PRSONA: Private Reputation Supporting Ongoing Network Avatars

by

Stan Gurtler

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

Waterloo, Ontario, Canada, 2021

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

I was the sole author for Chapters 4, 5, and 6, which were written under the supervision of Dr. Ian Goldberg and were not written for publication. This thesis consists in part of one manuscript written for publication. Exceptions to sole authorship of material are as follows:

Research presented in Chapters 1, 2, and 3:

This research was conducted at the University of Waterloo by me (Stan Gurtler) under the supervision of Dr. Ian Goldberg. I designed the methodology and carried out the analysis with consultations from Dr. Ian Goldberg. I drafted the manuscript and Dr. Ian Goldberg provided intellectual input on manuscript drafts.

Stan Gurtler and Ian Goldberg. SoK: Privacy-preserving reputation systems. *Proceedings on Privacy Enhancing Technologies*, 2021(1):107–127, 2021.

As lead author of these three chapters, I was responsible for designing the methodology for review, carrying out paper collection and analysis, and drafting and submitting manuscripts. My coauthor provided guidance during each step of the research and provided feedback on draft manuscripts.

Abstract

Trust and user-generated feedback have become increasingly vital to the normal functioning of the modern internet. However, deployed systems that currently incorporate such feedback do not guarantee users much in the way of privacy, despite a wide swath of research on how to do so spanning over 15 years. Meanwhile, research on systems that maintain user privacy while helping them to track and update each others' reputations has failed to standardize terminology, or converge on what privacy guarantees should be important. Too often, this leads to misunderstandings of the tradeoffs underpinning design decisions. Further, key insights made in some approaches to designing such systems have not circulated to other approaches, leaving open significant opportunity for new research directions.

Acknowledging this situation, online communities in particular face a difficult dilemma. Communities generally want to provide opportunities for their members to interact and communicate with one another in ways that advance their mutual interests. At times, communities may identify opportunities where providing their members specific privacy guarantees would particularly aid those opportunities, giving members assurances that their participation would not have negative consequences for themselves. However, communities also face the threat of bad actors, who may wish to disrupt their activities or bring harm to members for their status as members of such groups. The privacy that the community wishes to extend to members must be carefully approached so that bad actors can still be held accountable.

This thesis proceeds in two parts. First, this thesis investigates 47 systems describing privacy-preserving reputation systems from 2003–2021 in order to organize previous work and suggest directions for future work. The three key contributions in this portion of the thesis are the systematization of this body of research, the detailing of the tradeoffs implied by overarching design choices, and the identification of underresearched areas that provide promising opportunities for future work.

Second, this thesis explores one particular opportunity for new research identified in the first section of the thesis. Whereas previous work has overlooked the needs of certain kinds of small, tight-knit communities, this work features a novel design for a privacy-preserving reputation system that is targeted to fill that gap. The nature of its design is discussed particularly in contrast to the identified patterns of design present in previous works. Further, this thesis implements and benchmarks said system to determine its viability in real-world deployment. This novel construction addresses shortcomings with previous approaches and provides new opportunities for its intended audiences.

Acknowledgements

I would like to thank my family and my partner, whose unfailing support allowed me to continue and complete this work even when I was not confident in myself. I would like to thank W. and W., whose passions and fruitful discussions helped to inspire this work. I would like to thank the members of the CrySP lab, who have been both personally and academically some of the most inspiring individuals I could hope to meet. I would like to thank the members of my committee, Urs Hengartner and Florian Kerschbaum, and my supervisor Ian Goldberg, whose guidance and support proved invaluable over the course of this work to help keep me on the rails. And of course, there are many more people without whose support I would not be where I am today. I have a truly marvelous benediction for each and every one of them which this margin is too narrow to contain. From the bottom of my heart, thank you all.

This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo.

Dedication

For those who have felt strange feelings they are certain no one else ever has, and the fear and the loneliness which accompanies them. May you know the euphoria of hearing another person casually describe that which you hold most private.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Significant attention has been given to the internet and its ability to connect people with unprecedented amounts of information, as well as to large-scale changes to society that followed this connection. *Disruption* is the term of art, with companies like Amazon, AirBnB, and Uber all causing dramatic changes to the industries they inhabit [Alt16]. Social media platforms like YouTube and Facebook have been identified as disrupting a variety of areas, such as "online payments" [Cho19b], "credit cards" [Lee19], and "the future of media and entertainment" [LBW16]. Rugnetta [Rug13] terms this pattern of disruption (and the disintermediation underlying it) "the deconstructive internet", saying that "talk surrounding disintermediation also has a certain tenor to it. It suggests that there was infrastructure which was built and existed and now the internet is tearing it down, unbuilding it, or at the very least supporting its dereliction and eventual implosion. [...] It's not bad; for the most part it's not destructive, it is deconstructive" [Rug13, 5:00]. In many cases, this disruption is founded upon the ability to connect people with each other, rather than with information. Without the ability to refer to feedback shared by others on their experiences purchasing goods, staying in short-term rentals, or ridesharing, users would have a great deal of difficulty placing their trust into online platforms like those above.

While significant attention is placed on this aspect of the internet, there is more to the internet than its role as society's wrecking ball. It is easy to understand why so much attention is placed on this function of the internet; after all, those who might have the soapbox from which to note and decry this functionality are often those with entrenched interests in the various structures being disrupted. Traditionally, the internet is espoused as a means to access information like never before. UNESCO [UNE19] identifies four principles it argues should "[underpin] the growth and evolution of the Internet": Rights, Openness, Accessibility to all, and Multistakeholder participation. In several ways, the deconstructive internet does provide

this. In addition to allowing individuals access to information or media through services like YouTube, Netflix, and similar, the deconstructive internet allows them access to the people who create such material, such as through social media like Twitter. In some extreme cases, such as "Jeremy Renner Official", a shortlived smartphone app designed for fans of actor Jeremy Renner [Cho19a], entire pieces of the deconstructive internet can be designed for such access to minute pieces of the real world.

The internet, however, is not merely restricted to access to rote information. It has also served as a means by which individuals may access *each other*, creating communities. Rugnetta describes this role of the internet as "the constructive internet", which "connects locales not unreachable because of tolls or roadblocks but because there were no roads" [Rug13, 5:55]. The constructive internet is the realization of "connections created between people who, before now, would never or only with significant effort have crossed paths" [Rug13, 7:45]. The internet has been identified for its power for connecting various groups, such as LGBTQ individuals, even from geopolitical regions that may be hostile towards such individuals [Wel15]. Creating communities through the internet has made significant positive impact on the lives of countless individuals.

Naturally, though, such communities frequently encounter bad actors. Through sites like You Got Posted [Ste15], 8chan [All15], and Kiwi Farms [Ple16], malicious individuals gather to compare notes, stalk, and harass marginalized communities and their members, frequently infiltrating such groups to gather information on members. Other such infiltrations have been reported to be a part of the radicalization and recruitment of individuals to the alt-right [SC17]. Communities have responded to these infiltrations with varying success, up to and including warning local police of potential SWAT attempts or withdrawing entirely from the internet [All15]. Infiltrations have had effects on real-world events as well, causing the cancellations of gatherings in the physical world [Kil17] or the pre-emptive expulsion of individuals believed to be harmful to the group from such gatherings [Dic19]. Online communities have found it difficult to protect themselves from such infiltrations.

At large, the quick identification of fraudulent or malevolent actors is of interest beyond the needs of internet communities. In recent years, significant attention has been placed on social media and its role in propaganda, such as reports that Chinese disinformation campaigns targeted Twitter and Facebook to discredit pro-democracy protests in Hong Kong [WMF19] and the allegations of the US government that Russian troll farms were used in an attempt to manipulate American voters in 2016 [Lee18]. Social media sites have taken a variety of measures, not all public, to try to curb the usage of their platforms by malevolent actors.

Not all of these measures mesh well with the communities connected specifically by the constructive internet, however. Frequently such communities have very real concerns

about the impact of disclosure of their private lives on the public scene. Protections for LGBTQ individuals can be lacking, for example, and individuals may want to keep identifying information apart from their participation in such communities. Policies like Facebook's "real name policy" can lead to significant problems for such members [CBC18], who may be forced to choose between divulging sensitive information and being unable to participate in these communities.

Further, tying identity and a sense of community to attributes and restrictions of the physical world is a limitation that is not always the most productive for individuals in these communities. Individuals at times use participation in communities to experiment with identity, such as by creating pseudonyms that specifically use gender identities that do not reflect what they were assigned at birth. The ability to change attributes of such pseudonyms, or create new identities to interact with if old ones carry emotional baggage, are valuable properties to preserve in the creation and safeguarding of such communities. The ability to participate as multiple identities simultaneously, without those being linkable directly, can allow people to experiment in such ways in a "soft" manner, keeping a long-term identity that people already know and interact with while gaining the benefits of new experimentation in a safe manner. And in cases where interactions between members break down, allowing individuals to express their dissatisfaction with other members to the larger community without inviting retaliation from said members is an important property. These concerns identify issues of *privacy* that should be protected in any system supporting communities.

In order to incentivize good behaviour, and to identify and deal with bad behaviour, communities at times employ *reputation*. Platforms like Reddit provide a concrete example of this, and in these contexts, reputation is intended to serve as a measure of good-faith participation within the community. Reputation has already been used for a large portion of the internet's lifetime on sites like eBay and Amazon. In those contexts, it serves as an abstract measure of trustworthiness and/or quality, specifically of vendors that users may interact with and products that they may wish to purchase. Individuals may participate in transactions they wish to keep private, and there is a wealth of study on designing and implementing systems with privacy-preserving properties to implement reputation in these transactional situations. However, the privacy concerns around transactions are not the same as those in communities, where users are directly evaluating one another on an ongoing basis. In community settings, even seemingly innocuous data, such as ratings assigned to media, has been shown in previous work to reveal sensitive attributes about users [JWZG17].

Further, these extant reputation systems are typically deployed by individual companies, and these communities are frequently wary of such companies. YouTube [Ell19], Tumblr [Cut19], and TikTok [Gua19] all have encountered controversy with the LGBTQ community at large over concerns of moderation, and Reddit too is no stranger to controversy [Sta18]

over its policies on moderation. Further, even applications focused on the LGBTQ community, such as Grindr, have faced catastrophic privacy failures and revealed sensitive data about their users [DC21]. Consolidating the power to determine whether individuals in a community are bad actors into one centralized institution that may fail the community in a variety of ways is not a long-term strategy that will sufficiently safeguard the interests of such communities. As such, decentralized solutions may be better suited to protect these communities. Communities have already begun to form around technologies like Mastodon, which allows for a federated social network that communities are better empowered to moderate internally as necessary.

Technologies like Mastodon do not currently offer opportunities more fine-grained than federation and blocking to help protect communities, however. Further, though reputation may help in this regard, previous implementations frequently have problems handling the actual problem at hand: identifying misbehaviour and incentivizing members to correct it or face punishment. For example, in Reddit-style implementations of reputation, a user may amass a great amount of reputation and "coast" on the good graces of past actions, even as their reputation takes a hit from misbehaviour. Further, these systems do not always protect the privacy of their users as thoroughly as they might need. Metadata associated with votes cast on reputation may be able to de-anonymize users via long-term linkage of their activities, or linkage between different identities they wish to keep separate, or endanger them to abusive individuals by revealing that they have acted in some manner against that individual. Even where previous work has attempted to address these privacy concerns, they have not always done so in a scalable manner, allowing for the usage of such systems by large numbers of users, which can be a desirable property for the advantages provided by larger anonymity sets within a group.

Further, identifying and implementing alternative, frequently more complex conceptions of "reputation" (modeled as reputation functions) can help avoid the problems of Reddit-style reputation. Not giving malicious individuals the ability to coast on past activity (by limiting the way reputation can grow) is a valuable property to introduce into reputation systems. Some reputation functions that may accomplish this also have interactions with and mitigations for Sybil attacks, a common problem for any socially grouped system — by allowing people multiple nyms, but only a single vote per person. Such complex reputation functions are not currently possible in existing schemes.

Our goal in this work is to address these concerns and opportunities for future research in order to enable better protection and maintenance of communities in ways that continue to safeguard their privacy. In particular, we aim to prove the following thesis statement:

It is possible to build a secure reputation system that preserves the privacy of its participants and enables the implementation of multiple complex reputation functions.

