	A State containing: $d, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, L, t, a-1, d\}, E_{\Psi_{\mathfrak{J}}.\Phi}, m_\Phi,$ $\Psi^*_{\mathfrak{L}}.\{\beta, L, t, a-1, d\}, \Psi_{\mathfrak{J}}.\Phi$

function PRESENT$_{\text{ISSUE_INVITATION}}(\Psi_{\mathfrak{L}}, \Psi_{\mathfrak{R}})$

Generate $d, w, s, m_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $\quad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

Compute $P_{\mathfrak{R}} \leftarrow s \cdot \Psi_{\mathfrak{R}}.P$ $\quad Q_{\mathfrak{R}} \leftarrow s \cdot \Psi_{\mathfrak{R}}.Q$

Compute $(C_{\Psi_{\mathfrak{L}}}.\{\beta, t, L, a, d\}, C_Q, \pi_{\text{CRED\_SHOW}_{\mathfrak{L}}}) \leftarrow$
$\hookrightarrow$ PRESENT_HIDDEN$(P, Q, \Psi_{\mathfrak{L}}.\{\beta, t, L, a, d\})$

Compute $(C_{\Psi_{\mathfrak{R}}.\beta}, C_{Q_{\mathfrak{R}}}, \pi_{\text{CRED\_SHOW}_{\mathfrak{R}}}) \leftarrow$ PRESENT_HIDDEN$(P_{\mathfrak{R}}, Q_{\mathfrak{R}}, \Psi_{\mathfrak{R}}.\beta)$
▷ Algorithm 9

Compute $(E_{\Psi^*_{\mathfrak{L}}}.\Phi, E_{\Psi^*_{\mathfrak{L}}}.\beta, E_{\Psi^*_{\mathfrak{L}}}.L, E_{\Psi^*_{\mathfrak{L}}}.t, E_{\Psi^*_{\mathfrak{L}}}.a-1, E_{\Psi^*_{\mathfrak{L}}}.d, E_{\Psi_{\mathfrak{J}}.\Phi}, \pi_{\text{USER\_HIDDEN}}) \leftarrow$
$\hookrightarrow$ PREPARE_HIDDEN$(D, \Psi^*_{\mathfrak{L}}.\{\beta, t, L, a-1, d\}, m_\Phi, \Psi_{\mathfrak{J}}.\Phi)$ ▷ Algorithm 1

Generate $\pi_{\text{USER}} = \{\pi_{\text{CRED\_SHOW}_{\mathfrak{L}}} \wedge \pi_{\text{CRED\_SHOW}_{\mathfrak{R}}} \wedge \pi_{\text{USER\_HIDDEN}},$
$\wedge\, \Psi_{\mathfrak{R}}.\beta = \Psi_{\mathfrak{L}}.\beta \wedge \Psi_{\mathfrak{L}}.\beta = \Psi^*_{\mathfrak{L}}.\beta$
$\wedge\, \Psi_{\mathfrak{R}}.t = \delta \wedge \Psi_{\mathfrak{L}}.t = \Psi^*_{\mathfrak{L}}.t,$
$\wedge\, \Psi_{\mathfrak{L}}.a > 0 \wedge \Psi^*_{\mathfrak{L}}.a = \Psi_{\mathfrak{L}}.a - 1,$
$\wedge\, \Psi_{\mathfrak{L}}.d = \Psi^*_{\mathfrak{L}}.d \wedge \Psi_{\mathfrak{L}}.L = \Psi^*_{\mathfrak{L}}.L\}$

Set REQUEST $\leftarrow (P, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, L, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}}.\beta, C_{Q_{\mathfrak{R}}}, D,$
$E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, L, t, a-1, d\}, E_{\Psi^*_{\mathfrak{J}}}.\Phi, \pi_{\text{USER}})$

Set STATE $\leftarrow (d, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, L, t, a-1, d\}, m_\Phi, \Psi^*_{\mathfrak{L}}.\{\beta, t, L, a, d\}, \Psi_{\mathfrak{J}}.\Phi)$

Store STATE
Send REQUEST to the LA $\text{ISSUE}_{\text{ISSUE\_INVITATION}}$(REQUEST) ▷ Algorithm 20
end function

Algorithm 20 Issue Invitation Server Issuing: Upon receiving an ISSUE_INVITATION request, the LA verifies the input for both $\Psi_{\mathfrak{L}}$ and $\Psi_{\mathfrak{R}}$ using Algorithm 10 and if successful, issues the updated $\Psi_{\mathfrak{L}}$ and $\Psi_{\mathfrak{J}}$. $\Psi_{\mathfrak{L}}.\Phi$ a well as $\Psi_{\mathfrak{J}}.\Phi$ are both jointly created with the LA and $\Psi_{\mathfrak{L}}.\{\beta, d\}$ are used for $\Psi_{\mathfrak{J}}.\{\beta, d\}$. All calculations are proved to be computed correctly with π_{ISSUE} and the response is returned to the user to be handled.

Input From Client

P	P component of MAC tag on Lox attribute
$\Psi_{\mathfrak{L}}.\Phi$	Revealed attributes of $\Psi_{\mathfrak{L}}$
$C_{\Psi_{\mathfrak{L}}}.\{\beta, t, L, a, d\}$	Hidden attributes of $\Psi_{\mathfrak{L}}$
C_Q	MAC for $\Psi_{\mathfrak{L}}$
$C_{\Psi_{\mathfrak{R}}}.\beta$	Hidden bucket attribute of $\Psi_{\mathfrak{R}}$
$P_{\mathfrak{R}}$	P component of MAC tag for migration credential
$C_{Q_{\mathfrak{R}}}$	MAC on migration credential attributes
$E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, L, t, a-1, d\}$	Client encrypted attributes
$E_{\Psi_{\mathfrak{J}}.\Phi}$	Invitation ID of invitation to be issued
π_{USER}	Proof of correct user blinding

Output To Client HANDLE_RESPONSE

RESPONSE A Response containing: $(P, E_Q, j_\Phi, T_\Phi, T_\beta, T_t, T_L, T_a, T_d, \Psi_{\mathfrak{J}}.P, E_{\Psi_{\mathfrak{J}}}.Q, j_{\Phi,\mathfrak{J}}, T_{\Phi,\mathfrak{J}}, \Psi_{\mathfrak{J}}.t = \delta, T_{\beta,\mathfrak{J}}, T_{d,\mathfrak{J}}, \pi_{\text{ISSUE}})$

function ISSUE$_{\text{ISSUE_INVITATION}}(P, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, L, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}}.\beta, C_{Q_{\mathfrak{R}}}, D, E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, L, t, a-1, d\}, E_{\Psi_{\mathfrak{J}}.\Phi}, \pi_{\text{USER}})$

if VERIFY$(P, \Psi_{\mathfrak{L}}.\Phi, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, L, a, d\}, C_Q, P_{\mathfrak{R}}, C_{\Psi_{\mathfrak{R}}}.\beta, C_{Q_{\mathfrak{R}}}, \pi_{\text{USER}})$ **then** ▷ Algorithm 10

Generate $d, w, v, j_\Phi, j_{\Phi,\mathfrak{J}} \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

Compute $P_{\mathfrak{J}} \leftarrow v \cdot B$ $Q_{\mathfrak{J}} \leftarrow v \cdot B$

$s, E_{Q_{\mathcal{R}}} \leftarrow$ ISSUE_REVEALED$(P_{\mathfrak{J}}, t = \delta)$ ▷ Invitation date of issue Algorithm 2

$E_{Q_{\mathcal{H}}}, T_\Phi, T_\beta, T_t, T_L, T_a, T_d, j_\Phi \leftarrow$ ISSUE_HIDDEN$_{\mathfrak{L}}(E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, t, L, a-1, d\})$

$E_{Q_{\mathcal{H}\mathfrak{J}}}, T_{\Phi,\mathfrak{J}}, T_{\beta,\mathfrak{J}}, T_{d,\mathfrak{J}}, j_{\Phi,\mathfrak{J}} \leftarrow$ ISSUE_HIDDEN$_{\mathfrak{J}}(E_{\Psi_{\mathfrak{J}}}.\Phi, E_{\Psi_{\mathfrak{L}}}.\{\beta, d\})$ ▷ Algorithm 4

Compute $E_Q \leftarrow E_{Q_{\mathcal{R}}} + E_{Q_{\mathcal{H}}} + E_{Q_{\mathcal{H}\mathfrak{J}}}$ ▷ Homomorphically compute MAC

Generate π_{ISSUE} ▷ Generate proof as in Section 5.4.1

Set RESPONSE$_{\mathfrak{L}} \leftarrow (\Psi_{\mathfrak{L}}.P, E_Q, j_\Phi, T_\Phi, T_\beta, T_t, T_L, T_a, T_d, \pi_{\text{ISSUE}})$

Set RESPONSE$_{\mathfrak{J}} \leftarrow (\Psi_{\mathfrak{J}}.P, E_{\Psi_{\mathfrak{J}}.Q}, j_{\Phi,\mathfrak{J}}, T_{\Phi,\mathfrak{J}}, \Psi_{\mathfrak{J}}.t = \delta, T_{\beta,\mathfrak{J}}, T_{d,\mathfrak{J}}, \pi_{\text{ISSUE}})$

Send RESPONSE to the Client: HANDLE(RESPONSE$_{\mathfrak{L}}$) ▷ Algorithm 7

Send RESPONSE to the Client: HANDLE(RESPONSE$_{\mathfrak{J}}$) ▷ Algorithm 7

end if
end function

5.5.5 Redeem Invitation Credential

As decribed in Chapter 4, the redeem invitation protocol allows users with a valid invitation credential, $\Psi_{\mathfrak{I}}$, to join Lox with a trust level of $L = 1$. To do this, the user presents their invitation credential $\Psi_{\mathfrak{I}}$ to the LA as depicted in Figure 5.6. The invitation must not have been presented before and it must be proved in zero knowledge that $\Psi_{\mathfrak{I}}.t$ is less than 15 days old. This is detailed in the $\text{PRESENT}_{\text{REDEEM}}$ function in Algorithm 21.

The user sends their request to the LA with $\text{ISSUE}_{\text{REDEEM}}$ which verifies and then issues the new credential $\Psi_{\mathfrak{L}}$ with attributes $\Psi_{\mathfrak{I}}.\{\beta, d\}$ that match with the user who gave the invitation to the new user. This is detailed in Algorithm 22.

