PRIVACY PRESERVING SYSTEMS WITH CROWD BLENDING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Mohsen Minaei

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Aniket P. Kate, Chair

School of Computer Science

Dr. Mikhail J. Atallah

School of Computer Science

Dr. Bruno Ribeiro

School of Computer Science

Dr. Ninghui Li

School of Computer Science

Approved by:

Dr. Kihong Park

Head of the Departmental Graduate Program

To my beloved wife, who stood by me through thick and thin, and to my loving parents, whose sacrifices and selfless support made everything possible.

ACKNOWLEDGMENTS

It gives me great pleasure to express my gratitude to a large number of people who contributed to my success over the years of my Ph.D. journey.

My most sincere gratitude goes to my advisor, Professor Aniket Kate, for his genuine support, unconditional care, and invaluable guidance. It is my absolute honor and pleasure to be given a chance to work with such an inspiring and productive mentor and learn to become a better independent researcher. This dissertation would not have been possible without his motivation and help. Thank you.

I would like to thank my committee members, Professor Mikhail Atallah, Professor Bruno Ribeiro, and Professor Ninghui Li for their valuable suggestions and advice.

This dissertation would not have been possible without my collaborators. I would like to thank Mainack Mondal, Chandra Mouli, Duc Le, Pedro Moreno-Sanchez, Tiantian Gong, Bruno Ribeiro, Patrick Loiseau, and Krishna Gummadi.

A special thanks go to Dr. Mainack Mondal, who was along my side in this journey and helped with most of the projects in this thesis. He is not only a great collaborator but also a tremendous friend. He gave me an enormous amount of encouragement and advice through the tough times, and thanks for all of it.

I am grateful to have spent time with all my past and present friends at Freedom Lab, including Dr. Pedro Moreno-Sanchez, Dr. Sze Yiu Chau, Duc Le, Easwar Mangipudi, Debajyoti Das, Donghang Lu, Adithya Bhat, and Tiantian Gong. They provided a productive and friendly environment to study and work. I'm surely going to miss our endless debates and discussions.

Last but not least, I would like to express my most sincere appreciation to my wife, my sisters, my parents, and my dear friends for their emotional support. I would not be the same person without them. Above all, I thank God the Almighty for blessing me with the strength, health, and will to prevail and finish this work.

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Minaei, Mohsen Ph.D., Purdue University, December 2020. Privacy Preserving Systems with Crowd Blending. Major Professor: Aniket Kate Professor.

Over the years, the Internet has become a platform where individuals share their thoughts and personal information. In some cases, these content contain some damaging or sensitive information, which a malicious data collector can leverage to exploit the individual. Nevertheless, what people consider to be sensitive is a relative matter: it not only varies from one person to another but also changes through time. Therefore, it is hard to identify what content is considered sensitive or damaging, from the viewpoint of a malicious entity that does not target specific individuals, rather scavenges the data-sharing platforms to identify sensitive information as a whole. However, the actions that users take to change their privacy preferences or hide their information assists these malicious entities in discovering the sensitive content.

This thesis offers *Crowd Blending* techniques to create privacy-preserving systems while maintaining platform utility. In particular, we focus on two privacy tasks for two different data-sharing platforms— i) concealing content deletion on social media platforms and ii) concealing censored information in cryptocurrency blockchains. For the concealment of the content deletion problem, first, we survey the users of social platforms to understand their deletion privacy expectations. Second, based on the users' needs, we propose two new privacy-preserving deletion mechanisms for the next generation of social platforms. Finally, we compare the effectiveness and usefulness of the proposed mechanisms with the current deployed ones through a user study survey. For the second problem of concealing censored information in cryptocurrencies, we present a provably secure stenography scheme using cryptocurrencies. We show the possibility of hiding censored information among transactions of cryptocurrencies.

1. INTRODUCTION

In recent years, with the emergence of data-sharing and social platforms, the problem of data privacy has become increasingly prominent as the content and activities of individuals are traceable and accessible worldwide. To cope with such privacy concerns, many regulations within different jurisdictions have been introduced; Yet, the long-term exposure of data still raises numerous longitudinal privacy concerns for users.

The boundaries of what is considered private differ among societies and individuals and mutate with time. Therefore, from the perspective of a global observer that does not focus on specific users, it is difficult to pinpoint sensitive information. However, users' actions taken to hide such information can help the observer in identifying sensitive content. A closely associated phenomenon is called the "Streisand Effect" [1], which suggests that any attempt to hide certain information has the unintended consequence of drawing public attention to it.

Consequently, as long as the content and actions of an individual are blended with sufficient similar content and actions, an outsider observer will not be able to pinpoint the sensitive content of a user. A similar concept exists in the survivalist community known as the Gray Man Theory [2,3], which elaborates on techniques of disappearing into the crowd and being able to move unnoticed when disaster strikes.[1] Similar techniques can be used in data-sharing platforms to protect the privacy of users in the presence of a persistent observer that monitors the platforms in large-scale, not particularly devoted to specific users.

In this thesis, we use crowd blending techniques to create privacy-preserving systems by hiding private and sensitive data (activity) of a user among similar data

[1]The same tactic is used in nature by many plants and animals to avoid predators or sneak up on prey.

(activity) of other users within the system. We study this technique in two parts: i) concealment of deletions within social media platforms and ii) concealment of censored information in cryptocurrency blockchains. Although the two fields seem to differ significantly, we show that similar techniques can be leveraged to create privacy-preserving systems.

1.1 Concealing Content Deletion from Persistent Observers

People freely open up about their personal life and opinions on online social platforms (e.g., Facebook, Twitter). The shared information remains available (to intended recipients as well as unintended observers) and is archived by archival services until (and if) the information is eventually deleted (or confined) by its creator. This long-term exposure of shared data raises numerous longitudinal privacy concerns [4–6]. Both celebrities and non-celebrities are regularly harassed and blackmailed by data scavengers. These scavengers stalk their victims to identify and abuse sensitive content from the shared data. Nevertheless, the sensitivity of a post is relative; it varies from person to person and changes through time, based on life events.

Thus, effective (high precision and recall) mining of available large-scale data to find suitable victims is not always feasible for scavengers. The task should have become more difficult as platforms and Internet archives honor users' requests to delete their data. However, these deletions leave the users more vulnerable to the scavengers who can now focus only on the withdrawn posts to find sensitive content. We find this problem associated with content deletions to be very feasible—today multiple web services find and hoard deleted content across different social platforms. Politwoops [7] for Twitter, ReSavr [8] and Uneddit [9] for Reddit, StackPrinter-Deleted [10] for Stack Overflow, and YouTomb [11] for Youtube are some of the prominent examples. These services enable attackers to specifically mine deleted posts of users for nefarious purposes.

In this thesis, we study the full-scale problem of deletion privacy within social and archival platforms. First, we perform a qualitative and quantitative study on the users of social media platforms to investigate their deletion experiences and privacy expectations. Next, we introduce two new deletion mechanisms that aim to provide privacy for damaging and sensitive deletions. Finally, we analyze factors that govern the effectiveness and usefulness of the introduced and existing deletion mechanisms.

1.2 Censorship Resistant Rendezvous using Cryptocurrencies

One of the most ubiquitous and challenging problems faced by the Internet today is the restrictions imposed on its free use. Repressive and totalitarian governments censor the Internet content to their citizens. Censors employ several techniques ranging from IP address filtering to deep-packet inspection to block disfavored Internet content [12]. Censored users are thereby prevented from accessing information on the Internet and expressing their views freely. Given that, several circumvention systems have been proposed over the last decade [13]. Nevertheless, censorship remains a challenge to be fully resolved.

Nowadays, Bitcoin [14] has gained a worldwide presence. This presence is also prevalent in countries with large-scale censorship, such as China [15]. The same holds for other cryptocurrencies focused on smart contracts as in Ethereum [16] or privacy-preserving coin transfers as in Zcash [17] and Monero [18].

The availability of cryptocurrencies across different geopolitical contexts makes them a suitably distributed rendezvous to post steganographic messages. In fact, censored users can leverage from their highly cryptographic structure to encode censored data while maintaining undetectability. To that end, in the presence of a global censor that observes all network communications, we use cryptocurrencies as a censorship circumvention rendezvous. More specifically, we study the feasibility of blending the steganographic messages among the normal daily transactions of users within different cryptocurrencies.

In summary, this thesis focuses on demonstrating the following statement:

> *In the presence of a global adversary that observes and scavenges data-sharing platforms for sensitive content, it is possible to create a privacy-preserving system that blends sensitive content (activity) of a user with similar content (activities) of other users within the system.*

1.3 Contributions

The technical contributions of this thesis can be broadly partitioned as followed.

i. Unveiling the users' perceptions of deletion privacy in social platforms. For the first time, we investigate the systematic access rules to regulate the discoverability of deletion events. This work takes the first step towards understanding and operationalizing user perceptions of deletion privacy. In particular, quantify the need for deletion privacy in social platforms with a 191 participant user survey. We establish a strong user-need for ensuring deletion privacy in social platforms. Our results show that users indeed care for deletion privacy. Further, we demonstrate the context-dependency of the rules for preserving deletion privacy. We identify key contextual factors that future developers should consider to better align their system functionalities with user expectations.

ii. Proposal for the next generation of social and archival platforms to enhance deletion privacy and evaluating their effectiveness.

We present two new deletion mechanisms that can be adopted by the next generation of social platforms. First, we propose *Lethe* [19], a novel solution to this problem of content deletion in the presence of a persistent observer. *Lethe* employs an intermittent withdrawal mechanism that protects privacy by toggling the observable state of the posts between up (or visible) and down (or hidden) states. As *Lethe* is applied to all the available (non-deleted) posts, the adversary observing a post in a down

(hidden) phase, cannot immediately discern whether the post is hidden by *Lethe* or deleted by the user.

Next, we propose Deceptive Deletions [20], which raises the bar for the adversary in identifying damaging content using a special deception technique. Given a set of damaging posts (posts that adversary can leverage to blackmail the user) that users want to delete, the Deceptive Deletion system (also known as a challenger) *selects* k additional posts for each damaging post and deletes them along with the damaging posts. The system-selected posts, henceforth called the *decoy posts*, are taken from a pool of non-damaging non-deleted posts provided by volunteers. Since a global adversary can only observe all of these deletions together, his goal is to distinguish deleted damaging posts from the deleted (non-damaging) decoy posts. Intuitively, Deceptive Deletion is more effective if the selected decoy posts are similar to the damaging posts. These two opposite goals create a minmax game between the adversary and the challenger that we further analyze.

Finally, by conducting a survey from 158 participants, we compare the currently deployed deletion mechanisms with our proposed mechanisms based on their effectiveness in preserving deletion privacy. Furthermore, we highlight the key factors that lead to the usefulness of these mechanisms.

iii. Introducing cryptocurrencies as a new medium for bootstrapping the censorship resistance proxies. We present *MoneyMorph* [21], a secure and privacy-preserving censorship-resistance bootstrapping scheme using cryptocurrencies. *MoneyMorph* enables an entity residing outside of the censored region to transmit bootstrapping credentials of an entry point for a censorship-circumvention protocol (e.g., Tor Bridge [22]) to the censored user in the presence of the censor. We describe how *MoneyMorph* works using Bitcoin, Zcash, Monero, and Ethereum as rendezvous. We carry out a comparative study of the different rendezvous by thoroughly evaluating their tradeoffs in terms of available bandwidth, monetary costs, and percentage of sibling transactions to blend in the bootstrapping transactions.

1.4 Outline of the Thesis

This dissertation is organized in two parts. Part I includes how crowd-blending can be used to conceal content deletion from persistent observers and comprises Chapter 2-5. In particular, in Chapter 2 we unpack the users' perceptions of deletion privacy in social platforms. In chapters 3 and 4, we propose two new deletion mechanisms namely Lethe and Deceptive Deletions to protect the users' damaging and sensitive deletions. In chapter 5, the last chapter of this part, we compare the different deletion mechanisms based on their effectiveness and usefulness. Part II focuses on how we can use crowd-blending techniques for bootstrapping censorship resistance tools using cryptocurrencies as rendezvous which consists of chapter 6. Finally, we summarize this dissertation in Chapter 7.

Part I

Concealing Content Deletion from Persistent Observers

People freely open up about their personal life and opinions on online social platforms (e.g., Facebook, Twitter) today. The shared information remains available on these platforms (to intended recipients as well as unintended observers) and is archived by archival services until (and if) the information is eventually deleted (or confined) by its creator. This long-term exposure of the shared data raises numerous longitudinal privacy concerns [4–6] for the users: not only celebrities but non-celebrities get regularly harassed and blackmailed by data scavengers, who stalk their victims to identify sensitive content from the shared data. Nevertheless, the sensitivity of a post is relative; it varies from person to person, and also with life events and time in general. Thus, effective (high precision and recall) mining of available large-scale data to find suitable victims are not always feasible for scavengers.

The task should have become more difficult as platforms and Internet archives honor users' requests to delete their data. However, these deletions in fact leave the users more vulnerable to the scavengers who can now focus only on the withdrawn posts to find sensitive content.[2] Indeed, we found this problem associated with content deletions to be very practical—today multiple web services find and hoard deleted content across different social platforms. Politwoops [7] for Twitter, ReSavr [8] and Removeddit [23] for Reddit, StackPrinter-Deleted [10] for Stack overflow, and YouTomb [11] for Youtube are some of the prominent examples. A malicious data-collector can simply leverage these notifications to flag deleted posts as possibly *damaging* and further use them against the users [24–26]. Importantly, the hand-picked politicians and celebrities are *not* the only parties at the receiving end of these attacks. We find that malicious data-collector can develop learning models to automate the process and perform a non-targeted (or global) attack at a large-scale; e.g., Fallait Pas Supprimer [27] (i.e., "Should Not Delete" in English) is a Twitter account that collects and publishes the deleted tweets of not only the French politicians and celebrities but also noncelebrity French users with less than a thousand followers.

[2]Closely associated phenomenon, "Streisand effect," suggests that any attempt to hide some information has the unintended consequence of bringing particular attention of public to it.

In this part of the thesis, we do a full-scale study on the problem of deletion privacy. We first investigated the prior experiences of the participants regarding their post deletions and corresponding deletion privacy expectations. We then leverage the contextual integrity theory [28] to identify key contextual factors (as perceived by users) for regulating access and ensuring the preservation of deletion privacy. Next, we introduce two new deletion mechanisms that aim to provide privacy for the damaging and sensitive deletions of the users. Finally, we unearth the factors governing the usefulness of the introduced and already existing deletion privacy preservation mechanisms.

2. USERS' PERCEPTIONS OF DELETION PRIVACY

Preserving deletion privacy involves designing and enforcing access control rules to regulate when and how information about the deletion events (and deleted content) is revealed to others, and that earlier research did not investigate systematic access rules to regulate the discoverability of deletion events. In general, there is no prior work on understanding the *need* for providing deletion privacy to general social media users. In other words, there was no evidence quantifying the importance of preserving deletion privacy for social platform users. This works takes the first step towards understanding and operationalizing user perceptions of deletion privacy.

In our study, we collected quantitative and qualitative data from 191 participants spanning both Europe and the US regarding their perceptions about deletion privacy. We first investigated the prior experiences of the participants regarding their post deletions and corresponding deletion privacy expectations. We then leverage the contextual integrity theory [28] to identify key contextual factors (as perceived by users) for regulating access and ensuring the preservation of deletion privacy. Specifically, we investigate the following research questions (RQ).

RQ1: *Have users faced violation of deletion privacy in social platforms? In other words, did some other users or organizations focused on their deleted social posts? How?* (Section 2.3.1)

We investigated this RQ by asking each participant detailed questions regarding their experience about deletions on the social platforms. We note that 82% of our participants have deleted some of their posts. Interestingly, 51% of the participants felt that a deleted post is indeed *sensitive, damaging or embarrassing* to its owner. Furthermore, 54 participants had their deletions noticed in social platforms, and even

within our small sample of fewer than 200 participants, nine participants proclaimed that when their deletions were noticed by others, it resulted in discomfort.

While establishing the need for deletion privacy was a prime goal of this work, a social platform would also need to know if its users feel (un)comfortable in revealing their deletions in certain contextual factors. We explore this question next.

RQ2: *On what contextual factors (such as recipient) do policies regarding acceptability of revealing deletion events depend on? How?* (Section 2.3.2)

We used the contextual integrity theory [28] to create a set of contextual variables (e.g., recipients) and enumerated possible values for each set of variables (e.g., family member, friend, coworker, a company, government). We then collected user feedback for combinations of all of those contextual variables in our survey. We observe that majority of the users seek to preserve deletion privacy against large-scale data collectors (e.g., corporations and government), but not so much against their family, friends, and even co-workers.

In summary, we begin to quantify the need for deletion privacy in social platforms with a 191 participant user survey. Our contributions include:

1. Establishing a strong user-need for ensuring deletion privacy in social platforms. Our results show for the first time that users indeed care for deletion privacy.

2. Showing the context-dependency of the rules for preserving deletion privacy. We identify the key contextual factors that future developers should consider to better align their system functionalities with user expectations.

2.1 Background and Related Work

In this section, we place our work in the context of related research on different deletion techniques and contextual integrity.

2.1.1 Deletion in Social Platforms

Deletion or the ability to remove content is a crucial functionality in social platforms—often needed due to social nature of these platforms as well as personal nature of social content. Earlier work studied in detail the reasons behind social content deletion. These reasons range from removing regrettable content to removing content which became irrelevant over time [6, 29–33]. However, even though deletion is crucial and widely adopted, in some cases, the removal of a post can be a simple and very effective indication about the sensitive and/or damaging nature of that social content [19, 20]. Thus simple removal might create an opportunity for an attacker to potentially harass and blackmail the users. Such deletion based surveillance is not only relevant to public figures, but also for normal social platform users, e.g., a French Twitter account, @FallaitPasSuppr, identify and re-publishes the deleted content of both French public figures as well as normal users [27]. However, no earlier work investigated if this shortcoming of social content deletion is truly affecting the general populace. We fill this gap.

2.1.2 Deletion Privacy as Contextual Integrity

Contextual Integrity (CI) [28] theory provides a systematic framework for studying privacy norms and expectations. CI defines privacy as appropriate flows of information. Each information flow consists of five parameters about the information: subject, sender, recipient, information type (or attribute), and transmission principle. The appropriate information flows conform to the socially acceptable values of these parameters. Earlier work demonstrated that we can infer privacy norms (i.e., rules regulating acceptable information flow) by measuring the acceptability of different information flows (created with varying combination of CI parameter values) [34, 35]. For example, users in general might be comfortable when a fitness tracker (*sender*) sends user's heart rate (*attribute*) to the doctor (*recipient*) to monitor the health sta-

tus (*transmission principle*), but uncomfortable if the *recipient* is a health insurance provider.

In this section, we aim to discover the effect of context on the acceptability of deletion events getting noticed (RQ2). Thus, we leveraged CI to systematically unroll the contextual factors in the scope of deletion privacy. We selected the CI parameter values relevant to deletion privacy by surveying earlier work and conducting pilot studies. Table 2.1 contains the full list of our CI parameter values. Note that, this list is not exhaustive. However, as a first, it does cover a range of information flows in the scope of deletion privacy and demonstrates the generality of our approach. Section 2.2.1 details the exact questions asked in our survey. Next we present our CI parameter values.

Sender & Subjects for each deletion, the sender of the information flow will be the user itself. However, a deleted post can have different subjects. We consider prior work on ego networks and social circles to design four distinct subjects [36, 37]—(i) the user, (ii) family members, (iii) friends, (iv) coworkers/acquaintance. We also included "not specifying a subject" as *null* i.e., control condition.

Recipients for social content deletion is the individual (or organization) that notices the user's deletion. We include users' social circles in our list recipients along with two other entities–a company that collects and archives the deletions of the users and the government.

Transmission Principles in this scenario is the method that a recipient uses to discover the deletion. We consider three discovery methods—(i) discovery due to checking/observing the user profile regularly to observe any change in the user's profile (ii) discovery due to an interaction with the post (e.g., liking, commenting, reposting, sharing, etc.), (iii) not specifying a discovery method (*null* i.e., control condition).

Attributes, we consider the reason of the deletion to be the attribute in the information flows. We adapt the categories defined by Zhou et al. [38] for the regrettable deleted tweets as attributes. We further add "fixing spelling/grammar" from earlier

Table 2.1.: Contextual integrity (CI) parameter values used to generate information flow of deletion events

Sender	**Transmission Principle**
user itself	because they were checking/observing your user profile regularly
Subject	because they were mentioned in the post or interacted with the post
that contained some information about yourself	*null*
that contained some information about your family members	**Attributes**
that contained some information about your friends	Post did not get enough attention
that contained some information about your coworkers	Fixing Spelling/Grammar
null	Cleaning up profile for new job
Recipient	Cleaning up profile for new relationship
your family member	Racial/Religious/Political reason
your friend	Being irrelevant due to time passing
your coworker/acquaintance	Removing sexual content
a company	Removing drug/alcohol related content
the government	Removing violence/cursing related content
anyone	Removing health related content

work [39] as well as two other reasons—"post did not get enough attention" and "being irrelevant due to time passing" based on our study pilot. We obtained feedback on acceptability for each of the information flows generated using the combination of all of these CI parameter values.

2.2 Methodology

In this section, we discuss the design and methodology of our study in detail. We begin with our survey instrument that paves the path for understanding the need for deletion privacy, unrolling the deletion privacy norms using Contextual Integrity (CI), and evaluating the effectiveness of deletion mechanisms in providing privacy to the deletions.

2.2.1 Survey Instrument

Our survey contained two sections: (1) Experiences about prior post deletions, (2) CI-parameter based questionnaire about deletion privacy. Our full survey instrument can be accessed at `https://tinyurl.com/y4zceoma`.

Experiences about prior post deletions (RQ1). We started by asking participants about the usage of different social platforms and whether they have ever deleted

any of their content. We further asked how old was the content at the time of deletion as well as the reasons behind their deletions. Next, we asked the participants to explain whether they have ever faced any discomfort because of their deletions and what problems do they anticipate in the future for their possible deletions. We then inquired if the participants are aware of other users noticing their deletions and whether they have noticed other users' deletions. Lastly, we investigated how the users feel about the sensitivity of deleted content by asking them whether they agree or disagree with the statement—"when someone deletes a social media post, it indicates that the content of that post is sensitive/damaging/embarrassing to that individual."

CI-based questionnaire: deletion privacy (RQ2). Taking inspiration from earlier research [34,35], we adapted a CI-based questionnaire to investigate the users' expectations of deletion privacy in social platforms. We detail our adaptation of CI to the scope of deletion privacy in Section 2.1.2. Here, we will focus on the setup of our survey.

Recall that we needed to obtain users' perception of acceptability for the information flows created by all combinations of the parameter values presented in Table 2.1. In total, we have 900 distinct information flows (5 *subjects* $\times$ 3 *transmission principles* $\times$ 6 *recipients* $\times$ 10 *attributes*), and asking each participant to evaluate all the flows is infeasible. Therefore, we divided the flows into 30 blocks with 30 information flows each[1].

To randomly assign participants to one of these blocks, each participant was randomly assigned to a **fixed** value for the *subject* and *transmission principle* variable. That participant was also randomly assigned to one of the two pre-defined sets of *recipient* variable values[2] (each set contained three *recipient* values).

[1]Each block was assigned to at least 6 participants.

[2]First set of *recipient* variable values or recipient_A: [your family member, your close friend, your coworker]. Second set of *recipient* variable values or recipient_B:[anyone, a company, the government]

As a result, in each block, we repeated the below matrix question three times by replacing the *recipient* variable with the values from the assigned recipient set (recipient_A or recipient_B), but keeping the same *subject* and *transmission principle* values each time. The rows of this matrix question represent the *attribute* variable values (an *attribute* value represents the reason behind a deletion). Therefore, each row of a question signifies one of the information flows, which the participants were asked to rate its acceptability, using a five-point Likert scale: Completely Acceptable, Somewhat Acceptable, Neutral, Somewhat Unacceptable, Completely Unacceptable.

CI-Q: *"We are putting a few possible reasons behind post deletions in the table below (leftmost column). Imagine a situation where you deleted a post [subject] from one of your social media accounts due to that reason. In each of these situations, please indicate how acceptable is it for you that [recipient] notices your deletion [transmission principle]?"*

Figure 2.1 presents part of a question block. This example belongs to a block with the subject "your coworkers" and transmission principle "because they were checking/observing your user profile regularly". Then in the presented question matrix we set the recipient to "anyone" and iterated through all the attributes to create final information flows.

Quality control. To ensure the quality of responses, we incorporated multiple attention check questions in the survey. In particular, we repeated two of the multiple-choice questions in random locations in the survey and compared the answers with their previous responses to the same question. Moreover, we added a fake social platform named "Cybersocial" in questions that asked about their usage of social platforms, to monitor whether they indicate using this platform or not. Further, we used time-based filtering to ensure that participants gave attention while watching the videos in our survey.

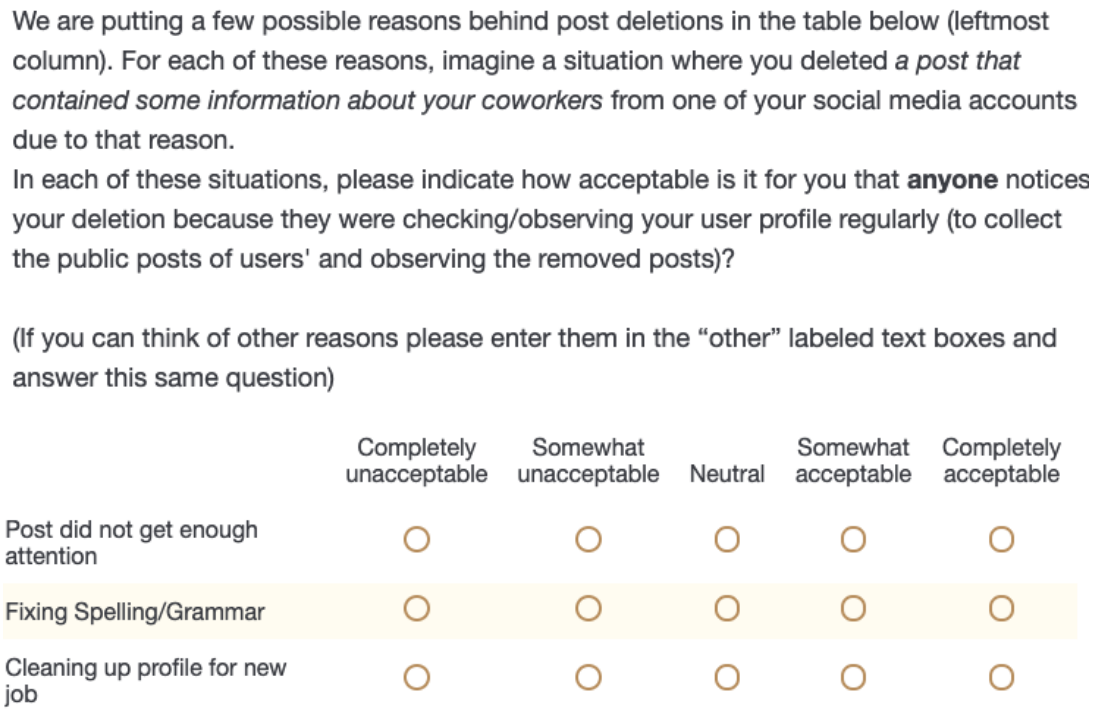


Fig. 2.1.: Measuring acceptability of information flow with a fixed subject (your coworkers), fixed transmission principle (because they were checking/observing your user profile regularly), fixed recipient (anyone) and varying attributes (not all are shown).

2.2.2 Pilot Studies

Prior to the deployment of the survey, we conducted two pilot studies to evaluate the procedure of the study, determine the average duration, and test the comprehensibility of the questions.

In the first pilot, we tested the study on ten colleagues (without prior knowledge of the study and its goals) from different departments in our university. As a result, we removed four questions from the questionnaire as they were somewhat redundant or too imprecise. Moreover, some of the choices in the questions were modified/added to eliminate any invasiveness and confusion. For example, in one of the questions, we had "stalking a profile" as a method of noticing other people's deletions which the word stalking seemed to make the participants uncomfortable. We ultimately

changed the choice to "checking the individual's profile regularly". Further, we observed that in some questions, a significant number of participants entered the same answer in the "other" choice box (free-text boxes). Therefore, we added them to the predefined choices in the final version. For example, we added "Post not getting enough attention" and "Irrelevant due to time passing" as choices for deleting a post.

After applying the changes above, in the second stage of the pilot, we deployed the survey on the Prolific Academic [40] and recruited ten participants. The results from the qualitative responses showed that the changes made from the first pilot were effective and participants had a good understanding of the questions. This point was also confirmed by the responses that the participants gave to our cognitive question at the end of the survey—"Did you find the questions in this survey to be understandable?". Eight of the participants responded with "completely understandable" and "mostly understandable." The remaining two responded with "neutral" and "mostly not understandable", however, without any feedback on which sections they had difficulty in understanding. Further, we asked—"How fair do you find the compensation of the survey, compared to the amount of time you took for completion?". Nine participants responded with "very fair" and "fair", giving us confidence in a fair payment.

2.2.3 Recruitment

We recruited our participants from Prolific Academic [40], a platform regularly used for advertising academic surveys [41]. We screened participants to ensure they were 18 years old or above, had not taken our pilot study, had taken a minimum of 50 prior surveys on the platform, had a minimum approval rate of 95%, fluency in English, and having a social media account currently or in the past.[3]. While designing our survey instruments we strongly aimed to minimize bias (i.e., leading or priming) and ambiguity. We did not screen our participants based on deletion behavior and

[3]These settings produced high-quality pre-study responses.

designed our recruitment text and strategy accordingly. We carefully avoided priming participants by not using words like "security" or "privacy" in our study.

The survey was advertised as "A study about social media usage and post deletions", and deployed in same sized batches (i.e., 20 participants at a time) over a one-week period, at different times of the day. We did this to counter anomalous time dependency in our results due to the effect of events happening at a specific time [42]. The average time for completion of the survey was 12.5 minutes and compensation was $1.5. In total we obtained 205 responses (103 from the US and 102 from Europe).

2.2.4 Participant Demographics

A total of 205 participants completed the survey. We discarded the responses that did not pass the validity checks (see Section 2.2.1) and we were left with 191 (93 from the US and 98 from Europe).

Our population sample was nearly gender-balanced; 50.8% identified as female, 47.6% as male, and 1.1% as other. The sample skewed young, with 26.7% between 18 and 24, 39.3% between 25 and 34, 20.4% between 35 and 44, and 13.6% age 45 or older. Our participants were slightly more educated than the general U.S. population [43], where 55% of the participants either had a bachelor or a graduate degree. The median annual household income of the participants was $40,000 - $59,999, where the majority had an income of $20,000 - $39,999 (23%). Despite the fact that participants in crowdsourcing platforms (e.g., Amazon Mechanical Turk and Prolific) are considered to be tech-savvy [44], 67% of our participants reported that they do not have any background (e.g., study, work, etc.) experience in the IT field. We present the detailed demographics of our population in Table 2.2.

The usage pattern of different social platforms by the participants are shown in Figure 2.2. Our participants are active users of popular social media platforms. We note that 74.9% of the participants reported the usage of at least one social

Table 2.2.: Demographics

	US # (%)	**Europe** # (%)
Gender		
Female	48 (52%)	49 (50%)
Male	42 (46%)	49 (50%)
Other	2 (2%)	—
Age		
18 - 24	29 (31%)	22 (22%)
25 - 34	36 (39%)	39 (40%)
35 - 44	18 (19%)	21 (21%)
45 - 54	4 (4%)	12 (12%)
55 - 64	4 (4%)	3 (3%)
65 - 74	1 (1%)	1 (1%)
Native Language		
English		
Other languages		
Ethnicity		
White or Caucasian	59 (64%)	83 (85%)
Hispanic or Latino	13 (14%)	1 (1%)
Black	9 (10%)	3 (3%)
Asian	9 (10%)	6 (6%)
Multiple races	2 (2%)	2 (4%)
Prefer not to answer	—	1 (1%)
Education		
Bachelor degree	35 (38%)	34 (35%)
Some college	20 (22%)	22 (22%)
Graduate degree	15 (16%)	20 (20%)
High school degree	11 (12%)	15 (15%)
Associate degree	10 (11%)	4 (4%)
Less than high school	1 (1%)	2 (2%)
Prefer not to answer	—	1 (1%)

	US # (%)	**Europe** # (%)
Marital Status		
Single, never married	55 (60%)	49 (50%)
Married/domestic partner	31 (34%)	44 (45%)
Divorced	4 (4%)	4 (4%)
Separated	2 (2%)	—
Prefer not to answer	—	1 (1%)
Employment		
Full-time employment	40 (43%)	52 (53%)
Part-time employed	17 (18%)	13 (13%)
Unemployed	14 (15%)	6 (6%)
Student	11 (12%)	17 (17%)
Other	4 (4%)	2 (2%)
Full-time uncompensated	4 (4%)	5 (5%)
Retired	2 (2%)	2 (2%)
Prefer not to answer	—	1 (1%)
Income		
$0 - $19,999	9 (10%)	22 (22%)
$20,000 - $39,999	18 (20%)	26 (26%)
$40,000 - $59,999	14 (15%)	16 (16%)
$60,000 - $79,999	15 (16%)	13 (13%)
$80,000 - $99,999	9 (10%)	7 (7%)
$100,000 or more	23 (25%)	5 (5%)
Prefer not to answer	4 (4%)	9 (9%)
Background in IT		
Yes	29 (32%)	31 (31%)
No	62 (67%)	66 (67%)
Prefer not to answer	1 (1%)	1 (1%)

platform daily and 91.1% use a platform at least once a week, showing the suitability of the participants for this study. Facebook, Youtube, Instagram, WhatsApp, and Twitter were the most frequently used social platforms in our population.

2.2.5 Analysis Method

Coding Free Text Answers. We coded free text answers obtained from our survey to uncover users' perceptions. In our analysis, two researchers independently coded free-text responses using a shared codebook. Across questions, Cohen's κ (inter-rater agreement [45]) ranged from 0.7 to 1, indicating substantial to perfect agreement. The coders met to resolve disagreements and choose a final code.

Statistical Analysis. We leveraged statistical hypothesis testing to investigate significant deletion privacy norms. Specifically, for such analysis, we converted five-point

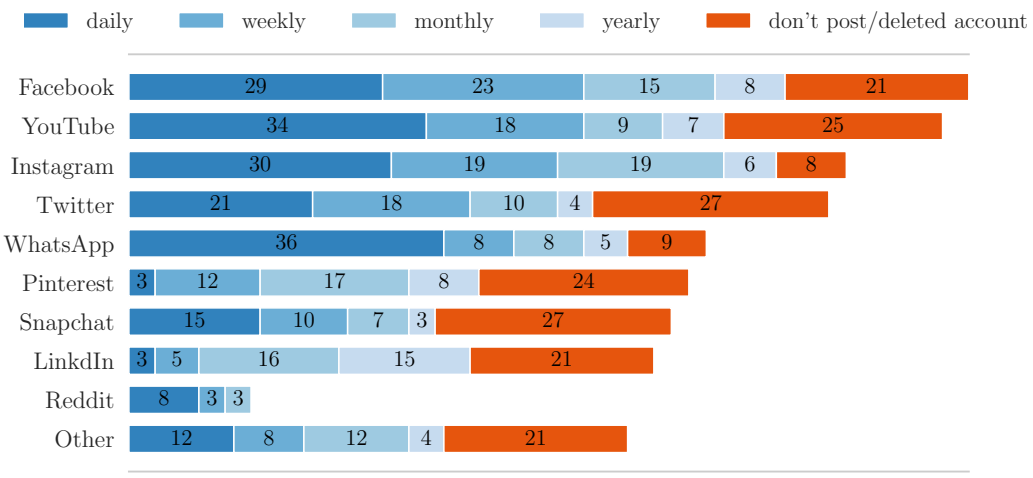


Fig. 2.2.: The usage pattern (in %) of different social platforms by the participants.

Likert scales to the ordinal variable as follows: Completely Acceptable (2), Somewhat Acceptable (1), Neutral (0), Somewhat Unacceptable (-1), Completely Unacceptable (-2). Unless otherwise stated, we used the nonparametric Mann Whitney U test to compare the responses across different groups. For all tests, the level of significance (α) was 0.05 and further adjusted using Bonferroni multiple-testing correction.

2.2.6 Ethical Considerations

We have taken great care to adhere to principles of ethical research. In the recruitment process, each participant was informed the purpose of the study, that they can withdraw at any time without giving any reasons, and that we would not store any personally identifying information (PII). We also informed the participants about the estimated duration of the study and their compensation in our consent form. Respondents who did not consent were not allowed to proceed with the study. Our study protocol was thoroughly examined and approved by the lead author's Institutional Review Board (IRB).

2.2.7 Limitations

We have planned and conducted our study thoroughly. However, our sampling approach introduces certain limitations. We used the Prolific Academic to recruit our participants, which might have resulted in younger and more tech-savvy users. Moreover, as our survey and videos were in English, we required the participants to be fluent in English, which could have created a language as well as cultural bias. However, since the majority of popular social platforms today have a bias towards younger English-speaking users, we strongly believe our study still captures the perceptions of a very important part of the population. Future research could validate and extend our findings to a more diverse sample.

2.3 Results

In this section, we present our findings of the deletion privacy exploration. We begin with the users' past deletion experiences.

2.3.1 Users' Perception of Content Deletion (RQ1)

Following the survey instrument in Section 2.2.1 we present our key observations from the participants responses.

Many users delete their outdated posts. Among the 191 participants, a significant majority of 82% reported that they have deleted a post(s) in the past. 78% report that they have deleted a post(s) from Facebook, 46% from Instagram, and 34% from Twitter. These numbers match with earlier work on social content deletion [6, 46].

We further asked the participants, who have deleted posts in the past, to indicate how frequently they have deleted them in different periods after their publication (i.e., the percentages of deletions made in different periods). The frequency results are shown in Figure 2.3. We note that the after-the-fact responses may have resulted

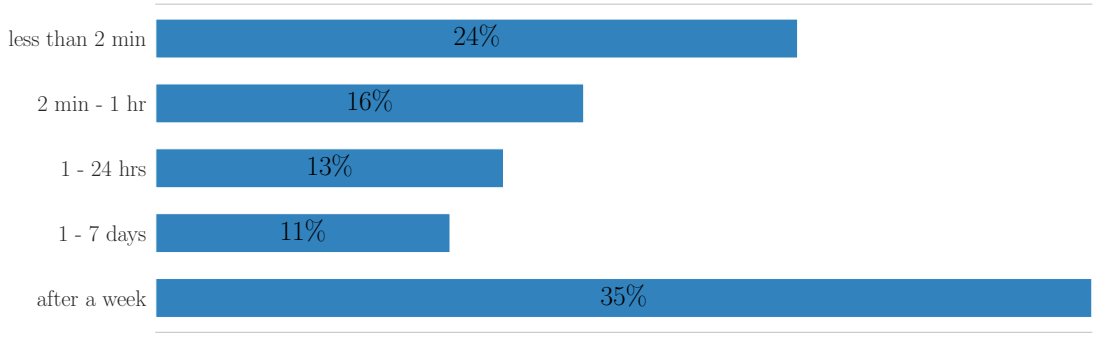


Fig. 2.3.: Self-reported percentage of deleted content removed at different time periods after publishing the post

in approximate percentages for each period; However, as will see next, the responses are in-line with prior works.

We see that 24% of the deletions are within less than two minutes from the publishing time (similar to [47], reporting 22% of deletions happening within a minute of their publication in Twitter), perhaps hinting at fixing misspelling or grammar.

The "2 minute - 1 hour", "1 - 24 hours", and "1 - 7 days" periods have similar frequency percentages (11-16%). These are periods where possible feedbacks are given by the users' family/close friend group (that closely follow the user's activities), coworkers/social friends (that check on their acquaintance's activities daily), and a much larger audience when posts go viral after a couple of days.

Interestingly, the largest category belongs to the deletions "after a week" with more than one third (35%) of all the deletions (similar to [32], where they report that one third of all posts are deleted after 6 years of their publication). This indicates that old posts on social platforms are not necessarily ignored, and users actively care about them and remove the unwanted ones.

Users delete their posts for non-obvious reasons. So far, we observed that over 80% of the participants have deleted at least one of their posts within the social platforms. The next question that comes to mind is what are the reasons behind users' deletions. We followed up with the 156 participants (that had deleted a post before) and asked what were the reasons behind their deletions. Nine different rea-

sons (shown in Table 2.3) were presented to them (taken from previous works [38, 39] and pilot studies) and further given an "other" option that they could have provided additional ones in a free text box. After categorizing the text responses we added four other reasons (i.e., "removing due to controversy/harassment", "removing embarrassing content", "personal reason" and "other") to the list.

The participants reported that the main reason for deleting their posts was that they became irrelevant as time passed (i.e., 64% of the users). This highlights the fact that 35% of deletions occurred after a week. About half of the participants indicated that they have removed a post due to the obvious reason of fixing spelling/grammar and factual checking. This reason is in-line with the 24% of deletions happening within a very short time of publishing (less than 2 minutes).

Observing Table 2.3, we see that a significant number of the participants have reported sensitive topics (drug, alcohol, race, politics, carnal, violence, etc.) as the reason for their deletion(s). In addition, 23 of the participants (15%) self-reported that they have deleted their posts to remove contents that were embarrassing to them or caused some controversy and harassment. This raises the question—*Are deletions an indication of hiding some sensitive or damaging content?*

Table 2.3.: Reasons of Deletion

Reasons	# (%)
Being irrelevant due to time passing	100 (64%)
Fixing Spelling/Grammar/FactCheck	77 (49%)
Post did not get enough attention	46 (29%)
Cleaning up my profile for new relationship	36 (23%)
Cleaning up my profile for new job	36 (23%)
Removing drug/alcohol/sexual related content	18 (12%)
Removing Racial/Religious/Political content	15 (10%)
Removing due to controversy/harassment	13 (8%)
Removing violence/cursing related content	11 (7%)
Removing embarrassing content	10 (6%)
Removing health related content	8 (5%)
Personal reason	7 (4%)
Other	6 (4%)

Table 2.4.: Users' agreement with the statement—"deletions indicate that the content of that post is sensitive/damaging/embarrassing to that individual."

Strongly agree	17 (9%)
Somewhat agree	81 (42%)
Neither agree nor disagree	45 (24%)
Somewhat disagree	35 (18%)
Strongly disagree	13 (7%)

Table 2.5.: Reasons for considering deletions as sensitive/damaging or not.