To this end, we design our own reputation system, PRSONA, or Private Reputation Supporting Ongoing Network Avatars, for the purpose of enabling said community protection and maintenance by using complex reputation functions. Further, we implement and benchmark this system, and this implementation is available online at https://git-crysp.uwaterloo.ca/tmgurtler/PRSONA.

Chapter 2 discusses reputation as it has been used and theorized in previous works, and its implementations into reputation systems. Chapter 3 more formally discusses previous work including previous study on reputation systems. Chapter 4 discusses the design of PRSONA, including its threat model. Chapter 5 outlines the implementation of PRSONA and an analysis of its practicality for real-world deployment. Chapter 6 concludes.

Chapter 2

Reputation Systems

In its most basic conception, we understand *reputation* to be what people believe about an individual. However, reputation is frequently less concerned with what Alice may think of Bob than with what Alice has heard from others *about* Bob, and thus we may take a more fine-grained approach. We could instead define reputation as the belief a person A may have about what some group of people C believe about an individual B. The distinction this raises is precisely that between an individual's opinions and a group's consensus. While Alice's opinions about Bob are tied directly to her, consensus can be more diffuse, the collective understanding of what a community thinks is believed about an individual. Reputation can be punctuated with specific anecdotes or the opinions of individuals, but it is more commonly used as the wider belief of some larger group of people.

Reputation has long been used by human society to organize how people interact with each other. As we learn more about established patterns of abuse in various domains through movements like "#MeToo", we observe countless cases where whisper networks had formed to try to, for example, help women communicate with one another about the reputations of various people they worked with. As a part of this movement, the women involved were at times empowered to share their previously private allegations to make this private reputation more public. Though the movement is large and not in any meaningful way centrally organized, one might surmise a hopeful set of goals, which is intended to be not exhaustive but perhaps illustrative: 1) to more effectively warn individuals not privy to the whisper networks of the dangers specific abusers may pose; 2) to incentivize abusers to mend their ways in order to repair their public image; and 3) to disincentivize others from following in abusers' footsteps. Indeed, these are all traditional roles that reputation has served for human interaction.

Reputation can serve as a tool in many facets of human society, and it is no surprise that it

has also long held an important role in the conducting of business and transactions. Credit scores are a simple form of reputation used by banks and other lenders to determine the reputation of individuals asking for loans. Knowledge about what other parties may not honour their transactions perfectly has played an important role in human economy for time immemorial, and it has naturally found its way to commerce on the internet as well. One of eBay's enduring contributions was the institution of its ratings systems, where sellers and buyers could rate each other in their interactions. This was one of the first *reputation systems*, where participants organized their beliefs about other individuals in a systematic manner to collect and distribute opinions about the trustworthiness of one another. In eBay, the intended role of this reputation system was to incentivize sellers to behave honestly and to send their wares to buyers in a timely manner, as well as to incentivize buyers to appropriately and honestly make payment to sellers for said wares. Ratings on eBay are organized by a "star" system, where users have star ratings displayed with their other information that are the average of all the ratings they have previously received. A dishonest transaction would afford a user a low rating, and thus negatively impact their displayed star rating.

This reputation system proved very popular on eBay, and over time similar systems found their way into other services. Companies like Lyft and Uber use reputation to incentivize their driver employees to provide positive ride experiences to customers, while also using it to incentivize customers against damaging the goods of or endangering drivers. It is used similarly in AirBnB for providers and customers of accomodations. Amazon uses ratings as a measure of the quality of specific goods sold on its site, as well as of the quality of interactions with the individual businesses that sell goods on Amazon. Yelp uses ratings to help users determine the quality of restaurants and other businesses in locales they find themselves in. Stack Overflow uses ratings and reputation to help determine what answers might best help solve questions users pose, as did the now-defunct Yahoo! Answers. And social networks like Reddit and Digg revolve around reputation, using ratings to promote popular material for audiences within subforums and in the discussions of specific submissions. These ratings themselves accrue to users. On sites like Reddit, reputation has become a tool with the intention to help users determine what users are trustworthy (in much the same way that forum posts made by members with very low post counts might be less trustworthy than those made by members who had participated frequently). Reputation even possibly has some level of monetary value, with reports claiming to have sold reddit accounts with high reputation scores to advertisers intent on astroturfing support for products they were selling [Rob16].

As the internet becomes ever more dominant in the ways we carry our lives and interact with one another, reputation systems seem to become ever more present in our day-to-day interactions. Infamously, China has instituted a Social Credit Score to encourage positive social behaviours [Elg19], but reputation has also found its way into more and more avenues

in North American life as well. Understanding reputation systems and how to design them in ways to help preserve the privacy and dignity of people subject to them will only become increasingly important as time goes on.

Previous work has not converged on a standard set of terms in order to describe their systems. In cases where the terms themselves are consistent, the definitions used with these terms have at times obscured important distinctions in underlying design. In Section 2.5, we examine the many different terms that have been used in previous work across all the categories we describe in this chapter. Throughout this thesis, we will use a few specific terms to refer to participants in reputation systems. We refer to a user who contributes feedback for another party as a "voter". We refer to a user for whom feedback is contributed as a "votee". In both cases, these users may refer to an individual or an organization as necessary. Where relevant, a user who requests a reputation of a votee is a "requester". Throughout this chapter, we elaborate on three key areas that have been addressed inconsistently in the past: architecture, reputation directionality, and privacy properties.

2.1 Architecture

In reputation systems, the integrity of reputations must be preserved. If votees were allowed to interfere with voters who would rate them negatively and prevent those ratings, they could artificially raise their own reputation scores. On the other hand, if malevolent actors were allowed to post ratings indiscriminately, they could artificially lower the reputation scores of votees. In Sybil attacks specifically, users may be able to perform this ballot-stuffing and badmouthing by creating arbitrary numbers of identities with which to participate in the system. This style of attack has been well described in previous work [LR04, YKGF06, TMLS09, GFM14, FYGL18, FGL20]. The designers of reputation systems may turn to a variety of strategies in order to protect against such malicious actions; largely, they rely on one of the following three:

Third-Party Mediation: A reputation system may designate one or more trusted third parties (TTPs) to be responsible for the integrity of the reputation scores. Reputation systems may also designate one or more TTPs to be responsible for the privacy of the users in the system. The TTPs' involvement can take several forms, and differing amounts of trust may be placed in them. In some systems, the TTPs bootstrap the system but may not be required for its ongoing operation, such as when group signature schemes are used. In others, the TTPs only serve to audit interactions. In still others, the TTPs intermediate all interactions. Some systems use only one TTP, others may use multiple, and still others require multiple, often to try to break apart centralized roots of trust. These systems are frequently called "centralized".

Ephemeral Mesh Topology: In some systems, reputation is not a global, persistent value, but is instead calculated when it is requested. Requesters are responsible for interacting directly with voters to solicit their individual evaluations of a votee. So long as requesters can confirm they are interacting with the voters they intend to, the procedure used to combine reputation scores in such systems guarantees that each participant may only contribute one evaluation. Some systems additionally allow requesters to weight the importance of voters' contributions by how much they themselves trust the voters. Requesters are typically free to choose which voters they intend to query and are not required to always choose the same voters for each request. We term these systems "user-defined decentralized".

Proofs of Validity: In some systems, voters contribute their feedback for votees directly to all other users, such as via an append-only public bulletin board. Proofs of the integrity and validity of votes, then, must be derived using additional information. This often takes the form of proofs of knowledge of specific secret values that indicate a voter has undergone a transaction with the votee, without specifying which transaction. In such a system, careful attention must be placed on how the bulletin board is maintained. The system would not be useful if users could not agree on which feedback is valid, and so the system must remove the potential for abuse in reaching this agreement. While this approach is certainly decentralized, it has clear differences in its manner of decentralization than systems that use an Ephemeral Mesh Topology. As such, we term these systems "system-defined decentralized".

While a majority of previous works in the literature use Third-Party Mediation, Proofs of Validity have been an increasingly attractive approach to designers of reputation systems. Methods of incorporating such proofs even where TTPs are still being used may help distribute trust in the system away from centralized nodes.

2.2 Reputation Directionality

eBay was one of the earliest-used reputation systems. In eBay's reputation system, buyers and sellers both participate in rating one another, and reputation has different roles in determining how to interact with buyers and sellers. In other systems, such as Amazon's, buyers rate sellers, but there is no clear mechanism for buyers themselves to be rated. In still other systems, such as Reddit's, all participants rate each other, with no distinctions being made between "types" of user. We suggest a classification of reputation systems into three kinds according to how their ratings are organized:

- *Simplex Reputation Systems* ($C \rightarrow S$): In a simplex reputation system, there are two sets of participants. One set, C, represents the clients or consumers in the system. The other

set, S, represents the servers or sellers in the system. Clients may assign ratings, but have no ratings associated with themselves; even when there is an overlap between C and S, a participant acting as a client does not have their server rating associated with their client activity. On the other hand, servers receive reputations, and have these reputations displayed in a manner that clients can observe and use to inform their decisions about future interactions. Amazon is an example of a simplex reputation system.

- *Half-Duplex Reputation Systems* ($C \leftrightharpoons S$): In a half-duplex reputation system, there are two sets of participants. One set, C, represents the clients or consumers in the system. The other set, S, represents the servers or sellers in the system. Clients may assign ratings to servers, and servers may assign ratings to clients. When there is an overlap between C and S, a participant has two different ratings that do not impact one another, and are only used in the appropriate settings where they behave as a client or as a server. As both clients and servers receive reputations, both clients and servers can observe the role-specific reputations of one another and base decisions about future interactions upon them. eBay is an example of a half-duplex reputation system.

- *Full-Duplex Reputation Systems* ($P \leftrightarrow P$): In a full-duplex reputation system, there is only one set of participants, P, representing the peers or participants in the system. Peers assign ratings to one another, and there are no structural distinctions between peers who give ratings and peers who receive ratings. Peers can observe the reputations of one another and base decisions about future interactions upon them. Reddit is an example of a full-duplex reputation system.

2.3 Privacy Properties

While privacy-preserving reputation systems must do something to protect user privacy, the exact nature of these privacy protections varies between systems. We highlight four privacy properties — two for voters and two for votees — that a privacy-preserving reputation system may provide. In all cases, it may be possible to provide said property with respect to one of the following sets: all parties not involved in a transaction, all parties except TTPs, or all parties without restriction. Following the example set by Kuhn *et al.* [KBS+19], we avoid the word "anonymity" in the names of these properties, as we feel that term may be unclear and overloaded.

2.3.1 Voter Privacy Properties

- *Voter-Vote Unlinkability*: In order to avoid concerns that a voter may face coercion or backlash for their vote, it may be desirable for a voter to cast a vote secretly. That is, Voter-Vote Unlinkability is provided when a voter cannot be associated with a vote they cast, or with the fact that they voted for a particular votee.
- *Two-Vote Unlinkability*: While voters may be unlinkable to their votes, this does not preclude the possibility that users may be able to identify that two votes came from the same voter. This may be undesirable, as more votes cast reduces a voter's anonymity set and allows behavioural tracking. Thus, Two-Vote Unlinkability is provided when it is not possible to distinguish whether two votes were cast by the same voter or not.

2.3.2 Votee Privacy Properties

- *Reputation-Usage Unlinkability*: It may be desirable for votees to be provided privacy as well. In these cases, reputation still must have some meaning, and must still be able to be accumulated, but it may be desirable for votees to produce a proof of their reputation without linking themselves to a long-term pseudonym associated with that reputation. Thus, Reputation-Usage Unlinkability is provided when a votee can display or use their reputation and accumulate new votes without enabling others to identify that another specific reputation use was also performed by the same votee.
- *Exact Reputation Blinding*: For the purposes that reputation serves, it can be sufficient to know that a votee's reputation is above some threshold. Displaying a votee's precise reputation score may in fact be undesirable, as it can be observed to track the votee across usages or to infer a voter's vote for a votee. Thus, Exact Reputation Blinding is provided when a system provides a mechanism for votees to display or use their reputation without giving an exact score.