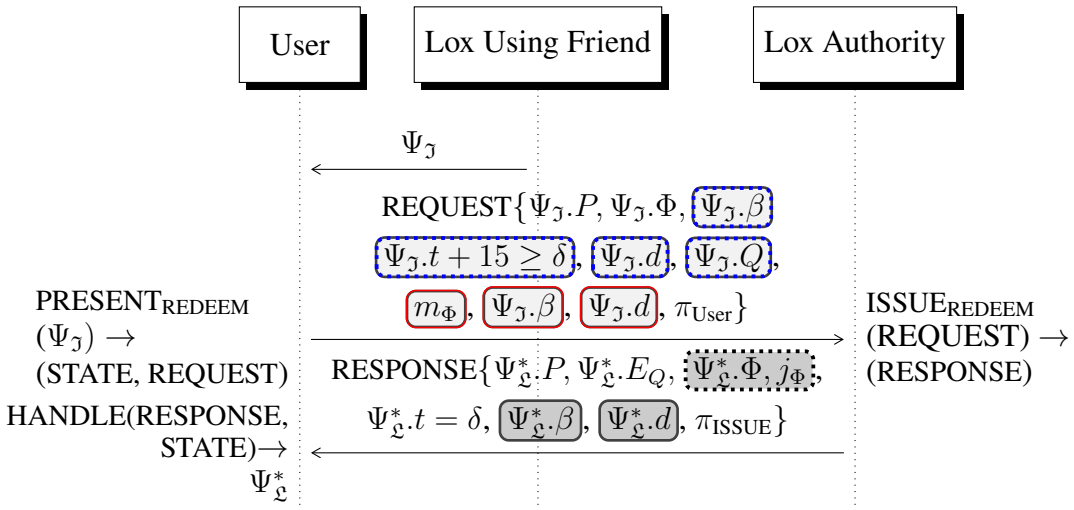


Figure 5.6: Redeem Invitation Credential Protocol

$$
\begin{aligned}
\text{Where:}\Psi^*_{\mathfrak{L}}\{ \quad & P : \Psi^*_{\mathfrak{L}}.P \\
& Q : \Psi^*_{\mathfrak{L}}.E_Q[1] - d \cdot \Psi^*_{\mathfrak{L}}.E_Q[0] \\
& \Phi : m_\Phi + j_\Phi \\
& \beta : \Psi_{\mathfrak{I}}.\beta \\
& L : 1 \\
& t : \delta \\
& a : 0 \\
& d : \Psi_{\mathfrak{I}}.d \qquad \}.
\end{aligned}
$$

Algorithm 21 Redeem Invitation Presentation: A new user that has received an invitation credential from a trusted user can redeem their invitation credential to enter the Lox system as a trusted user with a trust level $L = 1$ and a trusted bucket with 3 bridges. To do so, the user presents the credential ID Φ from their invitation credential $\Psi_{\mathfrak{I}}$ and the hidden bucket β and blockages d attributes while also proving that the credential is no more than 15 days old. The user prepares their Lox credential with encrypted β and d attributes from the invitation credential, including a Lox ID to be jointly created. Attributes presented for verification and prepared for issuing are proved to be computed correctly with π_{USER}.

Input Client

$\Psi_{\mathfrak{I}}$ Invitation credential

Output To Server

REQUEST A Request containing: $\Psi_{\mathfrak{I}}.\{P, \Phi\}, C_{\Psi_{\mathfrak{I}}}.\{\beta, d\}, C_{\Psi_{\mathfrak{I}}}.t, C_{Q,\mathfrak{I}}, D, E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \pi_{\text{USER}}$

Output To Client State

STATE A State containing: $d, D, E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\}, m_\Phi, \Psi^*_{\mathfrak{L}}.\{\beta, d\}$

function $\text{PRESENT}_{\text{REDEEM\_INVITATION}}(\Psi_{\mathfrak{I}})$

Generate $w, m_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

Compute $(C_{\Psi_{\mathfrak{I}}}.\{\beta, t, d\}, C_Q, \pi_{\text{CRED\_SHOW}_{\mathfrak{L}}}) \leftarrow$

$\hookrightarrow$ $\text{PRESENT\_HIDDEN}(P, Q, \Psi_{\mathfrak{I}}.\{\beta, t, d\})$ ▷ Algorithm 9

Compute $(E^*_{\Psi_{\mathfrak{L}}}.\Phi, E^*_{\Psi_{\mathfrak{L}}}.\beta, E^*_{\Psi_{\mathfrak{L}}}.d, \pi_{\text{USER\_HIDDEN}}) \leftarrow$

$\hookrightarrow$ $\text{PREPARE\_HIDDEN}(D, \Psi_{\mathfrak{I}}.\{\beta, d\}, m_\Phi)$ ▷ Algorithm 1

Generate $\pi_{\text{USER}} = \pi_{\text{CRED\_SHOW}} \wedge \pi_{\text{USER\_HIDDEN}} \wedge \Psi_{\mathfrak{I}}.t + 15 \geq \delta$

Set REQUEST $\leftarrow \Psi_{\mathfrak{I}}.\{P, \Phi\}, C_{\Psi_{\mathfrak{I}}}.\{\beta, d\}, C_{\Psi_{\mathfrak{I}}}.t, C_{Q,\mathfrak{I}}, D, E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \pi_{\text{USER}}$

Set STATE $\leftarrow (d, D, E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\}, m_\Phi, \Psi^*_{\mathfrak{L}}.\{\beta, d\})$

Store STATE

Send REQUEST to the LA $\text{ISSUE}_{\text{REDEEM\_INVITATION}}$(REQUEST) ▷ Algorithm 22

end function

Algorithm 22 Redeem Invitation Server Issuing: Upon receiving the REDEEM_INVITATION request from a user, the LA verifies the input for the invitation credential using Algorithm 10. If this is successful, the LA issues the new trusted Lox credential with $\Psi_{\mathfrak{L}}.\beta$ and $\Psi_{\mathfrak{L}}.d$ set to the values matching the invitation credential's $\Psi_{\mathfrak{I}}.\beta$ and $\Psi_{\mathfrak{I}}.d$. Then, $\Psi_{\mathfrak{L}}.t$ is set to today's date (δ) and $\Psi_{\mathfrak{L}}.L$ is set to 1. The hidden Lox credential attribute $\Psi_{\mathfrak{L}}.\Phi$ is jointly created as usual. All calculations are proved to be computed correctly with π_{ISSUE} and the response is returned to the user to be handled.

Input From Client

$\Psi_{\mathfrak{I}}.P$	P component of MAC tag on Lox attribute
$\Psi_{\mathfrak{I}}.\Phi$	Revealed attributes of $\Psi_{\mathfrak{L}}$
$C_{\Psi_{\mathfrak{I}}}.\{\beta, t, d\}$	Hidden attributes of $\Psi_{\mathfrak{L}}$
$C_{Q,\mathfrak{I}}$	MAC for $\Psi_{\mathfrak{I}}$
D	ElGamal public key
$E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\}$	Client encrypted attributes
π_{USER}	Proof of correct user blinding

Output To Client HANDLE_RESPONSE

RESPONSE A Response containing: $(\Psi_{\mathfrak{L}}.P, E_Q, j_\Phi, T_\Phi, T_\beta, T_t, \pi_{\text{ISSUE}})$

function $\text{ISSUE}_{\text{REDEEM\_INVITATION}}(\Psi_{\mathfrak{I}}.\{P, \Phi\}, C_{\Psi_{\mathfrak{I}}}.\{\beta, t, d\}, C_{Q,\mathfrak{I}}, D, E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \pi_{\text{USER}})$

if $\text{VERIFY}(P, \Psi_{\mathfrak{I}}.\Phi, C_{\Psi_{\mathfrak{I}}}.\{\beta, t, d\}, C_Q, \pi_{\text{USER}})$ **then** ▷ Algorithm 10

Generate $A, B, d, w, j_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $\quad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

$s, E_{Q_{\mathcal{R}}} \leftarrow \text{ISSUE\_REVEALED}(P, t = \delta, L = 1)$ ▷ Invited users have L=1, Algorithm 2

$E_{Q_{\mathcal{H}}}, T_\Phi, T_\beta, T_d, j_\Phi \leftarrow \text{ISSUE\_HIDDEN}_{\mathfrak{L}}(E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\})$ ▷ Algorithm 4

Compute $E_Q \leftarrow E_{Q_{\mathcal{R}}} + E_{Q_{\mathcal{H}}}$ ▷ Homomorphically compute MAC

Generate π_{ISSUE} ▷ Generate proof as in Section 5.4.1

Set RESPONSE $\leftarrow (P, E_Q, \Psi_{\mathfrak{L}}.t = \delta, j_\Phi, T_\Phi, T_\beta, T_d, \pi_{\text{ISSUE}})$

Send RESPONSE to the Client $\text{HANDLE}(\text{RESPONSE}_{\mathfrak{L}})$ ▷ Algorithm 7

end if

end function

5.5.6 Check Blockage and Blockage Migration

As described in Chapter 4, migrating to a new bucket β is a two-step process available to users with a trust level of $L \geq 3$ and some threshold number of blocked bridges in their bucket β. In our implementation, this number is set to 2 of 3 bridges. Since a perceived bridge blockage by the user

may be a result of a bridge server being temporarily down, we allow a user to check for a blockage multiple times without penalty. To initiate a migration due to blocked buckets, the user presents their Lox credential $\Psi_{\mathfrak{L}}$ to the LA as depicted in Figure 5.7. The user shows that their $\Psi_{\mathfrak{L}}.L \geq 3$ and that $\Psi_{\mathfrak{L}}.\Phi$ is not on the list of seen $\Psi_{\mathfrak{L}}.\Phi$. An encrypted $\Psi_{\mathfrak{L}}.\beta$ is included with the request to be issued in the migration key credential. This is detailed in the $\text{PRESENT}_{\text{CHECK\_BLOCKAGE}}$ function in Algorithm 23.

The user sends their request to the LA with $\text{ISSUE}_{\text{MIGRATION}}$ to issue the migration key credential. If the request verifies, the LA prepares a migration key credential as described in Section 4.2.2. This gives the user all of the information they require to start constructing both their $\Omega_{\mathfrak{M}}$ and updated $\Psi_{\mathfrak{L}}$ for the actual blockage migration. This is detailed in Algorithm 24.

Next the user presents their Lox credential $\Psi_{\mathfrak{L}}$, along with the migration token $\Omega_{\mathfrak{M}}$. The user must show that $\Psi_{\mathfrak{L}}.\Phi$ does not appear in the used Lox credentials list, that $\Psi_{\mathfrak{L}}.L \geq 3$, and prove in zero knowledge that $\Psi_{\mathfrak{L}}.d$ is less than the maximum allowable d as detailed in Table 4.2. The user must also prove in zero knowledge that $\Psi_{\mathfrak{L}}.\beta$ matches $\Omega_{\mathfrak{M}}.\beta_{\text{FROM}}$ and the prepared $\Psi^*_{\mathfrak{L}}.\beta$ matches $\Omega_{\mathfrak{M}}.\beta_{\text{TO}}$. This is detailed in the $\text{PRESENT}_{\text{BLOCKAGE\_MIGRATION}}$ function in Algorithm 25.

The user sends their request to the LA with $\text{ISSUE}_{\text{BLOCKAGE\_MIGRATION}}$ which verifies and then issues the new credential $\Psi^*_{\mathfrak{L}}$ with attributes $\Psi_{\mathfrak{L}}.d - 1$, $\Psi_{\mathfrak{L}}.L - 2$, $\Psi_{\mathfrak{L}}.a[L-2]$ (See Table 4.1) and the new $\Psi^*_{\mathfrak{L}}.\beta$ that matches with $\Omega_{\mathfrak{M}}.\beta_{\text{TO}}$. This is detailed in Algorithm 26.