Reasons	Agree # (%)	Neutral # (%)	Disagree # (%)
Embarrassing, inappropriate, emotional	**73 (74%)**	4 (9%)	3 (6%)
Irrelevant, factCheck, grammar, spelling	8 (8%)	12 (27%)	**29 (60%)**
Context dependent	2 (2%)	**24 (53%)**	14 (29%)
No attention	3 (3%)	—	1 (2%)
Privacy	5 (5%)	—	1 (2%)
Political	4 (4%)	—	—
Job related	6 (6%)	—	—
Racism	1 (1%)	—	—
Other	1 (1%)	5 (10%)	2 (4%)

Majority of the users consider deletions as an indication of hiding something sensitive. We asked the participants whether they agree or disagree with the following statement—"when someone deletes a social media post, it indicates that the content of that post is sensitive/damaging/embarrassing to that individual".

The participant responses are shown in Table 2.4. More than half of the participants, to some degree, agreed with the statement and 25% disagreed. The remaining 24% neither agreed nor disagreed.

The question was followed by asking the respondents to provide an example in support of their answer. We categorized the responses and present them in Table 2.5. 74% of the respondents that considered deletions to contain some sensitive/damaging content indicated that users delete their posts as it contains some embarrassing, inappropriate, or emotional content. For example, participant $P29$ wrote: *"Someone*

would probably delete posts that would be embarrassing or legally damaging for the public to know about".

On the other hand, we see that 60% of the participants that disagreed with the statement see deletions as removing irrelevant content or fixing grammatical and spelling mistakes, Hence, containing no sensitive or damaging content. For example, participant $P61$ wrote: *"I've posted things that looking back an hour later I just think are dumb, nothing embarassing or offensive".*

More than half of the participants that did not agree nor disagree with the statement indicated that it is dependent on the context of the posts and can be considered either damaging or non-damaging. Therefore, they chose to be neutral about the statement. For example, participant $P3$ wrote: *'Those could be the reasons why, but they could also just think it's irrelevant/outdated, stupid, not worth having up, or factually wrong.'.*

Many users are likely unaware of their deletions being noticed. Previously we saw that 82% of the participants reported that they have deleted their posts in the past. To see if these deletions have been noticed by anyone, we asked those participants—"have you ever become aware of someone noticing that you have deleted one of your posts?". The results are presented in Table 2.6.

Table 2.6.: Number of participants that their deletions have been noticed by others and the number of participants that noticed others' deletions.

	Participant's deletion noticed by others	**Participant noticed others' deletions**
Deletion Notice		
Yes	**54 (35%)**	**130 (68%)**
No	82 (53%)	60 (31%)
Don't know	20 (13%)	1 (1%)
Social Circles [Who]		
Family member	22 (41%)	22 (17%)
Friends	50 (93%)	91 (70%)
Coworkers/Acquaintances	12 (22%)	43 (33%)
Stranger	**5 (9%)**	**55 (42%)**
Prefer not to say	—	2 (2%)

A significant minority of 54 participants (i.e., 35%) self-reported that they have became aware that someone noticed their deletions. The remaining 65% either said no or didn't know if someone has noticed their deletions.

We repeated the question by changing the roles, meaning that we asked the participants whether they have noticed anyone deleting their posts. Among all the 191 participants, 68% (130) of them reported that they have noticed at least one other user's deletion(s). The difference between these two settings hint that many users are likely unaware of their deletions getting noticed.

A significant percentage of deletions are being noticed by people outside the users' close social groups. In the previous subsection, we observed that many users become aware of others' deletions. Following this observation, we set to answer who are the individuals that are noticing the deletions? To affirmatively answer these questions, we asked the participants (i.e., those that are aware of others noticing their deletions) to identify the social group(s) that have noticed their deletion(s). Once again, we changed the roles and asked the participants (i.e., those that have noticed someone else's deletions) to specify whose deletions they have noticed. The results for these two sets of questions are presented in Table 2.6.

As shown in Table 2.6, the majority (93%) of the participants' deletions were noticed by their friends' group. Their family members took second place by noticing 41% of the deletions and not surprisingly, a very small number (9%) of strangers noticed the participants' deletions.

Conversely, these percentages change in the scenario of participants noticing other users' deletions. Again, the friend group stands out as the highest, but with a lower percentage of 70%. Surprisingly, the second highest-ranked is the stranger group with 42% (an increase of 33 percentage points). This point hints that many of the users' deletions are being noticed by people outside the users' close social groups, and the users are not aware of it.

Majority of the users anticipate some sort of a discomfort due to others notice their deletions. We saw that majority of the users consider deletions to

contain some sensitive, damaging, or embarrassing content. To see if participants have had any negative experiences from others noticing their deletions, we asked—"did you face any issues/problems/discomforts due to others noticing your deletions?". Nine individuals (17%) among the 54 participants (i.e., those that their deletions were noticed by others) reported some sort of an issue or discomfort. For example, participant $P56$ stated— *"It was an awkward conversation as I'd posted the post in a fit of anger/stress and then felt differently a day or so later. My friend wanted to talk about it and I didn't."*.

Further, to get a perspective about the potential issues and discomforts that the users foresee for their future deletions, we asked the participants—"Suppose you were to delete one or more of your social media posts; What possible issues/problems/discomforts do you think you might face if you become aware of someone noticing your post deletion(s)?"

We categorized the responses into seven different categories shown in Table 2.7. One-third of the participants report that there will be no to minimum consequences for others noticing their deletions. However, the remaining 67% feel that they will face some sort of a discomfort. 28% report that they will be needing to explain and justify the reasons that they have deleted their posts, while 17% of the users think that they will face some sort of a harassment and dislike from others noticing their deletions. Participant $P18$ wrote—*"I would probably receive potential backlash. Also, maybe I might get some questioning, whether it is supposed to be funny (like ironic),*

Table 2.7.: Possible discomforts due to deletions.

Possible Discomforts	# (%)
Minimum consequence	59 (33%)
Explaining/justifying/worried	49 (28%)
Publicizing/harassment/dislike	31 (17%)
Embarrassment/shame/insecure	25 (14%)
Loss of privacy	7 (4%)
Loss of archival value	2 (1%)
Other	5 (3%)

or serious". In another example, Participant $P40$ wrote—*"I could be banned from the platform or suffer some kind of angry mob that follows me around pointing out my socially unacceptable views"*.

Loss of privacy was another discomfort that 7 of the participants mentioned in their responses. For example, participant $P93$ stated—*"I might be uncomfortable because that means someone might be visiting my profile a lot and stalking it. This would make me wary about the media I post."*, and participant $P70$ wrote—*"If someone I don't know notices all the posts that I have ever deleted I would become uncomfortable and I would block them"*.

This shows that although many users have not yet faced any issues or problems from their deletions, a significant majority of them do think that their deletions may lead to negative consequences.

2.3.2 Uncovering Contextual Norms of Deletion Privacy

Next, we analyze the contextual norms of deletion privacy using data collected from the CI-driven questions. We visualize the average acceptability scores, ranging from -2 (completely unacceptable) to 2 (completely acceptable) as explained in Section 2.2.5, for all information flows with the pairs of recipient and subject, transmission principle, or attribute in Figure 2.4 using heatmaps.

Users seek deletion privacy against large-scale data collectors. The average acceptability scores of information flows that have "the government" as their recipient are mostly negative (Figure 2.4). This indicates that most of the participants consider these flows as "completely unacceptable" or "somewhat unacceptable". Although less severe, the same is true for "a company" as the recipient. In contrast, the average scores of flows with the recipient "family members", "friends", and "coworkers" are all positive and mostly above one. This score difference between flows with large-scale data collectors (governments and private companies) as recipients versus the closely connected individuals (family, friends, and coworkers) holds regardless of other

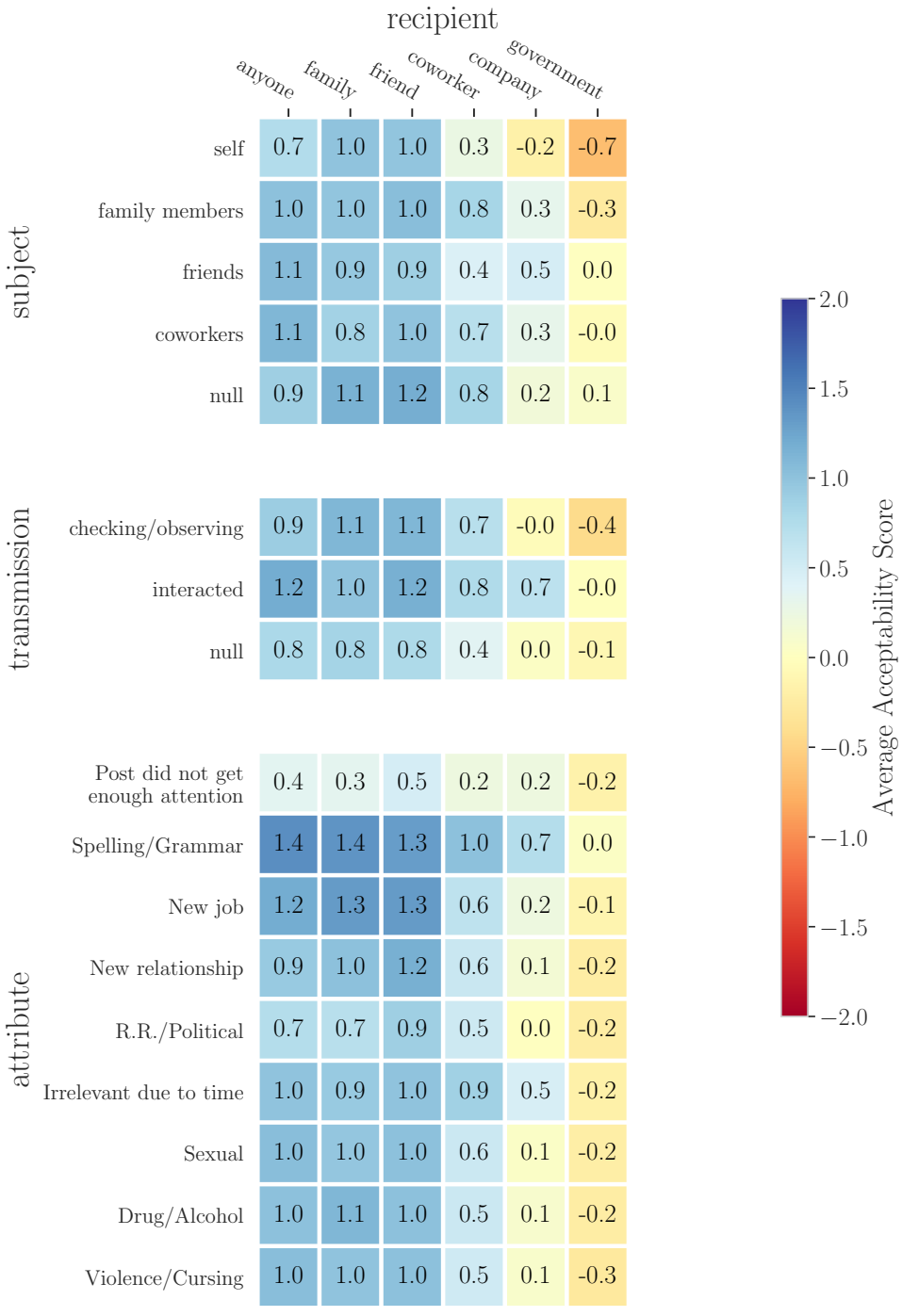


Fig. 2.4.: Average acceptability scores of information flows grouped by recipients and subjects, transmission principles, or attributes. Scores range from -2 (completely unacceptable) to 2 (completely acceptable).

CI parameters (information transmission principle, subject, attribute). On average, information flows with large-scale data collectors as the recipient are 0.94 Likert-scale points less acceptable than their other recipient counterparts ($p < 0.00001$).

This result provides the first quantitative evidence that social platform users are indeed concerned about their deletions being noticed by third party services and state agencies. This result urges the social platform developers to proactively create different deletion privacy policies for different classes of recipients.

Deleted posts that are about the users themselves need more protection. One of the CI parameters that we explored in this study is the subject of the deleted posts. We performed a pairwise statistical test (i.e., 20 tests) between all the possible subjects (the significant value was adjusted to 0.05/20) to see any significant differences. The only post subject that has acceptability scores with a different distribution than all other subjects is the "user itself" ($p < 0.002$). This difference is shown in the average scores, where the flows with the subject "user itself" have a 0.35 Likert-scale point less acceptability.

Not knowing how deletions were noticed is less acceptable to the users. In Section 2.1.2, we defined two methods of noticing the deletions (i.e., the transmission principle). We also considered the null transmission principle where no discovery method is specified to the participants. By statistically comparing the distributions of the flows with the null transmission principle versus the non-null transmission principles we see that there is a significant difference ($p < 0.00001$). The mean of the null transmission principle is 0.23 Likert-scale points smaller (less acceptable) than the non-null transmission principles. This is consistent with human desire for cognitive closure [48] that not knowing how deletions were noticed is less acceptable.

Users want to hide their non-popular posts more than any other post. The *attribute* parameter, of the CI flows, in this study corresponds to the reasons for removing a post. We identified ten different reasons for the deletions shown in Table 2.1 (bottom portion). Similar to the *subject* parameter, we performed a

pairwise statistical test (i.e., 90 tests) between all the attributes (the significant value was adjusted to 0.05/90).

We found that the attributes "Fixing Spelling/Grammar" and "Post did not get enough attention" are statistically different ($p < 0.0001$) than all other attributes. These two deletion reasons are at the opposite ends of the acceptability scores. Flows with the attribute "Fixing Spelling/Grammar" have the highest average acceptability score (0.98 on the Likert-scale points), and the flows with the attribute "Post did not get enough attention" have the lowest (0.24 on the Likert-scale points). We further analyzed the low-scored flows by checking the correlation between the scores given by users who had self-reported deleting content for that reason. Interestingly, for "Post did not get enough attention", the negative scores primarily came from people who did not delete content because of that reason. This finding hints at the fact that our participants *perceived* digging up forgotten posts by virtue of deletion as a serious violation of the deletion privacy. In other words, the platforms should consider providing stronger deletion privacy to non-popular posts than popular posts.

Effect of demographics on deletion privacy. We considered different demographic categories (gender, age, education, income, and marital status) to see their effect on the scores given to the information flows. We performed five comparison tests and set the threshold for significance to $\alpha = 0.05/5 = 0.01$ to account for the Bonferroni multiple-testing correction.

Among the above demographics, all of them had some significant difference ($p < 0.01$), as we explain below, except the income category. We divided the participants into two groups of higher household income ($80,000 and more) and lower household income (less than $80,000). The statistical test showed no significant difference between the two groups.

Female users and higher educated users are more concerned about their deletions. The average score of the information flows labeled by the female participants is 0.51 on the Likert-scale points and is significantly different from the average score of the male participants that scored an average of 0.7. We observe a similar

result with the same average scores between the participants that have a university degree (bachelor or graduate degrees) compared to all other participants.

Younger participants (millennials and generation Z) are less concerned about their deletions. We see that there is a significant difference between the average scores of participants that identified themselves between the age of 18-34 to all other older participants. The younger generation has a higher acceptability score with an average of 0.65 Likert-scale points compared to all other participants, with an average score of 0.46.

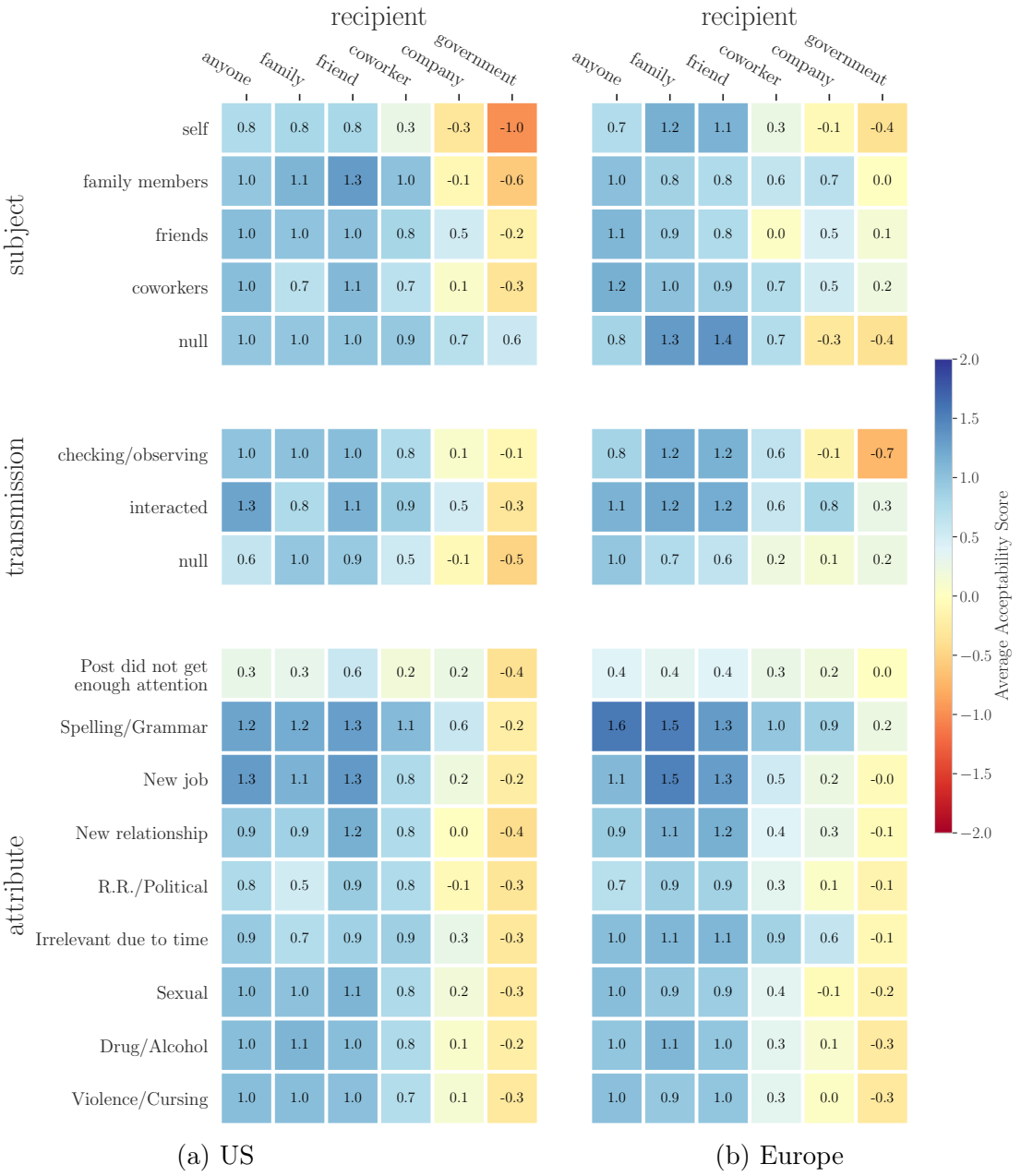
Individuals that have ended their relationship in the past are more conservative. We found that on average, individuals that had identified their marital status as divorced or separated had a lower acceptability score (0.22 Likert-scale points) compared to the individuals that identified themselves as single, never married, or in a current relationship (0.62 Likert-scale points).

Differences between the US and Europe. In what follows we highlight the main differences between the responses observed from the US and European participants.

Coworkers are considered a closer social group in the US. Observing Figure 2.5, we can see the differences between the US and Europe when the recipient is a "coworker". By applying the statistical test between the flows of the recipients "coworker" and "family" for the US participants we see no significant difference. However, considering the same conditions, there is a significant difference in Europe ($p < 0.00001$).

The opposite is true when comparing the distribution of the flow where the recipient is "coworker" and "a company". In this case, there is no significant difference between the distributions in Europe, but one exists for the US ($p < 0.00001$). We conclude that in the US, individuals consider their coworkers to be in a closer social group compared to European individuals.

Stalking is more of a concern in Europe. Earlier, we saw that users are much more comfortable if they know what transmission principle (method of noticing the deletion) is used to notice their deletions. However, we observe a difference between



(a) US (b) Europe

Fig. 2.5.: Average acceptability scores of information flows grouped by recipients and subjects, transmission principles, attributes, gender, age, or background IT information. Scores range from -2 (completely unacceptable) to 2 (completely acceptable).

the acceptability of the US users and European users when looking at the transmission principle "checking and observing the user profile regularly" (in other words stalking). We compared the scores that participants in the US and Europe gave to the

two defined transmissions separately. We observe no significant difference between the two transmission principles for the US participants. However, there is a significant difference between checking/observing the users' profile regularly (stalking) and interacting with the post in Europe ($p < 0.00001$). The stalking method is seen as a less acceptable means of noticing the deletions compared to noticing due to a prior interaction with a difference of 0.4 Likert-scale points.

2.4 Concluding Remarks

In this study, we observed that the majority of the users are deleting their posts every day. There is a strong user-need for ensuring deletion privacy in social platforms, as users consider deletions a tool for removing sensitive, damaging, and embarrassing content. Further, using contextual integrity we demonstrated the context-dependency of the rules for preserving deletion privacy. The study identified that it is acceptable for the users if the one-hop individuals (family members, friends) on their social graphs become aware of their deletions but not the large-scale data collecting actors (e.g., web-service data collectors or the government).

3. INTERMITTENT WITHDRAWAL MECHANISM TO CONCEAL CONTENT DELETION

The large-scale identification and hoarding of deleted content from social sites and archives pose a serious violation of "Right to be Forgotten" and the ill-effects of this phenomena on our social behavior will be far reaching. For example, in one case, singer Ed Sheeran's deletion of a tweet from 2011 was found and widely publicized in media [24] leading to his brief disappearance from Twitter. In another case, an SNL cast member's deletion of racist tweets back in September 2016 [25] were tracked by third parties and subsequently publicized. Not only celebrities but normal users also fell prey to this phenomenon when links delisted by Google in Europe (to honor Right to be Forgotten requests) were identified, publicized and scrutinized by media [26]. In general, the users today are extremely vulnerable due to the fact that, whatever content they delete (ironically, to protect their privacy) will possibly be identified, dissected and abused.

In spite of this threat, not surprisingly, without any better alternatives available, information exposure control in the form of deletions still remains a common phenomenon on the social platforms; Mondal et al. [6] observe that a significant fraction ($\sim$35%) of all Twitter users have now deleted or confined (i.e., made private) their public Twitter posts made in 2009. Consequently, as any persistent onlooker can keep track of such changes and go after the deleted posts, users aiming to make observers forget their posts are left with a "damned if I do, damned if I don't" dilemma. This work aims to provide a solution to the problem.

A trivial solution is to make users *not* publish sensitive content in the first place; but this is infeasible even for extremely careful users as the sensitivity of shared data changes drastically and unpredictably with time and life events. A growing number of users have now shifted to ephemeral social platforms such as Snapchat [49], where

everything gets deleted in a premeditated fashion. However, given the huge historical, cultural, and economical value of user-generated data, it is extremely unlikely that most next-generation social or archival platforms will adapt to this model.

This leaves us with a hard research question: *can we offer an alternative to the next-generation social or archival platforms that achieves the best properties of both deleting everything (i.e., privacy) and keeping an archive of posts and events (i.e., availability)?* The aim of this work is to answer this question affirmatively and develop a privacy mechanism that retains the archival values of posted content and still allows deletions while providing deniability and protection to the users after some time of deletion, i.e., those deletions will not be immediately discernible to even persistent onlookers.

A simple-yet-drastic proposal. We offer a simple-yet-drastic proposal towards mitigating the problem of concealing content deletions in presence of persistent observers while maintaining high availability of archived content. In our proposed system, *Lethe*[1], we very conservatively assume that the adversary has complete access to the archival platform and can view any post. We presume the platform administrator is working with the data creator (or owner) to protect the privacy of deletions. *Lethe* employs an *intermittent withdrawal mechanism* that protects privacy using two public, infinite-support time distributions—one we call the up (or online) distribution and the second is called down (or offline) distribution. Just before publishing a post, *Lethe* samples a time duration from the up distribution and for that time duration makes that post available (i.e., visible) to everyone. After the up duration passes *Lethe* takes an instance from the down distribution and for that time duration hides the post from viewers.

In the same way, *Lethe* continues to toggle between the up and down durations as long as the post has not been deleted or its privacy preference has not changed.

[1]In Greek mythology, *Lethe* was the river of forgetfulness: all those who drank from it experienced complete forgetfulness. The word *Lethe* also means oblivion, forgetfulness, or concealment.

Since *Lethe* also hides non-deleted posts, it will be confusing for the adversary to distinguish whether a post is hidden by *Lethe* or deleted by the owner.

In particular, we make four key contributions.

Firstly, to the best of our knowledge, this is the first systematic study of the problem with content deletion in the presence of persistent onlookers. We formalize the problem with content deletion in the presence of a very powerful adversary who can take snapshots of the whole platform at any point in time. We define and analytically quantify the necessary security notions: *privacy*–likelihood ratio of a post deleted or not at any particular time, *availability*–fraction of time the posts are visible and *adversarial overhead*–adversary's precision on detecting deleted posts. Based on our formalization, we propose and evaluate a novel scheme, *Lethe*, to provide privacy for users' deletions.

Secondly, we show that privacy is correlated with the up and down distributions: (i) inversely proportional to the hazard rate of up distribution, and (ii) inversely proportional to the complementary cumulative distribution function (CCDF) of down distribution. Moreover, we show that by picking geometric and negative binomial distributions as the up and down distribution, not only we achieve good privacy guarantees, but our notion of privacy is simplified to a *decision threshold* period—duration an adversary is willing to wait before identifying a (hidden) post in a down period as deleted.

Thirdly, we present the trade-offs between the notions mentioned above using data from Twitter. We show that in the case of 95% content availability, the adversary, with an uninterrupted access to the entire platform, will have a precision value associated with adversarial overhead below 20% even when a post has been down for more than 90 days. In the case of a more forbearing adversary that has a decision threshold of 180 days, the precision will only increase to 35%. However, the system administrator can reduce the availability of the system by a small fraction and set it to 90%, which drops the adversary's precision back to 20%. For a large-scale system such as Twitter, with trillions of tweets, even precision of 80% can result in a significant overhead for

the adversary (investigating 20 million non-deleted tweets falsely marked as deleted each day).

Finally, we evaluate the effect of our scheme on Twitter dataset to show the feasibility of *Lethe* in a real-world scenario. We show that our proposal, while maintaining a trade-off between availability and privacy, also allows interactions in the system without much interruption. Specifically, leveraging real-world interaction data from Twitter we show that, by applying *Lethe* the utility (i.e. user interactions with posts) remains above 99% even when content availability is 85%.

3.1 Context and Motivation

User-initiated spontaneous deletions. One of the widely employed form of content deletion today is user-initiated deletion; i.e., system operators remove content when the owners explicitly asked them to do so. Almost all real world social data sharing platforms today (e.g., Facebook, Twitter or YouTube) provide users option to delete their uploaded content. Recent studies [6,50] have shown that users extensively use this mechanism to protect the privacy of their past content—users delete around 35% of posts within six years of posting them. The European Union (EU) regulation of "Right to be forgotten" [26, 51] which is part of EU General Data Protection Regulation (GDPR) [52] is also trying to accomplish exactly this same, albeit at a much more elaborate scale. They wish to enable users to remove historical data about themselves from multiple systems, including removing results from leading search engines. Nevertheless, as we already suggest, those deleted content attract unwanted attention [26].

Premeditated withdrawals. Complementary to these user-initiated spontaneous deletions, a number of *premeditated* withdrawal methodologies have been proposed and employed today.

Many of those aim to protect content privacy via withdrawing *all* posts after a predefined viewership or time of posting; we call those the *age-based withdrawals*.

Recent ephemeral social content sharing sites like Snapchat [49] or Dust [53] are prominent examples of age-based withdrawal. Several academic projects also try to enforce age-based withdrawal in different context; e.g., Vanish [54, 55] in distributed hash tables (DHTs), EphPub [56] and [57] using DNS caches, and Ephemerizer [58] and its improvement [59] using trusted servers. A user's inability to a priori predict the right time (or viewership) for her content withdrawal remains to be the key issue with the age-based withdrawals. This prevents deriving the best possible content availability.

Mondal et al. [6] suggest *inactivity-based withdrawal* to eliminate the burden on the users to decide expiry times and to facilitate continued discussions around interesting content. Unlike in age-based withdrawals, where a post is withdrawn after a predefined time or viewership, in inactivity-based withdrawal posts can be withdrawn only when it becomes inactive over time, i.e., it does not generate any more interactions (e.g., sharing the post by other users). Recently proposed Neuralyzer [60] uses a similar concept to maintain the availability of content as long as there is sufficient demand for it, and leverages the caching mechanisms of DNS to keep track of the activity. A similar idea is also employed on sites like 4chan [61, 62], where posts are withdrawn as users stop contributing to them for a prolonged time.

Problems with premeditated withdrawals: No historical data. The above premeditated withdrawal methodologies remove every post from the public view eventually; thus, there is *no* archived history of user data. However, existence of archival data can be important to not only the system but also the users. A recent survey [4] shows that users have a keen interest in going back to the past social content they have uploaded, e.g., for reminiscing old memories. Moreover, as social media sites are often perceived as a mirror of the real world, reflecting events in the past and how people reacted to them, archiving the past uploaded content has immense historical value; e.g., US Library of Congress [63] is already archiving all uploaded public Twitter data.

Moreover, if a user deletes her post before the predefined time (or viewership) limit on the post, an adversary can be certain that it is a user-initiated content deletion. In this case, the current premeditated schemes provide no privacy or deniability to the user.

Our Approach. Our challenge is to devise a privacy mechanism that offers protection to user-initiated content deletions (from a persistent onlooker with pervasive access) without reducing the content's archival value. We demonstrate how to achieve these contrasting privacy and availability goals by systematically withdrawing and resurrecting non-deleted posts from public view.

3.2 Problem and Key Idea

3.2.1 System and Adversary Model

We model a user-generated data sharing platform (e.g., Twitter) as a public bulletin board where individuals can upload and/or view content. Below we define prominent players and their roles in our setup: *Platform* is the system, which maintains the bulletin board (used to upload and view user generated content); *Data Owner* is a user who uploads her posts to the bulletin board. *Adversary* can view the uploaded posts on the bulletin board and is constantly in search of posts which have been deleted by their owners (possibly to scavenge for the posts that are sensitive to their owners).

In our generic model, all the subscribers (including the adversary) have complete access to the bulletin board and can view the posts as they wish. After a data owner decides to delete a post, the post will be removed from the bulletin board and will not be visible to anyone. We expect the publisher to be honest and assist towards achieving the privacy goal.

Our adversary accesses the bulletin board continuously and takes snapshots at will. He can determine the deleted posts by comparing the two snapshots taken at different times and pinpointing the posts that existed in the first one but not in

the second one (the same strategy used to find deleted tweets in previous studies [6, 64]). The adversary is capable of adding posts and deleting them from the bulletin board; however, it will not be able to delete some other users' posts. Although the adversary is ultimate in terms of the data access, given the manual nature of the task of determining sensitive deletions, his goal will be to flag and analyze as few non-deleted posts as possible. In the real world, an adversary would be actually limited in its capability; consequently, all the privacy guarantees we observe in this work are actually lower bound (Section 8). Finally, we expect all aspects of our system and its parameters to be public, and the adversary to be aware of those.

3.2.2 Security Goals

Towards our goal to conceal deletions from the adversary without significantly affecting the availability, we propose the following security properties:

Deletion privacy is the uncertainty of the adversary about a post having been deleted or just temporarily withdrawn by the platform at a given point of time. In other words, it is the deniability of deleting a post for the data owner. As the post remains down for a longer duration, the adversary becomes more certain about its deletion, achieving a particular level of privacy is directly related to having a certain *Decision Threshold* on the observed down periods for declaring that posts are deleted beyond that point.

Platform availability represents the average availability of a post within a period. The goal is to provide privacy guarantees to users while obtaining high levels of availability. It is easy to observe that introducing down periods creates a trade-off between privacy and availability. For example, assuming the mean up duration is fixed, as the mean of down distribution increases the availability of the platform will decrease; however, when a post is deleted, it remains unnoticed to the adversary for longer periods due to higher decision thresholds.

It is natural to ask why the adversary cannot select his decision threshold independent of down distribution (and subsequently availability). The answer lies in the difficulty of distinguishing sensitive posts from non-sensitive ones. Sensitivity of a post varies from person to person, and also with life events and time in general, therefore, pinpointing sensitive posts for each user is a hard task. Moreover, there is a huge discrepancy between the content creation and deletion rates on social sites today (social sites are generating new content at the rate around ten times more than deletions).

This brings us to our third property of adversarial overhead as we expect our adversary to be concerned with flagging many non-deleted posts (false positives).

Adversarial overhead. is associated with the number of non-deleted posts falsely flagged as deleted (false-positives) that the adversary has to investigate along with the detected actual deleted posts (true-positives). We capture it by the precision measure:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Towards offering a balanced viewpoint, we also consider the recall measure capturing false-negatives (i.e., posts that are flagged as non-deleted but will eventually be deleted):

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}.$$

There is a trade-off between privacy and adversarial overhead similar to the trade-off between privacy and availability. Ideally, the adversary overhead should be high which implies that the precision should be low. If the adversary needs to keep its overhead low (less false positives), it has to provide better privacy (deniability) to its victim by increasing its decision threshold period.

3.2.3 Key Idea

We plan to provide privacy for a post deletion by intermittently withdrawing the non-deleted posts such that the adversary cannot distinguish between a temporarily withdrawn post and a permanently deleted post for some long time duration after the deletion. At its core, our intermittent withdrawal mechanism consists of choosing alternating up and down periods of random durations. This obviously adversely affects the availability of posts: increasing withdrawal time of a post can improve the deletion privacy; however, it reduces the overall availability. Therefore, our key challenge is to determine distributions (and their parameters) for these intermittent withdrawals such that we achieve a satisfactory level of deletion privacy without significantly affecting the availability of the posts.

We illustrate our distributions selection process through the following two Straw-man proposals.

Straw-man proposal I. As a simple example, consider the degenerate (or fixed-value) distribution for up and down duration of a post. With 90% availability in mind, we consider an alternating series of fixed up period of nine hours and fixed down period of an hour. Here, every post once withdrawn remains down for a complete hour. Thus, the adversary cannot flag a post as deleted until it remains down for more than an hour as any flagging during the first hour down time cannot be better than just randomly flagging the posts. However, the adversary becomes certain about the deletion right after this one hour of down period. Moreover, if the deletion occurs sometime during the up period of nine hours, the adversary can break the privacy immediately.

Although it is possible to increase down time while maintaining the same availability, the adversary can simply wait longer before becoming certain about the deletion. Larger down time may also not be acceptable to platforms expecting content to be highly available.

Straw-man proposal II. We can replace the above degenerate distribution by the uniform distribution with mean value of nine hours for the up distribution and mean value of one hour for the down distribution. Here, the deletion can happen anytime during the up duration without the adversary becoming certain about the deletion. However, the problem with the down period remains: with the finite support of the down distribution (two hours for our example), the adversary will be sure about deletion after two hours.

Towards *Lethe*. As we do not expect the platform and the users to accurately predict the waiting time (i.e., decision threshold) for the adversary, we propose to use the distributions with infinite support. Here, the adversary can *never* be certain about the deletions; but it is easy to see that once the post is deleted, the adversary becomes more certain about it as time progresses.

Towards building and analyzing *Lethe*, we measure privacy as likelihood ratio in Section 3.3, and find it to be inversely proportional to both hazard rate of the up distribution and complementary cumulative distribution of the down distribution. We measure availability as the ratio of mean up distribution and sum of means of both (up and down) distributions. In Section 3.4, we then explore different distributions with infinite support to select an up and down distribution that offers an excellent trade-off between deletion privacy, availability and adversarial overhead. Finally, in Section 3.5, we evaluate the system for the estimated Twitter dataset.

3.2.4 Non-Goals

Firstly, we consider all withdrawn posts to be *equal*, and do not consider the sensitivity of a post's content. Several other studies [65–67] investigated the sensitivity of posts in general and resulting privacy leaks. Those studies provide complementary privacy guarantees and can be used in addition to our approach.

Secondly, we do not take into account correlations between posts, and instead, assume individual posts to be independent in this first proposal for a very difficult

problem. Given extremely unpredictable and context-dependent nature of correlations between posts on social sites, considering correlations where they are apparent, will be an interesting future work.

Finally, similar to the usage of salting in password hashing against the dictionary (or rainbow table) attacks, our goal is to protect the privacy of withdrawn posts on a *large scale*, and our adversary scavenges through all the withdrawn posts to find as many sensitive deletions as possible. We do not aim to protect against a devoted stalker who stalks a particular user or post over a long duration. For example, an adversary with prior knowledge of users (e.g., posting patterns) will have an advantage that we do not consider. Nevertheless, as compared to the state-of-the-art, we aim at increasing the workload of devoted attackers and at delaying the deletion privacy loss at least by a few arguably important weeks.

3.3 Problem Formalization

3.3.1 Formalized Intermittent Withdrawals

In the proposed system, time is discretized in seconds. We denote by t_c the current time. We treat each post independently, and therefore, the privacy and availability analyses focus on an individual post. Let t_0 denote the creation time of the post.

The intermittent withdrawal mechanism introduces a disconnection between the real state of a post (deleted or non-deleted) and the observed state of the post (publicly visible or withdrawn). The real state of the post is available *only* to the platform and the owner, while the adversary can only see the observed state of the post.

Real state: Let $\mathcal{R}(t)$ denote the real state (either non-deleted or deleted) of the post at time t. By convention, we say that $\mathcal{R}(t) = 1$ if the post is not deleted at time t and $\mathcal{R}(t) = 0$ if the post is deleted. For example, at creation time t_0, $\mathcal{R}(t_0) = 1$.

We assume that a post cannot be undeleted (or resurrected) and thus can be deleted only once. Consequently, we define the deletion time $t_{del} > t_0$ such that $\mathcal{R}(t) = 1$ for all $t \in [t_0, t_{del})$ and $\mathcal{R}(t) = 0$ for all $t \geq t_{del}$. We also assume that t_{del}

is not a choice variable of the platform and remains unknown to the platform at any time before t_{del}.

Observable state: At any time t, by accessing the bulletin board, the adversary or any user only sees if the post is up (visible) or down (withdrawn). Let $\mathcal{O}(t)$ denote this observable state of the post at time $t \geq t_0$. By convention, we say that $\mathcal{O}(t) = 1$ if the post is up and $\mathcal{O}(t) = 0$ if the post is down.

For a post, the platform can decide $\mathcal{O}(t)$ for all $t_0 < t < t_{del}$. In particular, for each post, the platform chooses a sequence of positive integer values $(T_u^j)_{j\in\mathbb{Z}_+}$ and $(T_d^j)_{j\in\mathbb{Z}_+}$, interpreted as up and down time durations respectively. The observable state is set as follows.

$$
\begin{aligned}
&\text{For all } t \in [t_0, t_{del}) : \qquad (3.1)\\
&\quad \mathcal{O}(t) = 1 \text{ if, for some } i \geq 0,\\
&\qquad t \in \left[t_0 + \sum_{j=1}^{i} T_u^j + \sum_{j=1}^{i} T_d^j, t_0 + \sum_{j=1}^{i+1} T_u^j + \sum_{j=1}^{i} T_d^j \right);\\
&\quad \mathcal{O}(t) = 0 \text{ if, for some } i \geq 0,\\
&\qquad t \in \left[t_0 + \sum_{j=1}^{i+1} T_u^j + \sum_{j=1}^{i} T_d^j, t_0 + \sum_{j=1}^{i+1} T_u^j + \sum_{j=1}^{i+1} T_d^j \right).\\
&\text{For all } t \geq t_{del} : \mathcal{O}(t) = 0.
\end{aligned}
$$

Figure 3.1 illustrates the observable state (from an adversary's point of view) for a post due to the sequences of up and down duration. As the deletion time t_{del} is not known to the platform at any time before t_{del}, we can assume without loss of generality that large sequences $(T_u^j)_{j\in\mathbb{Z}_+}$ and $(T_d^j)_{j\in\mathbb{Z}_+}$ are chosen by the platform at the creation time t_0. As a result, the observable state in Equation equation 3.1 can be intuitively interpreted as follows. The post is initially up and stays up for a duration T_u^1. After the duration T_u^1, it goes down and stays down for a duration T_d^1 before coming up again. This process continues indefinitely until the post is deleted by the

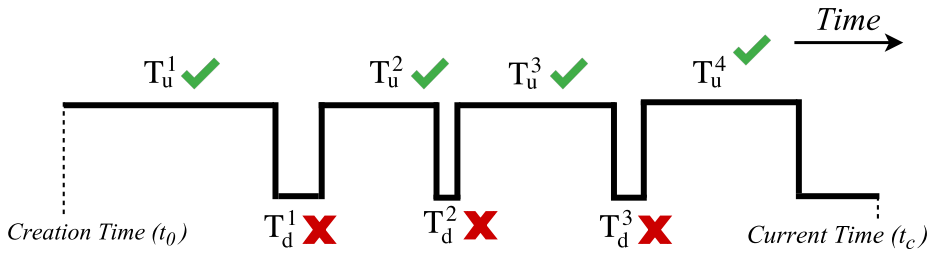


Fig. 3.1.: Timeline of a post. The post is created at time t_0, T_u^i is the duration of an up phase and T_d^i is the duration of a down phase. In up phases the post is visible to the adversary, in down phases it is not.

owner. Finally, when a post is deleted, it goes down immediately even if it is in middle of an up duration, and stays down forever.

Our objective is to control the observable state so that it becomes difficult for the adversary to be certain about the deletion of a post. In the proposed intermittent withdrawal mechanism, $(T_u^j)_{j\in\mathbb{Z}_+}$ and $(T_d^j)_{j\in\mathbb{Z}_+}$ are mutually independent i.i.d. sequences of random variables drawn from probability mass functions (PMFs) f_{T_u} and f_{T_d} respectively. We define the intermittent withdrawal mechanism as follows:

Definition 3.3.1 (Intermittent withdrawal mechanism) *We define $\mathcal{M}_{IW}(f_{T_u}, f_{T_d})$ as an algorithm that draws mutually independent i.i.d. sequences $(T_u^j)_{j\in\mathbb{Z}_+}$ and $(T_d^j)_{j\in\mathbb{Z}_+}$ from f_{T_u} and f_{T_d} respectively, and sets $\mathcal{O}(t)$ as in Equation equation 3.1.*

As elaborated later in Section 3.4 and onwards, We choose parameters PMFs f_{T_u} and f_{T_d} of the $\mathcal{M}_{IW}$ to satisfy the contrasting privacy, availability, and adversarial overhead requirements. Throughout the analysis, F_{T_u} and F_{T_d} represent the cumulative distribution functions (CDFs), and $\overline{F_{T_u}}$ and $\overline{F_{T_d}}$ represent the complementary cumulative distribution functions (CCDFs) of f_{T_u} and f_{T_d} respectively. We assume that the platform can efficiently sample values from distributions f_{T_u} and f_{T_d}, and that these distributions are known to the adversary.