2.4 Reputation Functions

In order to provide simple and interpretable reputation scores for users to observe, reputation systems typically feature a method of reducing the frequently large set of ratings received about participants into a single value. In the real world, reputation systems tend to feature one of two different such ways of reducing reputation ratings. First, reputation may be

represented as an average of ratings (occasionally weighted in various ways, giving priority to more reputable participants or to more recent ratings). Systems with stars typically do this, such as Yelp, Uber, Lyft, or AirBnB, where the actual scores are merely the (possibly weighted) mean of all ratings received about a participant. Second, reputation may be represented as a sum of ratings. Reddit and Stack Overflow are two examples of such systems, where reputations are measured in terms of raw difference between positive ratings and negative ratings. We term such reductions *reputation functions*.

Though reputation systems in practice largely fall into one of these two categories, previous work [SPT11, CSK13] has investigated a wider range of potential reputation functions, and we further suggest our own additions that we feel address major inadequacies in the fairness and intuitiveness of previous reputation functions. We present here a non-exhaustive list of reputation functions that have been used and suggested in previous work or in real-world deployment, along with our additions.

Note: for all functions, V represents the options of what a user may rate another user, S represents the displayed reputation rating of the function, U represents the set of users involved in the system (for half-duplex or simplex reputation systems, U is typically the set of users being rated, but the reputation functions themselves are agnostic to the directionality of the reputation system using them), $\overrightarrow{x_u} \in V^*$ represents an arbitrary length set of votes assigned to a user $u \in U$, and $x_{u,i} \in \overrightarrow{x_u}$ represents the i-th of these votes.

2.4.1 Voter-agnostic Reputation Functions

The most common reputation functions in use do not require information about the voter who assigned a rating in order to output a reputation score. While the reputation systems that employ them typically use some mechanism to ensure that users cannot merely spam ratings, the functions themselves would be able to accept arbitrary ratings from voters and use all of them to create a rating score. This is evident in systems like Reddit's, where users may vote positively or negatively on posts made by a specific other user, and all of those votes have impacts on that user's reputation score.

- *Accrue Stars*: Accrue Stars takes three parameters, $m \in \mathbb{R}^+, c \in \mathbb{R}^+, n \in \mathbb{N}$. In this reputation function, $V = \{m\}$ is a singleton set; users merely vote to indicate approval (or disapproval, as the case may be). $S = \{0, 1, ..., n\}$ represents n different ratings (we might visualize them as a number of stars) that users can obtain. The function $\Phi : V^* \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_u}) = \begin{cases} 0 & \text{if } 0 \leq s < c \\ 1 & \text{if } c \leq s < 2c \\ \dots & \\ n & \text{if } nc \leq s \end{cases} \text{ where } s = \sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i}$$

As such, n represents a maximal number of stars, and once a user reaches the value needed for it, they eternally have n stars. Accrue Stars has been discussed (in this specific form) in previous work [SPT11]. We also outline some tweaks to this function below, which may be arbitrarily mixed and matched.

- *Accrue Stars — Exponential*: Accrue Stars — Exponential again takes three parameters, $m \in \mathbb{R}^+, c \in \mathbb{R}^+, n \in \mathbb{N}$. V and S are unchanged from Accrue Stars, but $\Phi : V^* \to S$ is now defined as follows:

$$\Phi(\overrightarrow{x_u}) = \begin{cases} 0 & \text{if } 0 \leq s < c \\ 1 & \text{if } c \leq s < c^2 \\ \dots & \\ n & \text{if } c^n \leq s \end{cases} \text{ where } s = \sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i}$$

  Though this is a relatively simple tweak, we belabour the point here to show that there are a large number of modifications one could make to the spacings of the thresholds needed for each reputation level. Accrue Stars — Exponential is our own contribution.

- *Accrue Stars — Unbounded*: Accrue Stars — Unbounded takes only two parameters, $m \in \mathbb{R}^+, c \in \mathbb{R}^+$. V is unchanged from Accrue Stars, but now $S = \mathbb{N}$, and $\Phi : V^* \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_u}) = \begin{cases} 0 & \text{if } 0 \leq s < c \\ 1 & \text{if } c \leq s < 2c \\ \dots & \end{cases} \text{ where } s = \sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i}$$

  The main change here from Accrue Stars is that reputation can eternally increase. In this specific scheme, the reputation rating corresponds to some linear function of votes (although the Exponential tweak or some other similar tweak could also be applied here). Accrue Stars — Unbounded has been extensively used in previous work [IBJR03, Vos04,

ACBM08, BSS10, SL03, BIJR04, MPRS08, Ker09, HS11, WCMA13, PLS14, ZWC+16, BSHB16, BPS+17, GMN17, BEJ18, BBB+18, SKCD16, SBHB16, ABH18, LAN+19].

- *Accrue Stars — Negative*: Accrue Stars — Negative takes four parameters, $m \in \mathbb{R}^+, c \in \mathbb{R}^+, h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0,1\}$. In this reputation function, V is now $\{m, -m\}$; that is, users can now express both positive and negative ratings. $S = \{\ell, \ell+1, ..., h-1, h\}$, and $\Phi : V^* \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_u}) = \begin{cases} \ell & \text{if } s < (\ell+1)c \\ \dots & \\ 0 & \text{if } 0 \le s < c \\ 1 & \text{if } c \le s < 2c \\ \dots & \\ h & \text{if } hc \le s \end{cases} \quad \text{where } s = \sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i}$$

  Now, even though the number of stars is still bounded, reputation ratings may still always have an impact, as users can use negative votes to bring reputations back down. Note that, by combining this with the Unbounded tweak (and correct choice of parameters, namely $m = 1, c = 1$), this is Reddit-style karma. Accrue Stars — Negative has been widely used in previous work [Vos04, HYLC07, SL03, BIJR04, MPRS08, Ker09, HS11, WCMA13, PLS14, ZWC+16, BSHB16, BPS+17, GMN17, BEJ18, BBB+18, SKCD16, SBHB16, ABH18, LAN+19].

- *Accrue Stars — Relative*: Accrue Stars — Relative takes three parameters, $m \in \mathbb{R}^+, c \in \mathbb{R}^+, h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0,1\}$. In this reputation function, V is unchanged from Accrue Stars, and S is the same as in Accrue Stars — Negative. $\Phi : V^* \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_u}) = \begin{cases} \ell & \text{if } s < (\ell+1)c \\ \dots & \\ 0 & \text{if } 0 \le s < c \\ 1 & \text{if } c \le s < 2c \\ \dots & \\ h & \text{if } hc \le s \end{cases} \quad \text{where } s = \sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i} - \frac{\sum_{v \in U} \sum_{x_{v,i} \in \overrightarrow{x}_v} x_{v,i}}{|U|}$$

  Through this, reputation scores are made to be relative to the average user's current score, such that even a user who accrues a large reputation may not appear so impressive

if every other user has also accrued a large reputation. Accrue Stars — Relative has only been discussed previously once, by Schiffner et al. [SPT11].

- *Average Stars*: Average Stars takes two parameters, $h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0,1\}$. In this reputation function, V is now $\{\ell, \ell+1, ..., h-1, h\}$ (sometimes excepting $\{0\}$), and $S = \{c \mid c \in \mathbb{Q} \text{ s.t. } \ell \leq c \leq h\}$. $\Phi : V^* \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_u}) = \frac{\sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i}}{|\overrightarrow{x_u}|}$$

  In this, the key difference from the Accrue Stars family is that instead of summing up all ratings, the reputation function now takes their mean and reports that instead. As such, users frequently have a more fine-grained set of options for votes (e.g. from 1 to 5 stars). It is easy to imagine alterations of this function that takes a weighted mean instead (perhaps weighted by recency of votes, or similar). Note that for correct choice of parameters, this is very nearly the style of ratings eBay uses, namely $\ell = 1, h = 5$. (The difference is that eBay rounds displayed scores to the nearest half-integer number of stars.) Average Stars does not have a meaningful relationship with any of the tweaks listed above, except perhaps Relative. Average Stars has been used in previous work [IBJR03, SL03, BIJR04, MPRS08, Ker09, HS11, WCMA13, PLS14, ZWC+16, BSHB16, BPS+17, GMN17, BEJ18, BBB+18, LM19, SKCD16, SBHB16, ABH18, LAN+19]

- *Rank*: Rank takes no parameters. In this reputation function, $V = \{1\}$ and $S = \{1, 2, ..., |U|\}$. Unlike the previous functions, this reputation function outputs scores for all users simultaneously, and must take as input all votes in the system. That is, $\Phi : V^{|U|,*} \to S^{|U|}$ is defined by the following procedure. First, calculate the sum of ratings for each user. Then, order each user by these sums, breaking ties randomly. Then, each user's rating is their index in this ordering. Rank can be combined with the Negative tweak above. Rank has been discussed in previous work [SPT11, CSK13].

- *Gompertz*: Gompertz takes three parameters, $b \in \mathbb{R}^-, c \in \mathbb{R}^-, \lambda \in (0,1]$. In this reputation function, $V = [0,1]$ and $S = [0,1]$. $\Phi : V^k \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_{u,i}}) = e^{be^{c\sum_{i \in \{1,2,...,k\}} \lambda^{t_k - t_i} x_i^{\text{norm}}}}$$

  This reputation function (named for its use of the Gompertz curve) is suggested by Huang et al. [HKH10] as a potential reputation function, particularly for use in participatory

sensing. In that setting, atypically for the systems we focus on in this work, the set of voters is a singleton, a server that evaluates the quality of data submitted by devices, which are the votees. The parameters b and c control the growth rate of the function, and λ is a weighting factor intended to be chosen in a way that forces reputations to slowly accumulate but be quickly destroyed. In this way, Gompertz is supposed to model the trust of humans in social interactions, which takes time to build up, but can be easily destroyed. Votes are real numbers between 0 and 1, inclusive, as are the scores output. The t_i values represent epochs, and are used to weight more recent epochs more highly in the output of the function. Finally, x_i^{norm} is a normalized version of the score given to the user at epoch i. It is normalized against all votes given that epoch across all users. Gompertz has only been discussed previously in one work, by Huang et al. [HKH10].

2.4.2 Voter-conscious Reputation Functions

Less commonly, reputation functions do take the voter who assigned a rating into account when outputting a reputation score. The most common cases of this are situations where voters can only give one rating per user but are allowed to update their ratings.

Note: in these functions, W represents specifically the set of voters.

- *Short-Term Memory Consensus*: Short-Term Memory Consensus takes two parameters, $h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0,1\}$. In this reputation function, $V = \{\ell, \ell+1, ..., h-1, h\}$, and $S = \{\ell|U|, \ell|U|+1, ..., h|U|-1, h|U|\}$. $\Phi : V^{|W|} \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_u}) = \sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i}$$

  As mentioned above, each voter has one mutable vote to assign per each user, and a user's rating is just what the voters currently think of them. Note that, in this definition, a voter must always vote for each user; care must be chosen to decide what an appropriate default vote should be. Φ can easily be changed to a simple mean function instead of a sum, which may be useful as a means to make reputation scores more legible. This reputation function can also accept Rank from before as a tweak, and output a user's relative ranking in reputation instead of their exact score. Short-Term Memory Consensus has been used in previous work [VHM05, KP03, PRT04, YTP07, NR09, AAG09, HBB10a, HBB10b, HBB12, HBBS13, ZXYM16, CSH17, ABHS18, BAH18].

- *Short-Term Memory Consensus — Median*: Short-Term Memory Consensus — Median takes two parameters, $h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0, 1\}$. In this reputation function, V is unchanged from Short-Term Memory Consensus, and $S = \{t \mid t \in \mathbb{Q} \text{ s.t. } \ell \leq t \leq h\}$. $\Phi : V^{|W|} \to S$ is defined by selecting the median rating assigned to a user by the voters. This can easily be extended to alternative percentiles by instead selecting for a different percentile than 50%. Like before, a user's rating is just what the voters currently think of them, but now this is calculated as a median (or other percentile) instead of a sum, which may yield an interesting alternative game theory approach to reputation. Short-Term Memory Consensus — Median is our own contribution.