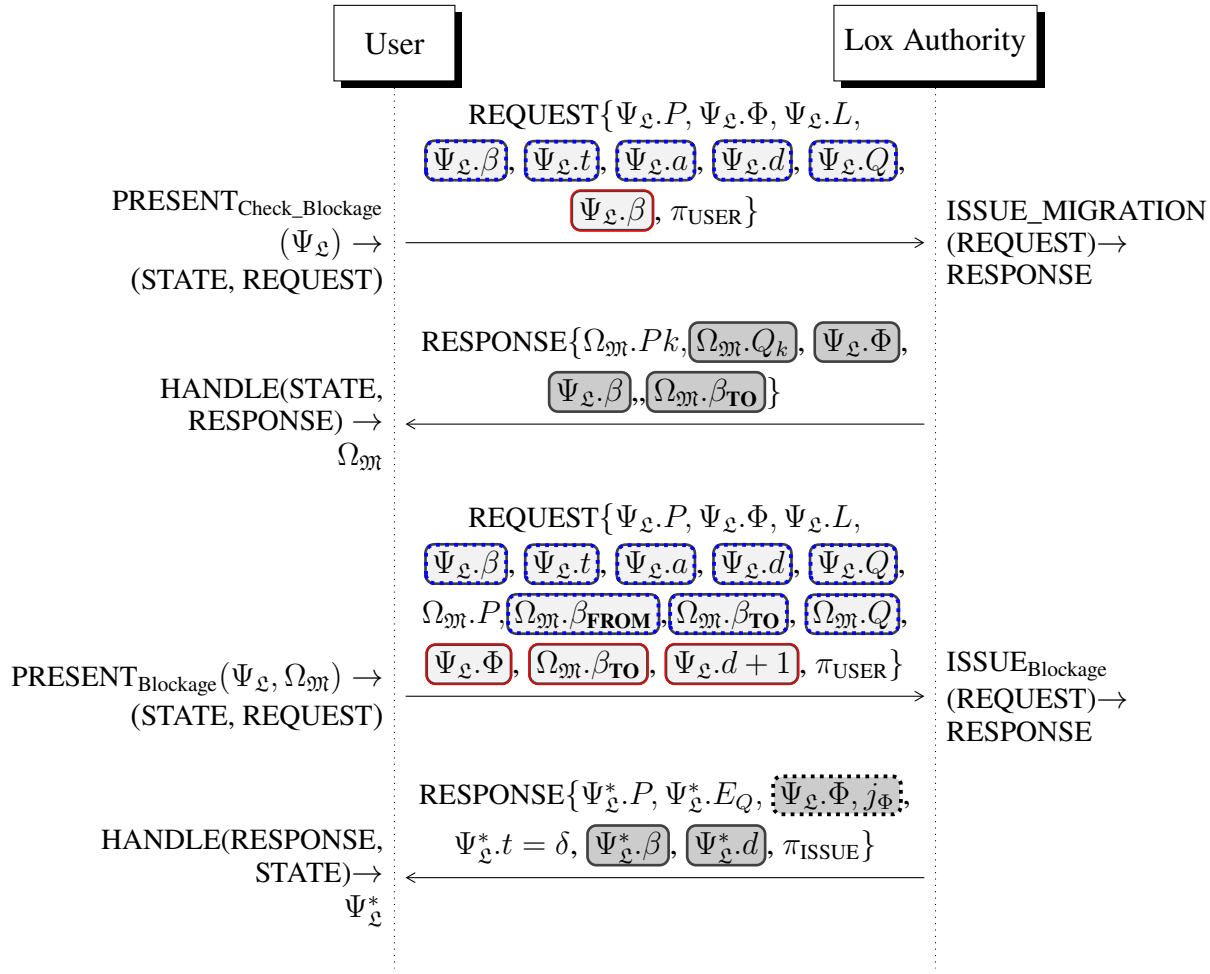


Figure 5.7: Overview of Check Blockage and Blockage Migration Protocols

$$
\begin{aligned}
\Omega_{\mathfrak{M}} : \{\quad & Pk : \Omega_{\mathfrak{M}}.Pk \\
& Q_k : E_{\Omega_{\mathfrak{M}}.Q_k}[0] - d - E_{\Omega_{\mathfrak{M}}.Q_k}[1] \\
& \Phi : \Psi_{\mathfrak{L}}.\Phi \\
& \beta_{\text{FROM}} : \Psi_{\mathfrak{L}}.\beta \\
& \beta_{\text{TO}} : \Omega_{\mathfrak{M}}.\beta_{\text{TO}} \\
& \text{Migration Type: Check Blockage} \quad \}
\end{aligned}
\qquad
\begin{aligned}
\Psi^*_{\mathfrak{L}}\{\quad & P : \Psi^*_{\mathfrak{L}}.P \\
& Q : \Psi^*_{\mathfrak{L}}.E_Q[1] - d \cdot \Psi^*_{\mathfrak{L}}.E_Q[0] \\
& \beta : \Omega_{\mathfrak{M}}.\beta_{\text{TO}} \\
& L : \Psi_{\mathfrak{L}}.L - 2 \\
& t : \delta \\
& a : a[\Psi_{\mathfrak{L}}.L - 2] \\
& d : \Psi_{\mathfrak{L}}.d + 1 \quad \}.
\end{aligned}
$$

Check Blockage

Algorithm 23 Check Blockage Presentation: To make a check blockage request, the user presents their hidden bucket β, time of the last level change t, and number of blockages seen d, to the Lox Authority. The user also shows the revealed trust level, L, showing that it is at least 3, and their credential ID. The user prepares their hidden bucket attribute to be issued by the Lox Authority as part of the migration credential. This can then be used towards a blockage migration. Hidden attributes presented for verification and prepared for issuing are proved to be computed correctly with π_{USER}.

Input Client
  $\Psi_{\mathfrak{L}}$ Lox credential
Output To Server
  REQUEST Contains: $P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, D, E_{\Psi_{\mathfrak{L}}}.\beta, \pi_{\text{USER}}$
Output To Client State
  STATE Contains: $d, D, E^*_{\Psi_{\mathfrak{L}}}.\beta, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.\beta$
function PRESENT$_{\text{CHECK_BLOCKAGE}}(\Psi_{\mathfrak{L}})$
  Generate $d, w \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$
  Compute $D \leftarrow dB$
  Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags
  $(C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, \pi_{\text{CRED\_SHOW}}) \leftarrow$ PRESENT_HIDDEN$(P, Q, \Psi_{\mathfrak{L}}.\{\beta, t, a, d\})$ ▷ Algorithm 9
  $(E_{\Psi_{\mathfrak{L}}}.\beta, \pi_{\text{USER\_HIDDEN}})) \leftarrow$ PREPARE_HIDDEN$(D, \Psi_{\mathfrak{L}}.\beta)$ ▷ Algorithm 1
  Generate $\pi_{\text{USER}} \leftarrow \pi_{\text{CRED\_SHOW}} \wedge \pi_{\text{USER\_HIDDEN}}$
  Set REQUEST $\leftarrow (P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, D, E_{\Psi_{\mathfrak{L}}}.\beta, \pi_{\text{USER}})$
  Set STATE $\leftarrow (d, D, E_{\Psi_{\mathfrak{L}}}.\beta, \Psi_{\mathfrak{L}}.\{\Phi, \beta\})$
  Store STATE
  Send REQUEST to the LA ISSUE$_{\text{CHECK_BLOCKAGE}}$(REQUEST) ▷ Algorithm 24
end function

Algorithm 24 Check Blockage Server Response: The LA receives the user's request, verifies the input from $\Psi_{\mathfrak{L}}$ and in particular that $\Psi_{\mathfrak{L}}.\Phi$ has not been seen before. Importantly, the revealed $\Psi_{\mathfrak{L}}.\Phi$ is not added to a seen list at this step allowing the migration key credential to be issued multiple times but redeemed only once. If the user's $\Psi_{\mathfrak{L}}.\beta$ is not blocked, their bucket will not be included in the migration table and no $\Omega_{\mathfrak{M}}$ can be created by the user.

Input Client

P	P component of MAC tag
$\Psi_{\mathfrak{L}}.\{\Phi, L\}$	Revealed attributes Φ, and L
$C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}$	Hidden Lox attributes β, t, a, and d
C_Q	MAC on Lox attributes
D	ElGamal public key
$E_{\Psi_{\mathfrak{L}}}.\beta$	Client encrypted β attribute
π_{USER}	Proof of correct user blinding

Output To Client

RESPONSE	A response to the client: $Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}}$

function ISSUE$_{\text{CHECK_BLOCKAGE}}(P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, D, E_{\Psi_{\mathfrak{L}}}.\beta, \pi_{\text{USER}})$
  if VERIFY$(P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, \pi_{\text{USER}}, \text{BLOCKAGE})$ **then** ▷ Algorithm 10
    $Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}} \leftarrow$ISSUE_MIGRATION$(D, \Psi_{\mathfrak{L}}.\Phi, E_{\Psi_{\mathfrak{L}}.\beta}, \text{BLOCKAGE})$ ▷ Algorithm 6
    Set RESPONSE $\leftarrow (Pk, E_{Qk}, E_{\text{TABLE}_{\mathfrak{M}}})$
    Send to HANDLE_MIGRATION(RESPONSE) ▷ Algorithm 8
  end if
end function

Blockage Migration

Algorithm 25 Blockage Migration Presentation: A user presents hidden attributes from their Lox credential and their migration credential to the Lox Authority, proving that the hidden bucket from their Lox credential and the from bucket in the migration credential are equal. The user prepares an updated Lox credential with encrypted credential ID and bucket, where the encrypted bucket matches the migration credential's to bucket. Attributes presented for verification and prepared for issuing are proved to be computed correctly with π_{USER}.

Input Client

$\Psi_{\mathfrak{L}}$	Lox credential
$\Omega_{\mathfrak{M}}$	Migration Token

Output To Server

REQUEST	A Request containing: $P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}\}, C_{Q_{\mathfrak{M}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta_{\text{TO}}, d\}, \pi_{\text{USER}}$

Output To Client State

STATE	A State containing: $d, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.d + 1, \Psi_{\mathfrak{L}}.\beta_{\text{TO}}, \Psi_{\mathfrak{L}}.L$

function $\text{PRESENT}_{\text{BLOCKAGE\_MIGRATION}}(\Psi_{\mathfrak{L}}, \Omega_{\mathfrak{M}})$

Generate $d, w, s, m_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P \qquad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

Compute $P_{\mathfrak{M}} \leftarrow s \cdot \Omega_{\mathfrak{M}}.P \quad Q_{\mathfrak{M}} \leftarrow s \cdot \Omega_{\mathfrak{M}}.Q$

Compute $(C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, \pi_{\text{CRED\_SHOW}_{\mathfrak{L}}}) \leftarrow$

$\hookrightarrow$ $\text{PRESENT\_HIDDEN}(P, Q, \Psi_{\mathfrak{L}}.\{\beta, t, a, d\})$ ▷ Algorithm 9

Compute $(C_{\Psi_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}\}, C_{Q_{\mathfrak{M}}}, \pi_{\text{CRED\_SHOW}_{\mathfrak{M}}}) \leftarrow$

$\hookrightarrow$ $\text{PRESENT\_HIDDEN}(P_{\mathfrak{M}}, Q_{\mathfrak{M}}, \Omega_{\mathfrak{M}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}\})$ ▷ Algorithm 9

Compute $(E_{\Psi^*_{\mathfrak{L}}}.\{\beta, \Phi, d\}) \leftarrow$

$\hookrightarrow$ $\text{PREPARE\_HIDDEN}(D, \Omega_{\mathfrak{M}}.\beta_{\text{TO}}, \Psi_{\mathfrak{L}}.\{\Phi, d + 1\})$ ▷ Algorithm 1

Generate $\pi_{\text{USER}} \leftarrow \{\pi_{\text{CRED\_SHOW}_{\mathfrak{L}}} \wedge \pi_{\text{USER\_HIDDEN}} \wedge \pi_{\text{CRED\_SHOW}_{\mathfrak{M}}}$
$\wedge\, \Omega_{\mathfrak{M}}.\beta_{\text{FROM}} = \Psi_{\mathfrak{L}}.\beta$
$\wedge\, \Omega_{\mathfrak{M}}.\beta_{\text{TO}} = \Psi^*_{\mathfrak{L}}.\beta$
$\wedge\, \Psi^*_{\mathfrak{L}}.d = \Psi_{\mathfrak{L}}.d + 1\}$

Set REQUEST $(P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\{\beta_{\text{FROM}}, \beta_{\text{TO}}, C_{Q_{\mathfrak{M}}}, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \pi_{\text{USER}})$

Set STATE $\leftarrow (d, D, E_{\Psi^*_{\mathfrak{L}}}.\{\Phi, \beta, d\}, \Psi^*_{\mathfrak{L}}.\Phi, \Psi^*_{\mathfrak{L}}.d + 1, \Psi^*_{\mathfrak{L}}.\beta, \Psi^*_{\mathfrak{L}}.L)$

Store STATE
Send REQUEST to the LA ISSUE(REQUEST) ▷ Algorithm 26
end function

Algorithm 26 Blockage Migration Server Issuing: Upon receiving the BLOCKAGE_MIGRATION request from a user, the LA verifies that the input for both the Lox and Migration credentials using Algorithm 10. If this is successful, the LA issues the new trusted Lox credential with $\Psi_{\mathfrak{L}}.t$ set to today's date (δ) and $\Psi_{\mathfrak{L}}.L$ set to $\Psi_{\mathfrak{L}}.L - 2$. The hidden Lox credential attribute $\Psi_{\mathfrak{L}}.\Phi$ is jointly created with the LA and new trusted bucket $\Psi_{\mathfrak{L}}.\beta$ that matches the value for $\Omega_{\mathfrak{M}}.\beta_{\text{TO}}$ remains hidden during issuing. All calculations are proved to be computed correctly with π_{ISSUE} and the response is returned to the user to be handled.