Next, we formally analyze our security goals in the context of $\mathcal{M}_{IW}(f_{T_u}, f_{T_d})$.

3.3.2 Deletion Privacy

The notion of deletion privacy should quantify the uncertainty of the adversary in distinguishing between a post being really deleted by the owner or just in one of its down durations. We define this likelihood of adversary detecting an actual deleted post as the likelihood ratio of the observed sequence of observable states since post creation conditioned on the post being deleted or not at the current time t_c.

Definition 3.3.2 *For any time t_c, we define the privacy of mechanism $\mathcal{M}_{IW}(f_{T_u}, f_{T_d})$ as a ratio (LR)*

$$LR = \frac{\sup_{t \leq t_c} Pr(\mathcal{O}_{\mathcal{M}_{IW}}([t_0, t_c]) \mid t_{del} = t)}{\sup_{t > t_c} Pr(\mathcal{O}_{\mathcal{M}_{IW}}([t_0, t_c]) \mid t_{del} = t)}, \tag{3.2}$$

where $\mathcal{O}_{\mathcal{M}_{IW}}([t_0, t_c])$ is the observed state for posts due to $\mathcal{M}_{IW}$ in the interval $[t_0, t_c]$.

The above ratio is the classical *likelihood ratio* (LR) statistic [68] for the test to determine if the post was deleted or not, i.e., the test with null hypothesis H_0 : $\{\mathcal{R}(t_c) = 1\}$ (equivalent to $\{t_{del} > t_c\}$) and alternative hypothesis $H_1 : \{\mathcal{R}(t_c) = 0\}$ (equivalent to $\{t_{del} \leq t_c\}$). It is known that likelihood ratio tests have good properties and are often the most powerful tests that the adversary can do to determine if the post was deleted [68]. Hence, limiting this likelihood ratio is the best way of limiting the possibility for the adversary of accurately testing if the post was deleted or not. Increase in the LR value for a post denotes increase in certainty of the adversary about a post deletion; in short, lesser value of LR denotes better privacy. Since the adversary knows the up and down time distributions it can compute the likelihood ratio of the deletion privacy. Our definition of deletion privacy parallels with the definition of differential privacy [69], however there is subtle difference between them which is explained Section 3.7.

We will use Equation (3.2) to analyze the privacy using the Frequentist approach. We refer interested readers to our paper [19] for the Bayesian analysis of Equation (3.2).

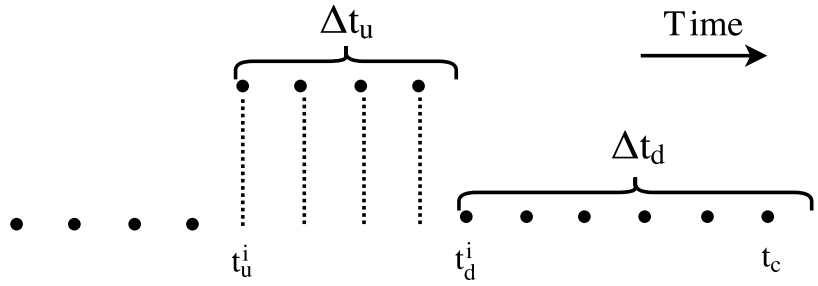


Fig. 3.2.: Observing the status of a single post from its creation and precisely looking at the last up and down duration, Δt_u and Δt_d respectively. t_c is the current time, t_u^i denotes the last up toggle and similarly t_d^i is the last down toggle time.

Deletion Privacy for the Intermittent Withdrawal Mechanism Using Frequentist Approach. As deletion privacy (or LR value) depends on $\mathcal{O}([t_0, t_c])$ (i.e., the sequence of observable states chosen by the platform) and consequently on the distributions f_{T_u} and f_{T_d}, we need to quantify this dependency to understand the deletion privacy offered by intermittent withdrawal mechanism.

In our intermittent withdrawal mechanism, up and down durations are drawn i.i.d. until the post is deleted. Therefore, the probability of the sequence is the product of probability of observing each duration which is the same regardless of if the post was deleted or not except for the last up and down durations; one of the last up and down durations could be cut by the deletion. As a result, the ratio LR depends only on the last up and down durations. We denote last up and down duration by Δt_u and Δt_d respectively and by extension by $\mathcal{O}(\Delta t_u, \Delta t_d)$ the observed state in those times (see illustration on Figure 3.2). Then the likelihood ratio on the lhs of equation 3.2 can be simplified as

$$LR = \frac{\sup_{t \leq t_c} Pr(\mathcal{O}(\Delta t_u, \Delta t_d) \mid t_{del} = t)}{\sup_{t > t_c} Pr(\mathcal{O}(\Delta t_u, \Delta t_d) \mid t_{del} = t)}. \tag{3.3}$$

Now we compute the numerator and denominator separately. The denominator is simply the likelihood of observing $\mathcal{O}(\Delta t_u, \Delta t_d)$ if the post was not yet deleted at time t_c (i.e., $\mathcal{R}(t_c) = 1$), which is

$$Pr(\mathcal{O}(\Delta t_u, \Delta t_d) \mid \mathcal{R}(t_c) = 1) = f_{T_u}(\Delta t_u) \cdot \overline{F_{T_d}}(\Delta t_d - 1). \tag{3.4}$$

As the post has not been deleted at time t_c (i.e., $\mathcal{R}(t_c) = 1$), the probability of observing Δt_u is $f_{T_u}(\Delta t_u)$. Moreover, since the post is in middle of a down period the probability of observing Δt_d is $Pr(T_d^i \geq \Delta t_d) = \overline{F_{T_d}}(\Delta t_d - 1)$.

For the numerator, we compute the probability of $\mathcal{O}(\Delta t_u, \Delta t_d)$ conditioned on the deletion time being t for each t between the last toggle t_d^i and the current time t_c and take the maximum of those probabilities. (It is *not* necessary to consider earlier deletion times since the probability of $\mathcal{O}(\Delta t_u, \Delta t_d)$ would then be zero.) We treat separately the case where $t_{del} = t_d^i$ which corresponds to a deletion happening during (or at the end of) an up period and the cases $t_{del} \in (t_d^i, t_c]$ which correspond to a deletion happening during a down period. In the second case, for $t \in (t_d^i, t_c]$, we have

$$Pr(\mathcal{O}(\Delta t_u, \Delta t_d) \mid t_{del} = t) = f_{T_u}(\Delta t_u) \cdot \overline{F_{T_d}}(t - t_d^i - 1),$$

which is maximized for $t = t_d^i + 1$ where $\overline{F_{T_d}}(t - t_d^i - 1) = \overline{F_{T_d}}(0) = 1$. In the case where $t_{del} = t_d^i$, then the last up period could have been either of exactly Δt_u or of more, hence

$$Pr(\mathcal{O}(\Delta t_u, \Delta t_d) \mid t_{del} = t_d^i) = \overline{F_{T_u}}(\Delta t_u) + f_{T_u}(\Delta t_u).$$

Since $\overline{F_{T_u}}(\Delta t_u) \geq 0$, we conclude that

$$\sup_{t \leq t_c} Pr(\mathcal{O}(\Delta t_u, \Delta t_d) \mid t_{del} = t) = \overline{F_{T_u}}(\Delta t_u) + f_{T_u}(\Delta t_u). \tag{3.5}$$

Substituting equation 3.4 and equation 3.5 into equation 3.3, we get the final expression of the likelihood ratio:

$$LR = \left(\frac{\overline{F_{T_u}}(\Delta t_u)}{f_{T_u}(\Delta t_u)} + 1 \right) \cdot \frac{1}{\overline{F_{T_d}}(\Delta t_d - 1)}. \tag{3.6}$$

Equation equation 3.6 captures the relation between the LR (i.e., deletion privacy) and the choice of up and down time distributions: (i) the LR is (almost) inversely proportional to the *hazard rate* $f_{T_u}(\Delta t_u)/\overline{F_{T_u}}(\Delta t_u)$ of the up distribution; and (ii) the LR is inversely proportional to the *CCDF* $\overline{F_{T_d}}(\Delta t_d - 1)$ of the down distribution. We need to optimize for these two functions while choosing up and down time distributions for controlling privacy guarantee of $\mathcal{M}_{IW}$.

3.3.3 Availability Property

The intermittent withdrawal mechanism provides deletion privacy at the cost of reducing availability of the post. The post is not visible to the adversary as well as any benign observer during the down periods. Intuitively, the availability of a post is simply the fraction of time the post is visible to an observer. Formally, for mechanism $\mathcal{M}_{IW}(f_{T_u}, f_{T_d})$ the availability is:

$$Availability = \frac{\mu_{f_{T_u}}}{\mu_{f_{T_u}} + \mu_{f_{T_d}}}, \tag{3.7}$$

where $\mu_{f_{T_u}}$ is the mean of the up time distribution f_{T_u} and $\mu_{f_{T_u}}$ is the mean of the down time distribution f_{T_d}.

The LR equation 3.6 and availability equation 3.7, both are functions of the up and down time distributions and thus are correlated. For instance, when posts in the archive are always down (e.g., f_{T_u} is a finite distribution and f_{T_d} is a distribution with infinite mean), the archive has zero availability and perfect privacy (the LR value is 1). On the other hand, when posts in the archive are always up (e.g., f_{T_d} is a uniform distribution with mean 0), the archive has perfect availability of 1 and no privacy

(LR value is ∞). In non-extreme cases, the relationship of availability and privacy is more intricate and depends on specific choices of up and down distributions. We explore this trade-off empirically in Section 3.5.

3.4 *Lethe* Design

We parameterized the security guarantees in section 3.3, but we still need to determine exact specifications for these parameters to effectively control the guarantees. The required parameters include the mean up (down) times for the up (down) distributions as well as choices of PMFs for those distributions. The key design challenge for *Lethe* is: *How to choose suitable parameters for Lethe to give good availability and privacy guarantees?* Here, we resolve this design challenge empirically.

3.4.1 Choosing Distribution Mean Values to Control Availability

Availability of *Lethe*, the average fraction of up time, depends upon the mean for up and down distributions (Equation equation 3.7). While choosing mean values of up and down time distributions, the platform operator needs to decide upon the required availability of the platform. From a practical perspective, we envision that the platform would need the availability to be around 90%.

The absolute value of the down time is also interesting from a usability viewpoint: Hypothetically if an operator expecting 90% availability sets the mean down time as one year and mean up time as nine years, a particular post will be hidden on average for one year. However, a year of down time on average is unacceptable in many real-world scenarios: the users may leave the system if the non-deleted content is not available for such large durations. Therefore, unless otherwise stated, we set mean for down time distributions as one hour and mean for up time distributions as nine hours.

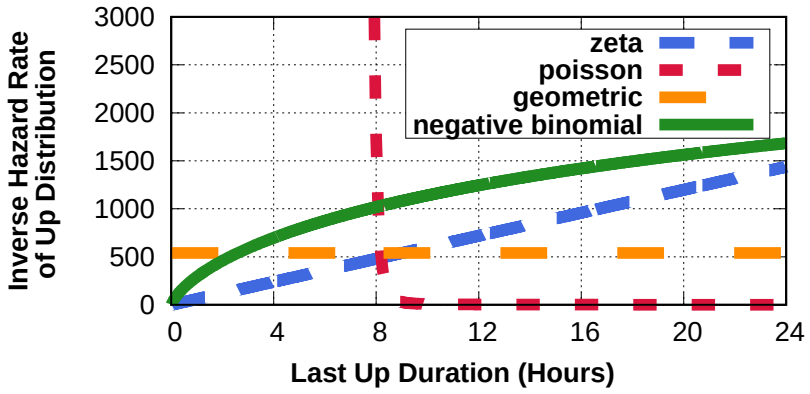


Fig. 3.3.: Variation of inverse hazard rate with time for four choices of up time distributions (with same mean). Increase in the rate signifies increase in LR value.

3.4.2 Selecting Proper Distributions to Control Deletion Privacy

The platform operator needs to control the deletion privacy guarantee of *Lethe* via setting some suitable choices for up and down time distributions (i.e., their PMFs). Her aim is to minimize the LR value.

Geometric distribution is a suitable choice for up time distribution. Recall that the value of LR, is inversely proportional to the hazard rate for the up time distribution (Equation (3.6)) at the last up duration. To select the up times distribution, we considered a wide range of distributions varying in their main characteristics and we present here four distributions with infinite support and the same mean of nine hours—zeta, poisson, geometric and negative binomial [70]—that illustrate the main rationales behind our choice. Figure 3.3 shows the inverse hazard rate for these four choices of up time distributions for different values of last up durations (ranging up to 24 hours). The trends remain similar for longer time durations. Note that, negative binomial distribution requires a parameter called the shape parameter or n, which is set to 0.15 in Figure 3.3 for demonstration. The take away in this figure remains the same for other values of n. The key observation is that only the memoryless geometric distribution has a constant inverse hazard rate for different last up durations. If we take geometric distribution as our up time distribution function, *any* value of last up

duration will have the same effect on the value of LR, i.e., the value of LR will not be affected even when a deletion happens in middle of an up duration (and effectively cut short the original up duration).

However, this is not the case for other distributions—their inverse hazard rate changes with the value of last up duration. Thus, aside from geometric distribution, any other choice of up time distributions poses two problems: (i) the inverse hazard rate (and consequently LR value) would be very high at some point for the last up duration, as evident from Figure 3.3 and (ii) if a post is deleted in the middle of last up duration the LR value will change for that post (since deletion effectively changes the original value of last up duration) compare to the case of no deletion. This phenomenon might provide additional hint to the attacker. Thus we strongly prescribe to use geometric distribution as a suitable choice of up time distribution.

We note that our choice is conservative—for other distributions, there will be instances where inverse hazard rate (and subsequently the LR value) is lower compare to geometric distribution (see Figure 3.3). However, we prefer predictability in the inverse hazard rate of geometric distribution (thus value of LR) for a deployment.

Negative binomial distribution is a suitable choice for down time distribution. Similar to up time distribution analysis, we have experimented with a wide range of distributions for down times. Recall that the LR value, is proportional to the inverse CCDF of a given down time distribution (Equation (3.6)). Figure 3.4 presents the inverse of CCDF of down time distribution in log scale for different values of last down time duration (ranging up to 24 hours) for our four representative choices—zeta, geometric, Poisson and negative binomial [70] (each with a mean of one hour). The trends remain similar for longer time durations. We first observe that for a small down duration, the Poisson distribution has the lowest inverse CCDF value (thus lowest LR). However, at the mean down duration, the value quickly jumps and becomes the highest amongst the different distributions tested. The reason is that most values in the Poisson distribution are concentrated around the mean. Hence, before the mean, the CCDF is close to 1 but quickly after the mean it becomes close

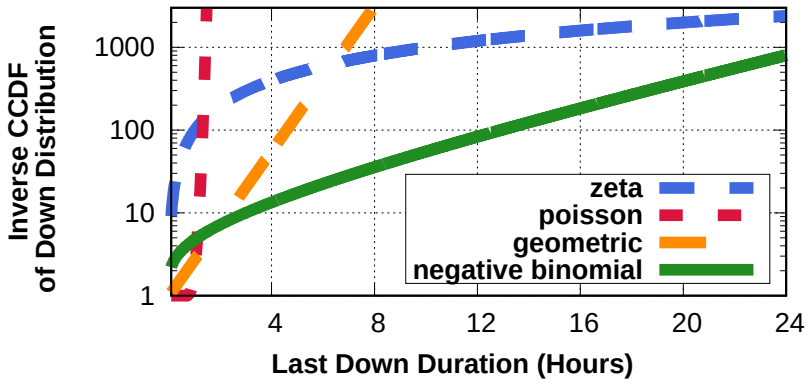


Fig. 3.4.: Variation of inverse of CCDF values in log scaled (proportional to value of LR) for four choices of down distributions in the last down duration.

to zero (intuitively, for Poisson distribution there is a negligible chance that a non-deleted post observes a down time much larger than the mean; thus observing one gives a very strong signal to the attacker).

Similarly, any other distribution with value concentrated around a mode would suffer the same limitation and it is preferable to select a distribution with a decreasing PMF such as geometric, zeta or negative binomial. Amongst those three, geometric has lowest LR for small down time durations, but it increases rapidly for large down time durations. Comparatively, zeta has higher LR for small down time durations and smaller values for large down time duration. This difference is because the geometric distribution has a light tail and its PMF decreases faster whereas the zeta distribution has a heavy tail and therefore assigns higher weights to very large values–hence observing even a very large value has a non-negligible probability to happen under no deletion if the down time distribution is zeta. Finally, the key observation from Figure 3.4 is that the inverse CCDF value of negative binomial distribution provides a balance between these two patterns and thus presents itself as a nice choice for down time distribution.

However, there is a challenge while using negative binomial distribution: it takes another parameter (in addition to mean down time), called the shape parameter and denoted "n". In Figure 3.4, n is set to 0.15 for demonstrating trends, but a practical

deployment of *Lethe* requires a systematic guideline for setting n. Specifically, if the platform operator can have an estimate θ^* for adversary's decision threshold, then it can choose n such that the value of LR is lowest for decision threshold θ^*. The platform operator may even base θ^* on user perception, e.g., operator decides that it is ok, if an adversary finds out deletion of a post after six months or more.

As evident in Figure 3.4, zeta distribution will outperform negative binomial distribution at some point in time. However, we claim that for all the decision thresholds that we have considered (even years), there exists a shape parameter for the negative binomial distribution that provides lower LR value for that threshold compared to zeta distribution. On the other hand, if the platform cannot come up with any reasonable θ^* it might use zeta distribution, since eventually it will perform better than negative binomial distribution; however, this comes at the cost of lowering privacy, i.e., increased LR value, for some period of time. In general, we expect the platform operators, based on their experience, to estimate the range of decision threshold θ^* values reasonably well.

Next, we discuss the procedure to calculate the value of the shape parameter (n) of negative binomial distribution (given the mean down time and the adversary's decision threshold) and its effect on the LR value.

3.4.3 Effect of Negative Binomial Shape Parameter

What is a suitable shape parameter for negative binomial distribution? We present an analytical approach to set an optimal n, given platform operator's estimate of decision threshold θ^*. Since we set the up time distribution as geometric distribution with a constant inverse hazard rate (which we will denote by "c"), Equation equation 3.6 becomes

$$LR = \frac{c+1}{\overline{F_{T_d}}(\Delta t_d - 1)}.$$

Table 3.1.: The best shape parameter n i.e. the lowest LR value when the estimated decision threshold for the adversary is θ^* days. The mean of our negative binomial distribution is one hour.

Estimate of decision threshold is θ^* days	30	60	90	120	150	180
Shape parameter n for lowest LR $\times 10^{-4}$	6	3	2	1.5	1.2	1

Ideally, the platform operator should set n such that, when the adversary's decision threshold is θ^* (i.e., the adversary flags a post as deleted after not observing the post for time θ^* or more), the post has the lowest LR value. In other words, LR value should be lowest when the last down duration is θ^*. Thus, by deciding negative binomial distribution with mean μ_d and shape parameter n, we would want $\overline{F_{T_d}}(\Delta t_d - 1)$ to reach a maximum at $\Delta t_d = \theta^*$. Thus, we take the derivative of $\overline{F_{T_d}}(\Delta t_d - 1)$ with respect to shape parameter n and equate it to 0 at $\Delta t_d = \theta^*$, i.e.,

$$\frac{\partial}{\partial n}\overline{F_{T_d}}(\theta^* - 1) = \frac{\partial}{\partial n} I_{\left(1 - n\frac{1-\mu_d}{\mu_d}\right)}(\theta^*, n) = 0 \tag{3.8}$$

where $I_x(a, b)$ is the incomplete beta integral. Now setting $\mu_d = 1$ hour, we solve for n to determine the best shape parameter for a given value of θ^*.

Table 3.1 shows the best shape parameters for different values of θ^*. An archive operator can choose any of these values according to her choice of θ^* or even calculate suitable values of n for her estimated θ^* using our analytical technique.

Effect of Negative binomial Shape Parameter on LR value: Furthermore, in Figure 3.5, we demonstrate how parameter n impacts the LR value by plotting the LR for some of the shape parameters in table 3.1 (corresponding to θ^* 1, 2 and 6 months). We have set the mean up and down time to nine and one hours respectively. As evident there is no shape parameter that performs best for all the times, however, we can observe that for each of the decision thresholds θ^* in Table 3.1 the corresponding shape parameter has the lowest LR value. We also observe that

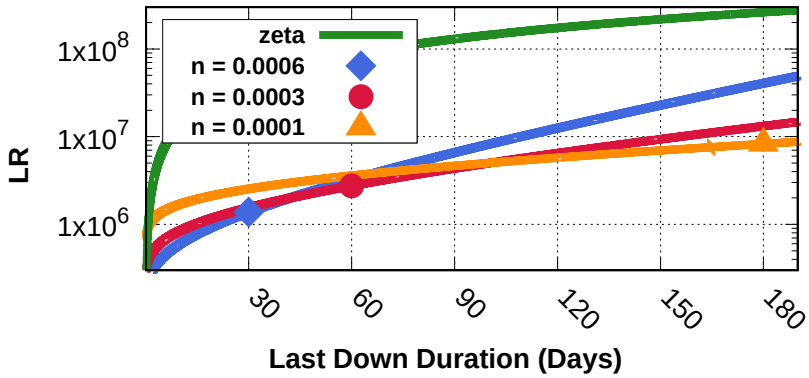


Fig. 3.5.: Variation of LR for zeta distribution and three choices of shape parameters for the negative binomial distribution. The x-axis is showing the last down duration in months and y-axis is showing the LR value in log scaled. The lowest LR value for each of the decision thresholds in table 3.1 is for the corresponding shape parameter. Choosing negative binomial distribution with any of the parameters for the given down durations results lower LR values then choosing zeta as the down distribution.

for all the chosen parameters, LR value for negative binomial is lower than the case of picking zeta as down distribution.

3.4.4 *Lethe* Algorithm

Input: platform availability percentage, mean down time, adversary's decision threshold.

Algorithm:

1. Acquire the mean up time based on the provided mean down time and availability values.
2. Obtain the shape parameter using the derivative procedure based on Equation equation 3.8 using the mean down time and decision threshold from input.
3. Initialize the up and down distributions by passing the mean up and down times along with the shape parameter for the down distribution.
4. Upon a post creation, set the real state of the post to 1 and instantiate the first up period from the up distribution. Set observable state of the post to 1.

5. Upon a toggle signal for a post, if the post was in a up period instantiate a down period from the down distribution and set the observable state to zero; Otherwise instantiate an up period from the up distribution and set the observable state to one.
6. Upon a deletion request for a post from the owner, set the real and observable state to zero and remove the post from the active set (i.e. posts that toggle).

These steps provide a platform operator the basic algorithm to run *Lethe*. However, from a system design point of view a relevant question is—how to efficiently implement these steps? For example, a simple but inefficient (not scalable) implementation for the platform is to just assign one process per post to track the observable state for that post (which is toggled due to *Lethe*). We find that pre-computing future up and down durations and updating them lazily results in efficient *Lethe* implementation. We direct interested readers to Appendix A.1 for an efficient *Lethe* implementation sketch.

Note that we expect the platform to run *Lethe* to provide privacy to their users; however, some platform may even let the post owners themselves enforce the intermittent withdrawal mechanism for their posts; Our analysis remains same in those cases as well.

3.5 Evaluation of *Lethe*

We evaluate the usefulness of *Lethe* by answering a key question: In practice, how hard is it for an adversary to detect deleted posts in presence of *Lethe* (adversarial overhead for identifying deleted posts)?

The posts, which are deleted by the users, will be in a down period for an infinite time. Thus, the down period of such posts will at some point exceed the adversarially chosen decision threshold θ (associated with the LR values) and be flagged by the adversary. These deleted posts, once correctly flagged by an adversary, constitute the true-positives TP_θ. Conversely, when a down period T_d^i for some non-deleted posts

exceed the decision threshold, these falsely flagged posts constitute the false positives FP_θ. On the other hand, the posts that are flagged as non-deleted but will eventually be deleted will be the false negatives FN_θ.

Thus, for a decision threshold θ set by our adversary, if his strategy gives the TP_θ, FP_θ and FN_θ, we measure the adversarial overhead as the precision $Precision_\theta = \frac{TP_\theta}{TP_\theta+FP_\theta}$ and the recall $Recall_\theta = \frac{TP_\theta}{TP_\theta+FN_\theta}$.

To evaluate usefulness of *Lethe* we empirically explore the relation between adversarial precision, availability and decision threshold set by the adversary.

Data Collection: Today, such an intermittent withdrawal mechanism does not exist in the domain of social media and archives. To evaluate the feasibility and performance of *Lethe*, we take Twitter data as a good model platform. To that end, we need numbers for non-deleted and deleted posts on Twitter, and the rate of deletion and new tweets addition in Twitter.

Using reports such as [71, 72], we estimate that there are one trillion non-deleted tweets in the Twitter platform as of 2015. To determine the rates of deletion/addition of tweets, we resort to the 1% random sample provided by Twitter [73]. Specifically, we collected 1% random sample for 18 months (from October 2015 to April 2017). In our 1% random sample, daily on average, 3.2 million tweets are created, i.e. in the whole Twitter 320 million new tweets are created daily. Further, the 1% sample also provides us deletion notices; using those notices we determine how many of archived tweets are deleted daily [50]. We found that on average around 1 million tweets are deleted daily from 1% sample. So daily, on average 100 million tweets are deleted from the whole Twitter archive. Thus, the ratio between the volume of deleted and non-deleted tweets in the Twitter platform is approximately 0.01%. As time passes, this ratio will become smaller (assuming deletion volume will not change too much). Finally, daily 220 million non-deleted tweets are added to the archive.

Experimental setup: For our experiment, we set 1 day as our time unit and pick three system availabilities to experiment—85%, 90% and 95%, all with the mean down time of one hour. Consequently, for 85%, 90% and 95% availability the mean up times

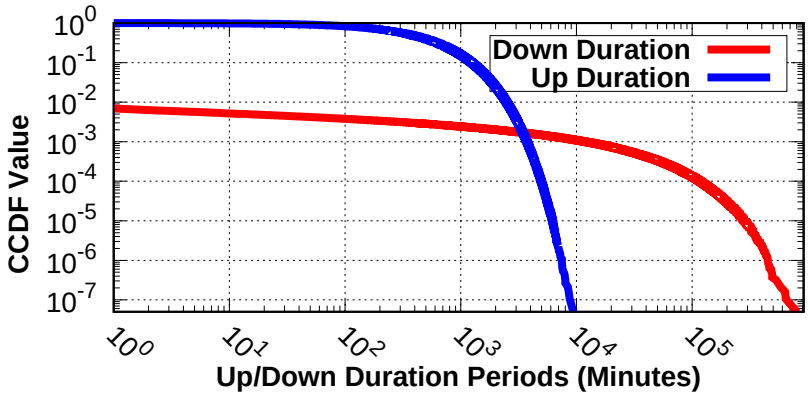


Fig. 3.6.: CCDF value of up and down durations. The up distribution is a geometric distribution with the mean of 9 hours. The down distribution is a negative binomial distribution with the mean of 1 hour.

are respectively 5.7, 9 and 19 hours. Next, we set the up and down time distributions as geometric and negative binomial respectively (as discussed in Section 3.4). We use Table 3.1 to set the shape parameter n for our negative binomial distribution.

To make the *Lethe* simulation feasible with our available resources, we scale down the absolute numbers of deleted/non-deleted tweets to 0.01% of their original values. In other words, we simulate *Lethe* on a scaled down version of Twitter (our archival platform). We consider that our platform contains 100 million non-deleted tweets (0.01% of 1 trillion) already archived in the platform Moreover, 32k tweets are created each day and 10k tweets are deleted (thus adding 22k non-deleted tweets each day) in our platform.

Experimental methodology: For the evaluation of *Lethe* we take the Frequentist design explained in Sections 3.3.2 and 3.4. We note that in order to simulate *Lethe* we don't need the exact timestamps for each post creation and deletion. *Lethe* is applied to the posts as if all of them were created on the first day of experiment. We take 1 day as our time unit and for our simulation, we assume that creation and deletion notifications are received in batch in every time unit. We continue this experiment for 10 years (considering creation and deletion of tweets each day). Figure 3.6 presents the CCDF value of the up and down durations for the chosen distributions at the

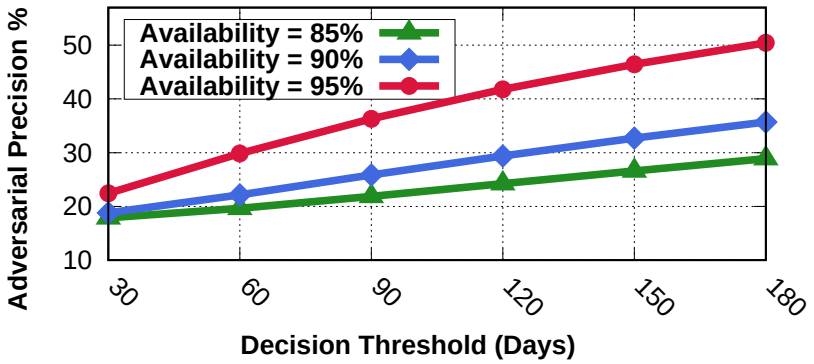


Fig. 3.7.: Variation of adversarial precision against decision threshold periods for different availability values in flag-once scenario. In this scenario, each tweet can be *flagged only once*.

90% availability. More than 99% of the down durations are less than or equal to one minute. The mean up duration in Figure 3.6 is 9 hours and more than 90% of the up durations are longer than 3 hours.

Leveraging our aforementioned experimental set-up we simulate *Lethe* and measure adversarial overhead (i.e. precision and recall) at different decision thresholds. In our set up the true positive for the adversary is simply: number of daily deletions $\times$ (experiment duration - decision threshold). The false positives for our adversary, on the other hand, are non-deleted tweets that get flagged based on the adversary's decision threshold. Further, we note that our adversary might decide to flag the false positives either once or multiple times (i.e., remove flag from a tweet when the tweet is resurrected after a long time and again flag it later). We consider these two scenarios separately.

Adversary investigates a flagged tweet only once: In this scenario, if a non-deleted post gets flagged the adversary will investigate it and after its investigation, it will remove that tweet from his consideration. Thus, the adversary will not consider the post again in the future, even though the post is visible again. We call this scenario *flag-once*. Figure 3.7 is showing the variation of adversarial precision for different decision thresholds in X-axis. As the decision threshold increases the adversary's

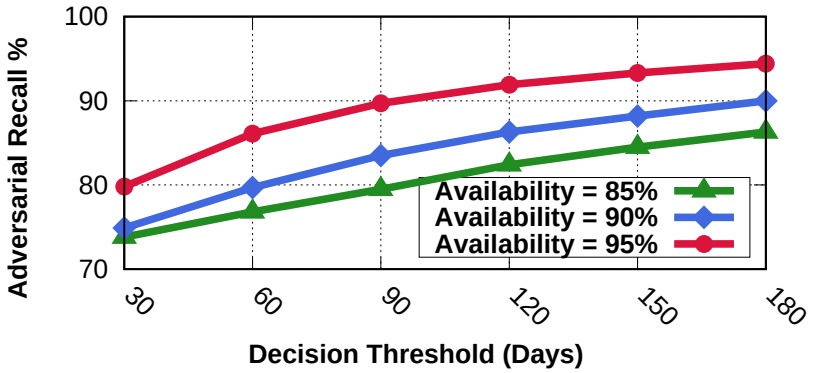


Fig. 3.8.: Variation of adversarial recall against decision threshold periods for different availability values in flag-once scenario.

confidence about a tweet being deleted also increases which result in higher precision values. Note that even for 85% and 90% platform availability the adversarial precision is around (or less than) 35% even when the decision threshold is as high as six months or 180 days , i.e., due to *Lethe* a deletion will go unnoticed for as long as six months.

This scenario checks a flagged post only once and will not consider it later again. Thus, it is possible that a non-deleted post flagged at time t will actually be deleted at a time later than t. So some posts might be deleted but not considered by the adversary, introducing false negatives. Figure 3.8 shows the variation of adversarial recall of deleted posts for different decision thresholds in X-axis. We make two observations. First, the adversary's recall increases with decision threshold. This is because, with increasing threshold, tweets that are not deleted at time t (but deleted later) will have more time to become visible (not getting flagged) before their actual time of deletion. Second, the recall increases with system availability. The reason is that the number of down periods decreases with increasing system availabilities and thus it is less likely to obtain larger down periods to flag tweets. This results in higher recall.

Adversary investigates a flagged tweet multiple times: This scenario is opposite of the previous one in the sense that once a non-deleted tweet has been flagged and investigated it will return to the set of non-deleted. We call this scenario *flag-*

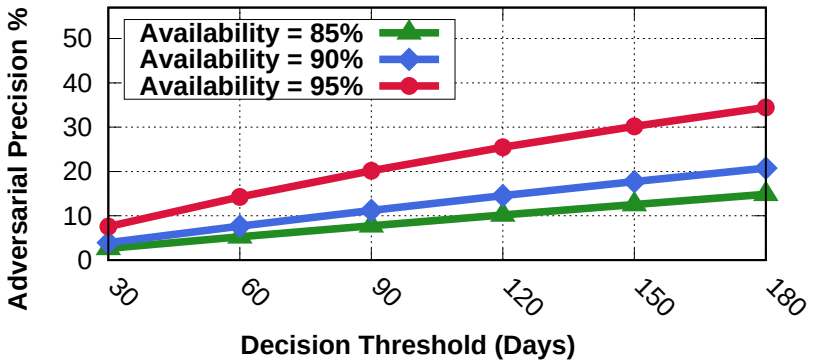


Fig. 3.9.: Variation of adversarial precision against decision threshold periods for different availability values in flag-multi scenario. In this scenario a tweet can be falsely *flagged multiple times*.

multi. The rationality behind this scenario is: it is true that the falsely flagged tweets are not deleted at the current time, but they might be at a future point in time, since sensitivity changes with time and life events. Thus the adversary would also like to take into consideration the real deletion of false positive tweets. Figure 3.9 shows the adversarial precision with varying decision thresholds. Compared to the scenario in Figure 3.7 the adversary has a lower precision for different thresholds for all values of platform availability. The reason is, in this case, a tweet can be flagged multiple times and result in higher false positives. Specifically, in Figure 3.9, for the case of 90% availability, *Lethe* keeps adversarial precision around 20% even when the adversary's decision threshold is as high as 6 months.

In this scenario if a post is flagged it can again be considered for investigation. Since, a deleted post will remain in a down period forever, the adversary will flag it as soon as the decision threshold is over. Thus, all the deletions will be identified eventually. Consequently, in this case there are no type II errors (false negatives) and recall will always be 100%.

Overhead of investigating falsely flagged tweets: Finally, we address one aspect of *Lethe* that we did not consider so far: the astronomical number of falsely flagged tweets that an adversary has to investigate (i.e., extra work) in either of these scenar-

ios. Table 3.2 presents the raw number of non-deleted tweets falsely flagged (i.e. false positives) for both of the aforementioned scenarios. In the worst case, the adversary falsely flagged 13 trillion tweets in the flag-multi scenario when the availability and decision threshold are respectively 85% and 30 days. As Table 3.2 shows, even in the best case, with 95% availability and 180 day decision threshold in the flag-once scenario, the adversary needs to investigate 340 billion falsely flagged tweets.

We have also considered one extreme case—setting the platform availability to 99% (results not shown), i.e., setting the mean down and up time respectively to 1 hour and 99 hours. Although the precision, in that case, is higher compared to the ones in Figure 3.7 and 3.9, we found that even with 99% availability, in the best case (decision threshold 6 months, flag-once scenario) the adversary still needs to investigate 70 billion falsely flagged tweets. In short, We emphasize that the number of falsely flagged tweets is astronomical, and without incurring very high infrastructural cost an adversary can not support such investigation. Thus, much higher decision thresholds are needed for the adversary.

Note that, if an adversary targets a subset of all users, then precision/recall values for both scenarios remain the same and it will only proportionately effect actual number of falsely flagged tweets mentioned in Table 3.2. For example, if the adversary

Table 3.2.: Falsely Flagged Tweets (FFT) with different availabilities, which the adversary needs to investigate under different scenarios. DT denotes decision threshold.

	#FFT (in trillions) for flag-once scenario and diff availability %			**#FFT (in trillions) for flag-multi scenario and diff availability %**		
DT (days)	**85%**	**90%**	**95%**	**85%**	**90%**	**95%**
30	1.64	1.54	1.23	13.05	8.7	4.35
60	1.45	1.24	0.83	6.39	4.26	2.13
90	1.25	1.01	0.62	4.18	2.78	1.39
120	1.09	0.84	0.48	3.07	2.04	1.02
150	0.95	0.71	0.40	2.40	1.60	0.80
180	0.84	0.61	0.34	1.96	1.30	0.65

is targeting 0.1% of all the users then number of falsely flagged tweets in Table 3.2 will be in billions instead of trillions. Furthermore, as the number of users decreases, the prior knowledge of the adversary about the deletion patterns of the users becomes more precise. This advantage results in a more accurate adversarial model that lowers the privacy of the users.

3.6 Effect of *Lethe* in Practice

Platforms would like to make sure that their users are able to normally interact with the content they want and thus utility of their system is preserved when *Lethe* is in place. This guarantee differs from availability since even with 99% availability, the 1% non-available content might be the ones users are interested in. We identify one key factor that captures the distinction between availability and utility—the interaction with content in many platforms go down with time passing. E.g., [74, 75] shows that tweets receive more than 60% of their retweets and replies within the first hour of posting and it quickly becomes negligible as time passes. Thus, in this section we investigate how *Lethe* preserves the utility and not hinder the normal platform operations.

3.6.1 Quantifying Utility of a Platform

In order to evaluate the effect on utility in a real world scenario, we leverage data from Twitter. But first, we need to concretely define the utility of each post as well as the utility of the platform in the context of Twitter.

Utility of a post and of the platform: We take "retweets" as a proxy for interactions (temporal utility) around a tweet. We quantitatively measure the collective utility of the platform to be the *fraction of retweets allowed* when *Lethe* is in place. Although retweets are only a subset of all interactions (other interactions might be replies or user mentions) and may not capture the entirety of interactions, it is still one of the widely employed proxies of activity around a tweet [6, 76, 77].

Collecting a utility dataset: We need to ensure that *Lethe* preserves utility for all normal users of our system. To create a collective random sample of such users, we first take all the users who appeared in the 1% random Twitter sample collected in the first week of November 2011. Then we divide the users into five exponential buckets based on their number of followers (i.e. by their popularity) and randomly sampled 500 users from each bucket. We did this subsampling in mid-February 2016. Thus we end up with 2,500 random users. We collected all the tweets posted by these users (respecting Twitter's limit of 3200 most recent tweets per user) and all the retweets of those tweets on end of February 2016. Out of 2,500, 6 users have made their account private between the time of subsampling and the time of all-tweets collection. So we end up collecting data from rest of the 2,494 users. There are a total of 4,858,014 tweets in our dataset. Among them there are 730,055 tweets with at least one retweet and these tweets have 8,836,706 retweets in total. We use this dataset to check the *Lethe*'s effect on platform utility.

3.6.2 How Does *Lethe* Affect Utility?

We simulate *Lethe* on our utility dataset with the following set-up for *Lethe*'s parameters.

Setup for measuring utility in presence of *Lethe*: We have experimented with setting the platform availability to 85%, 90% and 95%. We again set the mean down time to 1 hour and set mean up times to satisfy the availability requirements. The up and down distributions are geometric and negative binomial respectively. Recall that the negative binomial distribution needs a shape parameter along with the mean. Although we are not considering the adversary in the utility experiment, to be consistent with the privacy analysis, we repeat the experiment for the shape parameters from Table 3.1.

Specifically, we simulate *Lethe* for each of the posts in our utility dataset. Note that, an original retweet happening in a down duration (i.e., when the tweet is hidden)

Table 3.3.: Utility for Twitter in presence of *Lethe* operating with different availabilities and different decision thresholds. All cases the utility of the system is above 99%, and as the availability increases the utility increases. DT stands for Decision Threshold.

DT (days)		**30**	**60**	**90**	**120**	**150**	**180**
Availability	85%	99.25	99.46	99.55	99.61	99.63	99.68
	90%	99.50	99.66	99.72	99.76	99.79	99.82
	95%	99.76	99.83	99.87	99.89	99.90	99.91

is essentially missed and thus platform utility is affected. However, retweets happening in an up duration essentially remain unaffected. We count all the retweets that would have been missed if *Lethe* was in place and calculate the fraction of retweets missed due to *Lethe*. Note that, here we do not consider the effect of missed retweets on future retweets, modeling such effect are part of our future work. Finally, the utility of our system will be simply *1 - fraction of retweets missed*.

***Lethe* has minimal effect on system utility:** Table 3.3 shows the utility of the platform in presence of *Lethe* with varying decision thresholds (for each of them the optimal shape parameter is used). The table is showing the utility, i.e., the fraction of retweets allowed, for 85, 90 and 95% availability. For each of the availabilities, we have chosen six different decision thresholds with their corresponding shape parameter from Table 3.1. The key observation is: for all the cases the utility is quite high. Difference between the utilities are at most 0.5% for different availabilities, and if 99% utility is sufficient for the platform, the platform can simply choose 85% availability over 95% to provide better privacy to the users while maintaining utility.

In summary, *Lethe* can indeed hide deletion of users while having minimal effect on platform utility. For a successful *Lethe* deployment, even 85% or less availability might provide a good trade off between privacy, availability, adversarial overhead and platform utility.

3.7 Enhancements and Discussion

Real-world restricted adversary. In this work, we considered an adversary that can consistently observe each and every post of our platform and has full access to the *Lethe* up/down distribution parameters. However, a real-world adversary will have a much more restricted view of the platform (e.g., Twitter normally allows the developers to collect only 1% random sample of their data) or even of the *Lethe* deployment (e.g., the adversary has to estimate the exact parameters of up/down distribution). Further, in the real world, non-state-level adversaries will be severely limited by computing power and memory. Hence a possible extension of *Lethe* is to restrict the adversary model (i.e., capabilities of the adversary) with practical restrictions on the adversary's resources and considering the estimation overhead of *Lethe* parameters. The privacy guarantees provided by *Lethe* will significantly improve for such restricted, real-world adversaries.