- *Short-Term Memory Consensus — Weighted*: Short-Term Memory Consensus — Weighted takes two parameters, $h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0, 1\}$. In this reputation function, V is unchanged from Short-Term Memory Consensus, and S is the same as in Short-Term Memory Consensus — Median. Similar to Rank from before, this reputation function outputs scores for all users simultaneously, and must take as input all votes in the system. Unlike previous functions, this function is not agnostic to the directionality of the reputation system using it; it may only be used by full-duplex reputation systems, where $U = W$. Then, $\Phi : V^{|U| \times |W|} \to S^{|U|}$ is defined by organizing the set of all votes into a matrix (where rows indicate who a vote originated from and columns who their vote is for), and performing the PageRank algorithm on it. This will yield rankings weighted by the reputation of whoever gave a specific vote. Short-Term Memory Consensus — Weighted is our own contribution.

- *Short-Term Memory Consensus — Iterated Weighting*: Short-Term Memory Consensus — Iterated Weighting is very similar to Short-Term Memory Consensus — Weighted. Its parameters are the same ($h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0, 1\}$), and V and S are both unchanged from Short-Term Memory Consensus — Weighted. This reputation function also outputs scores for all users simultaneously, and must take as input all votes in the system. In this reputation system, $\Phi : V^{|U| \times |W|} \times S^{|U|} \to S^{|U|}$ is defined by two steps. First, as in Short-Term Memory Consensus — Weighted, the set of all votes is organized into a matrix (where rows indicate who a vote originated from and columns who their vote is for). Additionally, the current scores of all users are organized into a vector, where the individual entries of the vector are ordered in the same way as the order of the matrix (the first entry of this vector corresponds to the score of the same user as the first column of the vote matrix, and so on). With this, a matrix-vector multiplication is performed between the vote matrix and the score vector. Similar to Short-Term Memory Consensus — Weighted, this will yield rankings weighted by the reputation of whoever gave a specific vote, and if votes do not change for a long enough period of

time, Short-Term Memory Consensus — Iterated Weighting will in fact produce the same set of scores as Short-Term Memory Consensus — Weighted. However, the weighting is considerably simpler to calculate each time the output of the reputation function is desired. Short-Term Memory Consensus — Iterated Weighting is our own contribution.

- *Long-Term Memory Consensus*: Long-Term Memory Consensus takes three parameters, $h \in \mathbb{Z}^+, \ell \in \mathbb{Z}^- \cup \{0,1\}, \lambda \in (0,1]$. In this reputation function, V and S are both unchanged from Short-Term Memory Consensus. As with the Gompertz function, this reputation function outputs scores at specific epochs t. With that, $\Phi : V^{|U|} \times S \to S$ is defined as follows:

$$\Phi(\overrightarrow{x_u}, s_{u,t-1}) = \lambda \sum_{x_{u,i} \in \overrightarrow{x_u}} x_{u,i} + (1-\lambda) s_{u,t-1}$$

  Like with Short-Term Memory Consensus, each voter has one mutable vote to assign to each votee. A votee's rating is now a weighted average of what their score would currently be in Short-Term Memory Consensus (that is, what the voters currently think of them), and their score in the previous epoch $s_{u,t-1}$. As with Short-Term Memory Consensus, a voter must always vote for each user, so care must still be chosen to decide what an appropriate default should be. Additionally, the first time users' scores are calculated, an appropriate default must be chosen for their "previous" score (or, instead, Short-Term Memory Consensus is calculated only for that very first time). Φ can again be changed to a simple mean function instead of a sum, which may be useful as a way to make reputation scores more readily interpretable to users. This reputation function can also accept Rank from before as a tweak, and output a user's relative ranking in reputation instead of their exact score. Long-Term Memory Consensus has been used in previous work [HLTZ08, WH09, PRT04, YTP07, NR09, AAG09, HBB10a, HBB10b, HBB12, HBBS13, ZXYM16, CSH17, ABHS18]. (Note, however, that no previous works gave this reputation function a name.)

2.5 Comparison of Terminology from Previous Work

We noted previously that previous work has not converged on a standard set of terms in order to describe their systems. Even when the same terms are used across works, the meanings ascribed to the terms are often different, obscuring underlying differences in the systems. Here we provide specific mappings from the terms we use to the terminology from previous

work. This previous work is discussed further in Chapter 3; throughout this section, all tables include data from all works discussed in Chapter 3 (and no others), wherever said works used any term or reputation function which maps to the terms and reputation functions we define. Where a work does not appear in these tables, it either eschewed discussion of that set of terms entirely, or it did not specifically give a name to the terms or reputation functions it implicitly used or supported. Our discussion of reputation functions in this chapter is complete; there are no reputation functions discussed in these works (implicitly or explicitly) that does not appear under some name in this chapter.

2.5.1 Methodology

In this section and the systematization detailed in Chapter 3, we conducted our search for papers by starting with one seed paper, AnonRep [ZWC+16]. From this seed, we examined every paper that it cites and that cites it (first, as recognized by Google Scholar in September 2019, then repeated in July 2021 in order to include more recent works). In the initial examination, we found 73 such papers. Papers were then included in our systematization if and only if both of the following were true: first, they described systems that supported a "vote" operation, where one or more voters gave feedback representing their opinion of a votee. To allow for variety, this criterion was not specified further. Second, they preserved at least one of our recognized privacy properties during said vote operation. No considerations were made regarding venues that works were presented in, in the interest of providing a complete view on work conducted in the area. The set of papers resulting from this process was not so large as to require such a condition. We feel that such a consideration has the potential to miss valuable insights from works that, as a complete paper, may have found difficulty in finding publication. Said works may still include smaller pieces of information useful to other researchers.

These conditions captured 14 and excluded 59 papers from the set of 73 under examination. All 59 excluded papers failed to implement a vote operation meeting our definition. This procedure was iteratively repeated for all included papers from this set until convergence was reached, resulting in 42 systems described across 45 papers. When repeated in July 2021, 5 recently published papers were discovered, resulting in a final count of 47 systems described across 50 papers. The mapping of properties in Section 2.5, as well as the classification of systems in Chapter 3, were both coded solely by the author.

2.5.2 Mappings of Terminology

We first note that, though a scant few papers do recognize the different ideas encapsulated by our terms "Simplex", "Half-Duplex", and "Full-Duplex" in reference to the directionality of reputation systems, no previous works actually develop these differences into specific terminology. The same is true of our terms "Voter-agnostic" and "Voter-conscious" in reference to reputation functions.

Table 2.1. Mapping of Architecture Terminology in Previous Works

Third-Party Mediation	Ephemeral Mesh Topology	Proofs of Validity
Centralized [Vos04, PRT04, Ker09, HBB12, HBBS13, BSHB16, SKCD16, SBHB16, GMN17, CSH17, ABH18, ABHS18, BAH18, LAN+19]	Decentralized [PRT04, HBB10a, HBB10b, HBB12, HBBS13, SBHB16, BSHB16, CSH17, ABH18, ABHS18]	Decentralized [SKCD16, SBHB16, ABH18, BAH18, LAN+19]
Semi-centralized [SBHB16]	Distributed [KP03, Vos04, YTP07, Ker09, GMN17, ABHS18, BAH18]	
Decentralized [BSHB16, BAH18]		
Distributed [Vos04, ABHS18]		

Table 2.1, referring to the terms from Section 2.1, demonstrates one of the clearest case of similar terminology obscuring underlying differences. To describe what we call "Third-Party Mediation", most papers call the architectures of competing systems "centralized" (though a few papers we classify as Third-Party Mediation differentiate themselves as "decentralized" [BSHB16, BAH18] or "distributed" [Vos04, ABHS18] due to differences in how they use their TTPs; specifically, they only refer to their own works as such and not the entirety of what we term Third-Party Mediation). However, there is significant confusion between the "decentralized" of Ephemeral Mesh Topology and of Proofs of Validity. An alternate term, "distributed", also commonly refers specifically to what we term Ephemeral Mesh Topology; in this work, we distinguish these terms as "system-defined decentralized" and "user-defined decentralized".

Between voter and votee privacy properties, a majority of papers focus on voter privacy properties. As such, Table 2.2, referring to the terms from the voter privacy portion of

Table 2.2. Mapping of Terminology from Previous Works Concerning Voter Privacy Properties

Voter-Vote Unlinkability	Two-Vote Unlinkability
Anonymity [IBJR03, SL03, Vos04, MR06, HYLC07, HLTZ08, MPRS08, WH09, BSS10, PLZZ10, WCMA13, CRH+13, CSK13, PLS14, ZWC+16, BPS+17, BEJ18, BBB+18, LAN+19, SBHB16]	Unlinkability [Vos04, BSS10, HS11, ZWC+16, BPS+17, BEJ18, BBB+18, LAN+19, LM19, SBHB16]
Privacy [KP03, PRT04, BIJR04, VHM05, AG06, YTP07, NR09, AAG09, HBB10a, HBB10b, HS11, HKH12, HBB12, HBBS13, ZXYM16, GMN17, CSH17, ABHS18, BAH18]	Anonymity [LM19]
Peer-Pseudonym Unlinkability [ACBM08]	Pseudonym-Pseudonym Unlinkability [ACBM08]
Review-Payment Unlinkability [SKCD16]	Review-Review Unlinkability [SKCD16]
Transaction-Rating Unlinkability [BSHB16]	Rating-Rating Unlinkability [BSHB16]
Confidentiality [Ker09, LAN+19]	
Rating Secrecy [CSK13, LM19]	
"secret, unlinkable, and anonymous" [ABH18]	

Section 2.3.1, features the greatest diversity of terms. In order to descriptively term these properties, as inspired by Kuhn *et al.* [KBS+19], we chose to name them (excluding Exact Reputation Blinding, for which this approach seemed less appropriate) with respect to an unlinkability between two entities. Three papers took a similar approach, and though they do not use the same terms "Voter-Vote" and "Two-Vote", their choices (Peer-Pseudonym / Pseudonym-Pseudonym Unlinkability [ACBM08], Review-Payment / Review-Review Unlinkability [SKCD16], and Transaction-Rating / Rating-Rating Unlinkability [BSHB16]) embody a similar spirit. Aside from those, concerning Voter-Vote Unlinkability specifically, previous work was widely split between "anonymity" and "privacy", though a few used other terms like "confidentiality" or "rating secrecy". Where other papers did consider Two-Vote Unlinkability, they most commonly referred to it simply as "unlinkability", though one paper confusingly referred to it as "anonymity" as well [LM19]. This paper, in concert with Kuhn *et al.*'s [KBS+19]

wider suggestion to do so, particularly inspired our desire to avoid the word "anonymity" in the name of any of our terms.

Table 2.3. Mapping of Terminology from Previous Works Concerning Votee Privacy Properties

Reputation-Usage Unlinkability	Exact Reputation Blinding
Identity Anonymity [ZWC+16]	Reputation Budget [ZWC+16]
Signer Anonymity [BSS10]	"cloaking of reputation scores" [CRH+13]

Votee privacy properties were typically less commonly provided by systems, so it is not surprising that terminology is not as frequently developed. What terminology was developed can be observed in Table 2.3, referring to the terms from the votee privacy portion of Section 2.3.2. Reputation-Usage Unlinkability is closely related to Voter-Vote Unlinkability, and as "anonymity" was a common choice for that term, both examples we observed were modifications of anonymity ("identity anonymity" [ZWC+16] and "signer anonymity" [BSS10]). Exact Reputation Blinding is similarly obscure, and we felt that the only specific terms used previously ("reputation budget" and "cloaking of reputation scores") were not adequately descriptive of what the property accomplished.

Table 2.4. Mapping of Terminology from Previous Works Concerning Reputation Functions (Where Addressed Directly)

Accrue Stars	Average Stars	Gompertz function	Short-Term Memory Consensus	Long-Term Memory Consensus
Sum [PLS14]	Mean [SBHB16]	Gompertz function [HKH12]	Ordered Weighted Average [NR09]	N/A
(0, 1) / (0, 1, -1) [ABH18]	(0–5) [ABH18]			

Table 2.4, referring to the terms from Section 2.4, is very sparse. We believe this is due to the fact that a majority of papers we consider either completely ignored reputation functions in their system design, or only worked with one specific function and did not see a need to name it. What we term Long-Term Memory Consensus in particular, though used by several systems, is never named. The unusual "terms" in PrivBox [ABH18] (being actually just sets of values that can be used to vote with) come from the fact that in that paper, instead of naming

these as functions, the paper makes comparison directly to the choices of values to vote with instead. These sets are then used as a proxy for the functions themselves. We felt that our names were more descriptive of the actual functions they describe, with the exception of Gompertz, which we take from Huang *et al.* [HKH12].