Input From Client

P	P component of MAC tag on Lox attribute
$\Psi_{\mathfrak{L}}.\{\Phi, L\}$	Revealed attributes of $\Psi_{\mathfrak{L}}$
$C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}$	Hidden attributes of $\Psi_{\mathfrak{L}}$
C_Q	MAC for $\Psi_{\mathfrak{L}}$
$P_{\mathfrak{M}}$	P component of MAC tag for migration credential
$C_{\Omega_{\mathfrak{M}}}.\{\beta, \beta_{TO}\}$	Hidden attributes of $\Omega_{\mathfrak{M}}$
$C_{Q_{\mathfrak{M}}}$	MAC on migration credential attributes
$E_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta, d\}$	Client encrypted attributes
π_{USER}	Proof of correct user blinding

Output To Client HANDLE_RESPONSE

RESPONSE	A Response containing: $(\Psi^*_{\mathfrak{L}}.P, \Psi^*_{\mathfrak{L}}.t, \Psi^*_{\mathfrak{L}}.L, E_Q, j_\Phi, T_\Phi, T_\beta, T_a, T_d, \pi_{\text{ISSUE}})$

function $\text{ISSUE}_{\text{BLOCKAGE\_MIGRATION}}(P, \Psi_{\mathfrak{L}}.\Phi, \Psi_{\mathfrak{L}}.L, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\{\beta, \beta^*\}, C_{Q_{\mathfrak{M}}}, D, E^*_{\Psi_{\mathfrak{L}}}.\{\Phi, \beta^*, d\}, \pi_{\text{USER}})$

if $\text{VERIFY}(P, \Psi_{\mathfrak{L}}.\{\Phi, L\}, C_{\Psi_{\mathfrak{L}}}.\{\beta, t, a, d\}, C_Q, P_{\mathfrak{M}}, C_{\Omega_{\mathfrak{M}}}.\{\beta, \beta_{TO}\}, C_{Q_{\mathfrak{M}}}, \pi_{\text{USER}})$ **then** ▷ Algorithm 10

Generate $d, w, j_\Phi \xleftarrow{\$} \mathbb{Z}/\ell\mathbb{Z}$

Compute $D \leftarrow dB$

Compute $P \leftarrow w \cdot \Psi_{\mathfrak{L}}.P$ $\quad Q \leftarrow w \cdot \Psi_{\mathfrak{L}}.Q$ ▷ Rerandomize tags

$s, E_{Q_{\mathcal{R}}} \leftarrow \text{ISSUE\_REVEALED}(P, t = \delta, L = \Psi_{\mathfrak{L}}.L - 2, a = a[\Psi_{\mathfrak{L}}.L - 3])$ ▷ Algorithm 2

$E_{Q_{\mathcal{H}}}, T_\Phi, T_\beta, j_\Phi \leftarrow \text{ISSUE\_HIDDEN}_{\mathfrak{L}}(E_{\Psi_{\mathfrak{L}}.\{\Phi,\beta,d\}})$ ▷ Algorithm 4

Compute $E_Q \leftarrow E_{Q_{\mathcal{R}}} + E_{Q_{\mathcal{H}}}$ ▷ Homomorphically compute MAC

Generate $\pi_{\text{ISSUE}_{\text{BLOCKAGE\_MIGRATION}}} \leftarrow \pi_{\text{ISSUE}}$ ▷ Generate proof

Set RESPONSE $\leftarrow (\Psi^*_{\mathfrak{L}}.P, \Psi^*_{\mathfrak{L}}.t = \delta, E_Q, j_\Phi, T_\Phi, T_\beta, \pi_{\text{ISSUE}})$

Send RESPONSE to the Client $\text{HANDLE}(\text{RESPONSE}_{\mathfrak{L}})$ ▷ Algorithm 7

end if

end function

5.6 Chapter Summary

In this chapter we discussed the implementation details for our Lox protocols based on the design that we detailed in Chapter 4. We detailed each of our Lox protocols and provided an overview for each to demonstrate how the presentation, preparation and issuing algorithms interact to form the core functionality of the Lox system.

Lox uses several cryptographic constructs to protect the social graph of users and privacy of Lox usage beyond the first interaction with the LA. This includes blinded migrations, which allow trusted users to migrate to new buckets in the event that their bridges become blocked without the LA learning the blocked or destination bucket. Lox uses Chase et al.'s keyed-verification anonymous credential scheme [CMZ14], incorporating many insights from Lovecruft and de Valence's Hyphae [LdV17]. Since our Lox Authority acts as both the issuer and verifier of anonymous credentials, this is an appropriate scheme to allow for a credential holder to verify attributes of their credentials with an authority without revealing any linkage between transactions or unnecessary attributes. Lox credential attributes are issued using one of four possible attribute options: reveal ($\mathcal{R}$), hide ($\mathcal{H}$), server-selected ($\mathcal{S}$) and joint ($\mathcal{J}$). The initial Lox credential issued from an open-entry invitation has all server-selected attributes besides the credential ID. The credential ID for all protocols (Φ) is always issued using the joint option, which allows the user to remain anonymous when revealing their credential while preventing them from unilaterally selecting the ID. Lox protocols each differ in which attributes are revealed, hidden and server-selected with the exception of the bucket (β) attribute which is never revealed beyond the initial credential issuing.

In Chapter 6 we test and evaluate the performance of each of our implemented protocols and provide an analysis of the latency and load considerations for using Lox.

Chapter 6

Evaluation

Having fully detailed the design and cryptographic implementation for each Lox protocol, we developed an implementation of the Lox protocols to evaluate their practicality. In this chapter we present the performance results of running each protocol as well as an analysis of the performance our system achieved. We consider our system in relation to the existing Tor bridge distribution network, the number of users that are typically served by the existing infrastructure and how the addition of the Lox system would impact this.

6.1 Implementation Overview and Experiment Setup

We implemented each of the Lox protocols we described in Chapter 5 and wrote a test script to simulate a complete request, response, and response handling sequence for each Lox protocol. Our implementation was developed in Rust due to its high performance and memory safety features and is available online at https://git-crysp.uwaterloo.ca/iang/lox. We made extensive use of several dalek cryptography libraries; `curve25519_dalek` [LdV20a] to construct the prime order groups that Lox secret keys rely on, `ed25519_dalek` [LdV20b] to sign and verify open invitations, and `zkp` [LdV20c] to construct and verify zero knowledge proofs for each transaction between the Lox Authority and the user. Each of our experiments was run on a single core of an Intel Xeon E7-8870 at 2.40 GHz running Ubuntu 16.04. Noting that Tor metrics puts the number of active bridges just above 3500 [Tor22g], we test the performance of our protocols across bridge pools of various sizes from 900 to 9000 in increments of 900. For each test, the total number of bridges in the bridge pool are divided in half such that half of the bridges are sorted into open invitation buckets and the other half are sorted as hot spare bridges. None of the initial

bridges are sorted into trusted-user buckets until users request trust promotions. We note that it is not strictly necessary to divide the pool in half when deploying Lox but some percentage of the bridge pool should be set aside as hot spares to create new buckets when trusted users migrate to new buckets after blockage events. It is also possible to add new hot spare bridges to the bridge pool after deployment, which is necessary to prevent censors from overwhelming the system over time. The request size and time, response size and time and response handling time for each protocol were averaged over 10000 runs. The check blockage and blockage migration protocols were additionally run over several blockage scenarios that considered different percentages of bridges blocked to provide insight into the cost of running the check blockage protocol as the number of blocked bridges grows (and the migration table becomes larger). To measure this, we first allowed 10000 users to reach a trust level of 4 ($L = 4$) then measured the performance statistics for the check blockage (Algorithms 23 and 24) and blockage migration (Algorithms 25 and 26) protocols when 5% to 100% of bridges are blocked in 5% increments.

6.2 Results

The results of running each protocol with 3600 bridges total, 1800 open-entry buckets with one bridge each and 600 hot spare buckets with three bridges each, are displayed in Table 6.1. We note that for most of the reported protocols, the sizes and times are the same for all bridge pool sizes (i.e., bridge pools from 900 to 9000). The exceptions are the trust promotion and check blockage protocols, since both require the Lox authority to compute and send an encrypted hash table of the bridges eligible for migration. For these, the size and time required to receive and handle the response grows linearly with the number of open-entry buckets and the number of blocked bridges, respectively. For example, the server response time for the check blockage protocol is displayed in Figure 6.1. We additionally report the results for varying bridge pool sizes for the trust promotion and check blockage protocols in Figures 6.2a–6.2c and 6.3. All size results are reported in bytes and all timing results are reported in miliseconds.

Table 6.1: Performance statistics per user for Lox protocols initialized with 3600 bridges (1800 untrusted open-entry buckets and 600 hot spare buckets) over 10000 users. All times are reported in milliseconds (ms) and sizes are reported in bytes. Results for the check blockage protocol are reported for 5, 50, and 100% of trusted bridges being blocked as the performance changes accordingly. All times and sizes reported in the table are the same for all bridge pool sizes for all protocols except trust promotion and check blockage, which change linearly with a growing bridge pool.

Protocol	**Request Size**	**Request Time**	σ	**Response Size**	**Response Time**	σ	**Response Handling Time**	σ
Open Invitation	332	0.75	0.02	740	3.23	0.03	2.26	0.03
Trust Promotion($0 \rightarrow 1$)	2216	9.94	0.04	378104	364.2	0.3	2.3	0.2
Trust Migration ($0 \rightarrow 1$)	936	4.29	0.03	584	6.50	0.04	2.80	0.03
Level Up ($1 \rightarrow \cdots \rightarrow 4$)	3368	15.16	0.05	712	15.28	0.05	3.70	0.03
Issue Invitation	1672	8.00	0.04	1480	15.04	0.06	7.29	0.05
Redeem Invitation	1576	7.00	0.04	680	9.09	0.05	3.24	0.03
Check Blockage 5%	744	3.27	0.03	6404	11.22	0.02	0.25	0.01
Check Blockage 50%	744	3.27	0.03	63104	64.31	0.09	0.49	0.01
Check Blockage 100%	744	3.26	0.03	126104	122.5	0.1	0.75	0.02
Blockage Migration	1224	5.84	0.03	840	9.39	0.04	4.12	0.03