Providing privacy guarantees based on users' needs. We note that by choosing different up/down time distributions, a platform operator can provide a range of privacy guarantees for *Lethe*. For example, if a user needs privacy specifically for 2 or 3 days (e.g., during an uprising) then the system operator can provide short-term privacy by choosing appropriate distributions (where LR value is very low for a short term, then increase rapidly). On the other hand, some celebrity might want long term privacy, where the privacy guarantee is not very high, but it is relatively stable over time. In other words, another possible extension will be to match users' need for privacy by simply tweaking the parameters and distributions in *Lethe*. The privacy guarantees can further be improved in case a user does not mind deleting their content only in down periods. Recall that, we choose geometric distribution as a suitable up distribution primarily since it enables the users to delete their content in both up and down time durations without any effect on the privacy. In case post deletions are restricted only to down durations, we can also explore other choices for up distributions.

Will six month be sufficient?. *Lethe* provides plausible deniability guarantees for a deletion even after 3 to 6 months of deletion. We argue that delaying an adversary 3 to 6 months to detect deletions might be sufficient in many scenarios. The reason is twofold: (i) Recent work [78] modeled users of social platforms as limited memory information processing actors; these actors care less and less about old information. In fact, this model is supported by the phenomena that almost all large social media sites today show the posts in reverse chronologically. (ii) Usually, curious people may focus on some specific user's posts related to some offline (i.e., physical work) event (e.g., in the case of the SNL cast member [25], it was her joining the SNL); however due to the very same reason the user in focus might decide to delete her posts at that time. If *Lethe* can delay the revelation of this deletion even for a few days, it should be sufficient to dissuade the observers.

Opt-outs and Delayed Execution. In some cases, users wish to maintain uninterrupted availability of some of their posts infinitely (e.g., pinned tweets on Twitter) or for the first few days. *Lethe* can easily skip such posts specifically marked by the user. Although these posts do not affect privacy and only improve availability, they can improve adversarial precision: such posts are hardly deleted and thus, their continuous presence will result in lesser false positives. Nevertheless, given the very high utility provided by *Lethe*, we expect the number of such posts to remain limited.

Difference between Deletion Privacy and Differential Privacy. Our notion of deletion privacy has parallels with differential privacy [69] in that we consider the ratio of likelihood of observed states, but there is also a subtle difference. The privacy parameter defined in Definition 3.3.2 depends on the specific observed states $\mathcal{O}$. This is in contrast with differential privacy where the relevant ratio e^{ϵ} (for the parameter ϵ) is defined as a worst-case bound *for all* possible observations. The reason for choosing this definition instead of differential privacy is that it is not possible to find a meaningful bound on the ratio in Equation equation 3.2 valid for all observations: as time-since-deletion increases, the adversary becomes more certain about deletion. In

short we can interpret our deletion privacy definition as a way to capture the certainty of an adversary for detecting post deletion *with* his observed states over time.

Deception for Intrusion Detection and Surveillance Systems. *Lethe* can have interesting applicability beyond the content deletion scenario. Consider an intrusion detection or surveillance system that continuously monitors accesses to a system. Assume an intruder with a side channel that allows him to determine if the system is not functioning for maintenance, power outage or crash. The intruder wishes to exploit this side channel to attack the system; nevertheless, the attack might be time-consuming, and the stakes can be very high such that he does not like to get caught in action. *Lethe*'s approach can be used in this context as a deceptive technology, deterring the intruder even when the system goes down. It will be confusing for the intruder as it cannot determine if the system is in a sleep mode due to *Lethe* or has crashed. Interestingly, this approach will also be helpful towards making the surveillance system energy-efficient as it will not have to be online and operate constantly.

3.8 Concluding Remarks

In the world with perfect and permanent memory, we are in dire need of mechanisms to restore the ability to forget. Against an adversary who can persistently observe a user's data, the user's deletions make her more vulnerable by directly pointing the adversary to sensitive information. In this work, we have defined, formalized, and addressed this problem by designing *Lethe*.

In particular, we have formally defined a novel intermittent withdrawal mechanism, quantified its privacy, availability, and adversarial overhead guarantees in the form of a tradeoff. We leverage this mechanism to design *Lethe* which provides users deniability for their deletions while having very little impact on the system availability against an extremely powerful adversary having complete knowledge about the archival platform. Still, even in the case of such an adversary, leveraging real-world

data we have demonstrated the efficacy of *Lethe* in providing a good tradeoff between privacy, availability, adversarial overhead and platform utility. For example, we have shown that while maintaining 95% availability and utility as high as 99.7%, we can offer deletion privacy for up to 3 months from the time of deletion while still keeping the adversarial precision to 20%.

4. DECEPTIVE DELETIONS FOR PROTECTION OF DAMAGING POSTS

In Chapter 3, we observed that asking the users not to post regrettable content on social platforms in the first place may seem like a good first step. However, users cannot accurately predict what content would be damaging to them in the future (e.g., after a breakup or before applying to a job). Zhou et al. [79] and Wang et al. [80] propose multiple types of classifiers (Naive Bayes, SVM, Decision Trees, and Neural Networks) to detect regrettable posts using users' history and to proactively advise users even before the publication of posts. However, this proactive approach cannot prevent users from publishing future-regrettable posts. It is inevitable to focus on *reactive* mechanisms to assist users with protecting their post deletions.

In our first deletion mechanism proposal Chapter 3 we proposed an intermittent withdrawal mechanism to tackle this challenge of hiding user-initiated deletions. Lethe offers a deniability guarantee for user-initiated deletions in the form of an availability-privacy trade-off and ensure that when a post is deleted, the adversary cannot be immediately certain if it was actually deleted or temporarily made unavailable by the platform. The trade-off could be useful for future social and archival platforms; however, in current commercial social media platforms like Twitter, sacrificing even a small fraction of availability for *all the posts* may be undesirable.

To this end, our next research question is straightforward, yet highly relevant—*can we enhance the privacy of the deleted and possibly damaging posts at scale without excessively affecting the functionality of the platform?*

Contributions. In this chapter, we make the following contributions.

First, we demonstrate the impact of deletion detection attacks by performing a proof-of-concept attack on real-world social media posts to identify damaging content.

Specifically, we use a crowdsourced labeled corpus of deleted posts from Twitter to train an adversary (a classifier). We demonstrate that our adversary is capable of detecting damaging posts with high probability (an increase of 27 percentage points in its F-score). Thus, it is feasible for the adversary to use automated methods for detecting damaging posts on a large scale. In fact, we expect systems such as Fallait Pas Supprimer [27] to employ analogous learning techniques soon to improve their detection.

Second, to overcome the problem of detecting damaging deletions, we introduce a novel deletion mechanism, Deceptive Deletions, that raises the bar for the adversary in identifying damaging content. Given a set of damaging posts (i.e., posts that adversary can leverage to blackmail the user) that users want to delete, the Deceptive Deletion system (also known as a challenger) carefully *selects* k additional posts for each damaging post and deletes them along with the damaging posts. The system-selected posts, henceforth called the *decoy posts*, are taken from a pool of posts (i.e., non-damaging non-deleted) provided by volunteers. The deletions of the decoy posts will confuse the adversary in distinguishing damaging posts from the (non-damaging) decoy posts. Intuitively, Deceptive Deletion is more effective if the selected decoy posts are similar to the damaging posts. These two opposite goals create a minmax game between the adversary and the challenger that we further analyze.

Third, we introduce the Deceptive Learning Game, which formally describes the minmax game between the adversary and the challenger. We start by considering a static adversary that tunes the parameters of its system (e.g., classifier for determining the damaging posts) up until a certain point in time. However, powerful adversaries are adaptive and continually tune their models as they obtain more deletions including the decoy deletions made by the challenger. Therefore, in the second phase, we consider an adaptive adversary and describe the optimization problem of the adaptive adversary and challenger as a minmax game.[1]

[1]See [81] for another example of a minmax game in adversarial learning.

We identify conditions under which either only the adaptive adversary or only the challenger provably wins the minmax game and discuss the scenarios in-between these two extremes. To the best of our knowledge, this is the first attempt to develop a computational model for quantitative assessment of the damaging deletions in the presence of both static and adaptive adversaries.

Finally, we empirically demonstrate that with access to a set of non-damaging volunteered posts, we can leverage Deceptive Deletions to hide damaging deletions against both static and adaptive adversary effectively. We use real-world Twitter data to demonstrate the effectiveness of the challenger. Specifically, we show that even when we consider only two decoy posts per damaging deletion, the adversarial performance (F-score) drops to 42% from 75% in the absence of any privacy-preserving deletion mechanism.

4.1 Background and Related Work

In addition to the mentioned content deletion mechanisms prevalent in today's social media (see Section 3.1), we observe that Tianti et al. [46] offer intuitions for predicting posts deletions on Instagram with the goal of managing the storage of posts on the servers: Once a post is archived, it becomes computationally expensive to erase it; thus, predicting deletions can help in reducing the overheads of being compliant with the "right to be forgotten" regulations. These predictions in the non-adversarial setting, however, does not apply to our minmax game between the adversary and the challenger.

Recently Garg et al. [82] formalize the right to be forgotten using platforms as a cryptographic game. While being interesting, their definitions and suggested tools such as history-independent data structures are not applicable to our setting where the adversary has continuous and complete access to the collected data.

4.1.1 Obfuscation Using Noise Injection

There has been a line of work in the area of (non-cryptographic) private information retrieval [83–86] that obfuscates the users' interest using dummy queries as noise to avoid user profiling. Howe et al. proposed TrackMeNot [83, 87], which issues randomized search queries to popular search engines to prevent the search engines in building any practical profile of the users based on their actual queries. GooPIR [85] is a similar work that uses a Thesaurus to obtain the keywords to constructs $k-1$ other queries (dummy ones) and submit all k queries at the same time. This way, timing attacks by the search engines are eliminated. However, it only addresses single keyword searches; these schemes do not address full-sentence searches. Murugesan et al. propose "Plausibly Deniable Search" (PDS) [84] that analogous to GooPIR generates $k-1$ dummy queries using latent semantic indexing based approach. In their mechanisms, each real query is converted into a canonical query which protects against deanonymization attacks based on typos and grammar mistakes.

We note that all of the systems mentioned so far consider hiding each query separately. However, a determined adversary may be able to find a user's interests by observing a sequence of such obfuscated queries. Multiple works have investigated such weaknesses [86, 88, 89].

Some relatively new techniques further try to overcome these shortcomings by *smartly* generating the $k-1$ queries. For example, Petit et al. proposed PEAS [90], where they provide a combination of unlinkability and indistinguishability. However, apart from introducing an overhead for encrypting the user queries, their method also requires two proxy servers that are non-colluding, hence weakening the adversarial model. K-subscription [91] is yet another work that proposes an obfuscation based approach that enables the user to follow privacy-sensitive channels in Twitter by requiring the users to follow $k-1$ other channels to hide the user interests from the microblogging service. However, the K-subscription has a negative social impact for the user as the user's social connections will see the user following these dummy

channels. These shortcomings, both social and technical, motivated our particular design decision for Deceptive Deletions.

4.1.2 Adversarial Machine Learning

Traditional adversarial learning settings [92] involve two players: a classifier and an attacker. The classifier seeks to label the inputs whereas the attacker tries to *modify* the inputs such that the classifier will misclassify them. Adversarial machine learning has also been used as a defense with the roles reversed where the defender attacks the adversary's classifier. For example, in [93], the adversary tries to extract users' private attributes from their public data while the defender modifies the public data of the users in order to fool the adversary's classifier. Our setting is different in that we are *not allowed to modify* the examples. Instead, the challenger wishes to attack the adversary's classifier by injecting *hard-to-classify* examples into the adversary's train/test datasets (i.e., the deletion set). A key constraint for the challenger is that it has to *select the examples from a preexisting set* of volunteered posts. This is because the challenger can only delete existing posts, and cannot generate fake posts.

As we detail in the subsequent sections, the adaptive adversary trains on these injected examples as well. With a faint relation to our work, data poisoning attacks [94, 95] focus primarily on injecting poisoned samples into a classifier's training data with the sole purpose of deteriorating the classifier. In contrast, our primary goal is to inject examples only into the adversary's test dataset, especially because data poisoning attacks typically require the freedom to arbitrarily construct data samples, which is not possible in our setting.

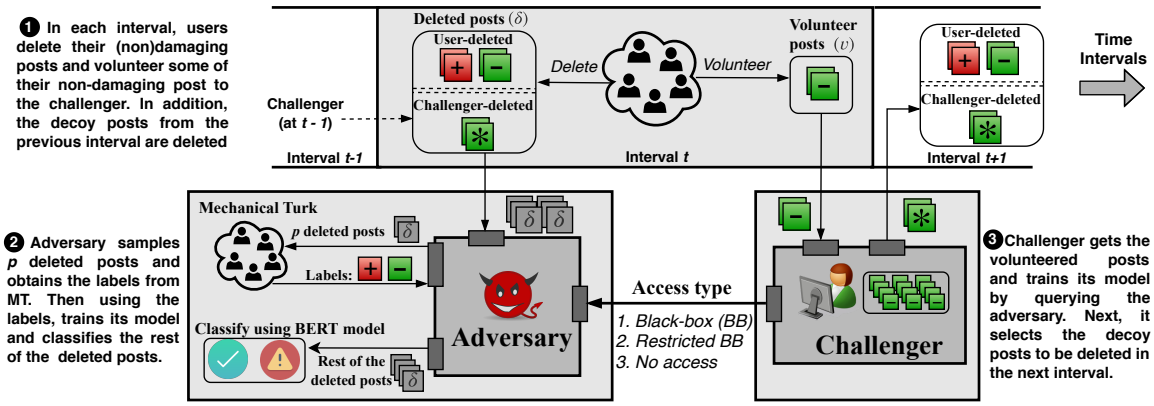


Fig. 4.1.: Overview of Deceptive Deletions. In each interval, the deletions are shown by gray squares with 'δ'. The deleted posts could be of three types: users' damaging deletions shown by red squares with '+', users' non-damaging deletions shown by green squares with '−' and challengers' decoys posts shown by green squares with '$*$'. Further, we denote the volunteer posts offered to the challenger during each interval by green squares with '−' to indicate that they are non-damaging.

4.2 System Model and Overview

4.2.1 System

We consider a data-sharing *platform* (e.g., Twitter or Facebook) as the public bulletin board where individuals can upload and view content. *Users* are the post owners that are able to publish/delete their posts, and view posts from other users. In this work, we consider discrete time intervals in which the users upload and delete posts (Figure 4.1 ❶). A time interval could be as small as a minute or even a week, depending on the platform. We define two types of posts.

- **User-deleted posts** A user could delete a post for two primary reasons [6, 19, 50]:
  - **Damaging posts:** the post contained *damaging* content to the user's personal or professional life, or
  - **Non-damaging posts:** the post was out-dated, contained spelling mistakes, etc.

An adversary's goal is to find the damaging posts among all the deleted ones that could be used to blackmail the corresponding owners of the post.

- **Volunteered posts** We consider a subset of non-deleted posts that users *willingly* offer to be deleted to protect the privacy of other users whenever needed. These volunteered posts are non-damaging and cannot be used by the adversary to blackmail the user of the post. We discuss the challenges of obtaining volunteered posts in Section 4.5.

  A challenger's goal is to select a subset of volunteered posts (i.e., non-damaging) and delete them such that the aforementioned adversary is unable to distinguish between the damaging and the non-damaging post deletions. We denote the posts selected by the challenger as ***decoy posts****.*

Notation. We use a subscript t to denote the time interval and superscripts $\delta, +, \mathrm{v}, *$ to denote the post type. In particular, $\mathbb{D}_t$ is all the uploaded and deleted posts in time interval t. Then we denote all the deleted posts (user- and challenger-deleted) in that interval as $\mathbb{D}_t^{\delta}$, the **damaging posts** as $\mathbb{D}_t^{+}$, and **volunteered posts** by $\mathbb{D}_t^{\mathrm{v}}$. The **decoy posts** that a challenger selects for deletion to fool the adversary is denoted by $\mathbb{G}_t^{*}$. Note that $\mathbb{G}_t^{*} \subseteq \mathbb{D}_t^{\mathrm{v}} \subseteq \mathbb{D}_t \backslash \mathbb{D}_t^{\delta}$.

4.2.2 Adversary's Actions and Assumptions

Task. At a given time interval, the task of the adversary is to correctly label all the deleted posts as being damaging to the post-owner or not. Similar to the threat model in Lethe Section 3.2.4, we *do not* focus on local attackers (or stalkers) targeting individuals or small groups of users.[2] Our global adversary instead seeks damaging

[2]Such stalkers can easily label their posts manually, and protecting against such an attack is extremely hard if not impossible. For example, consider the case that a stalker continuously takes snapshots of its targeted user profile with the goal of identifying the user's deletions. With its background/auxiliary information about the user (i.e., knowing what contents are considered sensitive to the target), the stalker can effectively identify the damaging deletions. We claim that, in the current full-information model, protection against such a local adversary is impossible.

deletions on a large scale, rummaging through all the deleted posts to find as many damaging ones as possible. Fallait Pas Supprimer [27] (from Chapter 4) is a real-world example of the global adversary.

Data access. At any given time interval, we assume that the adversary is able to obtain all the deleted posts by comparing different archived snapshots of the platform. Although this strong data assumption benefits the adversary tremendously, we show in Section 4.4.4 that Deceptive Deletions can protect the users' damaging deletions. Further, we discuss a few techniques that the platforms can use to restrict and limit the adversary's access to the users' profile in Section 4.5.3.

Labels. Our global, non-stalker adversary is not able to obtain the *true label* (damaging or non-damaging) of the post from the user. Instead, the adversary uses a crowdsourcing service like Mechanical Turk (MTurk) [96] to obtain a proxy for these true labels. Although the labels obtained from the Mechanical Turkers (MTurkers) reflect societal values and not the user's intention, following previous work [80], we assume they closely match the *true labels* in our experiments. This is reasonable as the adversary can expend a significant amount of effort and money to obtain these *true labels*, at least for a small set of posts, that will ultimately be used to train a machine learning model.

Budget. Since there is a cost associated with acquiring label for each deleted post from the MTurkers, the aim of the adversary is to *learn to detect the damaging deletions* under a budget constraint. We consider two types of budget constraints:

- **limited budget** where the adversary can only obtain the labels for a fixed number of posts B_{static}, and
- **fixed recurring budget** where the adversary obtains the labels for a fixed number of posts B_{adapt} *in each interval.*

The adversary with a limited budget is called the **static adversary** since it does not train after exhausting its budget. On the other hand, the adversary with a fixed

recurring budget keeps adapting to the new deletions in each time interval, and hence is dubbed the **adaptive adversary**.

Player actions. At every time interval t, the adversary obtains a set of posts $\mathbb{A}_t^\delta$ for training by sampling part of the deleted posts, say p, from $\mathbb{D}_t^\delta$, an operation denoted by $\mathbb{A}_t^\delta \overset{p}{\sim} \mathbb{D}_t^\delta$. The adversary uses MTurk to label the sampled dataset $\mathbb{A}_t^\delta$. After training, the task of the adversary is to classify the rest of the deleted posts of that time interval. Additionally, as the adversary gets better over time, it also *relabels* all the posts deleted from the past intervals. The test set for the adversary is all the deleted posts from current and previous time intervals that were not used for training; i.e., $\bigcup_{t' \leq t}(\mathbb{D}_{t'}^\delta \setminus \mathbb{A}_{t'}^\delta)$. Figure 4.1 ❷ shows the adversary's actions.

Note that although an adaptive adversary can sample $p = B_{\text{adapt}}$ deleted posts at *every* time interval and use MTurkers to label them, a static adversary can only obtain the labels until it runs out of the limited budget (after $\tau = B_{\text{static}}/p$ time intervals). After this period, a static adversary does not train itself with new deleted posts.

Performance metrics. The adversary wishes to increase *precision* and *recall* for the classification of deleted posts into damaging and non-damaging sets. At every time interval t, we report adversary's F-score[3] over the test set described above: deleted posts of all the past intervals, i.e., $\bigcup_{t' \leq t}(\mathbb{D}_{t'}^\delta \setminus \mathbb{A}_{t'}^\delta)$.

4.2.3 Challenger's Actions and Assumptions

Task. In the presence of an adversary as described above, the task of a challenger is to obtain volunteered posts (i.e. non-damaging and non-deleted posts) from users, select a subset of these posts and delete them in order to fool the adversary into misclassifying these *challenger-deleted* posts as damaging. The challenger is honest, does not collude with the adversary, and works with the users (data owners) to protect

[3]F-score $= 2 \cdot precision \cdot recall/(precision + recall)$

their damaging deletions. Other than the platforms themselves, third party services such as "tweetDelete" [97] can take the role of the challenger as well.

Data access. The challenger can be implemented by the platform or a third-party deletion service [97–99], that has access to the posts of the users. Additionally, we assume that there are users over the platform who volunteer a subset of their non-damaging posts to be deleted anytime (or within a time frame) by the challenger, possibly, in return for privacy benefits for their (and other users') damaging deletions.

Labels. The challenger is implemented as part of the platform (or a third-party service permitted by the user). Thus, unlike the adversary that obtains proxy labels from crowdsourcing platforms, it has access to the *true labels*— damaging or non-damaging, from the owner of the post. This is easily implemented: before deleting a post, the user can specify whether the post is damaging (and needs protection). This access to the *true labels* is an advantage that challenger has over the adversary and hence can train more accurate models.

Access to the adversary. The challenger not only knows the presence of a global adversary trying to classify the deleted posts into damaging and non-damaging posts but also can observe its behaviour.[4]. As a result, we consider three types of accesses to the adversary:

- **no access** where the challenger has no information about the adversary.
- **monitored black-box access with a recurring query budget of** B_g where the challenger can obtain the adversary's classification probability for a limited number of posts B_g every time interval, but the access is *monitored*, i.e., the adversary can take note of every post queried and treat them separately.
- **black-box access** where the challenger can obtain the adversary's classification probabilities for any post.

Here, *no access* is the weakest assumption that defines the lower-bounds for our challenger's success. Nevertheless, we expect the challenger to have some access to the

[4]Fallait Pas Supprimer [27] posts all its output on Twitter itself.

adversary's classification. An *unrestricted* black-box access serves as an upper bound for the challenger assuming that it can train a precise surrogate model of the adversary's classifier using its own training data. While employing such a surrogate model is common practice in the literature [100, 101], it can be hard to obtain in real world without knowing the adversary's exact architecture and training data. Our monitored black-box assumption with a recurring query budget (henceforth, interchangeably called the restricted black-box access) balances practicality of the access versus the feasibility of defending against an adversary with that access. In Section 4.3, we introduce three challengers (oracle, D^2 and random) corresponding to the three types of accesses.

Player actions. At every time interval t, the challenger receives new volunteer posts from the users and adds them to a set that stores the volunteered posts collected up until this point. Next, based on the type of access, it obtains the adversary's classification probabilities for some number of volunteer posts (the number is dependent on the access which we detail in Section 4.3). Finally, it selects *decoy posts*, a subset of the volunteered posts collected up until this point and deletes these posts in interval t+1 (hence the adversary sees these *challenger-deleted* posts in interval t+1 as part of the deleted set $\mathbb{D}^{\delta}_{t+1}$). Figure 4.1 ❸ shows the challenger's actions.

Performance metrics. The challenger, in direct contrast to the adversary, wishes to *decrease* adversary's precision and recall for the classification of deleted posts. Adversary's precision will decrease if it classifies the injected decoy posts as damaging (increased false-positives). On the other hand, adversary's recall will decrease if it learns to be conservative in order to ignore the decoy posts (increased false-negatives).

4.3 The Deceptive Learning Game

The deceptive learning game is a two-player zero-sum non-cooperative game over time intervals $t = 1, 2, ...$ (units) between an adversary who wishes to *find* users' damaging deletions, and a challenger who wishes to *hide* the said damaging deletions. The

challenger achieves this by deleting volunteers' non-damaging posts as decoys. While the adversary's goal is to maximize its precision/recall scores on the classification task, the challenger's goal is to minimize them.

We denote each post by (x, y), where $x \in \mathbb{X}$ represents the features of the post (i.e., text, comments, etc.) and $y \in \{0, 1\}$ denotes its true label such that $y = 1$ if the post is damaging and $y = 0$ if it is non-damaging. In the following subsections, we describe the actions of each player in the time interval t.

4.3.1 Adversary

We denote the adversary's classifier at the beginning of interval t by $a(\ \cdot\ ; \theta_{t-1}) : \mathbb{X} \to [0, 1]$ parameterized by θ_{t-1} such that $a(x; \theta_{t-1}) \coloneqq P(\hat{y} = 1 \mid x; \theta_{t-1})$ is the predicted probability of the post x being damaging. The adversary collects all the deletions that happen in this interval (i.e., $\mathbb{D}_t^\delta$) and samples p posts, denoted by $\mathbb{A}_t^\delta$. The adversary then uses MTurk to obtain a proxy for the true labels of these p posts.

The adversary uses this labeled training data in the following optimization problem to update its parameters,

$$\theta_t = \arg\min_\theta \mathcal{L}_{\text{NLL}}(\theta; \mathbb{A}_t^\delta)\ , \tag{4.1}$$

where $\mathcal{L}_{\text{NLL}}$ is the standard negative log-likelihood loss for the classification task, given by,

$$\mathcal{L}_{\text{NLL}}(\theta; \mathbb{A}_t^\delta) = \sum_{(x,y)\in\mathbb{A}_t^\delta} -y \log\left(a(x; \theta)\right) - (1 - y) \log\left(1 - a(x; \theta)\right)\ .$$

After training, the adversary uses the trained model $a(\ \cdot\ ; \theta_t)$ to predict the labels of the rest of the deleted posts of time interval t, i.e., $\mathbb{D}_t^\delta \setminus \mathbb{A}_t^\delta$ along with all the deleted posts that it had already predicted in the past. This way the adversary hopes to capture damaging posts that were missed earlier. Hence, we report the adversary's performance on all the past deletions (not including the training data): $\bigcup_{t' \le t}(\mathbb{D}_{t'}^\delta \setminus \mathbb{A}_{t'}^\delta)$.

Algorithm 1: Adversary

input : $\mathbb{D}^{\delta}$; `/* Deleted posts in this interval */`

1 Sample p posts $\mathbb{A}^{\delta} \overset{p}{\sim} \mathbb{D}^{\delta}$;
2 Query MTurk and obtain labels for $\mathbb{A}^{\delta}$;
3 Obtain optimal parameters θ^* by solving Equation (4.1) ;
4 **return** $a(\ \cdot\ ; \theta^*)$

Static vs Adaptive Adversary. Since the static adversary has a limited budget, first it chooses the number of time intervals for training, say τ, and accordingly samples $p = B_{\text{static}}/\tau$ posts for querying MTurk to obtain labels.

The adaptive adversary has a fixed recurring budget of B_{adapt} and hence, can sample $p = B_{\text{adapt}}$ posts every interval. This allows the adaptive adversary to train itself with new training data (of size B_{adapt}) every interval indefinitely. Algorithm 1 depicts adversary's actions within a time interval (subscript t removed for clarity).

4.3.2 Challenger

In the presence of such an adversary, the challenger's goal is to collect volunteered posts (non-damaging) from users and selectively delete these posts in order to confuse the adversary.

As described before, $\mathbb{D}^{\text{v}}_t$ is the set of posts volunteered by users in the time interval t. Let $\mathbb{G}^*_{\leq t}$ be the set of decoy posts deleted by the challenger in the current and past intervals. At the end of interval t, the challenger collects all the volunteered posts from the current and past intervals (except the posts that it has already used as decoys). The *available* set of volunteered posts is denoted by $\mathbb{D}^{\text{v}}_{\leq t} \equiv (\bigcup_{t' \leq t} \mathbb{D}^{\text{v}}_{t'}) \setminus (\bigcup_{t' \leq t} \mathbb{G}^*_{t'})$. Note that $(x, y) \in \mathbb{D}^{\text{v}}_{\leq t} \implies y = 0$, i.e., the volunteered posts are non-damaging by definition. For ease of notation, let $N^{\text{v}} := |\mathbb{D}^{\text{v}}_{\leq t}|$ be the number of volunteered posts collected till interval t.

Then, the goal of the challenger is to construct the decoy set $\mathbb{G}^*_{t+1} \subseteq \mathbb{D}^{\text{v}}_{\leq t}$ and delete these posts during the next time interval $t{+}1$ in order to fool the adversary into

misclassifying these challenger-deleted non-damaging posts as user-deleted damaging posts. Formally, we want to choose K decoy posts (denoted by a K-hot vector $\boldsymbol{w}$) that maximizes the negative-log likelihood loss for the adversary's classifier, given by the following optimization problem,

$$\begin{aligned} \boldsymbol{w}^* &= \arg\max_{\boldsymbol{w}} V(\boldsymbol{w}; \mathbb{D}^{\text{v}}_{\leq t}) \\ \text{s.t.} \quad & ||\boldsymbol{w}||_1 = K, \qquad \boldsymbol{w} \in \{0,1\}^{N^{\text{v}}} , \end{aligned} \tag{4.2}$$

where

$$V(\boldsymbol{w}; \mathbb{D}^{\text{v}}_{\leq t}) = \sum_{i=1}^{N^{\text{v}}} -w_i \cdot \log(1 - a(x_i; \theta_t)) , \tag{4.3}$$

and x_i is the i-th volunteered post in $\mathbb{D}^{\text{v}}_{\leq t}$. The cost function $V(\boldsymbol{w}; \mathbb{D}^{\text{v}}_{\leq t})$ in Equation (4.3) is simply the negative log-likelihood of the adversary over the set $\mathbb{D}^{\text{v}}_{\leq t}$ weighted by a K-hot vector $\boldsymbol{w}$. Equation (4.3) uses the fact that the set only contains non-damaging posts (i.e., $y_i = 0$).

Consequently, $\boldsymbol{w}^*$ optimized in such a fashion selects K posts from the set $\mathbb{D}^{\text{v}}_{\leq t}$ that *maximizes* the adversary's negative log-likelihood loss. The set of K selected posts can be trivially constructed as $\mathbb{G}^*_{t+1} = \{x_i : i \in \{1, \ldots, N^{\text{v}}\} \wedge w_i = 1\}$. The challenger deletes $\mathbb{G}^*_{t+1}$ over the next time interval $t{+}1$ (hence the adversary sees these posts as part of the deleted set $\mathbb{D}^{\delta}_{t+1}$). Note that the challenger uses the adversary's classifier $a(\ \cdot\ ;\ \theta_t)$ to create decoy posts for $t{+}1$. However, as per Section 4.3.1, in interval $t{+}1$ the adversary first trains over a sample of the deleted posts (including the decoy posts) and updates its classifier to $a(\ \cdot\ ; \theta_{t+1})$ before classifying the rest of the deleted posts of $t{+}1$. Hence, the challenger is always at a disadvantage (one step behind).

Next, we describe three challengers corresponding to the access types discussed in Section 4.2.3: *no access*, *black-box access* and *monitored black-box access with a query budget*.

Random challenger (no access). We begin with the case where the challenger has *no access* to the adversary's classifier and there is no side-information available to the challenger. With no access to the adversary's classification probabilities $a(\ \cdot\ ; \theta_t)$, the optimization problem in Equation (4.2) cannot be solved. We introduce the naive *random challenger* that simply samples K posts randomly from the available volunteered posts $\mathbb{D}^{\text{v}}_{\leq t}$ and deletes them, i.e., $\mathbb{G}^{*}_{t+1} \overset{K}{\sim} \mathbb{D}^{\text{v}}_{\leq t}$. This is the only viable approach if the challenger has no information about the adversary's classifier.

Oracle challenger (black-box access). Next we consider the challenger that has a black-box access to the adversary's classifier with no query budget, i.e., at any time interval t, the challenger can query the adversary with a post x and expect the adversary's predicted probability $a(x; \theta_t)$ in response without the adversary's knowledge. Armed with the black-box access, oracle challenger can simply maximize Equation (4.2) by choosing the top K posts with highest values for $a(x_i; \theta_t)$.

D^2 Challenger (monitored black-box access with query budget B_g). The oracle challenger assumes an *unmonitored* black-box access to the adversary *with an infinite query budget* which can be hard to obtain in practice. In what follows, we relax the access and assume a *monitored* black-box access *with a recurring query budget of B_g*. In other words, queries to the adversary, while being limited per interval, are also monitored and possibly flagged by the adversary. The adversary can simply take note of these queries as performed by a potential challenger, hence negating any privacy benefits from injecting decoy posts. Whenever the adversary sees a *deleted post* identical to one that it was previously *queried* about, it can ignore the post as it is likely non-damaging.

Here we design a challenger, henceforth dubbed D^2, that *trains to select decoy posts* from any given volunteered set. In other words, the D^2 challenger makes use of the monitored black-box access to the adversary only during training. Hence it can be used to find the decoy posts without querying the adversary; for example in a held-out volunteered set (separate from the training set). Additionally, the D^2 challenger queries the adversary for only B_g posts every time interval.

We denote the challenger's model at the beginning of interval t by $g(\ \cdot\ ; \phi_{t-1}) : \mathbb{X} \to \mathbb{R}$ parameterized by ϕ_{t-1}. For a given volunteer post x, $g(x; \phi_{t-1})$ gives an unnormalized score for how likely the post will be mislabeled as damaging; higher the score, higher the misclassification probability.

First, the D^2 challenger samples B_g posts for training from the available volunteered set $\mathbb{D}^{\text{v}}_{\leq t}$ collected till interval t. We denote the train and test sets of the D^2 challenger as $\mathbb{D}^{\text{v,train}}_{\leq t}$ and $\mathbb{D}^{\text{v,test}}_{\leq t}$ of sizes B_g and $N^{\text{v}} - B_g$ respectively. Then, the goal of the D^2 is to find optimal parameters ϕ_t by solving a continuous relaxation of Equation (4.2) presented below,

$$\phi_t = \arg\max_{\phi} \tilde{V}(\phi; \mathbb{D}^{\text{v,train}}_{\leq t}) \tag{4.4}$$

where

$$\tilde{V}(\phi; \mathbb{D}^{\text{v,train}}_{\leq t}) = \sum_{i=1}^{B_g} -\alpha(x_i; \phi, \mathbb{D}^{\text{v,train}}_{\leq t})\ \log(1 - a(x_i; \theta_t))\ ,$$

and

$$\alpha(x_i; \phi, \mathbb{D}^{\text{v,train}}_{\leq t}) = \frac{\exp\left(g(x_i; \phi)\right)}{\sum_{j=1}^{B_g} \exp\left(g(x_j; \phi)\right)}\ ,$$

is a softmax over the challenger outputs for all the examples in $\mathbb{D}^{\text{v,train}}_{\leq t}$. The softmax function makes sure that $0 \leq \alpha(\ \cdot\ ; \phi, \mathbb{D}^{\text{v,train}}_{\leq t}) \leq 1$ and $\sum_{j=1}^{B_g} \alpha(x_j; \phi, \mathbb{D}^{\text{v,train}}_{\leq t}) = 1$. The continuous relaxation in Equation (4.4) allows the D^2 challenger to train a neural network model parameterized by ϕ via backpropagation.

We now show that optimizing the relaxed objective in Equation (4.4) results in the best objective value for Equation (4.2).

Proposition 4.3.1 *For any given volunteered set $\mathbb{D}^v$ with N non-deleted posts,*

$$\max_{\phi} \tilde{V}(\phi; \mathbb{D}^v) = \max_{w_1, \ldots, w_N} V(w_1, \ldots, w_N; \mathbb{D}^v)$$

Algorithm 2: Challenger

input : $\mathbb{D}^{\text{v}}$, K, accessType

1 $\mathbb{G}^* \leftarrow \emptyset$;
2 **if** *accessType = none* **then**
    `/* Random challenger */`
3 $\quad \mathbb{G}^* \overset{K}{\sim} \mathbb{D}^{\text{v}}$;
4 **else if** *accessType = black-box* **then**
    `/* Oracle challenger */`
5 $\quad \mathbb{G}^* \leftarrow \{x_i : x_i \in \mathbb{D}^{\text{v}} \wedge a(x_i;\theta)$ is in the top $K\}$;
6 **else if** *accessType = monitored black-box (budget B_g)* **then**
    `/* D² challenger */`
7 $\quad$ Sample B_g posts for training $\mathbb{D}^{\text{v,train}} \overset{B_g}{\sim} \mathbb{D}^{\text{v}}$;
8 $\quad \mathbb{D}^{\text{v,test}} \leftarrow \mathbb{D}^{\text{v}} \setminus \mathbb{D}^{\text{v,train}}$;
9 $\quad$ Query $a(x_i;\theta)$ for all $(x_i, y_i = 0) \in \mathbb{D}^{\text{v,train}}$;
10 $\quad$ Obtain optimal parameters ϕ^* by solving Equation (4.4) ;
11 $\quad \mathbb{G}^* \leftarrow \{x_i : x_i \in \mathbb{D}^{\text{v,test}} \wedge g(x_i;\phi^*)$ is in the top $K\}$;
12 **return** $\mathbb{G}^*$;

Finally, the D^2 challenger with optimal parameters ϕ_t computes $g(x;\phi_t)$ for all $(x, y = 0) \in \mathbb{D}^{\text{v,test}}_{\leq t}$, and constructs G^*_{t+1} by choosing the examples with top K values for $g(\ \cdot\ ;\phi_t)$. Algorithm 2 shows the actions of the challenger within a time interval (subscript t removed for clarity).

4.3.3 Deceptive Learning Game

Algorithm 3 presents the game between the adversary and the challenger. In each time interval, users independently delete and volunteer posts (line 4). The platform/deletion-service additionally deletes the challenger-selected decoy posts (line 5). The adversary obtains all the deleted posts and queries the MTurk with a small subset of the posts for labels (if the adversary has not exhausted the budget). With this labeled set of deleted posts, the adversary trains its classifier (lines 6-7). The challenger collects new volunteered posts (line 8) and builds decoy posts to be injected

Algorithm 3: Deceptive Game

input : accessType, K

1 $\mathbb{G}_1^* \leftarrow \emptyset$;
2 $\mathbb{D}^{\text{v}}_{\leq 0} \leftarrow \emptyset$;
3 **for** $t \leftarrow 1$ **to** n **do**
4 $\quad \mathbb{D}_t^{\delta},\ \mathbb{D}_t^{\text{v}} \leftarrow \text{Users}(t)$; `/* deleted and volunteered posts of the users at interval` t `*/`
5 $\quad \mathbb{D}_t^{\delta} \leftarrow \mathbb{D}_t^{\delta} \cup \mathbb{G}_t^*$; `/* user- and challenger-deleted posts at interval` t `*/`
6 $\quad$ **if** *Adversary's budget has not exhausted* **then**
7 $\quad\quad a(\ \cdot\ , \theta_t) \leftarrow \text{Adversary}(\mathbb{D}_t^{\delta})$;
8 $\quad \mathbb{D}^{\text{v}}_{\leq t} \leftarrow (\mathbb{D}^{\text{v}}_{\leq t-1} \setminus \mathbb{G}_t^*) \cup \mathbb{D}_t^{\text{v}}$; `/* available volunteered set */`
9 $\quad \mathbb{G}^*_{t+1} \leftarrow \text{Challenger}(\mathbb{D}^{\text{v}}_{\leq t}, K, \text{accessType})$

in the next interval (line 9). This results in a real-life game between the adversary and the challenger, where each adapts to the other.

4.4 System Evaluation on Twitter Deletions

In this section we evaluate the efficiency of an adversary when Deceptive Deletions is applied to the real-world problem of concealing damaging deletions in Twitter. In this evaluation we first create and prepare sets of (non)damaging tweets. Then we use these sets to train the challenger and adversary classifiers and analyze their performance.

4.4.1 Data Collection

In this work, we select Twitter as our experimental social media platform. We note that it was certainly plausible to perform the exact experiment on other social platforms. However we chose Twitter due to its popularity and feasibility of data collection. Specifically, in order to evaluate the challenger we needed a real-world dataset which includes (i) both deleted and non-deleted tweets (i.e., Twitter posts)

and (ii) deleted tweets that contain both damaging and non-damaging tweets. To that end, we use two data sources to create such a dataset.

Deceptive Deletion dataset

We collected 1% of daily random tweet samples from the Twitter API from Oct 2015 - May 2018. Eliminating non-English tweets, we accumulated over one billion tweets. In the next step, we construct the damaging and volunteered sets.

To construct the damaging set, we first needed to identify the deleted tweets. We sampled 300,000 tweets from the aforementioned collected data, and leveraging the Twitter API, we identified the tweets that were deleted at the time of our experiment (Jan 30th, 2020). In total, we identified 92,326 deleted tweets. The next step was to obtain ground truth labels for the deleted tweets—i.e., detect and assign "true" labels to damaging tweets and "false" labels to rest. We used the crowdsourcing service Amazon Mechanical Turk (MTurk) [96] to obtain a proxy for these true labels. However, there were two challenges– First, it was impractical to ask our annotators to label 92,326 tweets. Second, since the dataset was highly imbalanced, a simple random sample of tweets for labeling would have resulted in a majority of non-damaging tweets.

Thus, we followed prior work [79,80] and filtered the deleted tweets using a simple sensitive keyword-based approach [79] (i.e., identify posts with sensitive keywords) to have a higher chance of collecting possibly damaging tweets. The complete list of keywords (over 1500 words) can be found in `http://bit.ly/1LQD22F`. This approach resulted in 33,000 potentially damaging tweets, and we randomly sampled 3,500 tweets to be labeled by annotators on MTurk. The mean number of sensitive keywords in each tweet within our data set was 2.55.

Note that, in addition to the cursing and sexual keywords, our sensitive keyword-based approach also considered keywords related to the topics of religion, race, job, relationship, health, violence, etc. Intuitively, if a post does not contain any such

sensitive keywords then the likelihood of the post being damaging is very low. We confirmed this intuition by asking MTurk annotators to label 150 tweets which did not contain any sensitive keyword as damaging/non-damaging. We noted that more than 97% of these 150 tweets were labeled as non-damaging by annotators. We surmised that in practice, the adversary will also leverage a similar filtering approach to reduce its overhead and increase its chances of finding damaging posts. Note that, in this experiment we have only considered the text of the tweets. However, the adversary can use additional user information, but labeling the posts (for training) based on the entire sets of posts of the users is infeasible for a large-scale attack.

In total, out of our sampled 3,500 deleted tweets, we obtained labels for 3,177 tweets (excluding annotations from Turkers who failed our quality control checks as described later). Among the labeled tweets, 1,272 were identified as damaging, and 1,905 were identified as non-damaging.