Chapter 3

Related Work

As discussed in Chapter 2, reputation systems have significant variety in the various forms in which they have been used. They are fairly well studied, and in particular there have been a number of works detailing not only reputation systems, but privacy-preserving reputation systems, which are of direct interest to this work. Of particular interest to this work is one specific such system, AnonRep [ZWC+16], from which our system draws significant inspiration. In this chapter, we discuss this line of research at large with its relevance to this work, and specifics about AnonRep in order to highlight our own novel contributions. The methodology we use to select works to compare to was discussed in Section 2.5.1.

3.1 Privacy-Preserving Reputation Systems

Though reputation systems have been widely used in the real world, systems that have been deployed frequently do not guarantee privacy to their users. Most commonly, these systems are highly linkable. That is, a user's actions in the reputation system can be linked together; these actions may include the votes they cast, the votes they receive, and the times they validate their reputation. Privacy-preserving reputation systems have been a line of research dating back to 2003 [IBJR03] on how to build reputation systems that can prevent these linkages while preserving the integrity of the reputation system. In Table 3.1 and Table 3.2, we systematize the strategies taken and the properties provided in privacy-preserving reputation systems in the literature. Table 3.2 specifically also includes Amazon, eBay, and Reddit as well-known reputation systems for comparison's sake. We compare systems using a variety of features, as follows:

Table 3.1. Privacy-Preserving Reputation Systems, Part 1

Name	*Year*	*Centralization*	*Directionality*	*Rep. Scope*	*Rep. Ownership*	*Correctness*	*Unlinkable to TTP*	*# TTP for Setup*	*# TTP Ongoing*	*More via Anytrust*	*V-V Unlinkability*	*2V Unlinkability*	*R-U Unlinkability*	*Exact Rep. Blinding*	*Acc. Stars*	*Acc. Stars — Neg.*	*Avg. Stars*	*Gompertz*	*STMC*	*LTMC*
System		**Structure**				**Trust**					**Privacy**				**Rep.**					
Coin-based Reputation Systems																				
Ismail *et al.*[IBJR03]	2003	★	↔	∀	◗	◐	-	2	2	-	●	●	◐	●	●	-	●	-	-	-
Voss [Vos04]	2004	★	↔	∀	◗	◐	-	1	1	-	◐	-	◐	●	●	●	-	-	-	-
Androulaki *et al.*[ACBM08]	2008	★	↔	∀	◗	◐	◐	1	1	-	●	●	◐	●	●	-	-	-	-	-
Dimitriou [Dim21]	2021	∴	↔	∀	◗	●	●	1	0	●	●	●	●	●	●	●	●	-	-	-
Signature-based Reputation Systems																				
iClouds [VHM05]	2005	★	↔	∀	⛛	-	-	1	N	-	◐	-	◐	-	-	-	-	-	●	-
Signatures of Reputation [BSS10]	2010	∴	↔	∀	⛛	●	●	1	0	-	◐	-	●	●	●	-	-	-	-	-
Reputation Transfer																				
Anwar and Greer [AG06]	2006	★	↔	∀	◗	◐	-	1	1	-	-	-	●	-	*	*	*	*	*	*
RuP [MR06]	2006	★	↔	∀	◗	-	●	1	1	-	-	●	●	-	*	*	*	*	*	*
DARep [HYLC07]	2007	★	↔	∀	◗	-	-	1	N	-	-	●	●	-	-	●	-	-	-	-
Hao *et al.*[HLTZ08]	2008	★	↔	∀	◗	-	-	1	1	-	-	●	●	-	-	-	-	-	-	●
Wei and He [WH09]	2009	★	↔	∀	◗	-	●	1	1	-	-	●	●	-	-	-	-	-	-	●
Peng *et al.*[PLZZ10]	2010	★	↔	∀	◗	-	●	1	1	-	-	●	●	-	*	*	*	*	*	*
Huang *et al.*[HKH12]	2012	★	→	∀	◗	●	-	1	1	-	-	-	●	●	-	-	-	●	-	-
IncogniSense [CRH+13]	2013	★	→	∀	◗	●	-	1	1	-	-	-	●	●	*	*	*	*	*	*
k-Anonymous Reputation [CSK13]	2013	★	↔	∀	◗	-	-	1	1	-	●	●	-	-	*	*	*	*	*	*
SMC-based Reputation Systems																				
Kinateder and Pearson [KP03]	2003	☍	↔	∃	⊙	-	-	0	N	-	●	●	-	-	-	-	-	-	●	-
DARS [PRT04]	2004	☍	↔	∃	⊙	●	●	0	0	-	●	●	-	-	-	-	-	-	●	●
PDSPP [YTP07]	2007	☍	↔	∃	⊙	●	●	0	0	-	●	●	-	-	-	-	-	-	●	●
3PRep [NR09]	2009	☍	↔	∃	⊙	●	●	0	0	-	●	●	-	-	-	-	-	-	●	●
CRDSPP [AAG09]	2009	☍	↔	∃	⊙	●	●	0	0	-	●	●	-	-	-	-	-	-	●	●
k-Shares [HBB10a, HBB10b, HBB12, HBBS13]	2010	☍	↔	∃	⊙	●	●	0	0	-	●	●	-	-	-	-	-	-	●	●
PFWRAP [ZXYM16]	2016	☍	↔	∃	⊙	●	●	0	0	-	●	●	-	-	-	-	-	-	●	●
Dyn-PDRS [CSH17]	2017	☍	↔	∃	⊙	●	●	0	0	-	●	●	-	-	-	-	-	-	●	●
M2M-REP [ABHS18]	2018	☍	↔	∀	⊙	●	●	0	0	-	●	-	-	-	-	-	-	-	●	●

Reputation Attributes

Centralization:	★ = Third-Party Mediation	☍ = Ephemeral Mesh Topology	∴ = Proofs of Validity
Directionality:	→ = Simplex	⇌ = Half-Duplex	↔ = Full-Duplex
Scope:	∀ = Global	∃ = Local	
Ownership:	⛛ = Votee-owned	◗ = TTP-owned	⊙ = Voter-owned
Correctness: via...	● = ...protocol guarantees	◐ = ...errors are traceable	- = ...TTP/miners
Trust Unlinkability: TTP can link...	● = ...nothing	◐ = ...misbehaviour	- = ...everything
Privacy Unlinkability:	◐ = Participants to a transaction can link each other		
Reputation:	* = This work considers reputation functions to be outside its scope.		

Table 3.2. Privacy-Preserving Reputation Systems, Part 2

Name	*Year*	Centralization	Directionality	Rep. Scope	Rep. Ownership	Correctness	Unlinkable to TTP	# TTP for Setup	# TTP Ongoing	More via Anytrust	V-V Unlinkability	2V Unlinkability	R-U Unlinkability	Exact Rep. Blinding	Acc. Stars	Acc. Stars — Neg.	Avg. Stars	Gompertz	STMC	LTMC
System		**Structure**				**Trust**					**Privacy**				**Rep.**					
Ticket-based Reputation Systems																				
[TTP Approaches]																				
Amazon	1994	★	→	∀	◗	-	-	1	1	-	-	-	-	-	-	-	●	-	-	-
eBay	1995	★	⇌	∀	◗	-	-	1	1	-	-	-	-	-	-	●	-	-	-	-
Reddit	2005	★	↔	∀	◗	-	-	1	1	-	●	●	-	-	-	●	-	-	-	-
TrustMe [SL03]	2003	★	↔	∀	◗	●	●	1	0	-	●	-	-	-	●	●	●	-	-	-
Boyd *et al.*[BIJR04]	2004	★	↔	∀	▽	-	-	1	1	-	●	●	-	-	●	●	●	-	-	-
ARM4FS [MPRS08]	2008	★	→	∀	⊙	-	-	1	2	-	●	●	●	●	●	●	●	-	-	-
Kerschbaum [Ker09]	2009	★	↔	∀	◗	◐	●	0	2	-	●	●	-	-	●	●	●	-	-	-
Hussain and Skillicorn [HS11]	2011	★	↔	∀	◗	-	●	1	1	-	●	●	-	-	●	●	●	-	-	-
ARTSense [WCMA13]	2013	★	→	∀	◗	●	-	1	1	-	-	-	●	-	●	●	●	-	-	-
Petrlic *et al.*[PLS14]	2014	★	↔	∀	◗	●	●	1	1	-	●	●	-	-	●	●	●	-	-	-
AnonRep [ZWC+16]	2016	★	↔	∀	◗	◐	◐	2	2	●	●	●	●	●	●	●	●	-	-	-
Bazin *et al.*[BSHB16]	2016	★	↔	∀	▽	◐	-	1	1	●	●	●	-	-	●	●	●	-	-	-
Busom *et al.*[BPS+17]	2017	★	⇌	∀	◗	●	●	1	1	-	●	●	-	-	●	●	●	-	-	-
Garms *et al.*[GMN17]	2017	★	↔	∀	◗	●	-	1	1	-	●	●	-	-	●	●	●	-	-	-
El Kaafarani *et al.*[EKKS18]	2018	★	→	∀	◗	◐	◐	1	1	-	●	●	-	-	●	●	●	-	-	-
Blömer *et al.*[BEJ18]	2018	★	↔	∀	◗	●	●	1	1	-	●	●	-	-	●	●	●	-	-	-
CLARC [BBB+18]	2018	★	↔	∀	◗	◐	●	1	1	-	●	-	-	-	●	●	●	-	-	-
PrivRep [BAH18]	2018	★	↔	∀	⊙	-	●	1	1	-	●	●	-	-	-	-	-	-	●	-
pRate [LM19]	2019	★	↔	∀	▽	●	●	1	1	-	●	●	●	●	-	-	●	-	-	-
BPRF [JC19]	2019	★	→	∀	◗	●	◐	2	2	-	-	-	●	-	●	●	●	-	-	-
EARS [KYMX20]	2020	★	↔	∀	◗	●	●	1	1	-	●	-	●	●	-	-	-	-	●	-
Garms *et al.*[GNQT20]	2020	★	↔	∀	◗	◐	◐	1	2	-	●	●	-	●	-	-	●	-	-	-
PRSONA [this work]	2021	★	↔	∀	◗	●	●	2	2	●	●	●	●	●	-	-	-	-	●	●
[Public Log Approaches]																				
Beaver [SKCD16]	2016	∴	→	∀	◗	◐	●	0	0	-	●	●	-	-	●	●	●	-	-	-
Schaub *et al.*[SBHB16]	2016	∴	→	∀	◗	◐	●	0	0	-	●	●	-	-	●	●	●	-	-	-
PrivBox [ABH18]	2018	∴	→	∀	◗	◐	●	0	0	-	●	●	-	-	●	●	●	-	-	-
ARS-PS [LAN+19]	2019	∴	→	∀	◗	●	●	1	1	-	●	●	-	-	●	●	●	-	-	-

Reputation Attributes

Centralization: ★ = Third-Party Mediation; ☍ = Ephemeral Mesh Topology; ∴ = Proofs of Validity
Directionality: → = Simplex; ⇌ = Half-Duplex; ↔ = Full-Duplex
Scope: ∀ = Global; ∃ = Local
Ownership: ▽ = Votee-owned; ◗ = TTP-owned; ⊙ = Voter-owned
Correctness: via... ● = ...protocol guarantees; ◐ = ...errors are traceable; - = ...TTP/miners
Trust Unlinkability: TTP can link... ● = ...nothing; ◐ = ...misbehaviour; - = ...everything
Privacy Unlinkability: ◐ = Participants to a transaction can link each other
Reputation: * = This work considers reputation functions to be outside its scope.

Structure We identify four factors relating to a reputation system's structure, as follows:

Centralization We indicate centralization as defined in Section 2.1.

Directionality We indicate reputation directionality as defined in Section 2.2.

Reputation Scope We indicate the scope of reputation as global (each votee has one score that all requesters see) or local (each score is dependent on which voters a requester works with to obtain a score).

Reputation Ownership We indicate the owner of reputation as votee-owned (a votee displays its own score with appropriate validation), TTP-owned (a requester obtains a votee's score from some third party with appropriate validation), or voter-owned (a requester obtains votes from voters for each votee).