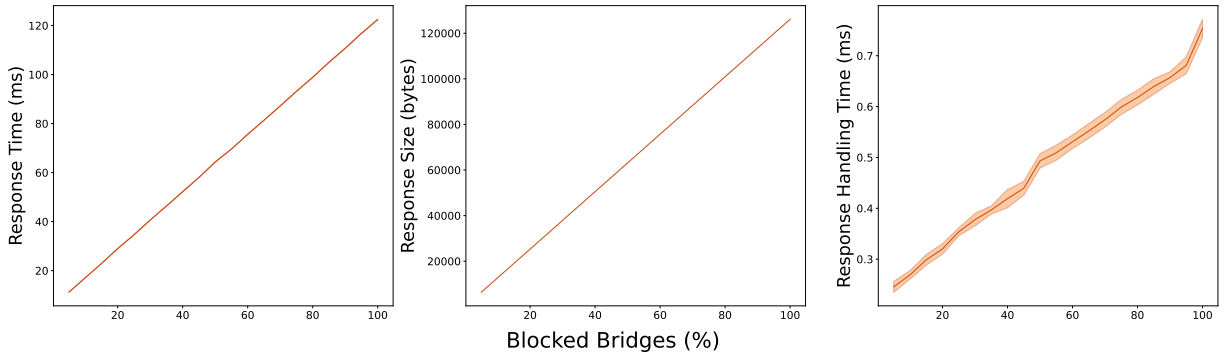
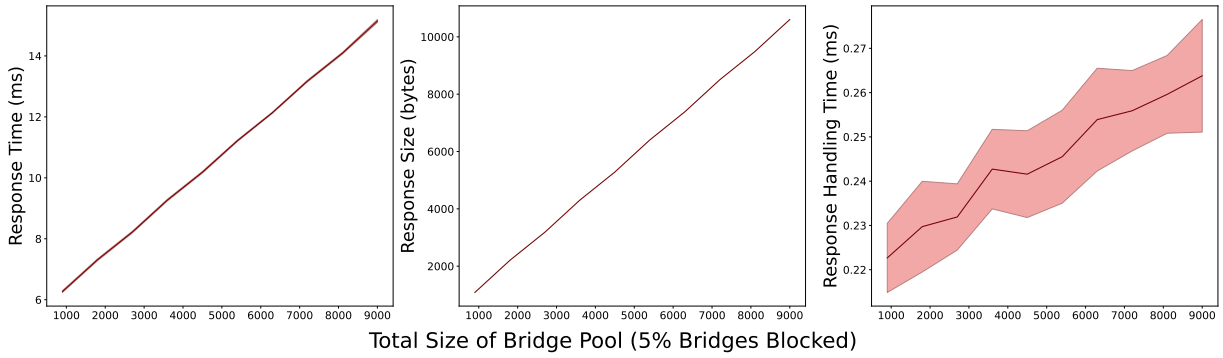
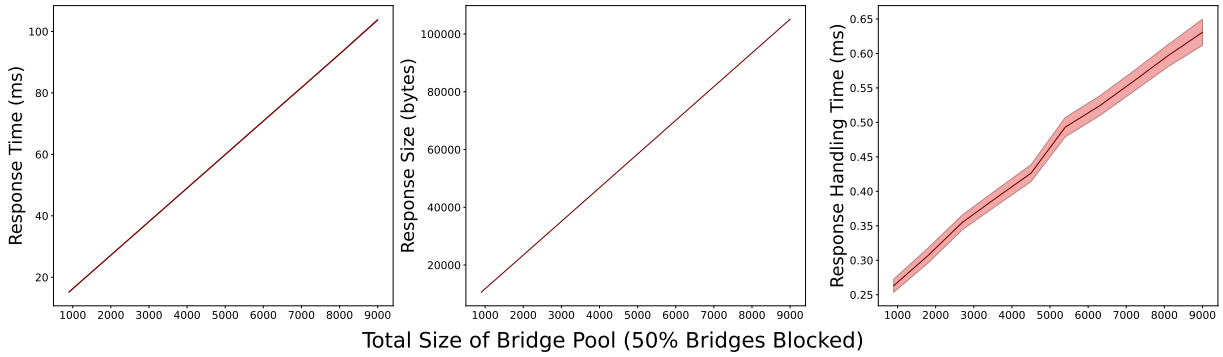


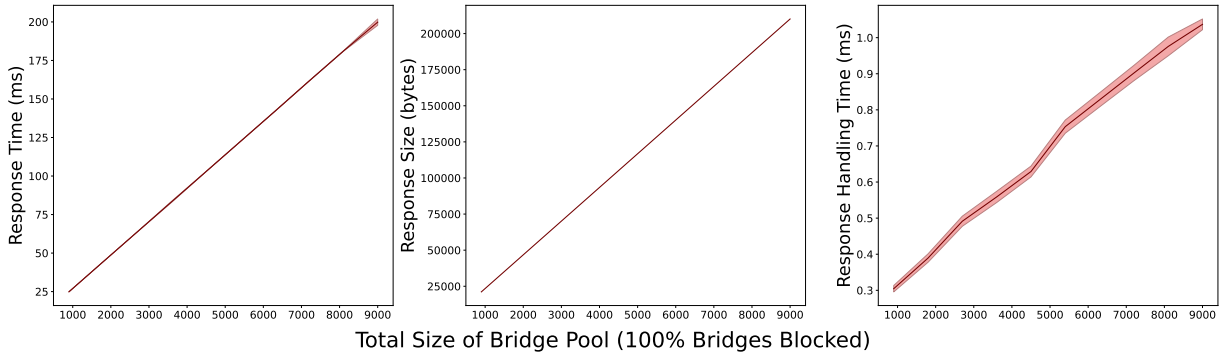
Figure 6.1: This shows the performance of the check blockage protocol with the standard deviations in shading and often too small to see. The latency and message size of the check blockage protocol grows linearly with the number of blocked bridges. This figure shows how the response time, response size and response handling time grow as the percentage of blocked bridges in the bridge pool increases.



(a) Performance of check blockage protocol with 5% blockage and increasing bridge pool



(b) Performance of check blockage protocol with 50% blockage and increasing bridge pool



(c) Performance of check blockage protocol with 100% blockage and increasing bridge pool

Figure 6.2: The performance of the check blockage protocol is shown with the standard deviations in shading. This figure shows that the response time, response size and response handling time grow linearly as the bridge pool increases from 900 to 9000 bridges. The figures show how the change in bridge pool size impacts a specified percent of bridges blocked, where Figure 6.2a shows 5% blockage, Figure 6.2b shows 50% blockage and Figure 6.2c shows 100% blockage.

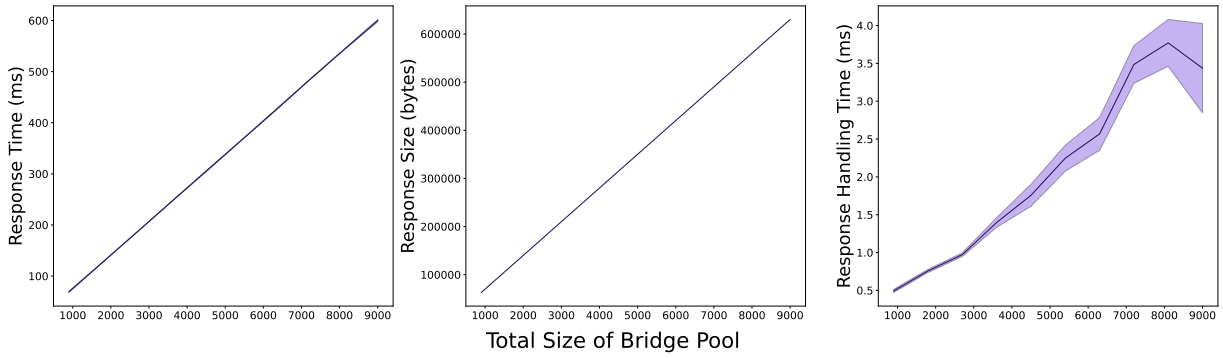


Figure 6.3: The performance of the trust promotion protocol is shown with the standard deviations in shading and often too small to see. This figure shows how the response time, response size and response handling time grows linearly with an increasingly large bridge pool beginning with 900 bridges and growing to 9000 bridges.

6.3 Analysis of Results

We can use these results to analyze the practicality of Lox in terms of the added cost to existing bridge users and to provide estimated system requirements, specifically the number of cores needed for deployment. We look to the Tor project as the largest publicly deployed bridge distribution network for insight into the demand on the system in terms of the expected number of users, users per bridge, and the likely request frequency of each of our protocols. We leave deployment to future work but strive to provide an analysis of the expected performance that would be useful for anyone considering deployment of Lox.

6.3.1 Latency and Overhead Considerations

The latency considerations for each Lox protocol can be calculated using our performance statistics from Table 6.1. Each would contribute additional latency and overhead to an existing bridge distribution system such as Tor's BridgeDB that only involves acquiring bridges through email, https or using the default moat bridges embedded in the browser with no further costs. However, as discussed in Section 3, the three channels of bridge distribution are the only defence Tor's BridgeDB provides against sock-puppet attacks and have proven ineffective in some regions, preventing many users from accessing bridges. By allowing users to anonymously gain trust for keeping bridges unblocked and build the user base by inviting friends, Lox rewards good behaviour and reduces the harm that sock-puppets can cause to users of the system.

Lox was designed to impose a minimal burden on bridge operators and users and could be implemented to use existing channels for bridge distribution by replacing directly distributed bridges with open invitations that users can then use to create their Lox credential. This would cause minimal disruption to the usage patterns of current users and bridge operators of a system like Tor's BridgeDB.

Client-side Latency

We find that the latency for individual users is reasonable for all operations. For trust promotion and check blockage protocols, server side response sizes and times are hindered by the hashtable of bridges that must be sent in the migration key credential and thus have the highest latency. Trust promotion consistently has the worst response time (1.2 s) and size (126 kB) but since every Lox user will only make a single trust promotion request and they will have access to an open-entry bridge to receive the response (i.e., a sufficiently high-bandwidth connection), this should not be a significant burden to the user. Below we discuss the number of cores Lox needs to support user requests in greater detail.

The Check Blockage protocol in the worst case where 100% of bridges are blocked, has almost identical performance (response time of 1.1 s and size 126 kB). However, if the user has lost access to all three of their bridges, they must first acquire a new open-entry bridge in order to run the check blockage protocol without exposing their IP address to the LA, adding to the overall burden of running this protocol. Much more than the trust promotion protocol, the check blockage protocol is likely to see spikes in requests frequency that coincide with bridges being blocked, which could further slow the response time for individual users. A strategic and patient censor could exploit this bottleneck in Lox to slow down or even deny successful blockage migrations to coincide with a particular event, such as an election or planned coup, as described in rBridge [WLBH13]. A targeted, long-term strategy like this by the censor would not only limit access to bridges at a critical moment, but could create a significant spike in check blockage and blockage migration requests while users migrate to unblocked bridges. If the censor subsequently blocks the newly migrated to bridges, trusted users will be effectively locked out of accessing trusted bridges, which could have a spillover effect on increased open invitation requests as well. Although the scenario described is possible for a determined censor, we note that it would require a censor to gain access to the system and leave bridges unblocked for genuine users for a minimum of 72 days, the sum of the first three values in the "Upgrade t" column in Table 4.1, as that is the minimum time that has to pass before an open-entry user can acquire enough trust to advance to a trust level of $L = 3$, where they are eligible for blockage migrations .

The check blockage protocol is also vulnerable to DoS attacks by a censor with a Lox credential that has a trust level of $L \geq 3$ and a blocked bucket. Since we do not record the

credential ID for the check blockage protocol with the understanding that a genuine user may mistakenly call check blockage when bridges are temporarily down rather than blocked, a censor could exploit this by calling check blockage repeatedly, such as every second. This would greatly increase the response time for all requests from the LA, causing it to re-create the migration table for each check-blockage request. To counter such an attack, we can have the LA maintain an expiring rate-limit list for specific credential IDs that have been seen within some time frame or else pair the check blockage request with a challenge-response type client puzzle as detailed by Stebila [Ste09].

Overall, we find that the latency added by Lox is reasonable and would not over-burden users or bridge operators. The ability for trusted users to migrate to new bridges when their bridges are blocked provides a strong incentive for users to compromise on some latency and overhead increases.