Data labeling using MTurk. We acknowledge that ideally, the tweet labels should have been assigned by the posters themselves. However, since we collected random tweets at large-scale using the Twitter API, we could not track down and pursue original posters to label their deleted tweets. Furthermore, following up with specific users for labeling their deleted posts is likely to cross the ethical boundary of this academic work (see Section 4.4.2). To that end, we note that there is a crowdsourcing based alternative which is already leveraged by earlier work to assign sensitivity labels [80,102,103]. Specifically, these studies determined the sensitivity of social media posts by simply aggregating crowdsourced sensitivity labels provided by multiple MTurk workers (Turkers). Thus, we took a similar approach as mentioned next.

On MTurk, tasks (e.g., completing surveys) are called Human Intelligence Tasks or HITs. Turkers can participate in a survey by accepting the corresponding HIT only if they meet all the criteria associated with that HIT (set by the person(s) who created the HIT). We leverage this feature to ensure the reliability of our results. Specifically we asked that the Turkers taking our survey should: (i) have at least 50 approved

HITs. (ii) have an assignment approval rate higher than 90%, and (iii) have their location set to United States. This last criterion ensured consistency of our Turkers' linguistic background. In our experiment each HIT consisted of annotating 20 tweets with true (damaging) or false (non-damaging) labels. We allowed the Turkers to skip some tweets in case they feel uncomfortable for any reason. We compensated 0.5 USD for each HIT and on average it took the Turkers 193 seconds to complete each HIT.

To control the quality of annotation by Turkers, we included two hand-crafted control tweets with known labels in each HIT. These control tweets were randomly selected from two very small sets of clearly non-damaging or damaging tweets and were inserted at random locations within the selection of 20 tweets. For example a damaging control tweet was: *"I think I have enough knowledge to make a suicide bomb now! Might need it New Year's Eve"* and non-damaging control tweet was: *"Prayers with all the people in the hurricane irma"*. If for a HIT, the responses to these control tweets did not match the expected label, we conservatively discarded all twenty annotations in that HIT.

We countered possible bias resulting from the order of presentation of tweets via randomizing the order of tweets in every HIT. Even if two Turkers annotated the same set of tweets, the order of those tweets was different. Furthermore, to ease the subjectivity of the labels from each participant, for each tweet we collected the annotations of multiple Turkers and took the majority vote. In our experiment, we created the HITs such that each tweet was annotated by 3 distinct Turkers. After receiving the responses, for each tweet we assigned the final label (indicating damaging or non-damaging) based on the majority vote.

We emphasize that in the real world, the burden of labeling the posts via crowdsourcing is on the adversary. The challenger, on the other hand, can be implemented as a service within the platform and can obtain the true labels directly from the post-owners. Therefore, existence of any mislabeled data will negatively impact only the adversary.

#Donttweet dataset

Recently Wang et al. [80] proposed "#Donttweetthis". "#Donttweetthis" is a quantitative model that identifies potentially sensitive content and notifies users so that they can rethink before posting those content on social platforms. Wang et al. created the training data for their model by (i) identifying possibly sensitive tweets by checking for the existence of sensitive keywords within the text and then (ii) using crowd-sourcing (i.e., using MTurk) to annotate the sensitivity of each tweet by three annotators.

The data collection approach used by "#Donttweetthis" (section 3 of [80]) is very similar to ours. Therefore, to enrich our dataset and be able to evaluate the challenger over more intervals, we acquired their labeled tweets. Using the Twitter API, we queried the tweets using their corresponding IDs and identified the deleted ones (at the time of writing, Jan 30th, 2020). In total, we obtained 851 deleted tweets, where 418 were labeled as sensitive (damaging), and the remaining 433 were labeled as non-sensitive (non-damaging). The mean of sensitive keywords in each tweet within this set was 1.7.

Summary of collected data. In summary, combining the two datasets explained above, we obtained labels for $4,028$ deleted tweets establishing the user deleted set. Among the deleted tweets $1,690$ were labeled as damaging constructing our damaging set ($\mathbb{D}^+$). As we will demonstrate in the results section, in our evaluation the four thousand labeled tweets (larger than that of prior works [79,80]) allows for 10 intervals for the game between the adversary and challenger.

Furthermore, for our experiment, we consider $k = 1, 2, 5$ (i.e., number of decoy posts for each damaging post). To accommodate these values of k and construct a volunteer pool that the challenger can make meaningful selections from, we sampled 100,000 non-deleted tweets uniformly at random from the 1% daily tweet samples posted between Jan 1st, 2018 – May 31st, 2018 to build the volunteered set. The non-deleted tweets are assumed to be non-damaging. We consider this assumption

to be reasonable as if a tweet contains some damaging content then its owner would not keep that post on its profile. In practice, we can forgo this assumption as the volunteer users themselves offer the volunteer posts. The average number of sensitive keywords in each tweet in this set was 0.41.

4.4.2 Ethical Considerations

Recall that in order to create our evaluation dataset we needed to show some deleted tweets to Turkers for the annotation task. Thus, we were significantly concerned about the ethics of our annotation task. Consequently, we discussed at length with the Institutional Review Board (IRB) of the lead author's institute and deployed the annotation task only after we obtained the necessary IRB approval. Next we will detail, how, in our final annotation task protocol we took quite involved precautionary steps for protecting the privacy of the users who deleted their tweets.

We recognize that, in the context of our evaluation, the primary risk to the deleted-tweet-owners was the possibility of linking deleted tweets with deleted-tweet-owner profiles during annotation. This intuition is supported by prior research [6, 104] who suggested applying selective anonymization for research on deleted content. Thus, we anonymized all deleted tweets by replacing personally identifiable information or PII (e.g., usernames, mentions, user ids, and links) with placeholder text. For example, we replaced user accounts (i.e., words starting with @) and url-links with "UserAccount" and "Link" respectively. Moreover, one of the authors manually went over each of these redacted posts to ensure anonymization of PII before showing them to Turkers.

4.4.3 Experiment Setup

Partitioning the data for different time intervals. Recall from Section 4.2 that we discretize time into intervals. In our experiments, we choose $T = 10$ intervals in total (a choice made based on the number of collected tweets). Consequently, we partition our dataset into 10 intervals. Ideally, the partitions should be based on the

creation and deletion timestamps of the tweets. Unfortunately however, the Twitter API does not provide deletion timestamps. Hence, we randomly shuffle the tweets and divide them into 10 equally sized partitions.

BERT model. In line with our approach to model the most-powerful adversary as best as we possibly can, we use a state-of-the-art natural language processing model: the BERT (Bidirectional Encoder Representations from Transformers) language model [105], both for the adversary and for the challenger. Specifically, we use $\text{BERT}_{\text{BASE}}$ model that consists of 12 transformer blocks, a hidden layer size of 768 and 12 self-attention heads (110M parameters in total). BERT has been shown to perform exceedingly well in a number of downstream NLP tasks [105]. We use HuggingFace's [106] implementation of the BERT model that was already *pre-trained* on masked language modeling and next sentence prediction tasks.

BERT uses WordPiece embeddings [107] to convert each word in the input tweet to an embedding vector. The concatenated embedding vector is passed to the BERT neural network model. In our experiments, we only give the text of the tweet as input to both the adversary and the challenger to make it amenable to the pre-trained BERT models. Other tweet features such as deletion timestamps, number of likes, etc. could be used by both the adversary and the challenger to improve their performance.

We *fine-tune* the BERT model on our datasets as prescribed by Devlin et al [105]. In each interval, the adversary's classifier is fine-tuned for the classification of tweets into damaging and non-damaging using the negative log-likelihood loss in Equation (4.1). We use a batch size of 32 and sample equal number of damaging and non-damaging tweets in each batch. This procedure results in better trained models as it avoids the scenario where a randomly sampled batch is too imbalanced (for example, no damaging tweet sampled in the batch). A separate BERT model is fine-tuned for the challenger using the loss function in Equation (4.4). Note that no balancing is required here since all the input tweets to the challenger model are non-damaging. We note that explaining the exact strategy employed by BERT models to classify text is an active research topic and complementary to our efforts. However, we highlight

that our challenger does not use any information about either the adversary's exact model or its parameters.

Budget constraints: We allow a limited budget of $B_{\text{static}} = 200$ deleted tweets for the static adversary and set $\tau = 1$, i.e., the static adversary only trains during the first out of the ten intervals. Similarly for the adaptive adversary, we allow a fixed recurring budget of $B_{\text{adapt}} = 200$ deleted tweets every interval. There are no budget restraints for random and oracle challengers (having no access and black-box access respectively). However, we restrict the D^2 challenger to have the same (recurring) query budget as the adaptive adversary's recurring budget to keep the game fair, i.e., $B_g = B_{\text{adapt}} = 200$.

We simulate the deceptive learning game described in Algorithm 3 with an adversary and a challenger, both implemented as BERT language models, with 10 different random seeds. We repeat the experiments for $k = 1, 2, 5$ where k denotes the number of decoy posts added per damaging deletion.

4.4.4 Results

Figures 4.2 and 4.3 show the F-scores (with 95% confidence intervals), precision and recall for different adversaries over 10 time intervals. We make the following key observations from the results.

Detection of damaging deletions in social media platforms is a serious concern. We start by considering the case where no privacy-preserving deletion mechanism is in place (i.e., no challenger to inject decoy deletions). In such a scenario, we compare the efficiency of different types of adversaries over ten intervals shown in Figure 4.2.

The random adversary labels the posts based on the prior distribution of the deleted tweets (around 42% damaging and 58% non-damaging every interval). As expected, the adversary achieves a 42% precision and 58% recall resulting in an F-score of about 48% in each interval.

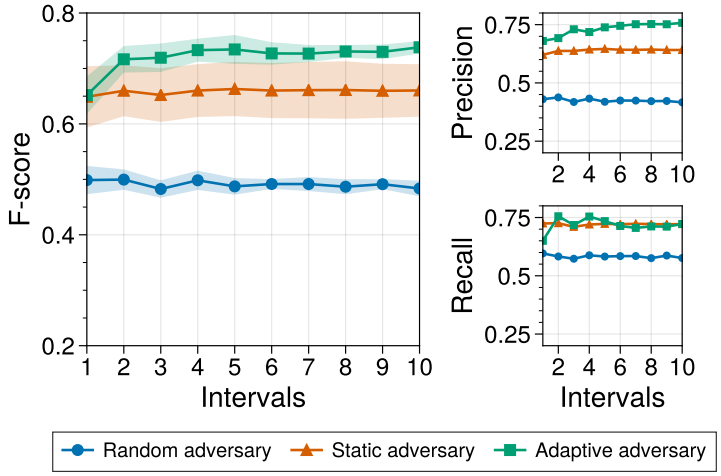
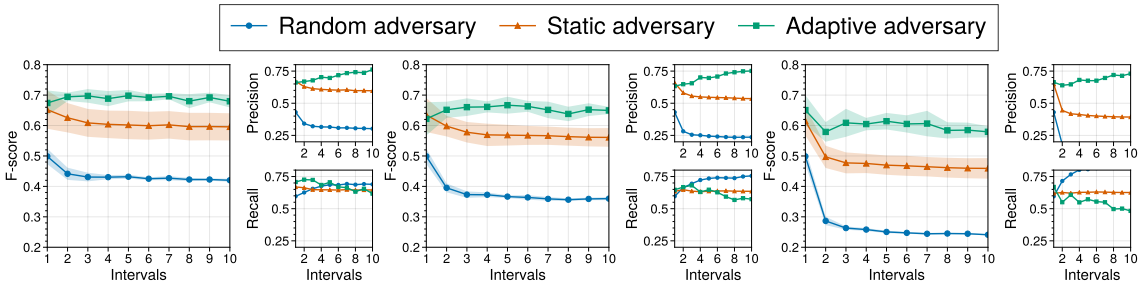


Fig. 4.2.: F-score of different adversaries (random, static, adaptive) when no privacy preserving deletion mechanism is in place. Shaded areas represent 95% confidence intervals.

As shown in Figure 4.2, in the first interval, the static adversary achieves a 17 percentage points (i.e., a 35%) increase in its F-score compared to the random adversary, and remains almost constant over the rest of the intervals. On the other hand, the adaptive adversary receives new training data every interval and trains its classifier continually, and hence is able to increase its F-score even further by about 10 percentage points (56% increase compared to the random adversary) at the end of the 10th interval.

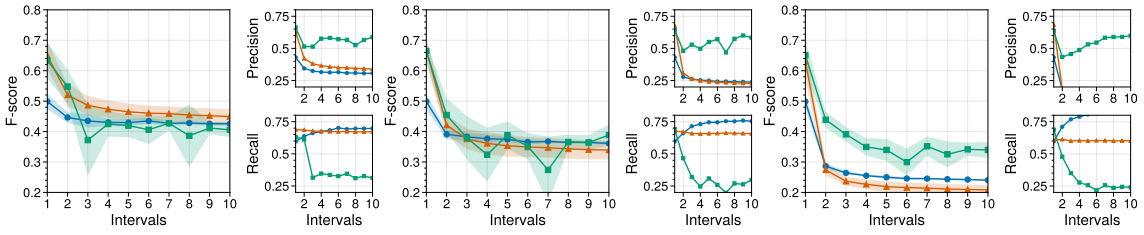
This shows that even normal users of social media platforms, not only celebrities and politicians, are vulnerable to the detection of their damaging deletions. Furthermore, the adversaries can automate this attack on a large-scale with an insignificant amount of overhead (access to a small dataset of posts with the corresponding labels), highlighting the necessity for a much-needed privacy-preserving mechanism for the users' damaging deletions in today's social platforms.

Injecting decoy deletions decreases the adversarial performance. As explained in Sections 4.2 and 4.3, we consider three challengers corresponding to the three types of accesses to the adversary's model – *no access*, *black-box access*, and *restricted black-box access*. In the following, we compare the performance of the



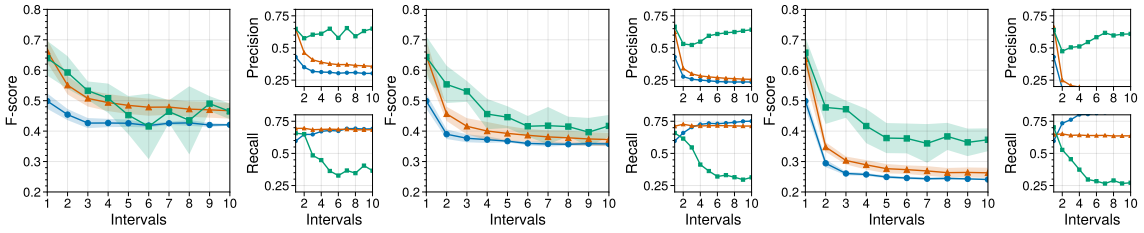
(a) Random ($k = 1$) (b) Random ($k = 2$) (c) Random ($k = 5$)

(No access.) Adversaries in the presence of **random challenger** with $k = 1, 2, 5$.



(d) Oracle ($k = 1$) (e) Oracle ($k = 2$) (f) Oracle ($k = 5$)

(Black-box access.) Adversaries in the presence of **oracle challenger** with $k = 1, 2, 5$.



(g) D^2 ($k = 1$) (h) D^2 ($k = 2$) (i) D^2 ($k = 5$)

(Restricted black-box access.) Adversaries in the presence of **D^2 challenger**

Fig. 4.3.: F-score (with 95% confidence intervals), precision and recall for the three adversaries (random, static and adaptive) in the presence of different challengers corresponding to different accesses with $k = 1, 2, 5$.

adversaries in the presence of the respective challengers against the adversaries' performance in the absence of any challenger.

No access: The top row of Figure 4.3 shows the performance of the three adversaries (random, static, and adaptive) in the presence of the random challenger. We

observe that although the F-score of both the static and the adaptive adversary decreases for all values of k, the reduction is not significant (only 7 percentage points for k=1 compared to the no-challenger case) In fact, both the adversaries still perform much better than the random adversary. This shows that protection of damaging deletions in the no-access scenario is possible but severely limited.

Black-box access: The middle row of Figure 4.3 shows the performance of the adversaries in the presence of an oracle challenger. Not surprisingly, this approach is very effective at lowering the (static and adaptive) adversaries' F-scores (close to random for k=1, 2; i.e., 20 and 35 percentage point reduction in the case of k=1 for the static and adaptive adversary respectively compared to the no-challenger case).

We also observe a major difference between the static and the adaptive adversaries in the presence of a competitive challenger. The static adversary retains the same recall performance (as in the no-challenger case) but loses drastically in precision, i.e., it classifies a large number of decoy posts as damaging. On the other hand, the adaptive adversary tries to *adapt* to the presence of decoy posts and becomes highly conservative – retains the same precision performance (as in the no-challenger case) but suffers heavily in the recall performance, i.e., it classifies a large number of damaging posts as non-damaging.

Restricted black-box access: The bottom row of Figure 4.3 shows the performance of the adversaries in the presence of the D^2 challenger. The performance of the D^2 challenger is comparable to the oracle challenger. The adversaries' F-scores in the presence of the D^2 challenger is close to 45% for the case of k=1 (20 and 30 percentage point reduction for the static and adaptive adversaries respectively compared to the no-challenger case). We also observe a precision-recall trade-off separating the static and the adaptive adversary (i.e., the static adversary loses in precision, whereas the adaptive adversary loses in recall) similar to the one described in the presence of an oracle challenger .

Overall, we conclude that the D^2 challenger is able to successfully raise the bar for the adversaries in identifying damaging deletions *without* requiring an unmonitored black-box access with infinite query budget.

The increase of decoy posts (k) results in lower adversarial performance with diminishing returns. While examining each row of Figure 4.3 individually, we see that the performance of the adversaries always decreases as k, the number of decoy deletions per damaging deletion, increases. However, we also observe that $k = 1$ is enough to reduce the F-scores of the adversaries to 45% (close to the random adversary). Further, the goal of most social platforms is to retain as many posts as possible, it would *not* be in the platform's best interests to use much larger values of k or to delete the entire volunteered set.

Observation of damaging and decoy posts. In Table 4.1, we show damaging tweets (as labeled by the AMT workers), decoy tweets (chosen by the D^2 challenger from a set of non-deleted tweets), and non-damaging tweets in our database. We observe that even though the decoy tweets typically seem to have sensitive words, they do not possess content damaging to the owner.

Table 4.1.: Sample tweet text extracts from the damaging, decoy, and non-damaging datasets. The real user accounts within the tweets have been replaced with @UserAccount. Some letters in the offensive keywords have been replaced by *.

Tweets' text extract	**Tweet Type**
"#GrowingUpInTexas Seeing a black person pass by ya front yard and telling your son to pass you the shotgun so you can play shoot em ups"	damaging
"@UserAccount its gods way of punishing you for your sins. fag**t."	damaging
"I think I might have the biggest douche for a boss hands down breaking point"	damaging
"Show up to work on meth once and your nickname is Tweaker for the rest of your life "	damaging
"gotta love watching two gay men having sex next to my homophobic parents"	damaging
"Listening to this deuchbag behind me at Chipotle diss every girl who comes in hot body but she has no face news check you re fugly"	decoy
"I grab a beer from the fridge put on my Bob Marley record crank that f**ker up and light up a fat one my professor is the shit"	decoy
"Kids having kids That sh*t is f**kin crazy to me I d rather be that cool a** uncle that buys the booze"	decoy
"This guy smacked his girlfriends ass in public That's disrespectful"	decoy
"I don't understand why people say that watermelon and fried chicken is for black people I love that sh*t"	decoy
"I want to eat to rid my emotions but I don't want the calories ya feel me"	non-damaging
"The middle is not the end, but a process you must grow through to get to your new beginning."	non-damaging
"@UserAccount @UserAccount Did you guys win it or did you burgle it from a classmate's house again?"	non-damaging
"Love is not about turning human relationships into billions of isolated couples."	non-damaging
"I'm pretty sure one of my professors has me mistaken for another black woman in my class."	non-damaging

4.5 Discussion

4.5.1 Adversarial Deception Tactics

The adversary can use different techniques to sabotage the challenger. Here, we mention some prominent systems attacks and their effects on the challenger.

Denial of Service attack. One of such attacks could be a simple Denial of Service (DoS), where the attacker submits requests for many damaging deletions to consume all the volunteer posts. First, we remind that the volunteered posts are a renewable resource, not a finite resource, as the users create, volunteer and delete posts in each time interval. Regardless, a DOS attack is possible wherein the adversary can use up all volunteered posts collected up until this point.

A standard way to avoid such attacks is to limit the number of damaging deletions that can be protected for each user in one time interval (we assume that the adversary can have many *adversarial users* to help with the DoS attack but is not allowed to use bots [108–113]). The challenger's defense is dependent on the distribution and number of volunteered posts. If there are more *adversarial users* than volunteers, then the adversary can win the game. We implemented the DoS attack as follows: in every interval, the adversary deletes as much as the standard deletions. We observed that the F-score did not change in this situation.

Volunteer Identification attack. In a volunteer identification attack, the adversary deletes a bunch of posts and uses the process of doing so to identify individuals who volunteer posts to the challenger for deletion. First, we note that in each time interval there is a large number of posts being deleted (> 100 million tweets daily [19]). Thus the posts deleted by the adversary (to try to identify volunteers) and the corresponding decoy deletions are mixed with other (damaging/non-damaging/decoy) deletions. In such a case, identifying the volunteers is equivalent to separating the decoy deletions from the damaging deletions; reducing to the original task. Additionally, the challenger does not delete the decoy posts at the same time as the original damaging deletion but does so in batches spread out within the time interval.

Further, the volunteers can also have damaging deletions of their own. Even if an adversary is able to identify volunteers, the adversary still needs to figure out which of the volunteer's deletions are decoys. If the adversary ignores all posts from volunteers, then a simple protection for the users is to become a volunteer, which helps our cause.

Adversary disguising as volunteer. In this attack, the adversary can take the role of a volunteer (or hire many volunteers) to offer posts to the challenger. Subsequently, the challenger may select the adversary's posts as decoys in the later intervals; however, these posts do not provide deletion privacy as the adversary will be able to discard these decoy posts easily. This effect can be mitigated with the help of more genuine volunteers and increasing the number of decoys per damaging deletion. This points to a more fundamental problem with any crowdsourcing approach: if the number of adversarial volunteers is more than the number of genuine volunteers, the approach fails.

Differentiating between different damaging categories. In this work, all the damaging posts are treated the same. However, in practice, the damaging posts fall into different categories, and some may be more harmful to the users than others. As a result, the adversary can focus on those categories more carefully. In such a case, the challenger's outputs and loss function need to be modified—the challenger needs to output a weight per damaging category for each decoy post (indicating the likelihood of fooling the adversary as a damaging post of that category). The challenger would also have to balance the different categories of decoy posts to keep the same distribution of categories as in the real damaging posts.

4.5.2 Obtaining Volunteered Posts From Users

Volunteer posts are a significant component of our system. We identify that there are already deletion services which enable users to delete their content in bulk (e.g., "twitWipe" [114] and "tweetDelete" [97] for Twitter, "Social Book Post Manager" [115]

for Facebook, "Cleaner for IG" [116] for Instagram, "Nuke Reddit History" [117], and multiple bots on RequestABot subreddit for Reddit). Our system can benefit from these bulk deletions to construct the volunteered posts pool. In such a scenario, whenever a user bulk-deletes it will mark its damaging posts and the remaining posts will be considered as "volunteered" with a guarantee that they will be deleted within a fixed time period.

We contacted the deletion services mentioned above and shared our proposal, Deceptive Deletions, for the privacy of users' damaging deletions. The responses that we received have been positive. They attest that, with Deceptive Deletions, an attacker that observes the deletion of users in large numbers will have a harder time figuring out which of the deleted posts contain sensitive material.

Nevertheless, other strategies could be more effective, for instance, one based on costs and rewards. Under such a strategy, each user seeking privacy for his/her damaging deletions is required to pay a cost for the service, whereas the users that volunteer their non-damaging posts to be deleted by the challenger (at any future point in time) are rewarded[5]. The costs and rewards can be monetary or can be in terms of the number of posts themselves (i.e., a user has to volunteer a certain number of her non-damaging posts to protect her damaging deletion). Nevertheless, in an ideal world, the volunteered set could also be obtained from altruistic users who offer their non-damaging posts for the protection of other users' deletions.

Finally, we emphasize that (as observed in Section 4.4.4) even when there is one decoy post for each damaging post ($k = 1$), the task of the adversary becomes significantly harder. Therefore, we can reckon that obtaining the pool of volunteer posts is realizable.

[5]similar concept exist for other distributed systems such as BitTorrent [118, 119].

4.5.3 Rate Limiting the Adversary's Data Access

In this work, we consider a very powerful adversary in terms of data access—it is capable of taking snapshots of the entire platform at different times to identify deleted posts (see Section 4.2.2). However, in practice, platforms can use rate-limiting techniques to restrict access of the adversary to the users' profile. Client-side strategies [120, 121], deferred responding [122], and the common limitations on source IP address, user, and API key [122, 123] are some of the well-known practices. A more sophisticated approach is to use computational puzzles, where the adversary can only access the data after successfully computing a puzzle given by the data platform. Sample domains include data breach mitigation [124, 125], DDOS [126, 127], spam-prevention [128], and practical cryptocurrencies [129]. These types of data limiting restrictions are interesting future work and will only improve our results. In such a case, the adversary will not be able to observe all the users' profiles constantly, or it will have blackout periods of the users' profiles (not observing the deletions).

4.6 Concluding Remarks

In this chapter, we show the necessity for deletion privacy by presenting an attack where an adversary is able to increase its performance (F-score) in identifying damaging posts by 56% compared to random guessing. Such an attack enables the system like Fallait Pas Supprimer to perform large-scale automated damaging deletion detection, and leaves users with "damned if I do, damned if I don't" dilemma.

To overcome the attack, we introduce Deceptive Deletions (which we also denote as challenger), a new deletion mechanism that selects a set of non-damaging posts (decoy posts) to be deleted along with the damaging ones to confuse the adversary in identifying the damaging posts. These conflicting goals create a minmax game between the adversary and the challenger where we formally describe the Deceptive Learning Game between the two parties. We further describe conditions for two extreme scenarios: one where the adversary always wins, and another where the chal-

lenger always wins. We also show practical effectiveness of challenger over a real task on Twitter, where the bar is significantly raised against a strong adaptive adversary in automatically detecting damaging posts. Specifically, we show that even when we consider only two decoy posts for each damaging deletion the adversarial performance (F-score) drops to 65%, 42% and 38% where the challenger has no-access, restricted black-box access and black-box access respectively. This performance indicates a significant improvement over the performance of the same adversary (75% F-score) when no privacy preserving deletion mechanism is in effect. As a result, we significantly *raise the bar* for the adversary going after damaging deletions over the social platform.

Our work paves a new research path for the privacy-preserving deletions which aim to protect against a practical, resourceful adversary. In addition, our deceptive learning game can be adapted for current/future works in the domain of Private Information Retrieval [83–86] that have a similar setting for injecting decoy queries to protect the users' privacy. Further, the challenger introduced in this work is considered to be honest and to not misuse the damaging deletions against the users. Considering distributed or federated protocols with multiple challengers as well as private multiparty computation [130–133] can be a promising future work to mitigate the complete trust of the challenger.

5. EVALUATING THE EFFICACY OF DELETION MECHANISMS

In the previous chapters, we observed different deletion mechanisms that are deployed in the current social platforms (selective deletions and prescheduled deletions). We further proposed two new deletion mechanisms: Intermittent withdrawal (Lethe Chapter 3) and Deceptive Deletions (Chapter 4). In this chapter, we aim to unearth the factors governing the usefulness of these four privacy preservation deletion mechanisms. Specifically, we looked into the efficacy of the deletion-privacy enhancing mechanisms, and identifying the key factors that led to the usefulness of mechanisms to preserve deletion privacy. To that end, we ask: *Are existing mechanisms useful for enhancing deletion privacy? Why or why not?*

We base this part of our study using four short videos. Those videos explained the high-level functionalities of four different deletion mechanisms. These mechanisms provide varying guarantees to protect deletion privacy. Users find selective deletions, the current deletion mechanism used by most social platforms, to be ineffective in protecting deletion privacy. The same users found other mechanisms more effective than selective deletion. However, they also identify the shortcomings of these mechanisms. We provide a principled analysis of the pros and cons of each of the existing mechanisms via mining the user perception for our participants.

5.1 Social Content Deletion Mechanisms

We identify four key mechanisms (the current deployed and our proposed mechanisms) for facilitating deletion in social platforms:

Selective deletions: Majority of the social platforms today provide a selective deletion mechanism–the posts are available on the platform until the user itself selects

an unwanted post and deletes it. However, as we mentioned earlier, selective deletion might attract unwanted attention to particular posts [19].

Prescheduled deletions: This mechanism automatically removes the users' contents when a specific criterion has been triggered (e.g., after a predefined time period or after prolonged inactivity around post [134]), e.g., Snapchat and Instagram Stories support this feature (e.g., delete content after 24 hours of posting). Prescheduled deletions ensure that an adversary cannot single out specific deletions to find damaging content (since all content will be deleted). However, on the down side, this mechanism removes *everything*, implying there will not be any archive of social content for users to reminiscence.

Intermittent withdrawal: Intermittent withdrawal [19] offers a deniability guarantee for the users' deletions in the form of an availability-privacy tradeoff. In this mechanism, all of the non-deleted posts are intermittently hidden for some amount of time. This hiding confuses an adversary while deciding if an unavailable post is deleted by the user or just temporarily hidden by the platform.

Decoy deletions: In the decoy deletions [20] mechanism, given a set of sensitive/damaging posts that users want to delete, the system selects k additional non-sensitive/non-damaging posts for each sensitive/damaging post and deletes them along with the damaging posts. The system-selected posts (decoy posts) are taken from a pool of non-damaging non-deleted posts provided by volunteers. Decoy deletions raises the bar for the adversary to identify deleted posts as they need to identify the sensitive or damaging post among the $k + 1$ deleted posts.

5.2 Methodology

The methodology of this survey is the same as explained in Section 2.2. In this section, we will only highlight the differences and refer the readers to Section 2.2 for detailed explanation.

5.2.1 Pilot Studies

Before deploying the survey, we conducted pilot studies to evaluate the procedure of the study, determine the average duration, and test the comprehensibility of the questions.

Initially, for this survey, we provided text descriptions of the threat model and the characteristics of the deletion mechanisms. However, the participants found the text descriptions to be monotonous, long, and more importantly hard to understand. Understandably, this issue seemed to be more prominent for the Intermittent Withdrawal and Decoy Deletions, as they have not yet been implemented in any of the popular platforms. As a result, we modified this part of the survey and created video explanations of the deletion mechanisms (similar to [135]). This allowed us to provide more information and present the mechanisms via examples and animations. We further created another video for the explanation of the threat model prior to displaying the mechanisms to the participants.

5.2.2 Survey Instrument

In this survey we captured the effectiveness and usefulness of different deletion mechanisms from the perspective of the users. We presented the participants with four deletion mechanisms—"Selective deletions", "Prescheduled deletions", "Intermittent Withdrawal", and "Decoy deletions" (detailed in Section 5.1). We realize that this part involves possible hypothetical scenarios, as some participants may have never used some of the mechanisms. Thus, we took a visual (audio and video) driven approach to first educate the users on these mechanisms, and later ask about their efficacy. This approach is similar to the ones used in prior work on familiarizing participants on novel authentication mechanisms [135].

To have a fair comparison between the different mechanisms, prior to introducing the mechanisms, we needed to present the threat model that we are considering protection against. In this work, we adapted the same threat model considered in

earlier works on deletion privacy–intermittent withdrawal and decoy deletions [19,20]. *In this threat model, the adversary can observe the entire social platform. Therefore, it can continuously access the platform to take snapshots of the posts with the goal of identifying the damaging/sensitive deleted posts to use against the users. The adversary aims to find as many as damaging/sensitive posts as possible and does not perform targeted attacks on particular users.* We demonstrated this threat model using a short video at the beginning of this section and asked the participants whether they have encountered attacks from such a malicious entity or how vulnerable do they find themselves against it.

After presenting the threat model, we showed short (1-2 minute) videos (with audio and subtitles) for each of the deletion mechanisms. Following each video, we asked our participants how effective do they find the mechanism in hiding their damaging/sensitive posts in the presence of the explained malicious entity. Further, we asked them to explain under which scenarios the find the mechanism useful and/or not useful.

For quality control of the survey, we used time-based filtering to ensure that participants gave attention while watching the videos in our survey.

5.2.3 Limitations

The videos in our survey have been narrated by a non-native English-speaking researcher, which may have caused some issues with the accent and pronunciations. We made the best effort possible by recording multiple times and narrating according to a script. Further, the videos have been uploaded to Youtube where the participants have had the ability to use the subtitles if needed.

5.3 Evaluating Deletion Mechanisms

So far, we established the need for deletion privacy and uncovered its norms on social platforms. In this section, we will compare the utility of deletion mechanisms

for enhancing deletion privacy in the presence of a large-scale adversary. We begin by investigating if the participants have ever experienced a negative scenario with the explained adversary in Section 5.2.2.

Some users have been attacked by malicious entities

As detailed in Section 2.2.1, part II of the survey begins by explaining the malicious entity (see Section 5.2.2) considered in this work, followed by asking the participants—"Have you ever experienced a scenario where a malicious entity who collects all deleted posts from a large number of users caused any issues/problems/discomforts for you in any of the social media platforms?"

95% of the participants responded with "No"; However, 35% of them think that this scenario is likely to happen to them. 34% responded that they don't think it is a likely scenario, and the remaining 31% were unsure.

Unfortunately, most of the 5% of the individuals that had a negative experience did not provide details to the incident to extract meaningful patterns. However, Participant $P70$ wrote—*"I saw one of my deleted photos on a Pinterest account that was not mine. It seemed like it was some kind of ad for earrings but it made me a little uncomfortable."* In response to another question (usefulness of the Decoy deletion mechanism), Participant $P36$ wrote about an experience that one of his/her friend had in encountering such an attacker—*" ...such malicious entities surely exist. My friend was contacted by one and threatened that the malicious entity would send pictures to his mother.*

5.3.1 The Current Deletion Mechanism Is Ineffective

For each of the deletion mechanisms (explained in Section 5.1): Selective deletions, Prescheduled deletions, Intermittent Withdrawal, and Decoy deletions, a short video was shown to explain the mechanism and its characteristics. Next, we asked—"In your opinion, how effective is [Deletion Mechanism] in hiding your damaging/sensitive

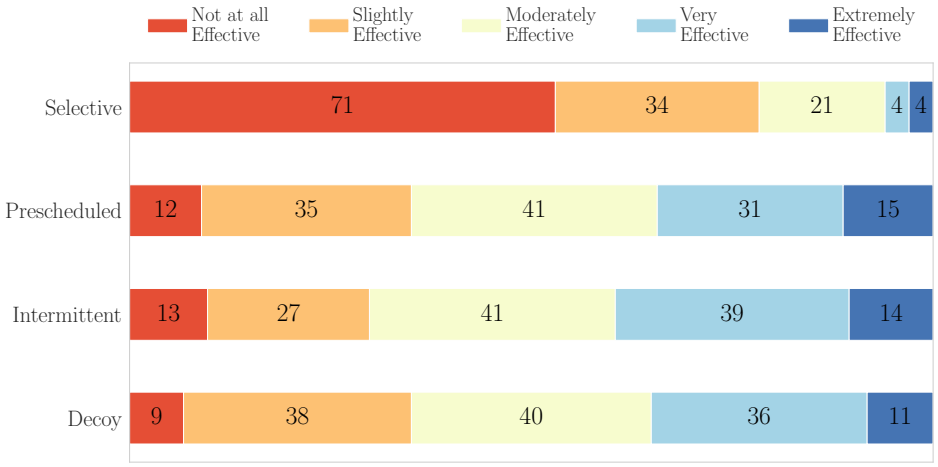


Fig. 5.1.: Effectiveness of deletion mechanisms

posts in the presence of a malicious entity who collects all deleted posts from a large number of users?" We used a Likert scale for the responses: Not Effective at all (0), Slightly Effective (1), Moderately Effective (2), Very Effective (3), Extremely Effective (4). The results are depicted in Figure 5.1.

We statistically compared the effectiveness of different deletion mechanisms by applying the Wilcoxon signed-rank test to find the likelihood that these four groups of scores come from the same distribution. We performed six (i.e., pairwise) such tests and set the threshold for significance to $\alpha = 0.05/6 = 0.008$ to account for the Bonferroni multiple-testing correction.

The results show that "Selective deletions" (i.e., the mechanism that is used by many social platforms today) has a significantly different distribution from all the other three mechanisms with a mean of 0.77 Likert-scale points compared to 2.01–2.1. The remaining deletion mechanisms do not have a significant difference among each other.

5.3.2 Useful Characteristics of the Deletion Mechanisms

In addition to the effectiveness question, in the form of a free-text box, we asked the participant to describe cases where they find the mechanisms to be useful and/or

Table 5.1.: Deletion Mechanisms' Characteristics.

Deletion Mechanism	Maintain Privacy	User in Control	Maintains Archive	No Assistance Needed	Limited No Privacy	Limited User Control	Limited No Archive	Need of Assistance
Selective	11 (8%)	**28 (21%)**	7 (5%)	—	**95 (70%)**	—	—	—
Prescheduled	**63 (47%)**	—	—	—	18 (13%)	14 (10%)	**56 (41%)**	—
Intermittent	**77 (57%)**	4 (3%)	7 (5%)	1 (1%)	23 (17%)	**29 (21%)**	—	—
Decoy	**66 (49%)**	1 (1%)	1 (1%)	—	15 (11%)	10 (7%)	9 (7%)	**27 (20%)**

NOT useful to them. We categorized the responses into eight categories[1], where each corresponds to a characteristic of a mechanism in Table 5.1. The first four characteristics cover the positive aspects of the deletion mechanisms, and the second four characteristics point out their shortcomings.

In the previous section, we observed that "Prescheduled Deletions", "Intermittent Withdrawals", and "Decoy Deletions" all have the same effectiveness in terms of protecting the users' damaging/sensitive deletions. In fact, providing privacy to the removal of sensitive content was noted as the highlight of the mechanism in the usefulness question as well. However, as we see in Table 5.1, each mechanism has a particular deficiency that may become a barrier for their use.

Prescheduled Deletions. For this mechanism, participants particularly disliked the fact that the platform will not have an archive of their posts and eventually everything is deleted. Participant $P19$ states: *"It could be EFFECTIVE (and thus, useful) because it would take care of the issue of sensitive material being used maliciously. However, effective doesn't mean that I like the idea of it! It would NOT useful because I like the idea of having access to my old content (great for memories, etc.) and do NOT like the idea of losing it forever because of some system."*

Intermittent Withdrawal. In this mechanism, all the non-deleted posts are intermittently hidden for some amount of time by the system. Therefore, the users felt a lack of control over their posts and profiles. Participant $P15$ said: *"This system could be useful if i made a sensitive post that I later decided to delete. However, it*

[1]An additional "other" category was also used which we omit from the analysis.

could also be problematic if a social platform randomly made an important post that I needed my audience to see, invisible for a period of time."

Decoy Deletions. In this mechanism, for each damaging/sensitive post, a set of decoy posts that are not damaging/sensitive to their owners (other users in the system) are selected to be deleted with the true damaging post. This procedure confuses the malicious entity in distinguishing which of the posts in the deleted set are the damaging/sensitive posts. Although many users found this tactic effective and novel, the dependability on a pool of decoy posts from other users prevented them from finding the mechanism useful. Participant $P119$ stated: *"this technique could be very effective if you want to delete a post and protect it from the entity. However it is hard too find the decoy post."*

Selective Deletions. As we observed previously, "Selective Deletions" was voted the least effective deletion mechanism in protecting the damaging deletions of the users. However, in the usefulness question, we see that it holds a unique characteristic that users admire. Giving the users full control of their posts and profile seems to be an advantage of this mechanism. Participant $P46$ stated: *"It's not effective at all but it is however the most popular among the bunch listed. People (even me) like to have full control over our social medias and tweets. Regardless if a malicious bot tries to collect sensitive information off of us."*

To no surprise, we see that in some cases, users will sacrifice their privacy over the usability of the system. This work is an initial step towards discovering the needs of the users and maintaining a balance between usability and deletion privacy.

5.4 Discussion and Future Work

5.4.1 Users Deeply Care About Their Old Posts

As we observed in Section 5.3, the major hurdle for using "Prescheduled Deletions", is the lack of users' archival posts. Prior to the deployment of the survey, we suspected that this problem would be a concern for the users. To that end, we added the

following two questions to see how important is the value of post archives for the users.

First, we asked—"How important is it for you that the social platform archives all your posts (new and old) and gives you the ability to access/view them at any time?". 74% of the participants stated that having access to their posts at a later time has some level of importance (24% extremely important, 33% very important, 17% slightly important). The remaining 26% was split between 15% neutral and 11%, not at all important.

We further asked—"How important is it for you that the social platform archives all your posts (new and old) and gives others (i.e., those who you have given permission to) the ability to access/view them at any time?". 44% of the participants stated this access pattern has some level of importance (9% extremely important, 14% very important, 21% slightly important) to them. The remaining 56% was split between 26% neutral and 30%, not at all important.

This highlights the fact that users care about their old posts and not only want to be able to access their own posts but also want others (to some level) to be able to access them at later times.

5.4.2 Users' Are Willing to Help to Enhance Deletion Privacy

In Section 5.1, we saw that "Decoy Deletions" benefits from a pool of volunteer posts to provide privacy to the sensitive/damaging deletions. We further saw Section 5.3 that users were worried about the need for assistance from other users' of the system for their sensitive/damaging deletions. Once again, before the deployment of the survey, we suspected that the construction of the decoy pool could be a burden for some of the users. However, to see the willingness of the users' in protecting the damaging deletions of themselves and others, we asked the participants—"Imagine that Decoy Deletions is available to you on a platform. Would you be willing to offer some of your non-sensitive/non-damaging posts that you won't mind getting removed

Table 5.2.: Users' suggestions for other deletion mechanisms

Deletion Mechanism	# of Votes	Platform overhead	Users overhead
Not posting regrettable content	19	None	High
Proactive approaches	8	High	Low
Rate-limiting the adversary	8	High	None
Enhancing existing mechanisms	17	High	Medium
Text Morphing	4	High	Low

from your profile to be added to the decoy pool in order to protect the sensitive/damaging deletions of yourself and other users?". 41% of the participants responded Yes (12% definitely yes, 29% probably yes), 39% of the participants responded No (13% definitely no, 26% probably no), and the remaining 20% responded with "might or might not".