Trust We identify five factors related to the level of trust placed on third parties in the system, as follows:

Correctness We indicate when correctness is guaranteed by the protocols used in a system, versus when errors can be recognized and flagged by anyone, versus when it is left to the TTP (or blockchain miners, who effectively act as a sort of distributed TTP) to handle correctness.

Unlinkable to TTP We indicate whether TTPs are relied on to protect the privacy of users (that is, whether or not users can always have their behaviour linked by a TTP).

 # TTP for Setup We indicate the minimum number of TTPs required to use a system in initial setup

 # TTP Ongoing We indicate the minimum number of TTPs required to use a system for ongoing usage.

More via Anytrust We indicate those systems that allow additional TTPs to be added in an *anytrust* relationship — that is, the system's guarantees are upheld if *any one* of the TTPs is honest.

Privacy We identify the privacy properties (as defined in Section 2.3) provided by each system.

Reputation We identify the reputation functions (as defined in Section 2.4) supported by each system.

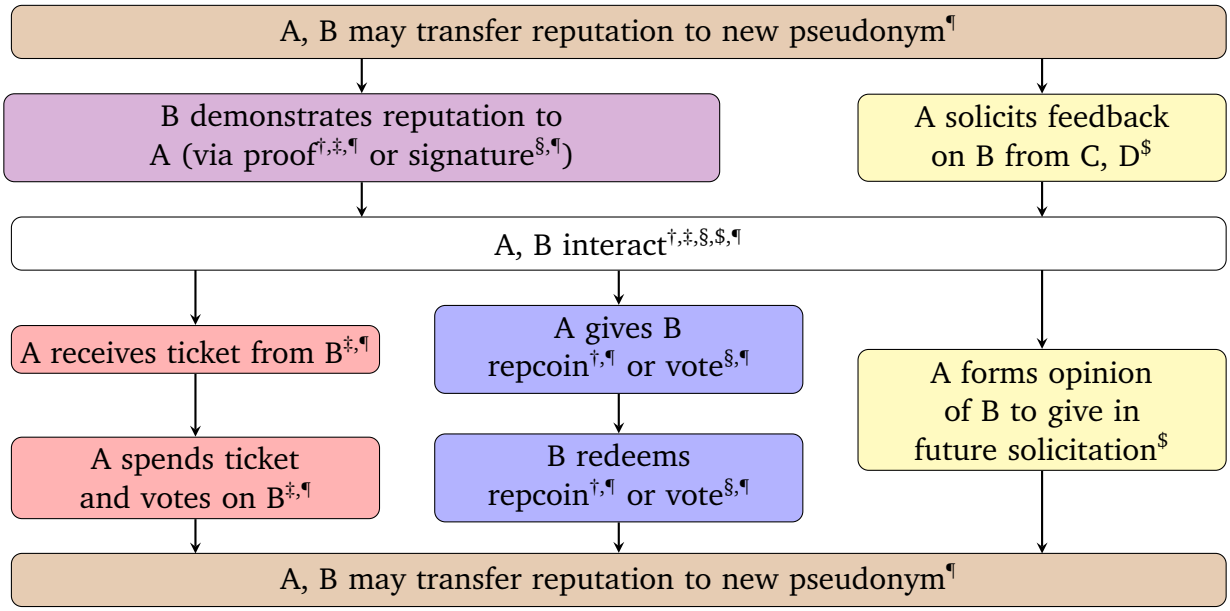


Figure 3.1. A system model visualization of the approaches discussed here. At each step, † (blue , violet) represents coin-based approaches, § (blue , violet) represents signature-based approaches, ¶ (brown) represents reputation transfer, $ (yellow) represents SMC-based approaches, and ‡ (red , violet) represents ticket-based approaches. A is a voter interacting with a votee B; C, D are other voters. Approaches skip forward when they cannot act.

In Table 3.1 and Table 3.2, we do not identify, nor does this section further elaborate on, evaluations of systems and threat models. Evaluations were not universally present in the systems under study. However, even if they were, it is difficult to compare evaluations that were performed in different environments and measured different components of the various systems. Explicit threat models, likewise, were not universally present in the systems under study. Though all works had implied threat models, comparing works on an implied threat model is imprecise.

In Table 3.1 and Table 3.2, we systematize the strategies taken in the literature to design privacy-preserving reputation systems into five approaches. Figure 3.1 visualizes the interactions typical in each approach and demonstrates how the interactions proceed. The rest of this section further elaborates on these five approaches:

- Coin-based Reputation Systems
- Signature-based Reputation Systems

- Reputation Transfer
- SMC-based Reputation Systems
- Ticket-based Reputation Systems

3.2 Coin-based Reputation Systems

Among privacy-preserving reputation systems, the earliest work was done on *coin-based reputation systems*. After a long period without new proposed coin-based reputation systems since 2008, a new system was proposed in 2021. As speculation, new work on such systems may have halted due to interest in other methods (like the similar ticket-based reputation system); the new work of 2021 may have been inspired by recent advances among cryptocurrencies such as zCash, which feature similar zkSNARK techniques.

Coin-based reputation systems are based upon e-cash designs; reputation is treated as a currency. Voters are granted reputation points (or "repcoins") that they may hand out to votees in the system. These repcoins are limited in some fashion to prevent reputation inflation; for example, voters may only get a set number of points to spend per epoch. Voters spend repcoins by sending them to votees. It is left open ended as to when this may occur; coin-based reputation systems do not typically require specific transactions to take place between participants in order to exchange repcoins. Upon receiving a repcoin, a votee then engages in a protocol to deposit the repcoin, raising their reputation score in the process. At any point, a votee may generate a proof that confirms their reputation level to a requester.

While all coin-based reputation systems in previous work have required a third party (which we will call "the bank") to facilitate portions of these transactions, this is not strictly required. Work on cryptocurrencies could potentially be adapted for use with a coin-based reputation system, much as they have (as we will see in Section 3.6) for ticket-based reputation systems.

A typical interaction might look something like the approach described by Androulaki *et al.* [ACBM08]: Alice and Bob seek to interact with one another, and generate unique pseudonyms for this particular interaction. They both generate proofs that the pseudonyms correspond to a votee with their reputation levels and exchange these proofs. Alice and Bob, considering the reputation levels of the other, decide to continue their interaction. After concluding, Alice decides to spend a repcoin on Bob. Receiving this repcoin, Bob deposits it in a two-step process. First, using his pseudonym, he exchanges the repcoin for a blind signature (as introduced by Chaum [Cha83]) from the bank. Then, under his long-term identity, Bob

unblinds the signature and transmits it back to the bank, which in turn increases his reputation score.

The bank is typically trusted to faithfully follow its protocols. It is responsible for distributing repcoins according to whatever limitations the system imposes. It is also responsible for maintaining records of each votee's deposited repcoins and corresponding reputation levels. However, although the bank is trusted to follow its protocols, the bank is not fully trusted with user privacy. In particular, the bank is not trusted to learn the linkage between a user and her pseudonym, or with what other users a given user may be interacting.

Coin-based reputation systems tend to be limited in terms of what reputation functions they support, due to the nature of how they use repcoins. They do, however, tend to provide useful, uncommon privacy properties, such as Exact Reputation Blinding. Frequently, this property is provided by votees having the ability to present zero-knowledge proofs of statements. These statements might include that their reputation is above some threshold, as mentioned by Ismail *et al.* [IBJR03] and implemented by Androulaki *et al.* [ACBM08].

Ismail *et al.* [IBJR03] designed the first privacy-preserving reputation system work we identify, in 2003. They design the bank as two entities: TI (for "token issuer") and CA (for "certificate authority"). TI handles distributing repcoins and is trusted to see the interactions between users to verify that feedback comes from real interactions. CA handles maintaining reputation scores for votees, and restricts votees to only see their own scores directly. Votees disseminate their scores via a designated verifier scheme, so only their intended recipient may see their reputation.

Voss [Vos04] presents a system in which repcoins are used as collateral in interactions. Voters request a number of repcoins of their choosing as the collateral for any interaction, and may invalidate any or all of them as punishment for bad behaviour. Alternately, they may award a single repcoin as positive reinforcement for good behaviour. Privacy exists for users from one another but not from the bank in this scheme.

Androulaki *et al.* [ACBM08] form their guarantees against misbehaviour by threatening that users who misbehave will implicate themselves and reveal the secret key to their long-term identity by doing so. Uniquely, anonymous credential systems, similar in concept to those introduced by Chaum [Cha85], form the basis for the votees' proofs of reputation levels in this work.

Dimitriou [Dim21] proposes an update to the system of Androulaki *et al.* [ACBM08]. By using zkSNARKs and an append-only public ledger, Dimitriou's proposed system resolves long-standing issues of how to force users to accept negative coins. In this system, users must commit to accepting a new coin before they know its impact to their reputation, and this commitment is handled through Pedersen commitments and zkSNARKs. The append-only

public ledger also lessens the reliance on trusted third parties; while the ledger needs to be maintained in some way, it can be maintained by multiple parties rather than one central Bank entity. Although it does not provide any mechanism for vote updates, this system represents a significant advancement for transaction-based contexts.

3.3 Signature-based Reputation Systems

Signature-based reputation systems are another approach to manage reputation in an unlinkable way. Less work was performed on this type of reputation system than the others identified in our classification. As speculation, interest may have been limited in further development of this line of work due to the tight relationship between the design core of the main work in this line (Signatures of Reputation [BSS10]) and its inflexibility with regards to reputation functions. Being unable to support averaging votes, or negative votes, due to reasons central to its approach would be a difficult hurdle to overcome.

Signature-based reputation systems were designed specifically to address the problem of voters ballot-stuffing, inflating others' (or their own) reputations by spending multiple repcoins on a target. Importantly, in signature-based reputation systems, a voter may only vote for any single votee once. That is, they can vote for as many votees as they like, but for any individual votee, their reputation score is determined by the number of *unique* voters who voted for them. These votes come in the form of signatures, which include information to bind a vote to a voter's and votee's long-term identities. This binding is carefully constructed to avoid linking voters' or votees' actions.

A typical interaction might look something like the approach taken with Signatures of Reputation [BSS10]: Alice and Bob seek to interact with one another, and as before, generate unique, short-term pseudonyms for this interaction based off of their long-term pseudonyms. They both use these short-term pseudonyms and votes they have previously received to sign messages with a specialized signature scheme. This scheme is designed such that Alice can prove the input votes were given by r_A distinct voters, where r_A is Alice's reputation score (and similarly for Bob). Alice and Bob, considering the reputation levels of the other, decide to continue their interaction. After concluding, Alice decides to vote for Bob, and generates said vote using her long-term pseudonym. Receiving this vote, Bob retains it for future proofs; if Alice has never previously voted for Bob, his maximum claimable score increases by one.

Unlike coin-based systems, signature-based reputation systems by design have the property that voters may cast at most one useful vote for any given votee. These systems are also limited in what reputation functions they can support, due to the manner in which they use these

votes in their signatures for reputation. They do, however, present interesting opportunities to design a system which has a smaller or more decentralized approach to trust, either requiring only trusted platform modules (TPMs) run by users, or requiring a TTP only to set up the system, and not in its ongoing operation. Additionally, more advanced work to support a less common privacy property, Reputation-Usage Unlinkability, was done in this line of work early on.

The prototypical iClouds [VHM05] relied on TPMs in its design. The work was expected to be used on mobile phones for information dissemination networks, and Voss *et al.* proposed that TPMs would allow votees to carry and produce their own reputation scores. Specifically, when voting for others, voters encrypt their votes in such a way that only the TPM will be able to open it, and the TPM can add together scores. The TPM, upon seeing a new vote by the same voter for a recipient, simply replaces the original with the new vote. In order to participate, users first sign up with a central authority to be given a certified long-term public key, with which they conduct all actions. In this system, voters do link their own actions together, in the view of their votees, but the value of the votes they cast is private.

Signatures of Reputation [BSS10] have a much lower bound for trust, at the cost of functionality. A certificate authority is required so that users may only join the system with one long-term public key. However, from that key, users are able to generate as many short-term pseudonyms as they desire without needing to interact with any TTP. Importantly, votes cast for a votee cannot be rescinded; that is, Accrue Stars — Negative is not supported. The authors of this work argue that the reputation in their system is only meant to be a barrier against spam, but support for negative votes is recognized as a valuable area for future work.