Server-side Load

Lox requires periodic communication between bridge users and the Lox authority. Though a single user may communicate with the LA infrequently (twice in the same day for migration protocols, otherwise with larger gaps between protocols in most cases), we must consider how frequently each protocol may be called across the entire userbase to get a sense of the worst-case load on our system. Among our Lox protocols, the majority will be called at most one time by every new Lox user. However, users that have a trust level of $L = 4$ will be able to level up every 84 days and issue eight invitations over the same period. Using our results from Table 6.1, we can calculate the CPU cost of the level up protocol and the issue invitation protocol as:

$$\begin{aligned} \text{Level Up: } & 15.27\,\text{ms} \,/\, 84\,\text{days} \\ & = 0.18\,\text{ms} \,/\, \text{day} \\ \text{Issue Invitation: } & 15\,\text{ms} \cdot\ 8 \,/\, 84\,\text{days} \\ & = 1.43\,\text{ms} \,/\, \text{day} \end{aligned}$$

Check blockage and blockage migration protocols could be requested by trusted users as often as bridges become blocked, potentially creating a significant burden on the system.

Suppose there are b buckets, 1 bucket gets blocked every Δ days, and we remove blocked buckets from the eligible list of migrations after W days. Then there will be W/Δ blocked buckets in steady state, and any given bucket will get blocked every $b \cdot \Delta$ days. A user will run

the check blockage and blockage migration protocols every $b \cdot \Delta$ days and it will cost:

$$(5.31 + 0.065 \cdot W/\Delta + 9.4)\,\text{ms}.$$

Then, we can calculate the total for check blockage and blockage migration for $W = 365$:

$$\begin{aligned}&(5.31 + 0.065 \cdot W/\Delta + 9.4)\,\text{ms} \,/\, (b \cdot \Delta)\,\text{days}\\ &\quad = (14.71/\Delta + 23.725/\Delta^2)/b\,\text{ms} \,/\, \text{day}\end{aligned}$$

Thus, the total cost in steady state is:

$$1.61 + (14.71/\Delta + 23.725/\Delta^2)/b\,\text{ms} \,/\, \text{day}$$

We compute the total number of cores required for n users in steady state where we have b buckets and one bucket gets blocked on average every Δ days. Figure 6.4 shows the number of cores needed for each $n = 1000000$ users for varying bridge pools (the load is linear in n). We see that frequent bridge blocking increases the load on the system but this effect is mitigated by the size of the bridge pool as the same number of users are less impacted by blocked buckets if there are more buckets in the pool. Overall, we see that Lox can provide reasonable support for millions of users with a single core even in the worst case where all currently handed out bridges (but not the reserved hot spares) are blocked each day.

Protocol Frequency and Usage

The number of bridge users we discussed in Chapter 3 provides an estimate of the number of users Lox may need to serve on average and during censorship events. To recall, these data were presented in Figures 3.2 and 3.3. We see that there are approximately 300–400 new bridge fingerprints that join the bridge pool in a month given the results for bridge churn shown in Figure 3.1, which were collected from Tor metrics [Tor22d]. As Figure 3.3 shows, the majority of these newly added bridges have fewer than 20 average users each day while a few have between 20 and 55. We can use the maximum of each bridge's reported average users for the new bridge fingerprints (approximately 2500 total for the maximum of daily averages for each fingerprint) to make an estimate of the daily number of open-entry requests the LA may receive. However, since each user will only call the open-entry protocol once, this has a negligible effect on the server-side load, as discussed above. Tor's current number of bridge users is close to 100000 users as observed in Figure 3.2. Even if we double this number to get the maximum number of users Tor bridges have seen since 2016, Lox can comfortably handle them with a single core.

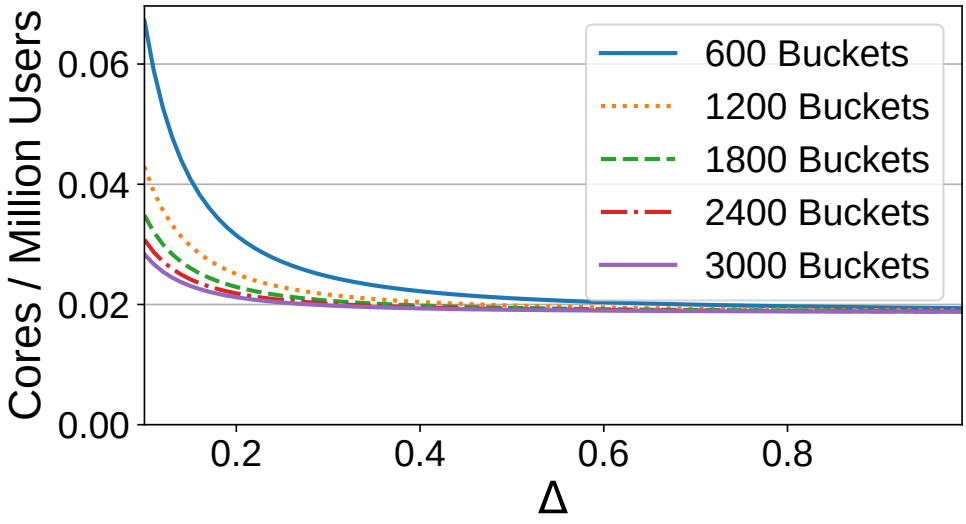


Figure 6.4: The number of cores required per million users for varied bridge pool sizes and one bucket being blocked on average every Δ days. Frequent bridge blocking increases the load on the system, but this is mitigated by a larger bridge pool for a constant user base.

6.4 Chapter Summary

In this chapter we presented the performance statistics and other results from experimentation on an implementation of our Lox system. We showed that each of our Lox protocols has reasonable performance with our two-step protocols for trust promotion and check blockage having increased latency due to the creation of the migration table for migration key credentials. Even still, we showed that in steady state, the load on the Lox server is minimal and can support millions of users on even a single core. Considering that Tor's BridgeDB has seen only 200000 users at its peak between 2016 and 2022, we have shown Lox to be capable of handling the steady state load of such a user base.

In Chapter 7, we discuss extensions to Lox that could make the system more appropriate for real-world scenarios and deployment.

Chapter 7

Improvements and Future Work

In this chapter we discuss some known attacks against Lox and areas where Lox's functionality can be extended for specific users and deployments in the real world.

7.1 Known Attacks and Limitations

7.1.1 Sock-puppet Attacks

The ability for censors to create sock-puppet accounts to enumerate and block bridges is a well known problem noted in prior work [LdV17, DRPC16, WLBH13, NFHG19]. Unlike prior work that limits use of the system to trusted users [WLBH13, LdV17] or relies on a third party to limit duplicate accounts [DRPC16] at the expense of protecting the social graph, we provide an open-entry avenue to join Lox with fewer protections. This means that we must contend with a greater number of censor controlled sock-puppet accounts attempting to gain access to bridges. In the previous chapters, we have detailed several features of Lox's design that aim to contribute to this. To summarize, in Lox we propose:

1. Limiting the number of bridges distributed to new users to a single bridge until the user is eligible to advance to the next trust level.

2. Limiting a user's privileges until they have spent some configurable amount of time with knowledge of unblocked bridges.

3. Allowing trusted users to invite friends incrementally.

4. Allowing trusted users to invite friends only to their bucket.

5. Using the concept of inheritance to prevent users who have migrated to a new bucket after a blocking to join with no record of blocking.

While these interventions together limit the effectiveness of sock-puppet attacks, we note that they are still imperfect and do not stop sock-puppets at the entry point to Lox. Using the entry pathways that the Tor project currently uses to distribute Lox open-entry invitations may lead to bridges being almost completely inaccessible in some regions.

Lox takes the position that protection of the social graph should not be sacrificed for the ability to monitor and punish untrusted malicious users and their friend groups since this could have consequences extending beyond the Lox system in the event of a compromise by the censor. In practice, this may be an idealistic position that results in genuine users being unable to access Lox through an open-entry pathway. Still, bootstrapping for trusted users and a system that protects their usage and social networks while rewarding their good behaviour, does not exist in currently deployed bridge distribution systems and we anticipate that this could benefit many users. Lox is intended as a proof of concept system that provides privacy protection and incorporates some of the benefits found in less privacy protective systems such as Salmon. We hope that our design can move the conversation on bridge distribution forward and act as a step towards an acceptable implementation that balances these inherently conflicting issues of protecting user's privacy and preventing sock puppets from winning.

7.1.2 Patient Censor's Recommendation Attack

Douglas et al. [DRPC16] discuss a patient censor that, upon learning a bridge address, allows the bridge to remain unblocked in order to gain trust and the privileges that come along with it. A patient censor that has gained enough trust to be able to recommend new users is able to perform a *recommendation attack*, where they can grow the number of agents in the system exponentially through invitations over time. Douglas et al. point out that while this attack can be very damaging to the overall system, particularly if the attack is coordinated to coincide with a political event, the bridges must be left unblocked for several months before the attack is successful. A similar attack is possible in Lox with the censor forced to make the same tradeoff of leaving bridges unblocked for a long period. However, a much more serious attack can be launched through Lox's blockage migration protocol by a group of sock-puppet patient agents that have reached a trust level of $L \geq 3$. In this attack, one of the agents could block their bridges, triggering the bucket to be added to the eligible migrations list for blockage migrations. One of the agents could complete a blockage migration and immediately block all bridges in the new bucket. Now that the bridges in

this new bucket are blocked, the Lox Authority will be triggered to replace the destination bucket with a new hot-spare bucket if one exists. Once this process has completed, a second agent can repeat the blockage migration and block the new bridges, rendering another three-bridge bucket of new, previously unused bridges unusable. Any legitimate users that happen to be caught in a bucket with a group of censors will be locked out of their trusted position in the system and will need to request new untrusted bridges. Censors can continue to enumerate through the cache of hot spares until they run out of trusted agents eligible for migration.

While an unmitigated attack like this would be extremely damaging to Lox, we stress that it is a highly inefficient way for the censor to enumerate bridges requiring a significant amount of time that bridges are left unblocked (at least 72 days for 2 of the 3 bridges per bucket). The open-entry path likely provides a much more efficient way of exhausting hot spares. Furthermore, the amount of time the Lox Authority takes to notice the blocked bucket and replace it with a hot spare may be arbitrary, making it difficult for the censor to time their attack. Finally, like in Salmon, a core set of trusted users that had established buckets without any censors in the group would remain safe and unimpacted by such an attack. We can further mitigate the ability for the censor to enumerate all hot spares by keeping track of the number of times a hot-spare bucket has been moved to replace a blocked destination bucket.

7.1.3 Censor Advantage over Casual and Event-driven Users

Since we assume that genuine users do not maliciously block bridges, they must always be on par or a step ahead of a censor's planned attacks, in order to have the best chance of preserving their status as a trusted user with unblocked bridges. This means Lox users should be diligent to make sure they upgrade their trust level as soon as they are able so that they can increase their chances of possessing bridges that will remain unblocked. While we do not consider this to be an undue burden on the users, it is less than ideal as human users are prone to forgetfulness and among other things, may use Lox only sporadically for legitimate reasons. To give an example, spikes in usage would be expected if Tor entry nodes or a particular VPN were blocked in a region during a contested election. Event-driven or casual users are at a severe disadvantage to a censor intent on compromising Lox. Censors can use their unlimited resources to specifically target Lox with preplanned blocking events that can be coordinated to induce maximal damage without warning to users. Censors can therefore game the timing of the migration table upgrade in order to upgrade from $L = 0$ to $L = 1$ even when bridges are blocked. Conversely they can wait until they have just reached the highest trust level required to migrate before blocking buckets, then could block the bucket they migrate to soon after receiving it, locking out all users who have already migrated.