This shows that although the construction of the decoy pool and the need for assistance from other users is a concern for some users, there are a significant number of participants that are willing to contribute to the pool.

5.4.3 Future of Deletion Mechanisms

Before ending the survey, we wanted to capture whether participants could think of other solutions for protecting sensitive deletions on social platforms. To that end, we asked the participants—"Can you think of any other technique that can protect user deletions in presence of the malicious entity mentioned above?"

In spite that many of the participants did not find any of the deletion mechanisms to be very effective (see Figure 5.1), 54% (73 out of 135) stated that they cannot think of any other technique for the protection of their deletions. Participant $P36$ wrote—*"I can't think of any other technique because although I know someone it happened to I never put much thought into the process"*. We summarize the remaining responses in Table 5.2

Not posting regrettable content and non-globalization of the data. Asking the users not to post regrettable content in the first place may seem like a good first step. Indeed 14% (19 participants) of the participant suggested the same point. For example, participant $P23$ responded with—*"To be honest I think the best thing people can do is think before they post"*, and participant $P83$ stated—*"The best defense is don't post sensitive stuff"*. Although it is an effective method, it is impractical as it puts lots of burden on the users. They cannot accurately predict what content would be damaging to them in the future (e.g., before applying for a job position or after a relationship breakup). On the positive side this method has no overhead challenge for the platform. Further, ten participants suggested a non-globalization of the user data by making the accounts private and only allowing the family members and close friends to view the posts.

Proactively preventing the publication of sensitive content. Eight participants pointed out different proactive approaches, similar to [38, 80]. In these proposals, multiple types of classifiers (e.g., Neural Networks, Naive Bayes, etc.) detect potential regrettable posts, to proactively advise users not to publish the posts. Participant $P186$ clearly explains the proactive solution in his/her response—*"Some kind of bot/AI that, based on language and keywords, warn the user that their post may be considered offensive or sensitive before they post it in the first place meaning they can delete it before posting. Sometimes, I am not sure people realise that once they post something, generally speaking the world online can see it. Even just asking 'Are you sure?' is enough to deter someone posting."*. Although helpful, in some cases, this proactive approach cannot prevent users from publishing future-regrettable posts. Further, these approaches create an overhead for the platform and slightly effects the free publishing of the posts on the user side.

Rate-limiting the malicious entity. Eight participants pointed out that the platforms should create some barrier for the malicious entities that collect the users' data in large-scale. For example, participant $P108$ states—*"Social networks could make efforts to thwart bots and scrapers that collect posts and also monitor profiles*

for deletions. Maybe IP limiting or some sort of CAPTCHA style tech?." In this approach the adversary will not be able to observe all the users' profiles constantly, or it will have blackout periods of the users' profiles (observing deletions with significant delay). The downside of this approach is that all the burden will be on the platforms and they have to be fully trusted.

Enhancing existing mechanisms. We further received 17 responses that pointed back to one of the deletion mechanisms that we already presented to them (i.e., 10 for Decoy, 5 for Prescheduled, and 2 for Intermittent). Some reiterated the mechanism in their own words and some provided extra features that they found helpful.

For the "Prescheduled Deletions", all five participants pointed out that the mechanism should have some sort of a selective archival procedure for at least the owner of the post. Participants $P16$ goes one step further and requests availability of the post for those who have interacted with the post as well—*"Scheduled Privacy. After a certain period of time/activity, all public posts automatically turn into private posts that can only be viewed by the owner and those who interacted with it."*

For the "Decoy Deletions", some participants mentioned that the users themselves may generate some random and non-sensitive posts (or even posts that contradict the original post) on their profiles and then at a later time delete them all together to confuse the adversary. In another example, participants $P55$ and $P60$ mention that the post that needs to be deleted should be hidden from the family members and friends of the individual immediately and later using decoy deletions the platform can remove it from the public (i.e., providing sufficient time to collect better decoy posts).

Morphing the posts' text. Finally, four of the participants suggest that rather than deleting a sensitive post, users can edit them and when they become comfortable enough with the edits then they can either delete it or just leave it on their profile. Participant $P62$ wrote—*"Altering the post completely and then delete it. If tried to recover, the post would be completely different."* This interesting proposal can be considered a feature of the platform itself, where this transition of the texts are

automated. If this transition converts the sensitive posts to benign ones without the users' input, then the platform itself will not know what is sensitive to the user. Hence removing the trust from the platform as well. The only burden that the users will face is that the audience of the posts at a later time will be only able to see a morphed version of the post and not the exact original text of the user.

5.5 Concluding Remarks

In this chapter, we saw that some users of the social platforms have experienced a discomforting scenario against a malicious entity that collects all the deleted posts from a large number of users. 35% of the remaining users that did not face such attacks believe that it is a likely scenario that can happen to them. Furthermore, we showed that selective deletions, the current deletion mechanism offered by many social platforms, is inefficient in providing protection to the users' deletions. Finally, we highlighted the key factors that future social platform designers should consider for attracting the users while providing them deletion privacy. These factors include but not limited to—i) providing an archive of the users' posts for future reference ii) giving the users complete control of their posts, and iii) minimum or no need for assistance from other users within the social platform.

Part II

Censorship Resistant Rendezvous using Permissionless Cryptocurrencies

6. MONEYMORPH

One of the most ubiquitous and challenging problems faced by the Internet today is the restrictions imposed on its free use. Repressive and totalitarian governments continue to censor Internet content to their citizens. Censors employ several techniques ranging from IP address filtering to deep-packet inspection in order to block disfavored Internet content [12]. Censored users are thereby prevented from not only accessing information on the Internet but also from expressing their views freely. Given that, several circumvention systems have been proposed over the last decade [13]. Nevertheless, censorship still remains a challenge to be fully resolved.

Nowadays, Bitcoin is observing a worldwide presence. Interestingly, this presence is prevalent in countries with large-scale censorship [15, 136], and although possible in theory, completely censoring Bitcoin may not be in the best interest of most countries [137]. The same holds true for other cryptocurrencies focused on smart contracts as in Ethereum [16] or privacy-preserving coin transfers as in Zcash [17] and Monero [18].

The availability of cryptocurrencies across different geopolitical corridors makes them a suitable distributed rendezvous to post steganographic messages. In fact, censored users can leverage their highly cryptographic structure to encode censored data while maintaining undetectability. In this work, we thoroughly study the feasibility of using the different available cryptocurrencies as a censorship circumvention rendezvous.

In preparation, inspired from the key encapsulation mechanism (KEM), we conceptualize the notion of *stego-bootstrapping (SB) scheme* that uses a two-way handshake between a censored user and an uncensored entity (i.e., decoder) where a decoder can transmit bootstrapping credentials of an entry point for a censorship-

circumvention protocol (e.g., Tor Bridge) to the censored user in the presence of the censor.

Our contributions. Firstly, we contribute *MoneyMorph*, our instantiation of the SB scheme. The cornerstone of *MoneyMorph* consists in reusing functionality from cryptocurrencies to make it seamlessly and interchangeably deployable with the major cryptocurrencies available today. In fact, *MoneyMorph* works using Zcash, Bitcoin, Monero, and Ethereum as rendezvous. Although we focus on widely deployed cryptocurrencies, our techniques can be leveraged in many other cryptocurrencies that share design principles with them. Supporting a wide range of cryptocurrencies increases the rendezvous available for the censored users and thus their chances of getting bootstrapping credentials.

Secondly, we carry out a comparative study of the different rendezvous by evaluating their effectiveness in terms of available bandwidth per transaction, monetary costs, and percentage of sibling transactions. In our study, Zcash is the preferable option with 1148 byte bandwidth per transaction costing less than 0.01 USD.

Finally, we have implemented *MoneyMorph* using Bitcoin, Ethereum, and Zcash as rendezvous, demonstrating its practicality and backward compatibility. Our evaluation shows that encoding/decoding operations can be completed in less than 50 milliseconds. Moreover, at given block creation rates, the decoder and censored user can simultaneously monitor several cryptocurrency blockchains looking for encoded data in real-time, even with their commodity equipment.

6.1 Related Work

The traditional censorship circumvention systems such as VPNs [138, 139], Dynaweb [140], Ultrasurf [141], Lantern [142], Tor [143], and others [144] benefit from establishing proxy servers outside of the censored area. However, these systems are vulnerable to blockage. Censors actively scan and block the IP addresses of the proxies. Circumvention systems respond with introducing new IP addresses. A prominent

example of such a cat-and-mouse game is between Tor [143] and the Great Firewall of China, which has resulted in introducing mirrors [145], bridges [22], and secret entry servers [146] in the Tor system. At the same time, multiple attacks (e.g., active probing and insider attacks) have been proposed to discover the Tor bridges [147–149]. In recent years domain fronting [150, 151] has been introduced as a way to resist IP address filtering. However, due to the high bandwidth and CPU usage, it can be costly for the hosts [152]. To reduce the cost, we can benefit from the use of content delivery networks (CDNs) namely CDNBrowsing [153, 154]. CDN's disadvantage is the unblocking of limited censored contents [153]. Moreover, as a central authority controls these services, their support for censorship circumvention is not reliable [155].

The most recent line of work in censor circumvention is the decoy routing approach [156–161]. Decoy routing, unlike the typical end-to-end approach, it is an end-to-middle proxy with no IP address. The proxy is located within the network infrastructure. Clients invoke the proxy by using public-key steganography to "tag" otherwise ordinary sessions destined for uncensored websites. Other anti-censorship mechanisms available in the literature leverage blog pings as communication medium [162] or hinder the harvesting attack by the censor relying on proof-of-work [163].

All of these approaches are orthogonal to what we present in this work. *MoneyMorph* exploits the new form of a communication channel, blockchain, that has been widely developed only recently. Hence, we believe it can coexist with current approaches and help augment the plethora of possibilities for anti-censorship.

Stealth Addresses (SA) is a cryptographic technique that allows detaching public keys from the intended receiver's identity, thus providing anonymity. However, SA does not define any data encoding mechanism. In this work, we not only ensure the anonymity of the receiver but also investigate how we can use different parts of the most used types of transactions in each of the cryptocurrencies to encode bootstrapping information.

Concurrently to this work, Tithonus [164] contributed a censorship-resistant communication tool that leverages the Bitcoin blockchain and the Bitcoin P2P network to

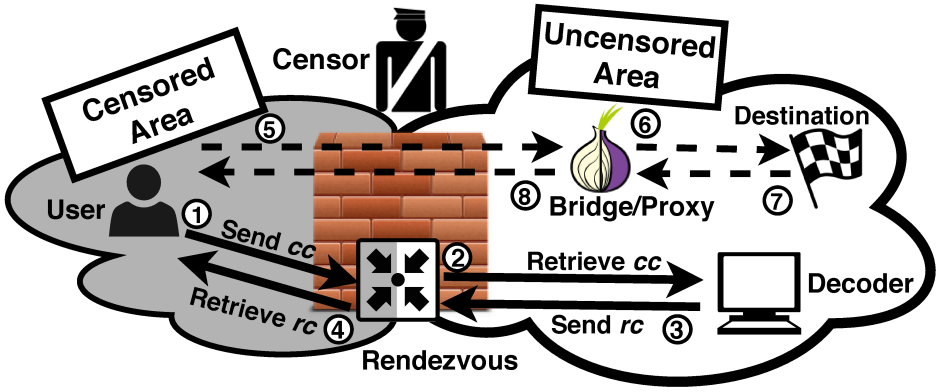


Fig. 6.1.: Censorship circumvention bootstrapping problem. Censored user sends a covertext to the decoder, who replies with another covertext including proxy's details. Then, the censored user can access censored information through the proxy. We focus on the bootstrapping process (solid arrows).

enable communication between the censored and uncensored areas. While Tithonus only considers Bitcoin as the communication channel, we instead define how to leverage other cryptocurrencies for rendezvous.

6.2 Problem Statement

We refer to the problem of bootstrapping communication into an uncensored area as *stego-bootstrapping*. As shown in Figure 6.1, a *censored user* wants to receive the credentials of a censorship-resistance protocol entry point (e.g., Tor bridge). For that, the user encodes a challenge message into a short covertext *cc* (e.g., a blockchain transaction) and sends *cc* to the rendezvous.

A *decoder* in the uncensored area provides the censored users with an entry point credentials. For that, the decoder continuously inspects the chosen rendezvous for covertexts, eventually getting the covertext *cc*, decoding it, and obtaining the *challenge* message from the user. Then, the decoder encodes the credentials in a new *response* covertext *rc* and adds it to the rendezvous. The censored user can then obtain *rc*, decode it, and get the bootstrapping details. What happens after this

point is out of the scope of this work. We rely on complementary solutions for the censorship-resistant communication.

The communication between censored user and decoder is hindered by the *censor*, an entity that decides what messages enter or exit the censored area. The censor can also run the protocol impersonating a censored user, learn the identity of the decoder, and easily stop the messages that are directly addressed to it. Therefore, we require a solution that communicates with the decoder without directly addressing messages to it.

6.2.1 Stego-Bootstrapping Scheme

The stego-bootstrapping (SB) problem can be seen as a *two-way handshake*, a *challenge* from the censored user to the decoder (*forward* direction), and the corresponding *response* from the decoder to the censored user (*backward* direction). The two-way handshake can be considered as two *independent* "one-way handshakes", each defined in terms of a public-key stegosystem [165], with a single setup, encoding and decoding algorithms. This approach, however, requires the decoder and censored user to know each other's public keys in advance. In practice, instead, the censored user knows the public key of the decoder, but the decoder does not know the public key of the censored user. Therefore, inspired by the key encapsulation mechanism, we consider the two-way handshake as a whole and only require that the censored user knows in advance the decoder's public key. Our SB scheme definition contains two pairs of encoding and decoding algorithms, the first for the challenge operations and the second for the response operations.

Here, λ is the security parameter; $\epsilon_1(\lambda)$, $\epsilon_2(\lambda)$ are negligible functions; $\mathcal{M}_c$, $\mathcal{M}_r$ are sets of challenge and response messages; $\mathcal{C}_c$, $\mathcal{C}_r$ are sets of challenge and response authenticated covertexts; and $\mathcal{T}$ is a set of tags. The f and b subscripts stand for forward (challenge operation) and backward (response operations) directions.

Definition 6.2.1 (Stego-Bootstrapping (SB) Scheme) *The SB scheme is a tuple of algorithms (SBSet, $SBEnc_f$, $SBDec_f$, $SBEnc_b$, $SBDec_b$) defined as below:*

- $vk_d, sk_d, \tau \leftarrow SBSet(\lambda)$. *On input the security parameter λ, output a key pair vk_d, sk_d and a tag $\tau \in \mathcal{T}$.*

- $\{((cc, \sigma), k), \perp\} \leftarrow SBEnc_f(vk_d, cm, \tau)$. *On input a public key vk_d, a challenge message $cm \in \mathcal{M}_c$ and a tag $\tau \in \mathcal{T}$, output either a tuple with an authenticated challenge covertext $(cc, \sigma) \in \mathcal{C}_c$ and a symmetric key k; or the symbol $\perp$ to indicate an error.*

- $\{(cm, k'), \perp\} \leftarrow SBDec_f(sk_d, (cc, \sigma), \tau)$. *On input a private key sk_d, an authenticated challenge covertext $(cc, \sigma) \in \mathcal{C}_c$ and a tag $\tau \in \mathcal{T}$, output either a tuple with a challenge message $cm \in \mathcal{M}_c$ and a symmetric key k'; or $\perp$.*

- $\{(rc, \sigma'), \perp\} \leftarrow SBEnc_b(sk_d, k', rm)$. *On input the private key sk_d, a symmetric key k' and a response message $rm \in \mathcal{M}_r$, output either an authenticated response covertext $(rc, \sigma') \in \mathcal{C}_r$; or $\perp$*

- $\{rm, \perp\} \leftarrow SBDec_b(vk_d, k, (rc, \sigma'))$. *On input the public key vk_d, a symmetric key k and an authenticated response covertext $(rc, \sigma') \in \mathcal{C}_r$, output a response message $rm \in \mathcal{M}_r$; or $\perp$.*

6.2.2 Threat Model

We consider the censor as a malicious adversary with network capabilities within the censored area and additionally knows the public key of the decoder. The censor does not control the decoder or its communications.

The censor is able to selectively inspect, fingerprint, block, or inject traffic within the censored area. We assume, however, that the censor is restricted in two ways. First, we assume that there are negative (economic) consequences for the censor to block all communications between censored users and the rendezvous system where

both censored user and decoder post their messages. While the censor can always prevent the access to a rendezvous as it has happened before with Telegram or SSL connections [12, 166], we believe that this assumption is realistic in practice using cryptocurrencies as rendezvous system. As the user base for different cryptocurrencies grows, even in countries with heavy censorship, banning them all may have economic consequences for the censor and the censored area [137]. In fact, this assumption is already followed by other works in the community [167]. Second, we assume that the censor cannot alter the information included in the rendezvous (e.g., the censor does control the majority of the Bitcoin network hash rate). We find this assumption realistic as it is required for the security of the rendezvous itself.

6.2.3 System Goals

Sibling Transactions. The SB system must maximize the number of messages that follow the structure of steganographic messages. Otherwise, the censor can deny service to the uncommon messages and yet maintain a functional system.

Cost-Efficiency. The SB system must provide a bootstrapping solution at a reduced cost for the honest censored users and the decoder. We measure the cost in the USD.

Bandwidth. The SB system must provide a bootstrapping solution that maximizes the bandwidth (number of Bytes available) between the censored users and the decoder for transferring the bootstrapping information.

6.3 Key Ideas

Blockchain as Rendezvous. We leverage the blockchain as rendezvous for censored messages encoded as blockchain transactions. The many blockchain systems existing today such as cryptocurrencies (e.g., Bitcoin), privacy-preserving cryptocurrencies (e.g., Zcash or Monero), or smart contracts (e.g., Ethereum) are managed by distributed users worldwide. We observe that the cryptographic structure of blockchain transactions can be used to encode censored data.

Using a blockchain system as rendezvous implies that covertexts remain visible even after the bootstrapping has finished. However, this cannot be leveraged by the censor because *MoneyMorph* is secure against chosen-covertext attacks. Therefore, the adversary cannot tell better than guessing whether a covertext contains bootstrapping data. The censor is thereby left with the choice of banning the complete blockchain system or allowing it completely.

Steganographic Tagging Scheme. We design a cryptographic construction to convert censored messages into ciphertexts that can then be encoded into a covertext transaction. The several public-key steganographic tagging schemes in the literature [165, 168–171] assume, in general, high bandwidth not available in blockchain transactions. We adapt the construction in [159] aiming to ciphertext succinctness: A ciphertext has a group element (e.g., an elliptic curve point) representing a public key and a random-looking bitstring of the size similar to the plaintext message. Moreover, the group element can be easily included in a blockchain transaction as it already handles public keys.

Fees. *MoneyMorph* introduces a fee overhead, which is inevitable due to the use of cryptocurrencies: A fee is paid to the miners to process and confirm transactions. Increased use of a cryptocurrency implies a fee raise. Fortunately, virtually all cryptocurrencies have built-in mechanisms to handle it.

Paid Services. Currently, many censorship circumvention systems are paid services, including even a premium account, to provide better performance [142, 150, 152, 172]. *MoneyMorph* can be used to bootstrap free of charge services (e.g., Tor) as well as the mentioned paid services. The fee of *MoneyMorph* compared to the actual cost of the mentioned services is negligible. Ultimately, *MoneyMorph* is a bootstrapping mechanism that is employed infrequently by the censored user, resulting in a low amortized cost over a long period of time.

6.4 Our Protocol

6.4.1 Building Blocks

Encoding Scheme. The *encoding scheme* allows to encode challenge and response data as a transaction compatible with the rendezvous. We defer our instantiations with different cryptocurrencies to Section 6.5.

Let $\mathcal{D}_c$ and $\mathcal{D}_r$ be a set of challenge and response data respectively. Let $\mathcal{A}_c$ and $\mathcal{A}_r$ be a set of challenge and response auxiliary information. Let $\mathcal{T}_c$ and $\mathcal{T}_r$ be a set of challenge and response transactions, respectively.

Definition 6.4.1 (Encoding Scheme) *An* encoding scheme *is a tuple of algorithms* $(TxEnc_f, TxDec_f, TxEnc_b, TxDec_b)$ *defined as below:*

- $\{ctx, \perp\} \leftarrow TxEnc_f(cd, ca)$. *On input challenge data* $cd \in \mathcal{D}_c$ *and the challenge auxiliary information* $ca \in \mathcal{A}_c$*, output a challenge transaction* $ctx \in \mathcal{T}_c$ *or the special symbol* $\perp$ *to indicate an error.*

- $\{cd, \perp\} \leftarrow TxDec_f(ctx)$. *On input a challenge transaction* $ctx \in \mathcal{T}_c$*, output challenge data* $cd \in \mathcal{D}_c$ *or the special symbol* $\perp$ *to indicate an error.*

- $\{rtx, \perp\} \leftarrow TxEnc_b(rd, ra)$. *On input response data* $rd \in \mathcal{D}_r$ *and the response auxiliary information* $ra \in \mathcal{A}_r$*, output a response transaction* $rtx \in \mathcal{T}_r$ *or the special symbol* $\perp$ *to indicate an error.*

- $\{rd, \perp\} \leftarrow TxDec_b(rtx)$. *On input a response transaction* $rtx \in \mathcal{T}_r$*, output response data* $rd \in \mathcal{D}_r$ *or the special symbol* $\perp$ *to indicate an error.*

Definition 6.4.2 (Encoding Scheme Correctness) *An encoding scheme is* correct *if for every challenge data* $cd \in \mathcal{D}_c$*, challenge auxiliary information* $ca \in \mathcal{A}_c$*, response data* $rd \in \mathcal{D}_r$ *and response auxiliary information* $ra \in \mathcal{A}_r$*, it holds that: (i) Let* $ctx \leftarrow TxEnc_f(cd, ca)$. *Then,* $cd^* \leftarrow TxDec_f(ctx)$ *and* $cd^* = cd$. *(ii) Let* $rtx \leftarrow TxEnc_b(rd, ra)$. *Then,* $rd^* \leftarrow TxDec_b(rtx)$ *and* $rd^* = rd$.

Non-interactive Key Exchange. A non-interactive key exchange (*NIKE*) is a tuple of algorithms $(NIKE.KGen, NIKE.ShKey)$, where $(vk, sk) \leftarrow NIKE.KGen(id)$ outputs a public-private key pair vk, sk for a given party identifier id. The algorithm $k \leftarrow NIKE.ShKey(id_1, id_2, sk_1, vk_2)$ outputs a shared key k for the two parties id_1 and id_2. We require a *NIKE* secure in the CKS model. Static Diffie-Hellman key exchange satisfies these requirements [173, 174]. Additionally, we require a function $ID(vk_u)$ that on input a public key vk_u returns the corresponding identifier id_u. We implement this function as the identity function.

Key Derivation Function. A key derivation function $KDF(k, l)$ takes as input a key k and a length value l and outputs a string of l bits. We use the hash-based key derivation function (*HKDF*) in [175] as the secure key derivation function.

Our Construction. In *MoneyMorph* (see Figure 6.2), we aim at optimizing the succinctness of the ciphertext. For that, we first use the Diffie-Hellman key exchange to generate a symmetric key (k_d) between the censored user ($SBEnc_f$, steps 1-2) and the decoder ($SBDec_f$, step 3). This symmetric key (k_d) shared between censored user and decoder becomes a master key for a key derivation function to derive three other keys (sk_s, k_c, k_r). Note that this key derivation function does not require interaction between the censored user ($SBEnc_f$, step 3) and decoder ($SBDec_f$, step 4).

The key k_c is used by the censored user to encrypt the challenge message (cm) along with a message tag τ ($SBEnc_f$, step 4). Correspondingly, the decoder uses k_c to decrypt the ciphertext created by the censored user ($SBDec_f$, step 5) and checks whether the decryption contains the tag τ ($SBDec_f$, step 6). The decoder thereby checks whether the ciphertext was the one created by the censored user or it does not contain any censored information otherwise. The key k_r is used similarly by the censored user and the decoder to encrypt ($SBEnc_b$, step 2) and decrypt ($SBDec_b$, step 4) the response message rm. Note that $SBEnc_b$ and $SBDec_b$ must be invoked with the same symmetric key k_d computed in $SBEnc_f$ and $SBDec_f$ to ensure correctness.

The last key (sk_s) becomes a fresh private key shared between the censored user and the decoder. From sk_s, the censored user can create a public key vk_p ($SBEnc_f$,

MoneyMorph

- $\boldsymbol{vk_d}, \boldsymbol{sk_d}, \tau \leftarrow \boldsymbol{SBSet}(\lambda)$.
  1. Generate $vk_d, sk_d \leftarrow NIKE.KGen(id_d)$
  2. Set $\tau \leftarrow \{0,1\}^{64}$; Return vk_d, sk_d, τ
- $\{(\boldsymbol{cc}, \boldsymbol{k}), \perp\} \leftarrow \boldsymbol{SBEnc}_f(\boldsymbol{vk_d}, \boldsymbol{cm}, \tau)$**.**
  1. Compute $vk_u, sk_u \leftarrow NIKE.KGen(id_u)$
  2. Compute $k_d \leftarrow NIKE.ShKey(ID(vk_u), ID(vk_d), sk_u, vk_d)$
  3. Compute $sk_s||k_c||k_r \leftarrow HKDF(k_d, \lambda + l_c + l_r)$
  4. Compute $vk_p \leftarrow vk_d^{sk_s}$ and set $ct_c := (\tau||cm) \oplus k_c$
  5. Compute $cc \leftarrow TxEnc_f((vk_u, H(vk_p), ct_c), sk_u)$
  6. If $cc = \perp$, return $\perp$. Else, return the tuple cc, k_d
- $\{(\boldsymbol{cm}, \boldsymbol{k'}), \perp\} \leftarrow \boldsymbol{SBDec}_f(\boldsymbol{sk_d}, \boldsymbol{cc}, \tau)$**.**
  1. Compute $cd \leftarrow TxDec_f(cc)$. If $cd = \perp$, return $\perp$
  2. Parse $vk'_u, H(vk'_p), ct'_c \leftarrow cd$
  3. Compute $k'_d \leftarrow NIKE.ShKey(ID(vk_d), ID(vk'_u), sk_d, vk'_u)$
  4. Compute $sk'_s||k'_c||k'_r \leftarrow HKDF(k'_d, \lambda + l_c + l_r)$
  5. Compute $vk_d \leftarrow g^{sk_d}$; $vk''_p \leftarrow vk_d^{sk'_s}$ and set $m' := ct'_c \oplus k'_c$
  6. Parse $\tau'||cm' \leftarrow m'$. Set $b := (\tau' = \tau) \wedge (H(vk'_p) = H(vk''_p))$
  7. If $b = 0$, return $\perp$. Else, return the tuple cm', k'_d
- $\{(\boldsymbol{rc}, \sigma'), \perp\} \leftarrow \boldsymbol{SBEnc}_b(\boldsymbol{sk_d}, \boldsymbol{k'}, \boldsymbol{rm})$**.**
  1. Compute $sk'_s||k'_c||k'_r \leftarrow HKDF(k', \lambda + l_c + l_r)$
  2. Compute $sk''_p \leftarrow sk_d \cdot sk'_s$; $vk''_p := g^{sk''_p}$ and set $ct_r := rm \oplus k'_r$
  3. Compute $rtx \leftarrow TxEnc_b((ct_r, vk''_p), sk''_p)$
  4. Parse rtx as (rc, σ') and return (rc, σ')
- $\{\boldsymbol{rm}, \perp\} \leftarrow \boldsymbol{SBDec}_b(\boldsymbol{vk_d}, \boldsymbol{k}, \boldsymbol{rc})$**.**
  1. Compute $rd \leftarrow TxDec_b(rc)$. If $rd = \perp$, return $\perp$
  2. parse $ct'_r, vk_p \leftarrow rd$
  3. Compute $sk_s||k_c||k_r \leftarrow HKDF(k, \lambda + l_c + l_r)$
  4. Set $rm := ct'_r \oplus k_r$
  5. If $vk_p \neq vk_d^{sk_s}$, return $\perp$. Otherwise, return rm

Fig. 6.2.: The *MoneyMorph* construction. We denote by l_c and l_r the number of bits for the challenge and response message respectively. We denote string concatenation by $||$. Here, H is a cryptographic hash as implemented in the encoding scheme and $\perp$ represents an error generated by the encoding schemes.

step 4) such that only the intended decoder knows the corresponding private key sk_p (as computed in $SBEnc_b$, step 2). We call the key pair vk_p, sk_p as the *paying key pair*. This key pair is used by the censored user to pay for the service provided by the decoder. The censored user can associate coins to vk_p so that when vk_p becomes a funded address in the blockchain, the decoder, knowing sk_p, can use those coins to cover the cost of sending the response covertext to the censored user.

Note that the decoder can use the coins at vk_p because our construction reconstructs the corresponding sk_p only at the decoder side. This allows the decoder to claim economic rewards without providing the response covertext. However, a rational decoder would arguably respond faithfully to keep the business with the censored users. Further, the decoder is trusted not to be running by the censor, as otherwise, it can trivially link the censored user to the chosen covertext that it can successfully decode. In practice, similar to the Tor directory, trusted decoders can be publicly identified by their public keys. We formalize our construction for *MoneyMorph* in Figure 6.2.

6.5 Cryptocurrency Encodings

6.5.1 Encoding Scheme in Bitcoin

Address and Transaction Format. A Bitcoin address is composed of a pair of signing and verification ECDSA keys. A Bitcoin address is then represented by the Base58 encoding for the hash of the verification key. Bitcoins are exchanged between addresses by means of a transaction. In its simplest form, a transaction transfers a certain amount of coins from one (or many) *input* address to one (or many) *output* address.

The Bitcoin protocol uses a scripting system called *Script* [176] that governs how bitcoins can be transferred between addresses within a transaction. In particular, coins are locked in an address according to SPKey, a Script excerpt that encodes the conditions to unlock the coins. The fulfillment of such conditions are encoded

Table 6.1.: Description of the Script excerpts used in the Bitcoin transactions. Text in blue denotes SPKEY and orange denotes the corresponding SSIG.

Lock's name	Script	Description of unlocking conditions
Pay2PKey	$\langle$*<pubKey> OP_CHECKSIG*$\rangle$ $\langle$*<sig>*$\rangle$	Including a signature $\langle$*<sig>*$\rangle$ of the Bitcoin transaction verifiable using the verification key $\langle$*<pubKey>*$\rangle$.
Pay2PKeyHash	$\langle$*OP_DUPOP_HASH160 < pubKeyHash > OP_EQUALVERIFYOP_CHECKSIG*$\rangle$ $\langle$*<sig> <pubKey>*$\rangle$	Including a verification key $\langle$*<pubKey>*$\rangle$ such that $\langle$*<pubKeyHash>*$\rangle$ = $H(\langle$*<pubKey>*$\rangle)$ and a signature $\langle$*<sig>*$\rangle$ of the Bitcoin transaction verifiable using the verification key $\langle$*<pubKey>*$\rangle$
Pay2ScriptHash	$\langle$*OP_HASH160 H(script) OP_EQUAL*$\rangle$ $\langle$*<sig> <redeem_script>*$\rangle$	Include a $\langle$*redeem_script*$\rangle$ such that H(*redeem_script*) = H(*script*) and *Eval* (redeem_script, $\langle$*<sig>*$\rangle$) returns `true`.
Pay2Null	$\langle$*OP_RETURN [data]*$\rangle$ $\langle$*<empty>*$\rangle$	Coins can never be unlocked. Data can contain up to 80 bytes [177].
Pay2Multisig	$\langle$*OP_M < pubkey1 >...< pubkeyn > OP_NOP_CHECKMULTISIG*$\rangle$ $\langle$*<sig1> ... <sigm>*$\rangle$	Including M signatures $\langle$*<sig1> ... <sigm>*$\rangle$ of the Bitcoin transaction, verifiable using the corresponding verification keys $\langle$*<pubkey1> ... <pubkeyn>*$\rangle$

in another Script excerpt called SSIG. A transaction is *valid* if coins unlocked (or spent) in the transaction have not been spent previously; the sum of input coins is greater or equal to the sum of output coins; and for each input SPKEY_i there exists a SSIG_i such that a function *Eval*($\text{SPKEY}_i, \text{SSIG}_i$) returns `true`, where *Eval* evaluates whether SSIG_i contains the correct fulfillment for the conditions encoded in SPKEY_i.

Possibilities for Encoding Data. Our approach consists on encoding the tagged message originated by the censored user as (some of) the conditions defined in the outputs SPKEY_i. We describe the different possible formats of the Bitcoin standard locking mechanism in Table 6.1. In the following, we describe how to re-use each of them to encode data within a transaction.

Pay2PKey: Instead of including an actual verification key within the $\langle$*<pubKey>*$\rangle$ field, it is possible to encode 33 bytes of data simulating thereby an ECDSA verification key. This encoding, however, implies the loss of locked coins as it is not feasible to guess a signing key corresponding to the data encoded as verification key.

Pay2PKeyHash: Instead of including the 20 bytes corresponding to the hash of a verification key within the $\langle$*<pubKeyHash>*$\rangle$ field, it is possible to encode 20 bytes of data. This encoding does not restrict the encoded data to an ECDSA verification key. Nevertheless, this encoding also implies the loss (or burnt in Bitcoin terms) of the locked coins since it is not feasible to come up with the pre-image of a random hash value.

Pay2ScriptHash: Similar to the Pay2PKeyHash, it is possible to encode 20 bytes of data replacing the field $H(script)$. This approach allows the inclusion of arbitrary random data at the cost of losing the locked coins.

Pay2Null: It allows the encoding of up to 80 bytes of data within the field $\langle [data] \rangle$. Although, this lock mechanism provides the maximum bandwidth so far, it also implies the loss of the locked coins.

Pay2Multisig: As only M verification keys are actually used in this lock mechanism, it is possible to encode 33 bytes of data in each of the remaining $N - M$ keys, simulating thereby an $N - M$ ECDSA verification keys. Advantage of this encoding is that the locked coins can be unlocked as the necessary M verifications are not modified. It is possible, however, to encode 33 bytes of data in each of the N verification keys at the cost of losing the locked coins.

Implementation Details.

We encode the censored data within locking mechanisms of the type Pay2PKeyHash (pkh) or Pay2ScriptHash (psh). Hereby, we use $\text{SPK}\textsc{ey}_{\text{pkh}}(H(vk))$ to denote the sequence $\langle OP\_DUP\ OP\_HASH160\ H(vk)\ OP\_EQUALVERIFY\ OP\_CHECKSIG \rangle$ and $\text{SPK}\textsc{ey}_{\text{psh}}(H(vk))$ to denote the sequence $\langle OP\_HASH160\ H(vk)\ OP\_EQUAL \rangle$. Similarly, we denote by $\text{SS}\textsc{ig}(tx, sk, vk)$ the condition defined as $\langle ECDSA.Sign(tx, sk)\ vk \rangle$. Finally, we denote by *Extract* an extraction function such that $Extract_{\{\text{pkh,psh}\}}(\text{SPK}\textsc{ey}(x)) = x$ and $Extract(\text{SS}\textsc{ig}(tx, sk, vk)) = vk$.

$\{\boldsymbol{ctx}, \perp\} \leftarrow \boldsymbol{TxEnc}_f(\boldsymbol{cd}, \boldsymbol{ca})$**.** Parse $vk_u, H(vk_p), ct \leftarrow cd$ and $sk_u \leftarrow ca$. If $|ct| > 20$ bytes, return $\perp$. Otherwise, create a Bitcoin transaction tx_1 as described below.
We assume that ct has been padded with pseudorandom bytes so that $|ct| = 20$ bytes and vk_u has been funded earlier with X `BTC`. This amount of coins encode the coins to be burnt at the output $\text{SPK}\textsc{ey}_{\text{psh}}$ as well as the amount of coins required by the decoder to pay for the transaction fee of the response covertext.

tx_1	
Inputs	$\text{SPKEY}'_{\text{pkh}}(H(vk_u))$, x BTC
Outputs	$\text{SPKEY}_{\text{psh}}(ct)$, γ_1BTC
	$\text{SPKEY}_{\text{pkh}}(H(vk_p))$, $(\text{x} - \gamma_1)$BTC
Signature	$\text{SSIG}(tx_1, sk_u, vk_u)$

$\{\boldsymbol{cd}, \perp\} \leftarrow \boldsymbol{TxDec}_f(\boldsymbol{ctx})$. If ctx does not have one input and two outputs or the lock mechanisms are not of the type Pay2PKeyHash and Pay2ScriptHash, return $\perp$. Otherwise, compute ct as $ct \leftarrow \mathit{Extract}(\text{SPKEY}_{\text{psh}}(ct))$. Compute $vk_u \leftarrow \mathit{Extract}(\text{SSIG}(tx, sk_u, vk_u))$. Compute $H(vk_p) \leftarrow \mathit{Extract}(\text{SPKEY}_{\text{pkh}}(H(vk_p)))$. Return the tuple $cd := (vk_u, H(vk_p), ct)$.

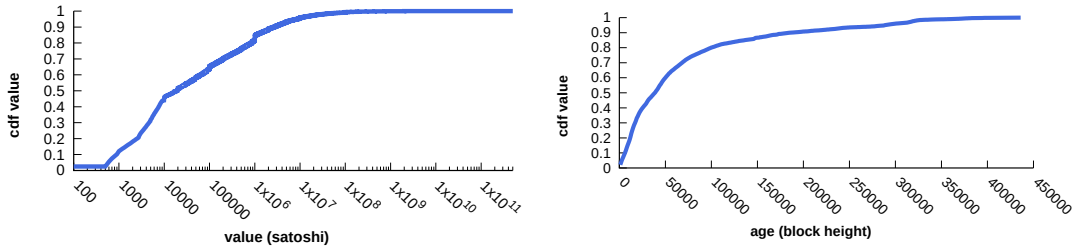
$\{\boldsymbol{rtx}, \perp\} \leftarrow \boldsymbol{TxEnc}_b(\boldsymbol{rd}, \boldsymbol{ra})$. Parse $ct, vk_p \leftarrow rd$ and $sk_p \leftarrow ra$. If $|ct| > 40$ bytes, return $\perp$. Otherwise, create a Bitcoin transaction tx_2 as described below. Return tx_2. As before, here we assume that ct has been padded with pseudorandom bytes so that $|ct| = 40$ bytes.

tx_2	
Inputs	$\text{SPKEY}'_{\text{pkh}}(H(vk_p))$, $(\text{x} - \gamma_1)$ BTC
Outputs	$\text{SPKEY}_{\text{psh}}(ct[0:19])$, γ_2BTC
	$\text{SPKEY}_{\text{pkh}}(ct[20:39])$, $(\text{x} - \gamma_1 - \gamma_2)$BTC
Signatures	$\text{SSIG}(tx_2, sk_p, vk_p)$

$\{\boldsymbol{rd}, \perp\} \leftarrow \boldsymbol{TxDec}_b(\boldsymbol{rtx})$. If rtx does not have one input and two outputs, return $\perp$. If the lock mechanisms are not of the type Pay2PKeyHash and Pay2ScriptHash, return $\perp$. Otherwise, compute $ct[0:19] \leftarrow \mathit{Extract}(\text{SPKEY}_{\text{psh}}(ct[0:19]))$ and $ct[20:39] \leftarrow \mathit{Extract}(\text{SPKEY}_{\text{pkh}}(ct[20:39]))$. Compute $vk_p \leftarrow \mathit{Extract}(\text{SSIG}(tx, sk_p, vk_p))$. Return $rd := (ct, vk_p)$.

System Discussion.

Sibling Transactions. We have downloaded a snapshot of the Bitcoin blockchain containing blocks 580,000 to 600,000 (June-Oct 2019), containing around 45 million Bitcoin transactions. In this dataset, we observe that the majority of the transactions contain one input and two outputs. Furthermore, we examined the outputs'



(a) value (b) age

Fig. 6.3.: CDF of the value and age, in the outputs of Pay2PKeyHash transactions that contain only one input and two outputs. The value is given in Satoshi (a Satoshi is 10^{-8} BTC). The age is given in block height (on average each block in the Bitcoin network is created every 10 minutes).

locking mechanisms and observed that the combination of one Pay2PKeyHash and one Pay2ScriptHash construct 32% of all transactions followed by two Pay2PKeyHash with 23%. To maximize the sibling transactions, we have selected the combination of Pay2PKeyHash and Pay2ScriptHash as the lock mechanism to be used by *MoneyMorph* to raise the bar for the censor targeting a single transaction type. Nevertheless, if the aforementioned transaction trend changes, any of the locking mechanisms mentioned earlier can be used to lock the coins while encoding censored data [178], which provide the same bandwidth as Pay2PKeyHash or more. Therefore, *MoneyMorph* users could *dynamically* adjust the encoding scheme based on the transaction output distribution in Bitcoin at different times.

Cost. *MoneyMorph* (BTC) requires to pay two transaction fees (tx_1 and tx_2) as well as burn coins three times because the SPKEY outputs used to encode the ciphertexts are no longer spendable. At the time of writing, the fastest and cheapest fee for a transaction is about $4,500$ Satoshi (0.34 USD) [179]. If the censored user is willing to wait two hours for its transaction to get into the blockchain, the cost is reduced to $2,700$ Satoshi (0.2 USD)

Furthermore, to find the minimum value for burning coins (γ_1 and γ_2) without trivially being censored, we investigated all previous transactions with one input and

two outputs. Section 6.5.1 shows the CDF value of the amounts in the outputs of such transactions. We observed that in order to blend with at least 25% of the outputs, the burned amount should be at least $2,500$ Satoshi (0.25 `USD`). We further analyzed the UTXO set and observed that about 50% of the outputs remain unspent for more than 9 months (Section 6.5.1) ; therefore, the *MoneyMorph* transactions could go unnoticed for a long period of time. We note that Bitcoin has a supply of 21 million Bitcoins. With the current amount of burnt coins per transaction, we would require $\sim 8 * 10^{11}$ transactions to terminate the supply, and thus burnt coins have a negligible effect on the economics of Bitcoin.

Bandwidth. *MoneyMorph* (`BTC`) uses SPKey fields to encode the data. Each SPKey field is leveraged to encode exactly 20 bytes (using Pay2PKeyHash or Pay2ScriptHash). This provides a 20 byte bandwidth for the challenge covertext, as one of the two SPKey fields is used. On the other hand, the response covertext has a 40 byte bandwidth as it uses both of SPKey fields to encode encrypted data.