3.4 Reputation Transfer

Unlike the previous two approaches, research on *reputation transfer* is agnostic to design choices around how voters may rate one another and how those ratings are tallied. There have been no new works in this line since 2013, perhaps because works in other lines have begun to incorporate core ideas from reputation transfer. Reputation transfer largely leaves the calculation of reputation up to the implementer, and instead focuses on one specific problem; namely, when users participate in any long-term system, having only one pseudonym for the entire length of participation is only a minor upgrade from using one's true identity.

As the calculation of reputation is typically left open, there is no typical full interaction for this approach. However, all works within this approach feature some procedure for votees to generate new pseudonyms that inherit the reputation of previous pseudonyms. These previous

pseudonyms become invalid for future use. An archetypical example of this behaviour is given by RuP [MR06], where systems operate in epochs, and transfers are allowed between adjacent epochs. These transfers are executed by first requesting a TTP to strip information pertaining to previous pseudonyms from information used to prove scores, then requesting the TTP to bind that score information to a new pseudonym.

At large, works within Reputation Transfer advanced a recognition of the usefulness of Reputation-Usage Unlinkability and draw attention to the fact that Reputation-Usage Unlinkability becomes significantly stronger when combined with Exact Reputation Blinding. However, they rely heavily on TTPs in order to accomplish their transfer. How to best decentralize this procedure is an open question.

Anwar and Greer [AG06] initiate this line of research by suggesting that enabling users to transfer their reputation between pseudonyms would help them safeguard their privacy. They suggest using third-party guarantors that users can trust to properly transfer reputation scores between pseudonyms upon request. In their design, the guarantors are fully aware of the transfer, and users must trust the guarantor to keep the relationship between the pseudonyms secret.

RuP [MR06] adds restrictions in the name of preserving the correctness of the reputation system. Users may transfer their scores between pseudonyms, but they are only allowed to use one pseudonym during any given epoch. The pseudonym they use is signed by a TTP for a given epoch. RuP also uses blind signatures, so that users do not have to rely on the TTP to protect the privacy of their pseudonym linkage.

DARep [HYLC07] uses TPMs to generate secret pseudonyms for each user in a consistent manner. All users change pseudonyms simultaneously by changing a parameter sent to the TPM. How this parameter is changed is left open to implementers; a central authority may direct it, or users may reach a consensus on the timing. Votees' reputations are held by other users, and this set of users changes whenever pseudonyms change. After this change, the users who previously held a votee's reputation can identify the new pseudonym, but other users do not have enough information to perform this linkage. Interestingly, in DARep, when a voter votes for a votee, there is a cost imposed to their own reputation, in order to disincentivize ballot-stuffing attacks.

Hao *et al.* [HLTZ08] in 2008 note that RuP is not robust against Sybil attacks, and identify the problem of users' exact score values potentially deanonymizing them on transfer. Hao *et al.*also make alterations to the blind signature scheme of RuP for efficiency gains, and Wei and He [WH09] propose similar alterations.

Peng *et al.* [PLZZ10] in 2010 also modify RuP to improve efficiency. They too recognize the issue of users' scores deanonymizing them on transfer. As a cost-saving alternative to

blind signatures, their design supports a protocol they call "group confusion", where several users with the same reputation all request reputation transfers at the same time. Their design does not mandate this behaviour, but it recognizes that the size of anonymity sets for users who transfer reputations is important.

Huang *et al.* [HKH12] further recognize this problem and specifically attempt to solve it. Inspired by *k*-anonymity, they fuzz reputation scores when publicly announcing them such that there is always a group of users for each reported reputation value at each time interval. Their system minimizes the difference between actual and reported score while preserving their *k*-anonymity goal. IncogniSense [CRH+13] takes a very similar approach, and specifically analyzes a set of different methods to cloak reputation scores for accuracy and usefulness.

k-Anonymous Reputation [CSK13] is also inspired by *k*-anonymity, but in a different manner. It recognizes that users may desire to keep pseudonyms for longer periods of time. Persistence of pseudonyms can be useful for users, as both name and score serve as markers of their reputation. As opposed to the technique used by Huang *et al.* [HKH12], where reputation scores are fuzzed, users are required to wait to get a new pseudonym until a large enough group willing to change pseudonyms forms, so that anonymity may be preserved for all of them.

3.5 SMC-based Reputation Systems

Secure multiparty computation (SMC) forms the basis for another early line of research on privacy-preserving reputation systems, *SMC-based reputation systems*. Unlike previously mentioned approaches, this line has persisted and new research has continued up to 2018. As many works in this line are iterative upon each other and feature mostly efficiency gains, the time for a new work to be proposed in the future may depend on identifying new techniques that can be applied to this line of work for further efficiency gains.

SMC-based approaches largely arose as a unique and useful application of SMC techniques, rather than as a tool for reputation within familiar paradigms of transactions and communications, and thus take a drastically different shape from most other approaches. In particular, SMC-based approaches tend to envision reputation as belonging to those who assign it, rather than belonging to those it describes. Put another way, other approaches typically view reputation as a score or value that a votee may display in a verified manner, and it certifies that over some period of activity they have accrued a specified reputation. However, SMC-based approaches view reputation instead as the collection of ratings of voters,

applied to a votee. To determine the nature of a requester's interaction with a votee, the requester will seek out and combine the ratings voters would give to the votee. A votee does not own their own reputation and does not display it; instead, they only display sufficient information for requesters to be able to identify the votee to voters.

A typical interaction might look something like the approach of Decentralized Additive Reputation Systems (DARS) [PRT04]: Alice is deciding whether to interact with Bob. Alice solicits feedback from Carol and Dave, which is securely composited into a reported score for Alice to consider. She decides to interact with him, and based on Carol and Dave's reported score and her own interactions with Bob, Alice forms her own score for him. Later, when Carol asks Alice (among others) for a new evaluation of Bob, Alice provides this score (as part of a securely composited score) to Carol.

SMC-based reputation systems start from a place of (user-defined) decentralization that other approaches frequently do not. These systems do not provide for a verified global scoreboard, as other systems do. Instead, requesters are expected to choose the voters from whom they solicit ratings themselves. Although this could be every eligible voter in the system, these systems are designed with the intention that requesters only request votes from a subset of voters (in particular, these systems are typically inefficient for large numbers of voters). Requesters are able to get pinpointed ratings from voters they trust, assuming they already trust some voters. Some systems allow requesters to weight ratings from voters based on how much stock they place in their recommendations. A different way of viewing this property is in terms of identifying a votee's reputation among specific communities or subgroups within a system. SMC-based approaches by their very nature turn voter-agnostic reputation functions into voter-conscious reputation functions, and consistently provide Voter-Vote Unlinkability and Two-Vote Unlinkability. However, again due to their nature, they are completely unable to provide Reputation-Usage Unlinkability, as they do not provide for short-term pseudonyms without needing to start over on reputation for each new pseudonym.

Kinateder and Pearson's [KP03] design is atypical compared to later approaches, in that it does not directly draw from cryptographic SMC techniques. Although the overall structure of the system is still similar (requesters soliciting ratings from voters about a votee in a manner that can be calculated without any individual voter's rating being revealed), this is accomplished through TPMs instead of cryptography. The design is more of a rough sketch than a fully fleshed out system, but it gives the overall idea of the approach.

DARS [PRT04] is more closely related to the later cryptographic approaches. In this work, the authors design a reputation system through an application of secure sum (through multiple different techniques, including secret sharing), an SMC technique. Further work among SMC-based reputation systems largely only modifies this approach or the output of

the algorithm, rather than using completely different SMC techniques.

One such work is the Private Distributed Scalar Product Protocol (PDSPP) [YTP07]. Instead of secure sum, it calculates a scalar product. Reputation is represented as the inner product of a vector of reputation ratings given by voters and a vector of ratings of trust a requester places in each voter. Put another way, it is a weighted sum of votes. The Collusion-Resistant Distributed Scalar Product Protocol (CRDSPP) [AAG09] expanded on this work and observed a security flaw in the PDSPP related to collusion between certain agents in the semi-honest model. The Privacy-Friendly Weighted-Reputation Aggregation Protocol (PFWRAP) [ZXYM16] further expanded and made efficiency gains.

3PRep [NR09] expands upon a non-privacy preserving decentralized reputation system, P2PRep [DdVPS03], attempting to add private computation of reputations to the system. Unlike other systems within the SMC approach, this system is not fully decentralized, at times relying on pre-trusted peers within the system to prepare certain computations or hold specific encryption keys separate from a requester. However, neither is it fully centralized; ratings still come directly from a variety of voters that work in concert to evaluate the "Ordered Weighted Average", an average that gives higher weighting to low or repeated reputation scores collected from voters.

A series of works defining k-Shares [HBB10a, HBB10b, HBB12, HBBS13] present several incremental improvements to the SMC-based approach. The works primarily improve communication complexity and, over their span, move from semi-honest models to malicious adversary models. In brief, these works largely derive their efficiency gains from reducing the burden on requesters to receive ratings from every voter in the system in order to guarantee that they do not collude, instead only requiring that they can receive ratings from k of them that they trust.

Dyn-PDRS [CSH17] focuses on a single specific problem within Ephemeral Mesh Topology designs. Specifically, the authors note that when users leave such decentralized systems, their ratings leave as well. Dyn-PDRS provides mechanisms for voters' ratings to continue to be used after a voter exits the system, by giving them to other voters to propagate. This does introduce the question of how long a voters' rating should still be considered accurate for a votee after the voter leaves the system. That being said, it is an interesting consideration to look at how systems handle the recommendations of voters who no longer continue to participate.

M2M-REP [ABHS18] has a different form than the other SMC-based works. In particular, it returns to a notion of global reputation. This is achieved by voters posting an encrypted version of their vote (and a proof that the vote is of a particular form) to a public bulletin board. The particular form of the vote takes advantage of the fact that votes can only be -1, 0,

or 1, and allows them to be combined in a way that outputs a plaintext summary score. Due to the construction used, this plaintext output can only occur when all votes are combined.

3.6 Ticket-based Reputation Systems

Ticket-based reputation systems are the most consistently researched approach for privacy-preserving reputation systems. Ticket-based reputation systems form a natural extension of coin-based reputation systems, and the first ticket-based system was proposed in the same year as the first coin-based system. Ticket-based systems have, like SMC-based systems, continued to be researched through contemporary work. More papers have been published in this approach than any other.

In a ticket-based reputation system, instead of being able to award coins for favourable interactions as in coin-based reputation systems, a voter is given some kind of authorization (or "ticket") to give a rating to a votee. This ticket is frequently the straightforward result of a one-to-one transaction — the archetypical example of this being a sale on a service like eBay, where the buyer and seller are given the opportunity through the service to rate each other after conducting their business. However, the ticket in some systems may be more naturally understood as one-to-many (such as, voters rating posts made by votees in a forum) or many-to-many (such as, all voters being given opportunities to revote once per some set epoch for all votees). In order to vote, a voter "spends" their ticket along with giving a rating value. A ticket may only be spent once by any individual voter in order to prevent ballot-stuffing attacks.

A typical interaction might look something like this. Alice and Bob seek to interact with one another. They both identify the others' reputation levels (sometimes via a proof, sometimes via a public bulletin board, sometimes via a TTP). Alice and Bob, considering the reputation levels of the other, decide to continue their interaction, in the process exchanging tickets for each other. After concluding, Alice decides to rate Bob. She redeems the ticket she received from Bob, posting her feedback either to a TTP or to a public bulletin board (typically in a way unlinkable to her long-term identity). This feedback directly affects future calculations of Bob's reputation. We note in particular that these two approaches for posting feedback constitute their own branches of this approach, which we term *TTP ticket-based reputation systems* and *public log ticket-based reputation systems*. Boyd *et al.* [BIJR04] provide a good archetype for a TTP ticket-based reputation system, and Beaver [SKCD16] is a good archetype for a public log ticket-based reputation system.