Though we recognize that we may not be able to fully overcome the censor's advantage, we have designed Lox to disincentivize a censor from earning trust by requiring them to keep bridges unblocked for long periods. The bootstrapping period also helps to ensure that censors do not gain access to a pool of trusted buckets. We may be able to further level the playing field by incorporating Lox into a client side system (e.g., the Tor browser) that automates trust level upgrade and migration requests as soon as it is appropriate to do so. While automation would make the censor's job easier as well, it would give them no greater advantage than a user seeking to use the system rather than compromise it.

7.1.4 Unoptimized Parameters

Our design of Lox, as well as the implementation and evaluation, include a number of parameters such as the number of bridges that are distributed to untrusted vs. trusted users, the time it takes to upgrade between trust levels, etc. We note here that our parameter selection is not meant to be generalized for every system and may not be suitable as is, for any particular deployed system. We chose parameters that we thought would best demonstrate a system that increases privileges based on a user's trust level and in the case of the expiration of a Lox credential (511 days) and Invitation credential (15 days), these values being $2^k - 1$, were selected to make the range proofs more efficient. Intentionally, nothing in Lox prevents altering the parameters we have chosen or optimizing them for a specific use case. Furthermore, nothing in Lox prevents incorporating an appropriate algorithm to determine how open-entry bridges are distributed, which we discuss further below.

Our hope with Lox was to provide a template for a deployable system that has privacy preserving features and does not prohibit anyone from optimizing the particular details of a suitable deployment further. We offer our thoughts and insight into extensions that we expect would further improve Lox, in Section 7.2.

7.1.5 Exponential Growth of Trusted Buckets

With trusted users being able to invite ever increasing numbers of friends to Lox, trusted buckets are in danger of having the number of users they support grow exponentially. Eventually, the bandwidth would be sufficiently limited for each individual in a bucket that they would use their utility. One way to prevent this would be to create an additional migration protocol that could become available when a bridge operator notices congestion on their bridge and notifies the LA. The implementation details of such a congestion migration protocol would require more thought to ensure that genuine users can continue to migrate to new trusted buckets when their

bridges pass some congestion threshold and that censors do not gain an additional advantage in enumerating bridges by flooding their bridges with sock-puppets once they are eligible. Some interesting directions for future work to explore would be to restrict migrations to users with sufficiently high trust levels, low d values and adding an additional attribute to the Lox credential that switches from a false to a true state in order to disallow any user from requesting a congestion migration more than once. Alternatively, once a bucket is determined to be congested, it can split into two buckets, randomly assigning everyone that requests the migration protocol to one of the two buckets.

7.2 Extensions

Lox was intentionally designed to be compatible with existing bridge distribution systems in order to remove obstacles to adoption. However, some extensions involving significant upgrades to the current protocols run by bridge operators, may help to improve the Lox system further. We detail these below.

7.2.1 Extensions for Bridge Operators

Bridge operators provide the bridge that users of the bridge distribution system can use to connect to the open Internet.

Bridge tokens could be distributed by a bridge operator for users who are able to prove bridge usage over some period of time each day. The LA could then require some number of valid bridge tokens, along with the other valid Lox credentials for upgrading trust and generating invitations for issuing. This would serve to make it even more challenging for a censor to gain access to trusted functionality without sinking significant time and resources into keeping the bridges up and functioning.

We assume that genuine bridge users would have acquired bridges in order to actually use them, so consistent usage for a predetermined amount of time over some number of days does not burden users. Bridge tokens also encourage trusted regular users to generate invitation tokens rather than users that are less invested in the system. We assume that these users would have the most interest in keeping their bridges available and usable and so would limit the invitations they gave out to trusted friends so as not to risk losing access to their own bridges or overwhelming them with users.

7.2.2 Extensions for User Privacy

Using the keyed verification anonymous credential scheme of Chase et al. [CMZ14] in Lox, described in Chapter 5, ensures that despite the LA being the issuer and verifier of credentials, these credentials protect the social graph of Lox users, and their anonymity as they use Lox, but this can be improved further. In our implementation and construction of Lox, the exact protocol being run (e.g., level up, issue invitation, etc.) is known to the LA, thereby leaking information about the time a specific protocol is requested and the frequency with which different protocols are run. We could eliminate even this leakage by blinding the protocol that is being requested with a zero knowledge array lookup. This change would require users to send exactly the same credentials and tokens for each request, even if they are no-op tokens. As an example, in our current scheme, a successful check blockage request from the user results in a valid migration token issued by the LA, allowing the user to migrate to a new bucket. To allow the same functionality without the LA learning which protocol was called, the LA would issue a migration token to the user for every request, regardless of whether or not it checks for blocked bridges. The vast majority of migration tokens issued would then be no-op tokens which would simply allow a user to migrate to the same bucket they are already in. If the check blockage protocol is called and bridges are found to be blocked, the migration table would be updated and the LA would issue a true migration token in zero knowledge. In both cases, credential issuance would continue in the same way for all protocols and the LA would not know which protocol was called.

These changes would increase the size and computation time required to complete each protocol but this may be an acceptable trade off for the added anonymity it would provide if bandwidth is not a concern and the rate of requests remains relatively small. Additionally, because many of the Lox protocols will have static timings and sizes, full implementations of Lox should take this into account and ensure that interactions with the LA would not disclose a Lox bridge.

7.2.3 Extensions to Distribute Bridges More Effectively

Location Based Distribution

Bridge distribution systems that operate in many different parts of the world may require some awareness of locale so that appropriate bridges can be distributed to users within particular regions and bridge blockages can be distinguished based on the regions from which they are inaccessible. The Lox design we have described does not include either of these features, but could be extended to support distribution based on a user's location.

Lox could provide location-based bridge distribution by having the LA sort pools of bridges based on their location and then create buckets from the bridges in these pools. When a user requests an open-entry invitation token, they could indicate their region or request bridges from a particular region, which could be encoded as an attribute of the invitation token. When the invitation is presented to the LA, a bucket within the requested region's bridge pool can be prioritized and a (hidden) location attribute can be added to the user's Lox credential for future migrations. Though we would have to assume that censors and users could both spoof their location, this does not necessarily create any greater advantage for the censor. Having open-entry buckets spread across locations may allow a censor to enumerate bridges across regions more quickly with sock-puppet requests than they would have been able to if regional distribution was not implemented. This also depends on the strategy chosen for distributing open-entry buckets.

Distributing bridges based on location introduces some novel considerations and requires reliable tools to check blocking across regions for systems with global reach. For example, region based distribution enables censors to acquire a bucket by indicating a spoofed region while blocking the bucket's bridges for users within their area of influence without consequence. While this seems like a cause for concern, it may be largely inconsequential for users in the censor's area of influence. This behaviour does not, after all, allow the censor to enumerate through buckets of bridges that are distributed to users in their area of influence faster than they would be able to without the location specific pooling. Even if a censor used this technique to block bridges that they knew of by also acquiring bridges that work within the censored region, when trying to migrate to new bridges, the censor would be penalized for the migration. This also does not change the threat for regular users.

It would seem that the most challenging aspect of distributing bridges based on region is ensuring that blockages can be monitored and reported accurately, reliably, and in a timely manner in each of the regions where a bridge distribution system operates. The LA must be able to verify that bridges are indeed blocked, rather than experiencing a temporary outage for reasons other than censorship. Locations of bridges and the differing capabilities and behaviours of censors around the world complicate this significantly.

Incorporation into Popular Browsers

For large scale projects in particular, incorporating the Lox client-side protocols into a browser that automates communication with the LA would vastly improve usability for users. The Tor browser is an obvious candidate due to the existing pool of bridge operators that could be rolled into a Lox implementation and users that may require bridges. Having the browser run an automated check for eligible credential updates and then run the appropriate protocols to update

those credentials could help reward genuine users whenever they open the Tor browser. If they have had the browser open for an extended period, a prompt with a CAPTCHA (or something equally challenging for an automated censor to evade but less of a usability nightmare for genuine users) could come up before the check is run to prevent censors from just leaving the window open to passively accrue credit.

Optimizations for Open-Entry Bridge Distribution

It is possible that how bridges are handed out to new users from the pool of open-entry bridges could be optimized to discourage a censor or limit the number of buckets they are able to occupy. Several parameters in Lox and adjacent to the Lox system are situation and system dependent. The amount of time required to move between trust levels in Lox is configurable and should be adjusted to complement each system's bridge pool size, rate of bridge churn, the algorithm used to determine which bridges are distibuted to which users, and the number of users that know about a single bridge. rBridge [WLBH13] for example, places 40 users in each bridge group where each user gets 3 bridges. Salmon [DRPC16], like Lox, allows trusted users to invite friends to the same bridges, sets the maximum number of users for each bridge group to 10, in order to reduce the number of censors likely to be in each group and to ensure good throughput for those users. Future work could look at creating an algorithm to distribute open-entry bridges in a way that optimizes for genuine users becoming trusted users and incorporates metrics such as the rate of bridge churn, the number of different channels of distribution, the number of users per bridge group, and the percentage of users that are censors. Research in this direction could help to improve Lox further by increasing the likelihood that new users and not censors tend to occupy new buckets.

Chapter 8

Conclusion

In this thesis, we showed that it is possible to design a bridge distribution system that is open to all users and leverages users' trust networks for bridge distribution while protecting their privacy as well as all connections in their social graphs. To this end, we designed and implemented Lox, a new bridge distribution system, that responds to known issues in Tor's BridgeDB and combines insights from prior work.

Lox allows users to advance trust levels towards elevated privileges for proven good behaviour (i.e., bridges remain unblocked) over time. Highly trusted users can invite friends, and migrate to new bridges in a blockage event. In order to protect the privacy of users and their social networks, Lox is implemented using keyed verified MAC anonymous, unlinkable credentials [CMZ14] that provide reasonable performance. Lox also introduces several features to limit the impact of a censor's malicious behaviour. We implemented the Lox protocols in Rust, and measured their performance to show that Lox can provide reasonable support for millions of users with even a single core. Finally we have discussed improvements to Lox that could be implemented for real-world deployment under certain conditions.

We hope that Lox provides a useful contribution for advancing the state of bridge distribution and inspires further exploration of privacy protective reputation systems.

References

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size Dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, pages 111–125, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BDF+19] Cecylia Bocovich, Roger Dingledine, Alexander Færøy, Kat Hanna, Philipp Winter, and Taylor Yu. Addressing Denial of Service Attacks on Free and Open Communication on the Internet: Final Report. Technical Report 2019-05-001, The Tor Project, May 2019. [Online; accessed February 21, 2022] https://research.torproject.org/techreports/dos-censorship-report2-2019-05-31.pdf.

[BG16] Cecylia Bocovich and Ian Goldberg. Slitheen: Perfectly Imitated Decoy Routing through Traffic Replacement. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1702–1714, New York, NY, USA, 2016. Association for Computing Machinery.

[BSR17] Diogo Barradas, Nuno Santos, and Luís Rodrigues. Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams. *Proceedings on Privacy Enhancing Technologies*, 2017(4):5–22, 2017.

[BSR18] Diogo Barradas, Nuno Santos, and Luís Rodrigues. Effective Detection of Multimedia Protocol Tunneling Using Machine Learning. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 169–185, Baltimore, MD, August 2018. USENIX Association.

[BSRN20] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS'20, pages 35–48, New York, NY, USA, 2020. Association for Computing Machinery.

[Cha85] David Chaum. Security Without Identification: Card Computers to Make Big Brother Obsolete. *Communications of The ACM*, 1985.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 56–72, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and Keyed-Verification Anonymous Credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1205–1216, New York, NY, USA, 2014. Association for Computing Machinery.

[CS97] Jan Camenisch and Markus Stadler. Proof Systems for General Statements about Discrete Logarithms. Technical report, ETH Zurich, 1997.

[CZJJ12] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching From a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 605–616, New York, NY, USA, 2012. Association for Computing Machinery.

[DCRS13] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol Misidentification Made Easy with Format-Transforming Encryption. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 61–72, New York, NY, USA, 2013. Association for Computing Machinery.