Limitations. *MoneyMorph* (`BTC`) suffers from two limitations: (i) it requires to burn coins; and (ii) forces the censored user to prepare a spendable input with the precise value amount to generate transactions tx_1 and tx_2. These two limitations can be mitigated by using the Pay2Multisig script in the encoding (as explained earlier in this section and similarly used in the community [164]), or adding an extra output as a change address. Both mitigations come at the cost of reducing the number of sibling transactions. Alternatively, *MoneyMorph* can mitigate these limitations by leveraging other cryptocurrencies discussed in this section.

6.5.2 Encoding Scheme in Zcash

Addresses and Transactions. Ben-Sasson et al. [17] proposed Zerocash, a privacy-preserving cryptocurrency scheme. The core idea behind Zerocash has been implemented in Zcash [180]. As the implementation slightly differs, we focus our description

on the cryptocurrency as detailed in the paper [17] and extend the implementation details when it applies.

Zerocash [17, 180] supports two types of addresses: *transparent* and *shielded*. A transparent address is defined by an amount x of coins that are locked according to a script excerpt SPKey that encodes the conditions to unlock the coins. The fulfillment of such conditions are encoded in another Script excerpt called SSig. More details are included in Section 6.5.1.

A shielded address (or *coin*) is a tuple of the form $(pk, \mathrm{x}, \rho, r, s, com)$, where pk is a public key generated as $PRF_{sk}(0)$ with PRF being a pseudo-random function and sk being a private key; x is the value associated to this coin, ρ, r and s are random seeds and com is a commitment that represents the coin. We describe the format of com later.

Zcash transactions transfer coins from input(s) to output(s) which can be both transparent and shielded addresses. Figure 6.4 shows an example of such transaction. Omit for a moment the transparent address in the input. Then, the rest of the transaction is an example of a user that wants to split the coin (i.e., c^{old}, sn^{old}) into two new coins c_1^{new} and c_2^{new}. For that, she creates a single shielded output $(rt, sn^{old}, com_1^{new}, ct_1, com_2^{new}, ct_2, h, vk^*, \sigma^*, \Pi_{pour})$ as follows: rt denotes the root of a Merkle tree whose leafs contains all the commitments $\{com_i\}$ included so far in the blockchain; $sn^{old} := PRF_{sk^{old}}(\rho)$ is the serial number associated to the coin being spent; com_1^{new} and com_2^{new} are the commitments formed as described for com but for new values x_1^{new} and x_2^{new}. The values ct_1^{new} and ct_2^{new} are two ciphertext that contain the corresponding $(\mathrm{x}_i^{new}, \rho_i^{new}, r_1^{new}, s_1^{new})$. These ciphertexts are encrypted for the corresponding payee. The payee can then locally reconstruct the complete coin information as $c_i^{new} := (pk_i^{new}, \mathrm{x}_i^{new}, \rho_i^{new}, r_1^{new}, s_1^{new}, com_i^{new})$. Finally, vk^* is a fresh ECDSA

Input	SPKey_0, x_0 ZEC
Output	$(rt, sn^{old}, com_1^{new}, ct_1, com_2^{new}, ct_2, h, vk^*, \sigma^*, \Pi_{pour})$
Sign.	SSig_1

Fig. 6.4.: Example of transaction in Zcash.

verification key, $h := PRF_{sk^{old}}(H(vk^*))$, σ^* is a signature of the complete output under sk^*, and Π_{pour} is a zero-knowledge proof of the correctness of the output (e.g., sn^{old} corresponds to one of the shielded coins ever created or σ^* can be correctly verified using vk^*). We refer readers to [17] for a detailed explanation.

A transaction can have several transparent inputs and any combination of shielded and transparent outputs. A transaction is valid if, for every shielded output, σ^* is a valid signature under verification key vk^* and Π_{pour} correctly verifies. Furthermore, for every transparent output, the coins unlocked (or spent) have not been spent previously; the sum of input coins is greater or equal to the sum of output coins; and for each input SPKEY_i there exists a SSIG_i such that a function $Eval(\text{SPKEY}_i, \text{SSIG}_i)$ returns `true`, where *Eval* evaluates whether SSIG_i contains the correct fulfillment for the conditions encoded in SPKEY_i.

Possibilities for Encoding Data. In a shielded output, there are several fields rt, sn^{old} or com_i^{new} that cannot be used to encode our data without making the zero-knowledge proof Π_{pour} fail. However, assume that a user creates a transaction to send a coin to herself, then the data encrypted in ct_i is not required as the intended payee is the user itself. Our insight then consists in encoding our data as the different ciphertexts available in the shielded outputs for coins sent to the user herself. We note that this ciphertext field contains an ephemeral public key for a Diffie-Hellman key exchange, followed by a bitstring of encrypted data with the corresponding symmetric key. The portion of encrypted data constitutes 584 bytes that can be reused to encode steganographic data in our system.

Furthermore, some data can be encoded in the conditions SPKEY of the transaction. In this work we consider the Pay2PKeyHash condition format that includes the 20 bytes corresponding to the hash of a verification key. The SPKEY $(H\,(vk))$ along with the SSIG of the transaction allows us to encode a verification key. For more details we refer to Section 6.5.1.

Implementation Details. Here, *Extract* is an extraction function working as follows: $Extract(\text{SPKEY}(x)) = x$ as well as $Extract(\text{SSIG}(tx, sk, vk)) = vk$. Additionally,

on input a field f and a transaction tx, it returns the value of f if present in the shielded output of tx.

$\{\boldsymbol{ctx}, \perp\} \leftarrow \boldsymbol{TxEnc}_f(\boldsymbol{cd}, \boldsymbol{ca})$**.** Parse $vk_u, H(vk_p), ct \leftarrow cd$ and $sk_u \leftarrow ca$. If $|ct| > 1148$ bytes, return $\perp$. Otherwise, create a Zcash transaction tx_1 as shown below and return tx_1. Here, we assume that vk_u has been funded earlier with X ZEC and that there exists an old coin with a value X^{old}, previously funded in a shielded output, whose serial number is sn^{old}.

tx_1

Input	$\text{SPKey}_1(H(vk_u))$, X ZEC
Output	$(rt, sn^{old}, com_1^{new}, H(vk_p)\|\|ct[0,563], com_2^{new}, ct[564,1147], h, vk', \sigma, \Pi_{pour})$
Sign.	$\text{SSig}(tx, sk_u, vk_u)$

The pair $(com_1^{new}, ct_1^{new})$ is set to an honest shielded coin for the censored user to get the remaining coins back. Therefore, com_1^{new} is well formed committing to a coin with value $\text{X} + \text{X}^{old} - \text{X}_y$. However, as the censored user is sending coins to herself, ct_1 is not required and can be used to encode the public key vk_p and $ct[0,563]$ (i.e., first 564 bytes). The value X_y must be enough to pay for the transaction fee. Finally, the pair (com_2^{new}, ct_2) is used to encode the rest of the ciphertext ct. For that, com_2^{new} is set to a commitment for a coin with value $\text{X} = 0$ and set $ct_2^{new} := ct[564, 1147]$. If $|ct| < 1148$ bytes, ct is padded with pseudorandom bytes. Finally, a fresh key pair vk', sk' is used for the signature σ and the hash h of the shielded output.

$\{\boldsymbol{cd}, \perp\} \leftarrow \boldsymbol{TxDec}_f(\boldsymbol{ctx})$**.** If ctx does not have a transparent input and a shielded output, return $\perp$. Otherwise, Compute $H(vk_p)||ct[0:563] \leftarrow Extract(ctx, ct_1)$, $ct[564:1147] \leftarrow Extract(ctx, ct_2)$, and $vk_u \leftarrow Extract(\text{SSig}(tx, sk_u, vk_u))$. Return $cd := (vk_u, H(vk_p), ct)$.

$\{\boldsymbol{rtx}, \perp\} \leftarrow \boldsymbol{TxEnc}_b(\boldsymbol{rd}, \boldsymbol{ra})$**.** Parse $ct, vk_p \leftarrow rd$ and $sk_p \leftarrow ra$. If $|ct| > 1168$ bytes, return $\perp$. Otherwise, create tx_2 as shown below and return tx_2.

tx_2

Input	$\text{SPKey}(H(vk_p))$, X^* ZEC
Output	$(rt, sn_1^{old}, com_3^{new}, ct[0,583], com_4^{new}, ct[584,1167], h', vk'', \sigma', \Pi'_{pour})$
Sign.	$\text{SSig}(tx_2, sk_p, vk_p)$

This transaction spends the coins on vk_p previously funded by the censored user in tx_1. The rest is constructed as in tx_1, with the difference that we don't include the paying public key and thus gaining an extra 20 bytes that we can use for the encoding of the response.

$\{\boldsymbol{rd}, \perp\} \leftarrow \boldsymbol{TxDec}_b(\boldsymbol{rtx})$**.** If rtx does not have a transparent input and a shielded output, return $\perp$. Otherwise, compute $ct[0:583] \leftarrow \mathit{Extract}(rtx, ct_3)$ and $ct[584:1167] \leftarrow \mathit{Extract}(rtx, ct_4)$. Extract $vk_p \leftarrow \mathit{Extract}(\text{SSIG}(rtx, sk_p, vk_p))$. Return $rd := (ct, vk_p)$.

System Discussion.

Sibling Transactions. We have obtained blocks 0 to 480,000 from the Zcash blockchain, containing about 4.2 million transactions. First, we observe that around 11% of the outputs are shielded. Although small, we note that shielded addresses have started to be used, and in Oct 2019, 19% of the transactions contained shielded outputs [181]. Second, we observe that there exist two new coins (and thus two pairs of (com, ct)) for the shielded outputs. Third, we observe, from Figure 6.5, that around 60% of shielded Zcash transactions included one transparent input. Therefore, to maximize the blending with other transactions, we use transactions that are structurally identical to the most widely used shielded transactions in Zcash blockchain. We further looked into the transactions from Aug 2019, and they showed the same trend of observations from earlier months.

Cost & Bandwidth. *MoneyMorph* (`ZEC`) requires to pay only two transaction fees. The rest of the coins are sent back to the censored user using the shielded coins. The price of the transaction fee is 0.0001 `ZEC` (0.003 `USD`). *MoneyMorph* (`ZEC`) uses the ciphertext ct_i of a Zcash transaction, encoding 1148 bytes for the challenge and 1168 bytes for the response covertexts.

6.5.3 Encoding Scheme in Monero

Addresses and Transactions. A Monero address is of the form (A, B), where A and B are two points of the curve ed25519, as defined in Monero. In order to avoid the linkability of different transactions that use the same public key, a payer does not send the coins to the Monero address of the payee. Instead, the payer derives a *one-time key* verification key vk and an extra random point R, from the payee's address using the Monero Stealth Address mechanism [182]. Given an arbitrary pair (vk', R') set as an output in the blockchain, a payee can use her Monero address to figure out whether the pair (vk', R') was intended for her and, if so, compute the signing key sk' associated to vk'. We note that while Stealth Addresses hide the intended receiver, they do not define a mechanism to encode censorable data. We refer the readers to [182] for further details.

A Monero transaction is divided into *inputs* and *outputs*. An input consists of a tuple $(\{vk_i\}, \{Com(x_i, r_i)\}, \{\Pi_i\})$, where $\{vk_i\}$ is a *ring* of one-time keys that have previously appeared in the blockchain, each $Com(x_i, r_i)$ is a cryptographic commitment of the amount of coins x_i locked in the corresponding public key vk_i, and each Π_i is a zero-knowledge range proof proving that x_i is in the range $[0 : 2^k]$, where k is a constant defined in the Monero protocol.

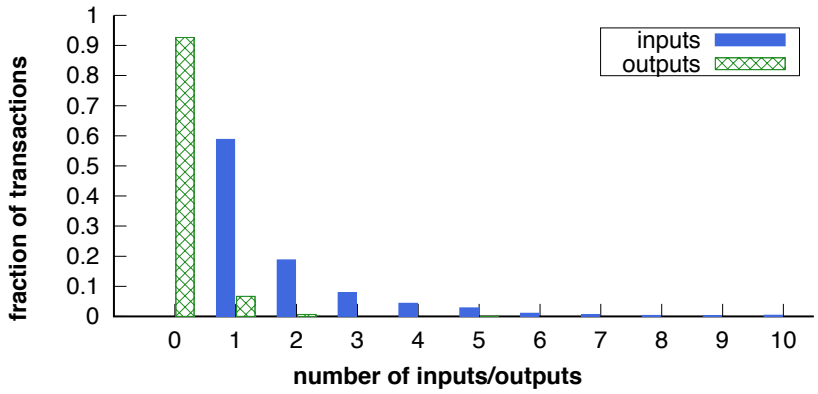


Fig. 6.5.: Distribution of the unshielded inputs and outputs for shielded transactions.

Inputs	$(\{vk_i\}, \{Com(x_i, r_i)\}, \{\Pi_i\})$
Outputs	$((vk'_1, R'_1), Com(x'_1, r'_1), \Pi'_1)$ $((vk'_2, R'_2), Com(x'_2, r'_2), \Pi'_2)$
Sign.	σ_{ring}

Fig. 6.6.: Illustrative example of a Monero transaction.

An output consists of a tuple $((vk', R'),\ Com(\mathrm{x}', r'),\ \Pi')$, where each element is defined as aforementioned. Finally, a transaction contains a signature σ_{ring}, created following the *linkable ring signature scheme* [183]. A Monero transaction is valid if the following conditions hold. First, σ_{ring} shows that the sender knows the signing key sk^* associated to a verification key within the set $\{vk_i\}$ and such key has not been spent before. Second, let x^* be the amount of coins encoded in the input commitment corresponding to the one-time key being spent. Then it must hold that x^* equals the sum of the output values. Finally, all zero-knowledge range proofs correctly verify that all amounts are within the expected range. An illustrative example of a Monero transaction with one input and two outputs is depicted in Figure 6.6.

Linkable Ring Signature Scheme. Let λ be the security parameter, and let $\mathbb{Z}_p$ be a group of prime order p. Moreover, let $\mathbb{G}$ be a cyclic group generated by the generator g as defined in the Monero protocol. Then, a linkable ring signature scheme $(LRS.KeyGen, LRS.Sign, LRS.Verify)$ is defined as:

- $vk, sk \leftarrow LRS.KeyGen(\lambda)$: Sample $sk \leftarrow_\$ \mathbb{Z}_p$ and compute $vk := g^{sk}$.
- $\sigma \leftarrow LRS.Sign((vk_1, \ldots, vk_{n-1}, vk_n), sk_n, m)$: Sample $r \leftarrow_\$ \mathbb{Z}_p$. Compute $\mathcal{I} := sk_n \cdot H(vk_n)$ and $h_0 \leftarrow H(m||g^r||H(vk_n)^r)$. Then, sample $s_1, \ldots s_{n-1} \leftarrow_\$ \mathbb{Z}_p^{n-1}$ and compute the following series:

$$h_i := H(m||g^{s_i} \cdot vk_i^{h_{i-1}}||H(vk_i)^{s_i} \cdot \mathcal{I}^{h_{i-1}})$$

  Now, solve s_0 such that $H(m||g^{s_0} \cdot vk_n^{h_{n-1}}||H(vk_n)^{s_0} \cdot \mathcal{I}^{h_{n-1}} = h_0$. For that, solve $g^{s_{n-1}} \cdot vk_{n-1}^{h_{n-2}} = g^r$, getting that $s_0 = r - h_{n-1} \cdot sk_n$. Finally, output $\sigma := (s_0, s_1, \ldots, s_{n-1}, h_0, \mathcal{I})$.

- $\{\top, \bot\} \leftarrow \mathit{LRS.Verify}((vk_1, \ldots, vk_n), m, \sigma)$: Parse $(s_0, s_1, \ldots, s_{n-1}, h_0, \mathcal{I}) \leftarrow \sigma$ and compute the series:

$$h_i := H(m || g^{s_i} \cdot vk_i^{h_{i-1}} || H(vk_i)^{s_i} \cdot \mathcal{I}^{h_{i-1}})$$

  Return $\top$ if $h_0 = h_n$. Otherwise, return $\bot$.

Possibilities for Encoding Data. As the information in an input tuple must previously exist in the blockchain, we cannot modify them. Our approach consists then in crafting an output tuple that encodes certain amount of data while maintaining the invariants for the validity of the transaction. In particular, our insight is that if a user transfer coins to herself, she does not need to create the pair (vk, R) from her own address (A, B), using the stealth addresses mechanism. However, the commitment and the range proof must be computed honestly, as otherwise transaction verification fails and the transaction does not get added to the blockchain.

Further, we can encode extra data within the signature (see Section 6.5.3). In particular, we observe that the *LRS.Sign* algorithm samples $n-1$ random values from $\mathbb{Z}_p$. Our insight is that instead of sampling random numbers, we use the corresponding bytes from a ciphertexts as random numbers. As values $s_1, \ldots, s_{n-1}$ are included in the signature of the transaction, they allow to increase the bandwidth. Currently, Monero establishes that the rings must contain 11 public keys and thus 10 random numbers can encode data.

Implementation Details. Here we detail the encoding of covertexts into Monero transactions.

$\{\boldsymbol{ctx}, \bot\} \leftarrow \boldsymbol{TxEnc}_f(\boldsymbol{cd}, \boldsymbol{ca})$**.** Parse $vk_u, H(vk_p), ct \leftarrow cd$ and parse $sk_u \leftarrow ca$. Create a transaction tx_1 as shown below. The ciphertext ct is split in chunks of 32 bytes and each chunk is included as a value s_i in the signature.

tx_1	
Inputs	$(\{vk_1^i\}, \{Com(\mathrm{x}_1^i, r_1^i)\}, \{\Pi_1^i\})$, $(\{vk_2^i\}, \{Com(\mathrm{x}_2^i, r_2^i)\}, \{\Pi_2^i\})$
Outputs	$((vk_u, R_1'), Com(\mathrm{x}_1', r_1'), \Pi_1')$, $((vk_p, R_2'), Com(\mathrm{x}_2', r_2'), \Pi_2')$
Sign.	$\sigma_{ring} := (s_0, s_1, \ldots, s_{n-1}, h_0, \mathcal{I})$, $\sigma'_{ring} := (s'_0, s'_1, \ldots, s'_{n-1}, h'_0, \mathcal{I}')$

$\{\boldsymbol{rtx}, \perp\} \leftarrow \boldsymbol{TxEnc}_b(\boldsymbol{rd}, \boldsymbol{ra})$**.** Parse $ct, vk_p \leftarrow rd$ and parse $sk_p \leftarrow ra$. Create a Monero transaction tx_2 as described below.

tx_2	
Inputs	$(\{vk_1^i\} \cup vk_p, \{Com(\mathrm{x}_1^i, r_1^i)\}, \{\Pi_1^i\})$, $(\{vk_2^i\}, \{Com(\mathrm{x}_2^i, r_2^i)\}, \{\Pi_2^i\})$
Outputs	$((vk_1', R_1'), Com(\mathrm{x}_1', r_1'), \Pi_1')$, $((vk_2', R_2'), Com(\mathrm{x}_2', r_2'), \Pi_2')$
Sign.	$\sigma_{ring} := (s_0, s_1, \ldots, s_{n-1}, h_0, \mathcal{I})$, $\sigma'_{ring} := (s'_0, s'_1, \ldots, s'_{n-1}, h'_0, \mathcal{I}')$

$\{\boldsymbol{cd}, \perp\} \leftarrow \boldsymbol{TxDec}_f(\boldsymbol{ctx})$**.** Extract the ciphertext ct by concatenating the values in the signature, and the pair vk_u, vk_p from each of the outputs. Return the tuple $cd := (vk_u, H(vk_p), ct)$.

$\{\boldsymbol{rd}, \perp\} \leftarrow \boldsymbol{TxDec}_b(\boldsymbol{rtx})$**.** Extract the ciphertext ct from the values included in the signature and extract vk_p from the input ring. Return $rd := (vk_p, ct)$.

System Discussion.

Sibling Transactions. We downloaded a snapshot of the Monero blockchain that contains blocks $1,890,000$ to $1,940,000$ (July-Oct 2019), with more than 200 thousand transactions. From this dataset, we have extracted the distribution of the number of inputs and outputs used by the transactions. We observe that transactions with one and two inputs, each consist of around 48% and 42% of the transactions. Furthermore, around 89% of them have two outputs. Therefore, to maximize our sibling

transactions and bandwidth goals, we opt for transactions with two inputs and two outputs.

Cost & Bandwidth. *MoneyMorph* (XMR) requires to pay only two transaction fees. The price of the transaction fee is 0.0005 XMR (0.03 USD). Each of the outputs provides us with ten fields of 32 bytes, totaling the bandwidth of 640 bytes.

Traditional Privacy-Preserving Currencies vs. *MoneyMorph*. Privacy-preserving currencies already support encrypted messaging; however, the straightforward use of private payments is disadvantageous with respect to *MoneyMorph*. Our approach for Zcash enables more than twice the bandwidth compared to the traditional use case. (i.e., the user pays to herself and memo must not include payment information for the receiver). The 32 bytes of the payment-id in Monero are not enough to send a key (i.e., already 32 bytes) and some extra query data (e.g., proxy type). Instead, *MoneyMorph* provides ten times the bandwidth provided by the straightforward use of Monero.

6.5.4 Encoding Scheme in Ethereum

In Ethereum, there exist two types of addresses: *external addresses* and *contracts*. An external address is formatted as in Bitcoin and holds a certain amount of ETH, the Ethereum native coin. They, however, differ in that unlike Bitcoin and Zcash, addresses in Ethereum can be used multiple times. A contract has associated a piece of software that implements a certain business logic. A contract invocation requires *data* as input for the contract execution. In this work, we focus on the use of contracts to encode steganographic data.

We have downloaded a snapshot of the Ethereum blockchain and extracted the transactions invoking contracts. We observed that among them, the contract *Etherdelta_2* (an exchange contract) [184] is the most invoked (8% of all transactions) and thus we use it in our encoding mechanism. Nevertheless, the approach described here can be easily extended if any other contract is invoked. The *testTrade* is a method in this

contract that checks whether a trade between two different addresses can take place. We will use this sample method to encode our data.
The signature of this method is as follow:

```
testTrade(address tokenGet, uint amountGet, address tokenGive,
uint amountGive, uint expires, int nonce, address user,
uint8 v, bytes32 r, bytes32 s, uint amount, address sender)
```

Each address is 160 bits and the uint values are 256 bits. To minimize the suspiciousness of the censor we only encode data in the *amountGet*, *amountGive*, *expires*, *amount* fields and use the lower 32 bits (to simulate realistic amounts). We get a total bandwidth of 20 bytes which is enough to bootstrap the censored user.

We denote $H(vk)$ by $\text{ADDRESS}(vk)$. We denote by $Extract(tx, tag)$ a function that returns the value of the field tag, e.g. $Extract(tx, Receiver) = H(vk_2)$.
$\{\boldsymbol{ctx}, \perp\} \leftarrow \boldsymbol{TxEnc}_f(\boldsymbol{cd}, \boldsymbol{ca})$**.** Parse $vk_u, H(vk_p), ct \leftarrow cd$ and $sk_u \leftarrow ca$. If $|ct| > 20$ bytes, return $\perp$. Otherwise, first create an Ethereum transaction tx_0 to fund the one-time key vk_p with X ETH as described below. The minimum value of X is 0.001 ETH (0.15 USD), which is equivalent to one transaction fee (γ') for calling the contract. The transaction fee (γ) for transferring Ether to external accounts is about 0.0002 ETH (0.03 USD).

tx_0 (Fund one-time key)

Field	Receiver	Amount	Fee	Signature	Data
Value	$H(vk_p)$	X ETH	γ ETH	$\sigma(sk_u), vk_u$	—

Next, create an Ethereum transaction tx_1. Split the ciphertext ct in chunks of 5 bytes. Each chunk is included as a value v_i in the low bit order of the *amountGet*, *amountGive*, *expires*, *amount* fields of the *data* field of a contract call. Rest of the fields are not changed and will contain proper addresses and values as aforementioned. Moreover, we assume that ct has been padded with pseudorandom bytes so that $|ct| = 20$ bytes. $H(vk_e)$ denotes the address of the *Etherdelta_2* contract. The value is set to zero and the only cost will be the transaction fee. Return $tx_0||tx_1$.

tx_1

Field	Receiver	Amount	Fee	Signature	Data
Value	$H(vk_e)$	0	γ' ETH	$\sigma(sk_u), vk_u$	v_1, v_2, v_3, v_4

$\{\boldsymbol{cd}, \perp\} \leftarrow \boldsymbol{TxDec}_f(\boldsymbol{ctx})$. Parse $tx_0||tx_1 \leftarrow ctx$. If tx_1 does not have $H(vk_e)$ as the receiver, return $\perp$. Otherwise, extract the ciphertext ct by concatenating the values in $data \leftarrow Extract(tx_1, data)$. Compute $\sigma(sk_u), vk_u \leftarrow Extract(tx_1, signature)$. Compute $H(vk_p) \leftarrow Extract(tx_0, receiver)$. If $H(vk_p)$ does not have at least 0.0003 ETH associated coins, return $\perp$. Otherwise, return the tuple $cd := (vk_u, H(vk_p), ct)$.

$\{\boldsymbol{rtx}, \perp\} \leftarrow \boldsymbol{TxEnc}_b(\boldsymbol{rd}, \boldsymbol{ra})$. Parse $ct, vk_p \leftarrow rd$ and $sk_p \leftarrow ra$. If $|ct| > 20$ bytes, return $\perp$. Otherwise, create an Ethereum transaction tx_2 as described below. Return tx_2. As before, here we assume that ct has been padded with pseudorandom bytes so that $|ct| = 20$ bytes.

tx_2

Field	Receiver	Amount	Fee	Signature	Data
Value	$H(vk_e)$	0	γ' ETH	$\sigma(sk_p), vk_p$	v_1, v_2, v_3, v_4

$\{\boldsymbol{rd}, \perp\} \leftarrow \boldsymbol{TxDec}_b(\boldsymbol{rtx})$. If rtx does not have $H(vk_e)$ as the receiver and vk_p as the verification key, return $\perp$. Otherwise, extract the ciphertext ct by concatenating the values of the *amountGet*, *amountGive*, *expires*, *amount* fields as contained in $data \leftarrow Extract(tx_2, data)$. Compute $\sigma(sk_p), vk_p \leftarrow Extract(tx_2, signature)$. Return the tuple $rd := (vk_p, ct)$.

6.5.5 Mining-Based Encoding

A miner creates blocks with the transactions to be added to the blockchain. The miner can decide on its own how to arrange the transactions within the block, except for the coinbase transaction, that must always be the first. Our insight is that the miner can arrange the transactions so that it conveys information for the intended user. (see Table 6.2 for a comparison on the effectiveness in different cryptocurrencies).

We denote by $\mathbf{tx'} \leftarrow \Pi(k, \mathbf{tx})$ a permutation function that takes as input a list of transactions sorted lexicographically $\mathbf{tx}$ and a permutation key k, and returns a permuted list of the transactions $\mathbf{tx'}$. We denote by $k \leftarrow \mathsf{GetPerm}(\mathbf{tx}, \mathbf{tx'})$ an algorithm that on input a list of transactions and its permutation, returns the key k used for the permutation. We observe that both of the operations can be performed efficiently in practice [185]. For simplicity, we denote by $tx(vk, vk', \mathrm{x})$ a transaction that sends x coins from vk to vk'. In practice, we use this transaction to pay for the service provided by the decoder. The details of this transaction depend on the cryptocurrency used. Finally, we assume that the mechanism provides β bytes of bandwidth. Here, $\beta = \log_2(N_{tx}!)$, where N_{tx} is the number of transactions in the block. .

$\{\boldsymbol{ctx}, \perp\} \leftarrow \boldsymbol{TxEnc}_f(\boldsymbol{cd}, \boldsymbol{ca})$**.** Parse $vk_u, H(vk_p), ct \leftarrow cd$ and $sk_u \leftarrow ca$. If $|ct| > \beta$ bytes, return $\perp$. Otherwise, create a block with the set of transactions $\mathbf{tx'} \cup tx(vk_u, vk_p, \mathrm{x})$, where $\mathbf{tx'} := \Pi(ct, \mathbf{tx})$, and publish it.

$\{\boldsymbol{cd}, \perp\} \leftarrow \boldsymbol{TxDec}_f(\boldsymbol{ctx})$**.** Compute $ct \leftarrow \mathsf{GetPerm}(\mathbf{tx}, \mathbf{tx'})$. Extract vk_u and $H(vk_p)$ from the extra transaction in the block of the form $tx(vk_u, vk_p, \mathrm{x})$. If x is not enough to pay for a transaction fee, return $\perp$. Otherwise, return the tuple $cd := (vk_u, H(vk_p), ct)$.

$\{\boldsymbol{rtx}, \perp\} \leftarrow \boldsymbol{TxEnc}_b(\boldsymbol{rd}, \boldsymbol{ra})$**.** Parse $ct, vk_p \leftarrow rd$ and $sk_p \leftarrow ra$. If $|ct| > \beta$ bytes, return $\perp$. Otherwise, create a block that contains the set of transactions $\mathbf{tx'} \cup tx(vk_p, vk', \mathrm{x})$, where $\mathbf{tx'} := \Pi(ct, \mathbf{tx})$. Finally, publish the block. Here vk' is a change address to recover the coins spent from vk_p.

Table 6.2.: Comparison of cryptocurrencies in terms of average number of transactions per block, bandwidth (bytes) provided per block and block creation rate (seconds).

Cryptocurrency	Tx per Block	Bandwidth	Block Time
Bitcoin	1500	1700	600
Zcash	9	2	150
Ethereum	115	77	20
Monero	5	1	120

$\{\boldsymbol{rd}, \perp\} \leftarrow \boldsymbol{TxDec}_b(\boldsymbol{rtx})$**.** Compute $ct \leftarrow \mathsf{GetPerm}(\mathbf{tx}, \mathbf{tx'})$. Moreover, extract vk_p from the extra transaction in the block of the form $tx(vk_p, vk', \mathrm{X})$. Finally, return the tuple $cd := (vk_p, ct)$.

6.5.6 Summary of Our Findings

We compare the feasibility of different cryptocurrencies as rendezvous in Table 6.3. We observe that shielded Zcash transactions provide the most bandwidth with 1168 bytes at a low cost of less than 0.01 USD. The downside is that only 11% of the transactions within Zcash are shielded, however, in the past month (Oct 2019) this number has increased to 19% [181]. Moreover, Zcash currently has the lowest market capitalization and exerts the lowest economic impact for a censor if it decides to ban it when compared to the other cryptocurrencies in this work.

Bitcoin (presented in Section 6.5.1) provides 20 and 40 bytes for the challenge and response messages correspondingly. Our Bitcoin-based solution relies on a transaction type used by more than 32% of the Bitcoin transactions, therefore hindering the censor's task. However, the fees in Bitcoin are the largest among all, and our encoding method entails the loss of coins as they are sent to unrecoverable addresses. Fortunately, it is possible to lower the cost by deploying the same encoding techniques

Table 6.3.: Comparison of the different rendezvous. Here, we consider the coins market value [186] at the time of writing (Nov. 27th 2019). We denote Zcash *transparent* by Z(Tr) and *shielded* by Z(Sh). Similar to Zcash(Tr), results for Bitcoin can be applied to Altcoins following the Bitcoin transaction patterns.

		Bitcoin	Z(Tr)	Z(Sh)	Monero	Ethereum
Challenge	Bandwidth	20	20	1148	640	20
	Tx fee	\$0.34	\$0.003	\$0.003	\$0.03	\$0.18
	Lost coins	\$0.18	\$0.01	—	—	—
Response	Bandwidth	40	40	1168	640	20
	Tx fee	\$0.34	\$0.003	\$0.003	\$0.03	\$0.15
	Lost coins	\$0.36	\$0.02	—	—	—
	Sibling txs	32%	81%	19%	42%	$> 8\%$
	Market cap \$	136B	230M	230M	950M	16B

over Zcash transparent transactions as they are conceptually identical to Bitcoin transactions. These transparent transactions have the lowest fees among all of the cryptocurrencies, with only 0.003 `USD`.

After shielded Zcash, Monero (Section 6.5.3) provides the most bandwidth with 640 bytes. Interestingly, the fee associated with the Monero transactions are the second lowest among the four cryptocurrencies. The type of transactions we consider in Monero blends in with 42% of all Monero transactions, making it difficult for the censor to block all such transactions. Ethereum provides the least amount of bandwidth (only 20 bytes in each direction according to the encoding scheme in Section 6.5.4) and a moderate cost of 0.18 `USD` compared to the other cryptocurrencies.

6.6 Implementation and Evaluation

We have developed a prototypical python implementation [187] to show the feasibility and practicality of *MoneyMorph*. We divide the implementation into two separate tasks: i) cryptographic operations and ii) cryptocurrency specific transaction encoding. In the remaining of this section, we detail each of the tasks.

6.6.1 Cryptographic Operations

As shown in Figure 6.2, the SB scheme requires two main cryptographic operations. First is the symmetric key derivation required by $SBEnc_f$ and $SBDec_f$, where each party (a user or the decoder) uses its private key along with the public key of the other party to perform the Diffie-Hellman key exchange. Next, using this shared key, it obtains symmetric keys for the encryption and decryption of the challenge and response messages. For the implementation, we leveraged the "ecdsa" [188] and "cryptography" [189] libraries to perform the Diffie-Hellman key exchange and derive the mentioned keys. The process of creating a fresh pair of keys takes the censored user about 120 milliseconds and deriving the shared key and symmetric keys for the encryption about 41 milliseconds on average.

Second, we have the encryption and decryption of the challenge and response messages. The timing of the encryption and decryption is dependent on the length of the messages. In our experiment, both encryption and decryption take less than 1 milliseconds. The challenge message sent by the user was chosen as *tor—obfs3* with the tag *00000000*. The hexadecimal representation of the message, key, and cipher is presented below.

```
Message:  30303030303030746f722d2d2d2d6f62667333
Enc Key:  399b8178571ea29a8094562e2200f6ad1b024f8f
Cipher:   09abb148672e92aaf4fb24030f2ddbc279643cbc
```

For the operations mentioned above, we used a personal commodity machine with an Intel Core i7, 2.2 GHz processor, and 16 GB RAM. Note that the blocks in all cryptocurrencies are generated with some specified creation rate, resulting in a time gap between the sequential blocks. We observe that a commodity machine, such as the one that we have used, will have more than enough capabilities to serve the users in multiple cryptocurrencies. Using higher performance machines will only improve the timings of the mentioned operations; However, the end result remains the same as the results will not be immediate for the users due to the time gap between the blocks. We further investigated the possibility of hosting the decoder on cloud platforms and observed that the price of a simple machine with 1TB of storage for multiple blockchains would be less than 100 USD per month [190, 191].

6.6.2 Transaction Encoding

After encrypting the challenge and response messages, the parties need to encode them into one of the cryptocurrencies' transactions. To show the feasibility and practicality of the instantiations mentioned in Section 6.5, we implemented the encoding scheme for three of the cryptocurrencies and deployed the challenge and response transactions to the corresponding test network.

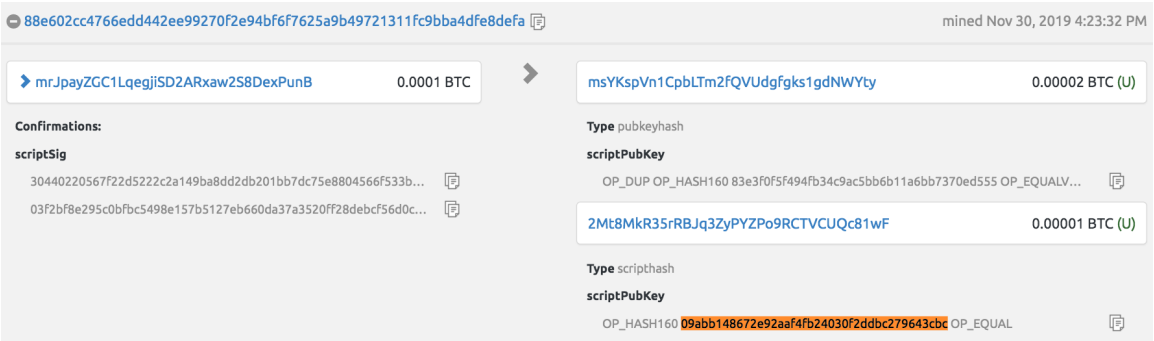


Fig. 6.7.: Snapshot of the challenge transaction on Bitcoin testnet. The highlighted hexadecimals are showing the Cipher mentioned above inside the scriptpubkey (SPKey).

Bitcoin. The implementation details of the encoding scheme in Bitcoin are provided in Section 6.5.1. In summary, using the "bitcoin-utils" library [192], we constructed and deployed two transactions on to the Bitcoin's blockchain. The first transaction, tx_1 (`https://tinyurl.com/wvmlyyz`) sent by the user, includes two outputs of the type Pay2PKeyHash and Pay2ScriptHash. The first output is the paying address that the decoder uses to send the response to the user. The second output is the 20 byte ciphertext mentioned in Section 6.6.1 (`09abb148672e92aaf4fb24030f2ddbc279643cbc`) and shown in Figure 6.7. Similarly, the decoder encodes the response message into tx_2 using the first output of tx_1 (`https://tinyurl.com/qsh5a7y`). We further analyzed the time it takes the decoder to process the transactions in a block. We observed that it takes less than 100 milliseconds to retrieve a block that only contains the hashes of its transaction. It takes an additional 2-5 millisecond for each of the transactions to be fetched and decoded. Overall, a block of Bitcoin containing about $2,000$ transactions can be examined by the decoder to find challenge messages in less than 15 seconds.

Ethereum. The implementation details of the encoding scheme in Ethereum are provided in Section 6.5.4. In summary, using the "web3" library, we encoded the challenge and response messages into transactions that were sent to the *etherdelta2* contract. However, since we were working on the testnet (the Rinkeby network), first we needed

to deploy the contract (presented in `https://tinyurl.com/s7w9sxe`). Next, the censored user funds the paying address (vk_p) using tx_0 (`https://tinyurl.com/rqy4ns9`) and sends the challenge covertext via tx_1 (`https://tinyurl.com/vq6b3qn`). Similar to tx_1, the decoder used the funds in vk_p (obtained from tx_0) and sends the response covertext through tx_2 (`https://tinyurl.com/uol385r`). Keep in mind that the challenge and response ciphertexts are fragmented into 4 pieces and placed as input values to the *testTrade* function. Similar to Bitcoin, we analyzed the time it takes the decoder to process the transactions in an Ethereum block. Overall, a block of Ethereum containing on average 100 transactions can be examined by the decoder to find challenge messages in less than 1 second.

Zcash. The implementation details of the encoding scheme in Zcash are provided in Section 6.5.2. We observed that, the most common transaction has one transparent input and two shielded outputs. This allows parties to send funds to themselves without loosing any coins (as the values and addresses of the shielded outputs are hidden) other than the transaction fee[1]. For the implementation we used the "zcash-cli" (command line client) to construct, send and receive transactions. The challenge and response data were included in the memo field of tx_1 (`https://tinyurl.com/wtku9ge`) and tx_2 (`https://tinyurl.com/vnleyhb`). We note that our tx_1 includes an extra transparent output for retrieving the remainder of the coins for testing purposes and similar to tx_2 (that has no transparent outputs) it can easily be omitted. Zcash, compared to the other two cryptocurrencies, has a small number of transactions per block. With an average of 5 transactions per block, it takes a decoder less than a second to process a block and look for challenge messages.

Transaction Finality. Previously, we saw that the time it takes to process a block to decode the challenge and response covertexts in all the three cryptocurrencies is insignificant. However, the procedure for sending the challenge and response covertexts is bounded by the transaction finality. We observe that transaction finality intro-

[1]This feature allows the users to send the shielded output directly to the shielded address of the decoder known by all.

duces a latency proportional to the rate of cryptocurrency blocks creation. Among the three cryptocurrencies, Bitcoin has the slowest block creation time at 10 minutes on average. Next is Zcash with an average block creation rate of 2.5 minutes and lastly Ethereum with a fast creation rate of 15 seconds. These results show that the throughput for any instantiation of *MoneyMorph* is only limited by the blocks generation rate. Fortunately, bootstrapping of censorship-resistance credentials does not have to be real-time; instead, it is carried out infrequently by each user (e.g., emails are currently used to find Tor bridges).

Simple Payment Verification. Although we recommend the censored users to run a full node for the cryptocurrency used as rendezvous if storage and computation overhead is prohibitive, they can run the Simple Payment Verification (SPV) client software to query expected blocks from available full nodes (multiple nodes for redundancy and security). As explained in Section 6.5, *MoneyMorph* ensures that clients know what stealth address should to query to retrieve the response, thereby reducing the number of required blocks. While improving the usability, this approach comes, however, with a tradeoff in security guarantees. First, SPV clients are more susceptible to eclipse attacks similar to those in [193,194]. Second, the selective query of transactions (e.g., using Bloom filters) could facilitate the detectability task of the censor, similar to [195]. Recent advances aim to mitigate those detectability issues in Bitcoin [196, 197] and Zcash [198], and similar techniques could be applied for Ethereum and Monero.

6.7 Concluding Remarks and Future Work

Despite the many academic and practical alternatives for censorship resistance, censorship remains an important problem that hinders numerous people from freely accessing and communicating information. We explore the use of the widely deployed blockchain technologies as a communication channel in the presence of a censor and we observe that the blockchain transactions enable communication channels offering

interesting tradeoffs between bandwidth, costs and censorship resistance. In particular, we describe for the first time communication channels fully compatible not only with Bitcoin but also with Zcash, Monero, and Ethereum that allow censored users to get bootstrapping credentials.

Nevertheless, this work only scratches the tip of the iceberg. The different permissionless blockchains are in continuous development, and new features are being added continuously that may come with yet unexplored possibilities to build a communication channel. For instance, the deployment of off-chain payment channels [199, 200] adds a new locking mechanism to Bitcoin and the alike cryptocurrencies with extra fields that can potentially be used to encode extra covertext bytes of the communication between censored user and decoder. This work sets the grounds for future research works exploring the use of blockchain for censorship resistance communications.

7. SUMMARY

In this dissertation, we have presented Crowd Blending techniques that can be used to create privacy preserving systems while maintaining their utility. In particular, we focused on two privacy tasks for two different data-sharing platforms presented below.

Concealing content deletion on social media platforms. In the world with perfect and permanent memory, we are in dire need of mechanisms to restore the ability to forget. Providing users with the ability to delete their past data is insufficient. In fact, against an adversary who can persistently observe a user's data, the user's deletions make her/him more vulnerable by directly pointing the adversary to sensitive information.

To that end, to affirmatively understand the users' perceptions for deletion privacy in social platforms, for the first time ever, we conducted user study with 191 participants. We investigate their prior deletion experiences, their expectations of deletion privacy, and how effective do they find the current deletion mechanisms. We find that the participants are significantly more concerned about their deletions being noticed by large-scale data collectors (e.g., a third-party data collecting company or the government) than any other individual from their social circle. To address the concern of the users in protecting their damaging deletions against large-scale data collectors we propose two new deletion mechanism that significantly raises the bar for such adversaries in identifying the damaging deletions.

In the first mechanism we have defined, formalized, and addressed the problem of deletion concealment by designing *Lethe*. In particular, we have formally defined a novel intermittent withdrawal mechanism, quantified its privacy, availability, and adversarial overhead guarantees in the form of a tradeoff. Leveraging real-world data

we have demonstrated the efficacy of *Lethe* in providing a good tradeoff between privacy, availability, adversarial overhead and platform utility. For example, we have shown that while maintaining 95% availability and utility as high as 99.7%, we can offer deletion privacy for up to 3 months from the time of deletion while still keeping the adversarial precision to 20%.

In the second mechanism, we introduce another novel deletion mechanism, Deceptive Deletions, that raises the bar for the adversary in identifying damaging content. Given a set of damaging posts (posts that adversary can leverage to blackmail the user) that users want to delete, the Deceptive Deletion system *selects* k additional posts for each damaging post (similar to the damaging post) and deletes them along with the damaging posts. As the global adversary can only observe all of these deletions together, its goal will be to distinguish the damaging posts from the decoy posts. Using real-world Twitter data we demonstrated the effectiveness of the system. Specifically, we showed that even when we consider only two decoy posts per damaging deletion, the adversarial performance (F-score) drops to 42% from 75% in the absence of any privacy-preserving deletion mechanism.

Concealing censored information in cryptocurrency blockchains. Despite the many academic and practical alternatives for censorship resistance, censorship remains an important problem that hinders numerous people from freely accessing and communicating information. We explore the use of the widely deployed blockchain technologies as a communication channel in the presence of a censor and we observe that the blockchain transactions enable multiple communication channels offering interesting tradeoffs between bandwidth, costs and censorship resistance. In particular, we describe for the first time communication channels fully compatible not only with Bitcoin but also with Zcash, Monero and Ethereum that allows censored users to get bootstrapping credentials.

APPENDIX

A. CONTENT DELETION APPENDIX

A.1 An Overview of *Lethe* Implementation

In section 3.4 we presented the basic steps of *Lethe*. However in our work we considered that *Lethe* should be applied to each single post. So a very practical question is: How to efficiently implement *Lethe* in a platform? Here we provide a brief implementation sketch.

Basic setup for a platform. We assume a generic archival platform where each post is stored as an Active Store Object (ASO) [201]. ASOs are simply key-value pairs with some (optional) code to run on values. Traditionally this ASO code is written in terms of handlers (e.g., code to handle deletion). Each post ASO will have an unique post id as key, the user generated post content as value, identification of the owner (as authentication token) and some metadata (e.g., the real state flag for a post). We further assume that there is an internal trusted time server, which is used throughout the platform for synchronizing operations. The platform internally does not use any other timestamps. Any mention of timestamps in this section refers to this internal timestamp. Extending this set-up to traditional databases is simple and left to future work.

We use an architecture similar to Comet [201], where the platform operator as well as platform users (including adversary) have some specific Application Programmer Interfaces (API) to access/create/delete the posts. However note that in our adversary model, the adversary can just query the posts and can not change them in any way. Thus, unlike Comet in *Lethe* post ASO objects are immutable from the point of view of an adversary.

Straw man implementation. A straightforward implementation of *Lethe* is to add an "observable state" flag (binary) with meta data of each post ASO. Whenever

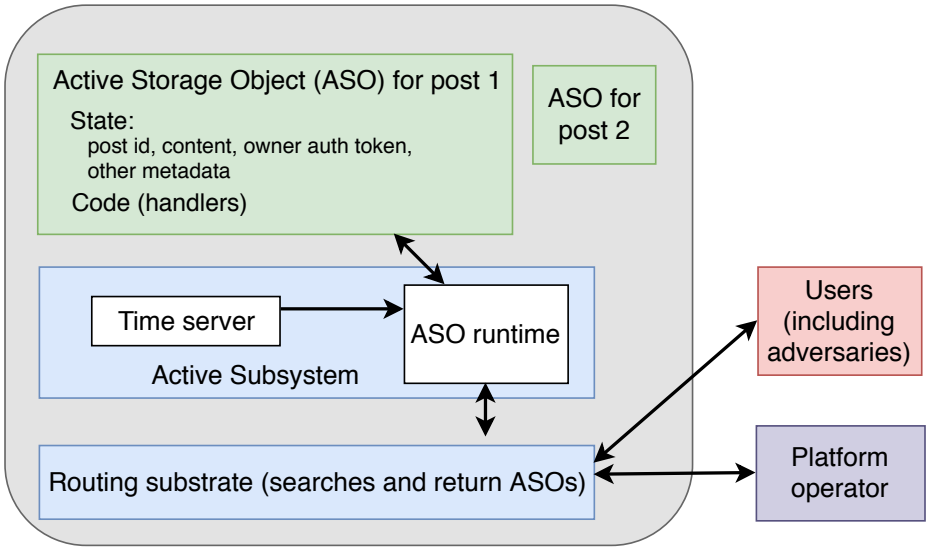


Fig. A.1.: A basic implementation schematic for *Lethe*. Each post is an ASO, and using APIs and code handlers these ASOs can be accessed. An operator can add more metadate to the ASO content according to requirement of the platform.

a post is created, the platform operator assigns a process (or a thread) to the post. That process will apply *Lethe* algorithm to toggle the observable state. In case of a view request, another user initiated process will seek the required ASO or ASOs, check the "observable state" flag and return a post if the post is observable (i.e., observable state flag is TRUE). However, this design if definitely not scalable for a platform with billions of posts. Thus we need an improved implementation.

Key insight. Our key insight is simple—the platform can precompute the timestamps for future up and down durations and then lazily update those duration timestamps. At any current time, for a view request, the platform operator can use the current timestamp to determine if the post should be in up or down duration (using the precomputed up/down duration timestamps) and return a post in case the post is in up duration or return null otherwise. The only exception is if the data owner requested to view her own post, the post should be returned, irrespective of up/down duration.

An improved *Lethe* implementation. We note that instead of keeping track of the observable state, a process can simply compute the observable state of an ASO

using the current timestamp and the precomputed up/down duration timestamps. Thus, when the platform operator adds each post ASO, they should also add (in bulk) the timestamps corresponding to up and down durations for a large time period in future (e.g., for next one year). Specifically we present the basic schematic of our proposed implementation in Figure A.1. Each post ASO contains a state which includes the post content, an authentication token to identify the owner of the post (who can delete the post) and timestamps for future up and down durations. Both users (including adversaries) and platform uses a routing substrate to find ASOs in the distributed storage (e.g., via a hash table of keys). The active subsystem contains a trusted time server and the ASO runtime, which converts platform and user API calls to ASO handlers and executes the ASO handler code.

Table A.1 contains the summary of API and ASO handler code descriptions. We use authentication (or auth) tokens to identify a user (to determine data owner or not). Any user can create a post using her auth token with *put* or delete her posts using *delete*. Handler code for call *get* first checks the auth token and if the request is from data owner the platform always returns the post (if it is not deleted). If the *get* request is not from a data owner, then (using the precomputed up/down durations) the handler code checks if the current timestamp is within the up of down time duration. If the current timestamp falls in an up duration for the post then the platform returns the post's content to the requesting user, otherwise the platform returns null. In addition to *get*, *put* and *delete*, the platform operator internally runs multiple processes with $updateTS$ function to keep adding future up and down time durations for ASO objects. The "post_ids" to update (given to $updateTS$) should be divided in these processes based on a hash table of ASO keys. The mapping between API and ASO handler codes is in 3^{rd} column of Table A.1.

Possible optimizations of this implementation sketch. We emphasize that this is just a sketch *Lethe* implementation with scopes for further optimization. E.g., $updateTS$ can additionally delete up/down timestamps lesser than current timestamp to optimize storage or there can be batch garbage collection after multiple calls to

APIs for the user (including adversary)			
Name	**Parameter**	**Description**	**Associated ASO handlers**
put	post_content, authentication_token	Creates a post ASO for the data owner, returns a post_id	*onPut*
get	post_id, authentication_token	Returns a post ASO or null depending on (i) ownership and (ii) if the post is in up/down duration.	*onGet*
delete	post_id, authentication_token	deletes associated post and returns null.	*onDelete*
Internal APIs for the platform			
Name	**Parameter**	**Description**	**Associated ASO handlers**
updateTS	list of post_ids to update	Updates the future up/down times in ASOs with post_ids.	*onUpdate*
Handlers in ASOs			
Name	**Parameter**	**Description**	**Associated ASO handlers**
onPut	post_content, authentication_token, current_timestamp	Creates an ASO object and assigns up/down timestamps covering next 1 year.	-
onGet	post_id, authentication_token, current_timestamp	Check current timestamp and if in up duration return post content, else return null.	-
onDelete	post_id, authentication_token, current_timestamp	Assign one down timestamp—infinity; remove post content.	-
onUpdate	post_id, authentication_token, current_timestamp	If current set of up/down times cover less than 1 year, create more up/down times.	-

Table A.1.: Summary of API and ASO handler code descriptions (and the mapping between them) for *Lethe* sketch implementation. Note that data owner always gets back her non-deleted posts irrespective of up/down duration.

delete. Further the input to *updateTS* can be chosen more intelligently e.g., by keeping a min-heap to determine the ASO objects which are in immediate need to update up/down timestamp. We leave exploration of these concrete system challenges to future work.

REFERENCES

REFERENCES

[1] S. C. Jansen and B. Martin, "The streisand effect and censorship backfire," 2015.

[2] "Gray man – how to blend in by hiding in plain sight," https://www.skilledsurvival.com/gray-man/.

[3] "Gray man theory: The art of blending in during disaster," https://www.thebugoutbagguide.com/gray-man-theory/.

[4] L. Bauer, L. F. Cranor, S. Komanduri, M. L. Mazurek, M. K. Reiter, M. Sleeper, and B. Ur, "The post anachronism: The temporal dimension of facebook privacy," in *ACM WPES '13*.

[5] O. Ayalon and E. Toch, "Retrospective privacy: Managing longitudinal privacy in online social networks," in *SOUPS'13*.

[6] M. Mondal, J. Messias, S. Ghosh, K. P. Gummadi, and A. Kate, "Forgetting in social media: Understanding and controlling longitudinal exposure of socially shared data," in *USENIX SOUPS '16*.

[7] "Politwoops," http://politwoops.sunlightfoundation.com/, (Accessed on February 2018).

[8] "Resavr," https://www.resavr.com/, 2017, (Accessed on February 2018).

[9] "Uneddit," https://web.archive.org/web/20170824002119/http://uneddit.com/, 2017, (Accessed on February 2018).

[10] "Stackprinter," http://www.stackprinter.com/deleted, 2017, (Accessed on February 2018).

[11] YouTomb, https://web.archive.org/web/20141029040225/http://youtomb.mit.edu/, 2017, (Accessed on February 2018).

[12] M. C. Tschantz, S. Afroz, Anonymous, and V. Paxson, "SoK: Towards Grounding Censorship Circumvention in Empiricism," in *(S&P)*, 2016, pp. 914–933.

[13] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg, "Sok: Making sense of censorship resistance systems," *PoPETs*, vol. 2016, no. 4, pp. 37–61, 2016.

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[15] "Bitcoin Purchase in China," https://paxful.com/china/buy-bitcoin.

[16] V. Buterin and E. Foundation, "A next-generation smart contract and decentralized application platform," https://github.com/ethereum/wiki/wiki/White-Paper.

[17] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," in *S&P*, 2014.

[18] N. van Saberhagen, "Cryptonote v 2.0," https://cryptonote.org/whitepaper.pdf, 2013.

[19] M. Minaei, M. Mondal, P. Loiseau, K. Gummadi, and A. Kate, "Lethe: Conceal content deletion from persistent observers," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 1, pp. 206–226, 2019.

[20] M. Minaei, S. C. Mouli, M. Mondal, B. Ribeiro, and A. Kate, "Deceptive deletions for protecting withdrawn posts on social platforms," 2020.

[21] M. Minaei, P. Moreno-Sanchez, and A. Kate, "Moneymorph: Censorship resistant rendezvous using permissionless cryptocurrencies," *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 404–424, 2020.

[22] "Tor:bridges," https://www.torproject.org/docs/bridges.

[23] "Removeddit," http://removeddit.com/, 2020.

[24] "24 tweets ed sheeran will probably delete soon," https://www.buzzfeed.com/mjs538/we-r-who-we-r-is-a-good-song-tho, 2017, (Accessed on February 2018).

[25] "Snl's first latina cast member is caught out deleting thousands of tweets, some of which were 'racist and offensive'," http://www.dailymail.co.uk/news/article-3805356/SNL-s-Latina-cast-member-caught-deleting-thousands-tweets-racist-offensive.html, 2016, (Accessed on February 2018).

[26] M. Xue, G. Magno, E. Cunha, V. Almeida, and K. W. Ross, "The right to be forgotten in the media: A data-driven study," *PoPETs*, vol. 2016, no. 4, pp. 389–402, 2016.

[27] "Collection of deleted tweets & annoying content," https://twitter.com/fallaitpassuppr?lang=en, 2019.

[28] H. Nissenbaum, *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, 2010.

[29] M. Sleeper, J. Cranshaw, P. G. Kelley, B. Ur, A. Acquisti, L. F. Cranor, and N. Sadeh, ""I Read My Twitter the Next Morning and Was Astonished": A Conversational Perspective on Twitter Regrets," in *Proc. CHI*, 2013.

[30] Y. Wang, G. Norcie, S. Komanduri, A. Acquisti, P. G. Leon, and L. F. Cranor, ""I Regretted the Minute I Pressed Share": A Qualitative Study of Regrets on Facebook," in *Proc. SOUPS*, 2011.

[31] L. Zhou, W. Wang, and K. Chen, "Tweet properly: Analyzing deleted tweets to understand and identify regrettable ones," in *Proceedings of the 25th International Conference on World Wide Web (WWW'16)*, 2016.

[32] M. Mondal, J. Messias, S. Ghosh, K. P. Gummadi, and A. Kate, "Forgetting in Social Media: Understanding and Controlling Longitudinal Exposure of Socially Shared Data," in *Proc. SOUPS*, 2016.

[33] Mainack Mondal and Günce Su Yılmaz and Noah Hirsch and Mohammad Taha Khan and Michael Tang and Christopher Tran and Chris Kanich and Blase Ur and Elena Zheleva, "Moving Beyond Set-It-And-Forget-It Privacy Settings on Social Media ," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS'19)*, 2019.

[34] N. Apthorpe, Y. Shvartzshnaider, A. Mathur, D. Reisman, and N. Feamster, "Discovering smart home internet of things privacy norms using contextual integrity," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 2, pp. 1–23, 2018.

[35] N. Apthorpe, S. Varghese, and N. Feamster, "Evaluating the contextual integrity of privacy regulation: Parents' iot toy privacy norms versus {COPPA}," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 123–140.

[36] R. A. Hill and R. I. M. Dunbar, "Social network size in humans," *Human Nature*, vol. 14, no. 1, 2003.

[37] V. Arnaboldi, M. Conti, A. Passarella, and F. Pezzoni, "Analysis of Ego Network Structure in Online Social Networks," in *Proceedings of the 4th ASE/IEEE International Conference on Social Computing (SocialCom'12)*, Amsterdam, The Netherlands, September 2012.

[38] L. Zhou, W. Wang, and K. Chen, "Tweet properly: Analyzing deleted tweets to understand and identify regrettable ones," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 603–612.

[39] H. Almuhimedi, S. Wilson, B. Liu, N. Sadeh, and A. Acquisti, "Tweets are forever: a large-scale quantitative analysis of deleted tweets," in *Proceedings of the 2013 conference on Computer supported cooperative work*, 2013, pp. 897–908.

[40] "Prolific academic," https://www.prolific.ac/, 2020.

[41] E. Peer, L. Brandimarte, S. Samat, and A. Acquisti, "Beyond the turk: Alternative platforms for crowdsourcing behavioral research," *Journal of Experimental Social Psychology*, vol. 70, pp. 153–163, 2017.

[42] S. Albakry, K. Vaniea, and M. K. Wolters, "What is this url's destination? empirical evaluation of users' url reading," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[43] "Educational attainment in the united states," https://www.census.gov/data/tables/2019/demo/educational-attainment/cps-detailed-tables.html, 2019.

[44] P. Hitlin, "Turkers in this canvassing: young, well-educated and frequent users," *P. Hitlin, Research in the crowdsourcing age, a case study*, pp. 20–30, 2016.

[45] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[46] R. Tinati, A. Madaan, and W. Hall, "Instacan: Examining deleted content on instagram," in *Proceedings of the 2017 ACM on Web Science Conference*, 2017, pp. 267–271.

[47] H. Almuhimedi, S. Wilson, B. Liu, N. Sadeh, and A. Acquisti, "Tweets Are Forever: A Large-Scale Quantitative Analysis of Deleted Tweets," in *Proc. CSCW*, 2013.

[48] A. Kruglanski and D. Webster, "Motivated closing of the mind: "seizing" and "freezing"," *Psychol Rev*, vol. 103(2), pp. 263–283, 1996.

[49] "Snapchat," https://www.snapchat.com/.

[50] H. Almuhimedi, S. Wilson, B. Liu, N. Sadeh, and A. Acquisti, "Tweets are forever: A large-scale quantitative analysis of deleted tweets," in *CSCW'13*.

[51] R. H. Weber, "The right to be forgotten more than a pandora's box?" *jipitec*, vol. 2, no. 2, 2011.

[52] "Art. 17, general data protection regulation, right to be forgotten," https://gdpr-info.eu/art-17-gdpr/, 2016, (Accessed on February 2018).

[53] "Dust," https://www.usedust.com/, 2016, (Accessed on February 2018).

[54] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data," in *USENIX Security Symposium '09*.

[55] R. Geambasu, T. Kohno, A. Krishnamurthy, A. Levy, H. M. Levy, P. Gardner, and V. Moscaritolo, "New directions for self-destructing data," University of Washington, Tech. Rep. UW-CSE-11-08-01, 2011.

[56] C. Castelluccia, E. De Cristofaro, A. Francillon, and M.-A. Kaafar, "Ephpub: Toward robust ephemeral publishing," in *ICNP'11*.

[57] S. Reimann and M. Dürmuth, "Timed revocation of user data: Long expiration times from existing infrastructure," in *WPES'12*.

[58] R. Perlman, "The ephemerizer: Making data disappear," Sun Microsystems, Inc., Tech. Rep. SMLI TR-2005-140, 2005.

[59] S. K. Nair, M. T. Dashti, B. Crispo, and A. S. Tanenbaum, "A hybrid PKI-IBC based ephemerizer system," in *SEC'07*.

[60] A. Zarras, K. Kohls, M. Dürmuth, and C. Pöpper, "Neuralyzer: Flexible Expiration Times for the Revocation of Online Data," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016.

[61] G. E. Hine, J. Onaolapo, E. De Cristofaro, N. Kourtellis, I. Leontiadis, R. Samaras, G. Stringhini, and J. Blackburn, "Kek, cucks, and god emperor trump: A measurement study of 4chan's politically incorrect forum and its effects on the web." in *ICWSM*, 2017, pp. 92–101.

[62] "4chan raids: how one dark corner of the internet is spreading its shadows," http://www.theconversation.com/4chan-raids-how-one-dark-corner-of-the-internet-is-spreading-its-shadows-68394, 2016, (Accessed on February 2018).

[63] "How tweet it is!: Library acquires entire twitter archive," http://blogs.loc.gov/loc/2010/04/how-tweet-it-is-library-acquires-entire-twitter-archive/, 2010, (Accessed on February 2018).

[64] K. S. Maddock, Jim and R. M. Mason, "Using historical twitter data for research: Ethical challenges of tweet deletions," in *CSCW '15 Workshop on Ethics*.

[65] B. Krishnamurthy and C. E. Wills, "On the leakage of personally identifiable information via online social networks," in *WOSN'09*.

[66] A. Srivastava and G. Geethakumari, "Measuring privacy leaks in online social networks," in *ICACCI'13*.

[67] B. Krishnamurthy, K. Naryshkin, and C. Wills, "Privacy leakage vs. protection measures: the growing disconnect," in *WEB 2.0 SECURITY & PRIVACY 2011*.

[68] G. Casella and R. L. Berger, *Statistical Inference*, 2nd ed. Duxbury Press, 2002.

[69] C. Dwork, "Differential privacy," in *ICALP'06*.

[70] C. Walck, "Handbook on statistical distributions for experimentalists."

[71] "Twitter puts trillions of tweets up for sale to data miners," https://www.theguardian.com/technology/2015/mar/18/twitter-puts-trillions-tweets-for-sale-data-miners, 2017, (Accessed on February 2018).

[72] "Twitter's evolving plans to make money from its data stream," https://bits.blogs.nytimes.com/2015/04/11/twitters-evolving-plans-to-make-money-from-its-data-stream, 2017, (Accessed on February 2018).

[73] "The streaming apis," https://dev.twitter.com/streaming/overview, 2017.

[74] D. van Liere, "How far does a tweet travel?: Information brokers in the twitterverse," in *MSM '10*.

[75] "Replies and retweets on twitter," https://sysomos.com/inside-twitter/twitter-retweet-stats/, 2017, (Accessed on February 2018).

[76] M. Conover, J. Ratkiewicz, M. R. Francisco, and B. Gonçalves, "Political polarization on twitter." 2011.

[77] D. Boyd, S. Golder, and G. Lotan, "Tweet, tweet, retweet: Conversational aspects of retweeting on twitter," in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*. IEEE, 2010, pp. 1–10.

[78] M. Gomez-Rodriguez, K. P. Gummadi, and B. Schölkopf, "*Quantifying Information Overload in Social Media and Its Impact on Social Contagions*," in *ICWSM'14*.

[79] L. Zhou, W. Wang, and K. Chen, "Tweet properly: Analyzing deleted tweets to understand and identify regrettable ones," in *Proceedings of the 25th International Conference on World Wide Web (WWW'16)*, April 2016.

[80] Q. Wang, H. Xue, F. Li, D. Lee, and B. Luo, "# donttweetthis: Scoring private information in social networks," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 4, pp. 72–92, 2019.

[81] C. Jin, P. Netrapalli, and M. I. Jordan, "Minmax optimization: Stable limit points of gradient descent ascent are locally optimal," *arXiv preprint arXiv:1902.00618*, 2019.

[82] S. Garg, S. Goldwasser, and P. N. Vasudevan, "Formalizing data deletion in the context of the right to be forgotten," in *Advances in Cryptology—EUROCRYPT*, 2020, pp. 373–402.

[83] D. C. Howe and H. Nissenbaum, "Trackmenot: Resisting surveillance in web search," *Lessons from the Identity trail: Anonymity, privacy, and identity in a networked society*, vol. 23, pp. 417–436, 2009.

[84] M. Murugesan and C. Clifton, "Providing privacy through plausibly deniable search," in *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM, 2009, pp. 768–779.

[85] J. Domingo-Ferrer, A. Solanas, and J. Castellà-Roca, "h (k)-private information retrieval from privacy-uncooperative queryable databases," *Online Information Review*, vol. 33, no. 4, pp. 720–744, 2009.

[86] S. T. Peddinti and N. Saxena, "On the privacy of web search based on query obfuscation: a case study of trackmenot," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2010, pp. 19–37.

[87] "Trackmenot," https://cs.nyu.edu/trackmenot/, 2017.

[88] S. T. Peddinti and N. Saxena, "On the effectiveness of anonymizing networks for web search privacy," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 483–489.

[89] E. Balsa, C. Troncoso, and C. Diaz, "Ob-pws: Obfuscation-based private web search," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 491–505.

[90] A. Petit, T. Cerqueus, S. B. Mokhtar, L. Brunie, and H. Kosch, "Peas: Private, efficient and accurate web search," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 571–580.

[91] P. Papadopoulos, A. Papadogiannakis, M. Polychronakis, A. Zarras, T. Holz, and E. P. Markatos, "K-subscription: Privacy-preserving microblogging browsing through obfuscation," in *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 2013, pp. 49–58.

[92] N. Dalvi, P. Domingos, S. Sanghai, D. Verma *et al.*, "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 99–108.

[93] J. Jia and N. Z. Gong, "Attriguard: A practical defense against attribute inference attacks via adversarial machine learning," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 513–529.

[94] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners." in *AAAI*, 2015, pp. 2871–2877.

[95] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in neural information processing systems*, 2017, pp. 3517–3529.

[96] Amazon, "Mechanical Turk," https://www.mturk.com/mturk/welcome, 2010.

[97] "Tweetdelete," https://www.tweetdelete.net/, 2017.

[98] "Twittereraser deletion service," https://www.tweeteraser.com/statistics, 2017.

[99] "Tweetdeleter: Delete many tweets with one click!" https://www.tweetdeleter.com, 2017.

[100] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[101] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 506–519. [Online]. Available: http://doi.acm.org/10.1145/3052973.3053009

[102] D. Correa, L. A. Silva, M. Mondal, F. Benevenuto, and K. P. Gummadi, "The Many Shades of Anonymity: Characterizing Anonymous Social Media Content," in *Proceedings of The 9th International AAAI Conference on Weblogs and Social Media (ICWSM'15)*, Oxford, UK, May 2015.

[103] J. A. Biega, K. P. Gummadi, I. Mele, D. Milchevski, C. Tryfonopoulos, and G. Weikum, "R-Susceptibility: An IR-Centric Approach to Assessing Privacy Risks for Users in Online Communities," in *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'16)*, 2016.

[104] J. Maddock, K. Starbird, and R. M. Mason, "Using historical twitter data for research: Ethical challenges of tweet deletions," in *CSCW 2015 Workshop on Ethics for Studying Sociotechnical Systems in a Big Data World. ACM*, 2015.

[105] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[106] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[107] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[108] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *Proceedings of the 26th annual computer security applications conference*. ACM, 2010, pp. 1–9.

[109] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia, "Detecting automation of twitter accounts: Are you a human, bot, or cyborg?" *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, pp. 811–824, 2012.

[110] J. P. Dickerson, V. Kagan, and V. Subrahmanian, "Using sentiment to detect bots on twitter: Are humans more opinionated than bots?" in *Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE Press, 2014, pp. 620–627.

[111] A. H. Wang, "Detecting spam bots in online social networking sites: a machine learning approach," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2010, pp. 335–342.

[112] A. Bessi and E. Ferrara, "Social bots distort the 2016 us presidential election online discussion," 2016.

[113] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The rise of social bots," *Communications of the ACM*, vol. 59, no. 7, pp. 96–104, 2016.

[114] "Twitwipe," http://twitwipe.com/, 2017.

[115] "Social book post manager," https://chrome.google.com/webstore/detail/social-book-post-manager/ljfidlkcmdmmibngdfikhffffdmphjae, 2020.

[116] "Cleaner for instagram," https://play.google.com/store/apps/details?id=ro.novasoft.cleanerig, 2020.

[117] "Nuke reddit history," https://chrome.google.com/webstore/detail/nuke-reddit-history/aclagjkmidmkcdhkhlicmgkgmpgccaod, 2020.

[118] I. A. Kash, J. K. Lai, H. Zhang, and A. Zohar, "Economics of bittorrent communities," in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 221–230.

[119] M. Ripeanu, M. Mowbray, N. Andrade, and A. Lima, "Gifting technologies: A bittorrent case study," *First Monday*, vol. 11, no. 11, p. 432, 2006.

[120] J. Mirkovic, G. Prier, and P. Reiher, "Attacking ddos at the source," in *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE, 2002, pp. 312–321.

[121] ——, "Source-end ddos defense," in *Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003.* IEEE, 2003, pp. 171–178.

[122] "Rate-limiting strategies and techniques," https://cloud.google.com/solutions/rate-limiting-strategies-techniques, 2020.

[123] "What is rate limiting? | rate limiting and bots," https://www.cloudflare.com/learning/bots/what-is-rate-limiting/, 2020.

[124] L. Vargas, G. Hazarika, R. Culpepper, K. R. Butler, T. Shrimpton, D. Szajda, and P. Traynor, "Mitigating risk while complying with data retention laws," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2011–2027.

[125] E. V. Mangipudi, K. Rao, J. Clark, and A. Kate, "Towards automatically penalizing multimedia breaches," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019, pp. 340–346.

[126] A. Juels, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *Proc. Networks and Distributed System Security Symposium (NDSS), 1999*, 1999.

[127] D. Dean and A. Stubblefield, "Using client puzzles to protect tls." in *USENIX Security Symposium*, vol. 42, 2001.

[128] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual International Cryptology Conference*. Springer, 1992, pp. 139–147.

[129] A. Miller, A. Kosba, J. Katz, and E. Shi, "Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 680–691.

[130] M. S. Riazi, "Towards a private new world: Algorithm, protocol, and hardware co-design for large-scale secure computation," Ph.D. dissertation, UC San Diego, 2020.

[131] ——, "Large-scale privacy-preserving matching and search," Ph.D. dissertation, 2016.

[132] M. S. Riazi, E. M. Songhori, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Toward practical secure stable matching," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 1, pp. 62–78, 2017.

[133] J. Darivandpour and M. J. Atallah, "Efficient and secure pattern matching with wildcards using lightweight cryptography," *Computers & Security*, vol. 77, pp. 666–674, 2018.

[134] M. Mondal, J. Messias, S. Ghosh, K. P. Gummadi, and A. Kate, "Longitudinal Privacy Management in Social Media: The Need for Better Controls," *IEEE Internet Computing*, vol. 21, no. 3, pp. 48–55, 2017.

[135] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, "Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication," in *41st IEEE Symposium on Secruity and Privacy (IEEE S&P)*. Oakland Association, 2020.

[136] "Bitcoin in Iran," https://localbitcoins.com/country/IR.

[137] E. Parker, "Can china contain bitcoin?" MIT Technology Review Blog Post. https://www.technologyreview.com/s/609320/can-china-contain-bitcoin, 2017.

[138] D. Nobori and Y. Shinjo, "Vpn gate: A volunteer-organized public vpn relay system with blocking resistance for bypassing government censorship firewalls." in *NSDI*, 2014.

[139] V. Perta, M. Barbera, G. Tyson, H. Haddadi, and A. Mei, "A glance through the vpn looking glass: Ipv6 leakage and dns hijacking in commercial vpn clients," *PoPETs*, no. 1, pp. 77–91, 2015.

[140] "Dynaweb," http://us.dongtaiwang.com/loc/download.en.php.

[141] "Ultrasurf," https://ultrasurf.us/.

[142] "Lantern," https://getlantern.org.

[143] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab, DC, Tech. Rep., 2004.

[144] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," in *CCS*, 2013, pp. 61–72.

[145] "Tor:mirrors," https://www.torproject.org/getinvolved/mirrors.html.en.

[146] "Tor:Hidden Service," https://www.torproject.org/docs/hidden-services.

[147] P. Winter and S. Lindskog, "How the Great Firewall of China is Blocking Tor," in *Free and Open Communications on the Internet, FOCI*, 2012.

[148] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall, "Analyzing the great firewall of china over space and time," *PoPETs*, no. 1, pp. 61–76, 2015.

[149] Z. Ling, J. Luo, W. Yu, M. Yang, and X. Fu, "Extensive analysis and large-scale empirical evaluation of tor bridge discovery," in *INFOCOM*, 2012, pp. 2381–2389.

[150] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, "Blocking-resistant communication through domain fronting," *PoPETs*, no. 2, pp. 46–64, 2015.

[151] "Meek transport," https://trac.torproject.org/projects/tor/wiki/doc/meek.

[152] "Summary of meek's costs, july 2016," https://lists.torproject.org/pipermail/tor-project/2016-August/000690.html.

[153] H. Zolfaghari and A. Houmansadr, "Practical censorship evasion leveraging content delivery networks," in *CCS*, 2016, pp. 1715–1726.

[154] J. Holowczak and A. Houmansadr, "Cachebrowser: Bypassing chinese censorship without proxies using cached content," in *CCS*, 2015, pp. 70–83.

[155] "Amazon Web Services starts blocking domain-fronting, following Google's lead," https://www.theverge.com/2018/4/30/17304782/amazon-domain-fronting-google-discontinued.

[156] E. Wustrow, S. Wolchok, I. Goldberg, and J. Halderman, "Telex: Anticensorship in network infratructure," in *USENIX Security*, 2011.

[157] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov, "Cirripede: Circumvention infrastructure using router redirection with plausible deniability," in *CCS*, 2011, pp. Pages 187–200.

[158] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. Mankins, and W. T. Strayer, "Decoy routing: Toward unblockable internet communication," in *FOCI*, 2011.

[159] E. Wustrow, C. Swanson, and A. Halderman, "Tapdance: End-to-middle anti-censorship without flow blocking," in *USENIX*, 2014, pp. 159–174.

[160] S. Frolov, F. Douglas, W. Scott, A. McDonald, B. VanderSloot, R. Hynes, A. Kruger, M. Kallitsis, D. G. Robinson, S. Schultze *et al.*, "An isp-scale deployment of tapdance," in *FOCI*, 2017.

[161] M. Nasr, H. Zolfaghari, and A. Houmansadr, "The waterfall of liberty: Decoy routing circumvention that resists routing attacks," in *CCS*, 2017, pp. 2037–2052.

[162] C. K. Luca Invernizzi and G. Vigna, "Message in a bottle: Sailing past censorship," *Computer Security Applications*, pp. 39–48, 2013.

[163] P. Lincoln, I. Mason, P. A. Porras, V. Yegneswaran, Z. Weinberg, J. Massar, W. A. Simpson, P. Vixie, and D. Boneh, "Bootstrapping communications into an anti-censorship system." in *FOCI*, 2012.

[164] R. Recabarren and B. Carbunar, "Tithonus: A bitcoin based censorship resilient system," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 1, pp. 68–86, 2019.

[165] M. Backes and C. Cachin, "Public-Key Steganography with Active Attacks," in *TCC*, 2005, pp. 210–226.

[166] "The Unexpected Fallout of Iran's Telegram Ban," https://www.wired.com/story/iran-telegram-ban/.

[167] F. Zhang, P. Daian, I. Bentov, and A. Juels, "Paralysis proofs: Safe access-structure updates for cryptocurrencies and more," in *Financial Crypto*, 2018.

[168] R. J. Anderson, "Stretching the Limits of Steganography," in *Information Hiding*, 1996, pp. 39–48.

[169] L. von Ahn and N. J. Hopper, "Public-Key Steganography," in *EUROCRYPT*, 2004.

[170] N. Hopper, "On Steganographic Chosen Covertext Security," in *ICALP*, 2005.

[171] N. Fazio, A. Nicolosi, and I. M. Perera, "Broadcast Steganography," in *Topics in Cryptology - CT-RSA*, 2014, pp. 64–84.

[172] "Psiphon," https://www.psiphon3.com/en/index.html.

[173] D. Cash, E. Kiltz, and V. Shoup, "The twin diffie-hellman problem and applications," *J. Cryptology*, vol. 22, no. 4, pp. 470–504, 2009.

[174] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson, "Non-interactive key exchange," in *PKC*, 2013, pp. 254–271.

[175] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *CRYPTO*, 2010, pp. 631–648.

[176] "Script - Bitcoin Wiki," https://en.bitcoin.it/wiki/Script.

[177] "Talk crypto blog OP_RETURN 40 to 80 bytes," http://www.talkcrypto.org/blog/2016/12/30/op_return-40-to-80-bytes/.

[178] A. Sward, I. Vecna, and F. Stonedahl, "Data insertion in Bitcoin's Blockchain," *Ledger*, vol. 3, 2018.

[179] "Bitcoin transaction fee estimator," https://bitcoinfees.earn.com/.

[180] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification," 2018.

[181] "Zcash Usage," https://explorer.zcha.in/statistics/usage.

[182] N. van Saberhagen, S. null, J. Meier, and R. Lem, "CryptoNote One-Time Keys," https://cryptonote.org/cns/cns006.txt, 2012.

[183] N. van Saberhagen, J. Meier, and A. M. Juarez, "CryptoNote Signatures," https://cryptonote.org/cns/cns001.txt, 2011.

[184] "Etherdelta source code," https://etherscan.io/address/0x8d12a197cb00d4747a1fe03395095ce2a5cc6819#code.

[185] "Algorithm to generate all possible permutations of a list?" https://stackoverflow.com/questions/2710713/algorithm-to-generate-all-possible-permutations-of-a-list.

[186] "Cryptocurrency market capitalizations," https://coinmarketcap.com/.

[187] "MoneyMorph Python Prototype," https://github.com/moneymorph.

[188] "ECDSA Python Package," https://github.com/warner/python-ecdsa.

[189] "Cryptography Python Package," https://pypi.org/project/cryptography/.

[190] "Amazon workspaces pricing," https://aws.amazon.com/workspaces/pricing/.

[191] "Digital Ocean Pricing," https://www.digitalocean.com/pricing.

[192] "bitcoin-utils Python Package," https://pypi.org/project/bitcoin-utils/.

[193] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *USENIX Security Symposium*, 2015, pp. 129–144.

[194] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin P2P network," in *ACM Conference on Computer and Communications Security*, 2014, pp. 15–29.

[195] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The Parrot Is Dead: Observing Unobservable Network Communications," in *S&P*, 2013, pp. 65–79.

[196] D. V. Le, L. T. Hurtado, A. Ahmad, M. Minaei, B. Lee, and A. Kate, "A tale of two trees: One writes, and other reads. optimized oblivious accesses to bitcoin and other utxo-based blockchains," *Proceedings on Privacy Enhancing Technologies*, pp. 519–536, 2020.

[197] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "BITE: bitcoin lightweight client privacy using trusted execution," *IACR Cryptology ePrint Archive*, vol. 2018, p. 803, 2018. [Online]. Available: https://eprint.iacr.org/2018/803

[198] K. Wust, S. Matetic, M. Schneider, I. Miers, K. Kostiainen, and S. Capkun, "Zlite: Lightweight clients for shielded zcash transactions using trusted execution," Cryptology ePrint Archive, Report 2018/1024, 2018.

[199] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[200] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *(SSS)*, 2015, pp. 3–18.

[201] R. Geambasu, A. A. Levy, T. Kohno, A. Krishnamurthy, and H. M. Levy, "Comet: An active distributed key-value store," in *OSDI'10*.

[202] M. Minaei, M. Mondal, and A. Kate, "'"my friend wanted to talk about it and i didn't": Understanding perceptions of deletion privacy in social platforms," 2020.

[203] T. Gong, M. Minaei, W. Sun, and A. Kate, "Undercutting bitcoin is not profitable," *arXiv preprint arXiv:2007.11480*, 2020.

[204] M. Minaei, P. Moreno-Sanchez, and A. Kate, "R3c3: Cryptographically secure censorship resistant rendezvous using cryptocurrencies." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 454, 2018.

[205] D. V. Le, L. T. Hurtado, A. Ahmad, M. Minaei, B. Lee, and A. Kate, "A tale of two trees: One writes, and other reads. optimized oblivious accesses to large-scale blockchains," *arXiv preprint arXiv:1909.01531*, 2019.

[206] M. Minaei, M. Mondal, P. Loiseau, K. Gummadi, and A. Kate, "Forgetting the forgotten with letheia, concealing content deletion from persistent observers," *arXiv preprint arXiv:1710.11271*, 2017.

[207] ——, "Forgetting the forgotten with lethe: Conceal content deletion from persistent observers," in *PETS 2019-19th Privacy Enhancing Technologies Symposium*, 2019, pp. 1–21.

[208] J. N. Nanduri, S.-J. Wang, and M. M. M. Bidgoli, "Accounting for uncertainty when calculating profit efficiency," Jan. 24 2019, uS Patent App. 15/655,452.

VITA

VITA

Mohsen Minaei received his Ph.D. in the department of computer science at Purdue University in September 2020. He worked as a research assistant under the supervision of Professor Aniket Kate. His doctoral research focused on the study and design of privacy-preserving systems using crowd blending techniques. Mohsen also received his Master's degree in computer science from Purdue University. Prior to joining Purdue, he received his B.S. degree in computer engineering from Sharif University of Technology. 2013 respectively. He was an intern at Microsoft Corporation for three consecutive summers of 2016-2018. Each year he took a different role including software engineer, program manager, and data scientist in Cloud+AI group working with fraud detection and Xbox knowledge platform teams. In the summer of 2019, he joined Visa Research as a research scientist intern for the blockchain team under the supervision of Mihai Christodorescu. At Purdue, on the way to the Ph.D., he got the CERIAS/Intel Research Scholarship from January to May 2017.

PUBLICATIONS

PUBLICATIONS

Mohsen Minaei, Mainack Mondal and Aniket Kate, “"My Friend Wanted to Talk About It and I Didn’t": Understanding Perceptions of Deletion Privacy in Social Platforms”, *ArXiv Preprint, 2020.*

Tiantian Gong, **Mohsen Minaei**, Wenhai Sun and Aniket Kate, “Undercutting Bitcoin is Not Profitable”, *ArXiv Preprint, 2020.*

Mohsen Minaei, S Chandra Mouli, Mainack Mondal, Bruno Ribeiro and Aniket Kate, “Deceptive Deletions for Protecting Withdrawn Posts on Social Platforms”, *Major Revision Submitted to Network and Distributed System Security Symposium (NDSS), 2021.*

Mohsen Minaei, Pedro Moreno-Sanchez, and Aniket Kate, “MoneyMorph: Censorship Resistant Rendezvous using Permissionless Cryptocurrencies” *Proceedings on Privacy Enhancing Technologies, 2020.*

Duc V Le, Lizzy Tengana Hurtado, Adil Ahmad, **Mohsen Minaei**, Byoungyoung Lee, Aniket Kate, “A Tale of Two Trees: One Writes, and Other Reads: Optimized Oblivious Accesses to Bitcoin and other UTXO-based Blockchains” *Proceedings on Privacy Enhancing Technologies, 2020.*

Mohsen Minaei, Mainack Mondal, Patrick Loiseau, Krishna Gummadi and Aniket Kate, “Lethe: Conceal Content Deletion from Persistent Observers”, *Proceedings on Privacy Enhancing Technologies, 2019.*

Jayaram NM Nanduri, Shoou-Jiun Wang and **Mohsen Minaei**, “Accounting For Uncertainty When Calculating Profit Efficiency”, *U.S. Patent #US20190026742, 2019.*