[DDK+21] Jakub Dalek, Nica Dumlao, Miles Kenyon, Irene Poetranto, Adam Senft, Caroline Wesley, Arturo Filastó, Maria Xynou, and Amie Bishop. No Access: LGBTIQ Website Censorship in Six Countries. Technical Report Citizen Lab Research Report No. 142, University of Toronto, August 2021.

[DGS+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018(3):164–180, June 2018.

[Din11] Roger Dingledine. Strategies for Getting More Bridges. Technical Report 2011-05-001, The Tor Project, May 2011. [Online; accessed February 21, 2022] https://research.torproject.org/techreports/strategies-getting-more-bridge-addresses-2011-05-13.pdf.

[Din12] Roger Dingledine. Obfsproxy: The Next Step in the Censorship Arms Race, Feb 2012. [Online; accessed February 21, 2022] https://blog.torproject.org/obfsproxy-next-step-censorship-arms-race.

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message Authentication, Revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 355–374, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[DM06] Roger Dingledine and Nick Mathewson. Design of a Blocking-resistant Anonymity System. Technical Report 2006-11-001, The Tor Project, November 2006. [Online; accessed February 21, 2022] https://research.torproject.org/techreports/blocking-2006-11.pdf.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. Technical report, Defense Technical Information Center, Fort Belvoir, VA, January 2004. [Online; accessed February 18, 2022] http://www.dtic.mil/docs/citations/ADA465464.

[Doe21] Eric Doerr. Securing our approach to domain fronting within Azure, March 2021. [Online; accessed May 10, 2022] https://www.microsoft.com/security/blog/2021/03/26/securing-our-approach-to-domain-fronting-within-azure/.

[DOG18] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing China's Blocking of Unpublished Tor Bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, Baltimore, MD, August 2018. USENIX Association.

[DRPC16] Frederick Douglas, Rorshach, Weiyang Pan, and Matthew Caesar. Salmon: Robust Proxy Distribution for Censorship Circumvention. *Proceedings on Privacy Enhancing Technologies*, 2016(4):4–20, October 2016.

[DWH13] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Zmap: Fast Internet-Wide Scanning and Its Security Applications. In *Proceedings of the 22nd USENIX Conference on Security*, SEC'13, pages 605–620, USA, 2013. USENIX Association.

[EFW+15] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pages 445–458, New York, NY, USA, 2015. Association for Computing Machinery.

[Elg85] Taher Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FDS+17] Sergey Frolov, Fred Douglas, Will Scott, Allison McDonald, Benjamin VanderSloot, Rod Hynes, Adam Kruger, Michalis Kallitsis, David G. Robinson, Steve Schultze, Nikita Borisov, Alex Halderman, and Eric Wustrow. An ISP-Scale deployment of TapDance. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI 17)*, Vancouver, BC, August 2017. USENIX Association.

[FHE+12] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Dan Boneh, Roger Dingledine, and Phil Porras. Evading Censorship with Browser-Based Proxies. In Simone Fischer-Hübner and Matthew Wright, editors, *Privacy Enhancing Technologies*, pages 239–258, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Fif17] David Fifield. *Threat Modeling and Circumvention of Internet Censorship*. PhD thesis, University of California, Berkeley, December 2017.

[Fif21] David Fifield. dnstt – DoH- and DoT-capable DNS tunnel, 2021. [Online; accessed February 21, 2022] https://www.bamsoftware.com/software/dnstt/.

[FLH+15] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and V. Paxson. Blocking-resistant Communication Through Domain Fronting. *Proceedings on Privacy Enhancing Technologies*, 2015:46 – 64, 2015.

[FT16] David Fifield and Lynn Tsai. Censors' Delay in Blocking Circumvention Proxies. *arXiv: Cryptography and Security*, 2016.

[FTZ17] David Fifield, Lynn Tsai, and Qi Zhong. Detecting Censor Detection. *ArXiv*, abs/1709.08718, September 2017. [Online; accessed February 21, 2022] https://arxiv.org/abs/1709.08718.

[FW20] Sergey Frolov and Eric Wustrow. HTTPT: A Probe-Resistant proxy. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*. USENIX Association, August 2020.

[FWT+19] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. Conjure: Summoning proxies from unused address space. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 2215–2229, New York, NY, USA, 2019. Association for Computing Machinery.

[FWW20] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting Probe-resistant Proxies. In *Proceedings 2020 Network and Distributed System Security Symposium*, San Diego, CA, 2020. Internet Society.

[Gal19] Ryan Gallagher. A New App Allows Readers in China to Bypass Censorship of The Intercept, August 2019. [Online; accessed February 21, 2022] https://theintercept.com/2019/08/08/china-censorship-bypass-app/.

[HZCB17] Amir Houmansadr, Wenxuan Zhou, Matthew C. Caesar, and N. Borisov. SWEET: Serving the Web by Exploiting Email Tunnels. *IEEE/ACM Transactions on Networking*, 25(3):1517–1527, June 2017.

[IH18] Jim Isaak and Mina J. Hanna. User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection. *Computer*, 51(8):56–59, 2018.

[JVS19] Rob Jansen, Tavish Vaidya, and Michael E. Sherr. Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1823–1840, Santa Clara, CA, August 2019. USENIX Association.

[KJQ+21] Carmen Kwan, Paul Justin Janiszewski, Shela Qiu, Cathy Wang, and Cecylia Bocovich. Exploring Simple Detection Techniques for DNS-over-HTTPS Tunnels. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*, FOCI '21, pages 37–42, New York, NY, USA, 2021. Association for Computing Machinery.

[KMDA10] George Kadianakis, Nick Mathewson, Roger Dingledine, and Yawning Angel. Pluggable Transport Specification, Version 1, 2010.

[LdV17] Isis Agora Lovecruft and Henry de Valence. Hyphae: Social Secret Sharing, 2017. [Online; accessed February 21, 2022] https://patternsinthevoid.net/hyphae/.

[LdV20a] Isis Agora Lovecruft and Henry de Valence. *curve25519-dalek: A Pure-Rust Implementation of Group Operations on Ristretto and Curve25519*, 2020. [Online; accessed February 18, 2022] https://doc.dalek.rs/curve25519_dalek/index.html.

[LdV20b] Isis Agora Lovecruft and Henry de Valence. *ed25519-dalek: Fast and Efficient ed25519 Signing and Verification in Rust*, 2020. [Online; accessed February 18, 2022] https://doc.dalek.rs/ed25519_dalek/index.html.

[LdV20c] Isis Agora Lovecruft and Henry de Valence. *zkp: Experimental Zero-knowledge Proof Compiler in Rust Macros*, 2020. [Online; accessed February 18, 2022] https://doc.dalek.rs/zkp/index.html.

[Lew10] Andrew Lewman. China blocking Tor: Round Two | Tor Blog, March 2010. [Online; accessed February 18, 2022] https://blog.torproject.org/china-blocking-tor-round-two.

[LLY+15] Zhen Ling, Junzhou Luo, Wei Yu, Ming Yang, and Xinwen Fu. Tor Bridge Discovery: Extensive Analysis and Large-scale Empirical Evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):1887–1899, 2015.

[LMD10] Karsten Loesing, Steven J. Murdoch, and Roger Dingledine. A Case Study on Measuring Statistical Data in the Tor Anonymity Network. In *Proceedings of the Workshop on Ethics in Computer Security Research (WECSR 2010)*, LNCS. Springer, January 2010.

[MML11] Damon McCoy, Jose Andre Morales, and Kirill Levchenko. Proximax: A Measurement Based System for Proxies Dissemination. In *Financial Cryptography and Data Security*, page 9, 2011.

[MTC17] Srdjan Matic, C. Troncoso, and Juan Caballero. Dissecting Tor Bridges: A Security Evaluation of Their Private and Public Infrastructures. In *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA, 2017. Internet Society.

[NFHG19] Milad Nasr, Sadegh Farhang, Amir Houmansadr, and Jens Grossklags. Enemy At the Gateways: Censorship-Resilient Proxy Distribution Using Game Theory. In *In Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, California, 2019. The Internet Society.

[NS14] Daiyuu Nobori and Yasushi Shinjo. VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 229–241, Seattle, WA, April 2014. USENIX Association.

[NZH17] Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. The waterfall of liberty: Decoy routing circumvention that resists routing attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2037–2052, New York, NY, USA, 2017. Association for Computing Machinery.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[PFX+21] Ramakrishna Padmanabhan, Arturo Filastò, Maria Xynou, Ram Sundara Raman, Kennedy Middleton, Mingwei Zhang, Doug Madory, Molly Roberts, and Alberto Dainotti. A Multi-Perspective View of Internet Censorship in Myanmar. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*, FOCI '21, pages 27–36, New York, NY, USA, 2021. Association for Computing Machinery.

[SF21] Adrian Shahbaz and Allie Funk. Freedom on the Net 2021: The Global Drive to Control Big Tech, 2021. [Online; accessed February 18, 2022] https://freedomhouse.org/report/freedom-net/2021/global-drive-control-big-tech.

[Ste09] Douglas Stebila. *Classical Authenticated Key Exchange and Quantum Cryptography*. PhD thesis, University of Waterloo, March 2009.

[Tor22a] The Tor Project. All Tor Bridge Users 2016-2022, 2022. [Online; accessed March 7, 2022] https://metrics.torproject.org/userstats-bridge-country.html?start=2016-03-07&end=2022-03-07&country=all.

[Tor22b] The Tor Project. Bridge Connections to Tor in Myanmar January 1st – June 1st, 2021, 2022. [Online; accessed March 7, 2022] https://metrics.torproject.org/userstats-bridge-country.html?start=2021-01-01&end=2021-06-01&country=mm.

[Tor22c] The Tor Project. Bridgedb, 2022. [Online; accessed February 18, 2022] https://bridges.torproject.org/.

[Tor22d] The Tor Project. Collector: Bridge pool assignments, 2022. [Online; accessed February 18, 2022] https://collector.torproject.org/archive/bridge-pool-assignments/.

[Tor22e] The Tor Project. Onionoo, 2022. [Online; accessed February 18, 2022] https://onionoo.torproject.org.

[Tor22f] The Tor Project. Relay Connections to Tor in Myanmar January 1st – June 1st, 2021, 2022. [Online; accessed March 7, 2022] https://metrics.torproject.org/userstats-relay-country.html?start=2021-01-01&end=2021-06-01&country=mm&events=points.

[Tor22g] The Tor Project. Servers — Tor Metrics. https://metrics.torproject.org/bridges-ipv6.html?start=2016-01-23&end=2022-03-23, 2022. [Online; accessed March 23, 2022].

[Tor22h] The Tor Project. Users — Tor Metrics. https://metrics.torproject.org/userstats-bridge-country.html?start=2021-09-01&end=2022-02-27&country=ru, 2022. [Online; accessed February 27, 2022].

[VFW+20] Benjamin VanderSloot, Sergey Frolov, Jack Wampler, Sze Chuen Tan, I. C. Simpson, Michalis Kallitsis, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. Running Refraction Networking for Real. *Proceedings on Privacy Enhancing Technologies*, 2020:321 – 335, 2020.

[WL12] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*, Bellevue, WA, August 2012. USENIX Association.

[WLBH13] Qiyan Wang, Zi Lin, Nikita Borisov, and Nicholas J. Hopper. rBridge: User Reputation Based Tor Bridge Distribution with Privacy Preservation. In *In Proceedings 2013 Network and Distributed System Security Symposium*, San Diego, California, 2013. The Internet Society.

[WPF13] Philipp Winter, T. Pulls, and Jürgen Fuß. ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship. *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, 2013.

[XF21] Maria Xynou and Arturo Filastó. Russia started blocking Tor, December 2021. [Online; accessed February 18, 2022] https://ooni.org/post/2021-russia-blocks-tor/.