Significant work in this approach has been focused on minimizing the trust placed in any

centralized party. There is, however, a direct tension between the efficiency and flexibility of such systems with the amount of trust placed in a centralized party to accomplish it. Centralized systems can be designed to compute reputation functions without incurring large costs associated with the cryptography usually needed to guarantee privacy in decentralized systems or systems that trust their TTPs less. Further, in order to provide a more diverse range of reputation functions (particularly complex ones), systems need more details about votes, such as the identity of the voter or their own reputation. This is challenging to provide in a privacy-preserving way. With the increasing popularity of blockchain technologies, there have been a number of systems intended for transaction-based contexts recently that prioritize decentralizing trust (that is, via blockchain), while systems in community-based settings have tended to focus on providing a wider array of privacy properties and/or reputation functions without centralizing trust.

3.6.1 Trusted Third Party Approaches

By definition, TTP ticket-based reputation systems involve entities in which some amount of trust must necessarily be placed. However, how much trust is placed and in how many such entities varies between systems. In some cases, the TTP's role is largely relegated to verifying new identities in order to prevent Sybil attacks; in others, the TTP sees most details associated with interactions in the system. In general, research has shifted over time towards smaller amounts of trust placed in TTPs, and distributing trust over multiple entities.

Notwithstanding this trend, in 2003 Singh and Liu present TrustMe [SL03], which places only a small level of trust in its TTP. In TrustMe, a bootstrap server is used to randomly assign each votee a set of different users in the system, called the trust-holding agents (THAs), who are responsible for holding, updating, and reporting the votee's reputation. A bootstrap server is used to preserve the correctness of this arrangement. When requesters broadcast reputation queries, the THAs return the reputation scores signed with an appropriate key given by the bootstrap server. Relatively little trust needs to be placed in the bootstrap server to randomly assign THAs, and the probability of THAs colluding to falsify a votee's score decreases with the size of the system and the number of THAs utilized. However, TrustMe's privacy protections largely amount to allowing voters to cast votes that votees will not see directly.

Boyd *et al.* [BIJR04], in contrast, use a more invasive TTP. Before initiating a transaction, voters and votees jointly register the transaction with the TTP, and the TTP responds with a token for the voter to use to prove their vote is the result of a valid transaction. After the transaction, the voter sends the token along with their feedback to the TTP and receives a new signed nonce, then sends that along with their vote to the votee. The votee verifies the

validity of the nonce, then acknowledges receipt of the vote to the TTP, who then provides a signed list of all vote values given for the votee. The votee uses this to display their reputation score.

ARM4FS [MPRS08] specifically examines the use case of file-sharing systems, where multiple users may have a file, but different users' versions of the files may be of varying qualities. One TTP is used for identity verification, so that users may not create arbitrary accounts in the system. Another TTP is used when a user uploads a file to the service, which tags the file with a nonce corresponding to that user. Each file receives a different nonce, even when the same user uploads them. Then, voters submit this nonce along with their vote, which the TTP directs to the correct user. This successfully allows a votee to use their reputation without linking themselves across files, which few other schemes achieve, but does not offer much to protect voter privacy.

Kerschbaum [Ker09] proposes a system using a combination of pairing-based cryptography and traditional asymmetric encryption to achieve its privacy properties. When two users engage in a transaction, they generate tokens for each other of specific forms such that quick verification of the validity of the token is possible. The voter then submits an encrypted vote to one TTP, along with the token. This first TTP periodically forwards all received votes to a second TTP, who can decrypt the vote and verify the tokens. The second TTP collates the votes and publishes votees' scores. So long as the two TTPs do not collude, voters' votes remain private. Petrlic *et al.* [PLS14] use a similar construction, but use homomorphic encryption such that only one TTP is needed. The TTP combines ratings, and gives the encrypted result to the votee to decrypt. This does change one important aspect, namely that the TTP is thus able to see when transactions between two users occur, which was not possible in Kerschbaum's system. Blömer *et al.* [BEJ18] also use a similar construction; their work is largely notable for approaching the problem in the Universal Composability Framework. As such, it places more emphasis on its security proofs than other works have.

Hussain and Skillicorn [HS11] describe a system based around what they term "personas". Their system describes using what are effectively anonymous credentials, such that service providers can be provided reputational feedback by their customers. This feedback is collated by a TTP, who observes the validity of the anonymous credential in order to accept feedback. CLARC [BBB+18] takes a similar approach, explicitly using anonymous credentials. In CLARC, however, the TTP is not needed to collate feedback; instead, feedback and proofs of validity are published openly for requesters to collate themselves. Rather, the TTP is used to trace misbehaving users and expel them from the system. This is done by requiring that all users register with the TTP separately from the other mechanisms of the system.

ARTSense [WCMA13] is a system focused on providing reputation to participatory sensing

in a privacy-preserving manner, and thus has a few unusual properties. Primarily, reputation votes come from a central server based on the performance of a user during data collection. In this system, after data is collected and sent to the central server, the central server responds with a ticket that contains the server's vote on the user's reputation, encrypted so the user cannot know whether it is positive or negative. The user then redeems the ticket, and their score is updated accordingly. A nonce inside the ticket is used so that users cannot replay previous tickets, and blind signatures are incorporated so that the central server cannot link a user redeeming a ticket to the instance it was issued to them. BPRF [JC19] also focuses on participatory sensing, but adds an append-only public ledger. This public ledger is used by votees to audit their scores and ensure the central server's transparency in delivering them.

Modeled as a reputation system for use in forum settings, AnonRep [ZWC+16] allows users to make posts and tag those posts with their own reputation. AnonRep also gives votees the ability to blind their own reputation, such that they may instead prove that their reputation is above a threshold of the votee's choosing. Voters use linkable ring signatures to cast votes on posts, so that they may only vote once or be traceable as misbehaving. Importantly, though, the TTPs are a set of servers that engage in a verifiable shuffle in order to allow users to periodically change their pseudonyms in a reliable, unlinkable manner. This has the effect that, for every epoch where this verifiable shuffle is performed, users may make posts and receive votes unlinkably from their own previous posts. Unlike almost all other systems, AnonRep notably allows the system to add additional TTPs in an anytrust relationship. That is, if any one TTP is honest, the system's guarantees are upheld, making it desirable to add additional TTPs. We go into further detail on AnonRep in Section 3.8.

In Bazin *et al.*'s work [BSHB16], the TTPs are less involved in transactions. Instead, they audit votees for honest reporting. When two users engage in a transaction, the votee gives the voter a blind signature on a ticket. The voter reports the transaction to the TTPs. The voter then unblinds the signature and sends their vote and unblinded signature directly to the votee through an anonymous channel. Periodically, all votees submit their feedbacks to the TTPs. The TTPs sign the votes, and votees go on to display their votes with this signature from the TTPs to new potential transaction partners. If a voter's feedback is missing, however, the voter can report this to the TTPs with proof that their vote was valid. As long as one TTP audits honestly, that TTP can be relied upon to enforce the system rules.

Busom *et al.* [BPS+17] recommend a multi-tiered system of reputation — the system is nominally half-duplex (voters and votees are distinct sets), but a caveat is added to this. Voters may *endorse* other voters' feedbacks as useful, and upon enough of these endorsements, a voter may gain additional status. Other than these tweaks, the reputation system is largely similar to Kerschbaum's [Ker09] or that of Petrlic *et al.* [PLS14]. Importantly, the addition of the endorsement mechanism does not force all of a voter's feedbacks to be linkable.

Garms *et al.* [GMN17], like AnonRep, examine reputation in a forum setting. The TTP is used as the manager in a group signature scheme, so that they and no one else can link those who submit a post to keep track of their reputation. All feedback is directed through the TTP, who updates votees' reputations periodically.

El Kaafarani *et al.* [EKKS18] take inspiration from AnonRep's use of linkable ring signatures. Instead of using linkable ring signatures, they revert to the more common group signatures, but add lattice-based linkable indistinguishable tags, which replace the linkability of linkable ring signatures (so that users cannot submit arbitrary multiple votes). Interestingly, the setting for their work is not forum-based, however, and to accomodate multiple different products that different users may or may not be able to rate in their transaction-based context, each ratable object is the subject of its own group signature. Their construction uses Merkle trees to allow new users to be added to the group signature of a product after purchase, and new group signatures are constructed for new products, making the system able to accomodate adding new items and users after initial setup. Managing these group signatures is all done through one group manager (although, strictly speaking, this does not need to be the same group manager for each product).

In PrivRep [BAH18], the TTP has an additional duty to decide which voters are considered trustworthy. Untrustworthy voters' votes are silently discarded and not used in reputation calculations. PrivRep also incorporates an idea from SMC-based reputation systems, where all voters get one vote for each votee (which may be updated on subsequent polls) rather than tying votes to things like transactions. The ticket in this case is just valid participation in the system, and the privacy of votes is secured through a series of non-interactive zero knowledge proofs of knowledge (NIZKPoKs) that allow the TTP to calculate an overall reputation score but that do not reveal any participant's individual ratings.

pRate [LM19] is particularly notable for drawing attention to the utility votees may gain by blinding their exact reputation scores. pRate instead allows for statements such as that a votee's reputation is above a specified threshold. pRate also specifically allows a user to prove statements about their reputation without linking to their long-term identity, much like coin-based systems and systems like AnonRep. It accomplishes this through pairing-based cryptography and NIZKPoKs.

EARS [KYMX20] uses a series of blind and partially blind signatures to enable its ratings, and as a result is particularly efficient. Unlike nearly all other ticket-based systems to this point, EARS also allows voters to update their votes, if they decide they have changed their opinion, making EARS one of very few systems (aside from the systems based on secure multiparty computation) to enable Short-Term Memory Consensus. To enable updating votes, EARS puts significant amounts of trust into the issuer of these blind and partially blind signatures.

Garms *et al.* [GNQT20] focus on a curious problem: how reliable are the votes that voters themselves give? It can be difficult to know why any voter feels any particular way about a votee, and generally outside of confirming that a voter and a votee have interacted, reputation systems have been reticent to interfere further with the tallying of a user's score. Their work adds a new feature: voters who give reputation votes close to the average that a votee already has can themselves gain additional reputation. To accomplish this, their system employs two non-colluding trusted third parties, linkable ring signatures, direct anonymous attestation, and public-key encryption.

Our own work, PRSONA, takes inspiration from AnonRep, and like that work examines reputation in a forum setting. Instead of using linkable ring signatures to cast plaintext votes anonymously, as AnonRep does, PRSONA has users sign their votes directly, but votes are encrypted such that only all servers colluding together would be able to see the contents of the vote. Further, when a user submits a vote, it does not necessarily indicate that they gave an updated vote for any individual user, or indeed, any user at all. TTPs are arranged in a verifiable shuffle arrangement similar to AnonRep, although the correctness of the system (specifically, correct decryption of the votes) requires a majority of servers to behave honestly, rather than any individual. That being said, anytrust still correctly models the protection guarantees for the privacy properties PRSONA supplies. PRSONA, unlike most other reputation systems (aside from SMC-based systems) enables multiple voter-conscious reputation functions (Short-Term Memory Consensus and Long-Term Memory Consensus).

3.6.2 Public Log Approaches

As mentioned above, public log ticket-based reputation systems have largely followed the rise of interest in blockchain technologies. As such, they carry a unique set of concerns from other systems. While TTP ticket-based reputation systems largely assume their TTPs have sufficient incentive to provide their services correctly and in a trustworthy manner, such a claim would naturally need greater elaboration when dealing with a set of blockchain miners. How systems consider their miners is only one important way they can vary. Their variance also comes from issues in common with TTP approaches, such as how tickets are formed.

Beaver [SKCD16] is directly associated with commerce settings. Votees' reputations are tied to the items they sell, and can choose whether to link these items together through NIZKPoKs of private keys associated with the other items. Voters are granted privacy in their evaluations through linkable ring signatures (as was done in AnonRep) across all public keys associated with a transaction for an item; anyone can vote once if they have participated in a transaction, but voting multiple times will implicate a voter. Voters are encouraged to

generate new keypairs for each transaction, but may use NIZKPoKs of values committed in previous reviews to link them together if they so desire.

Schaub *et al.* [SBHB16] use blinded tokens transferred during a transaction for voters to give a rating for votees. Voters wait for others to transact with a votee to enlarge their anonymity set, but how voters determine that other users have transacted with a votee after them is not obvious. The authors also propose a mechanic in which the currency associated with their system will be used by the votees themselves to generate tokens, so that they cannot issue arbitrary new tokens.

PrivBox [ABH18] only requires a public bulletin board as opposed to a full blockchain. While the bulletin board can of course be implemented via a blockchain, this work does not focus on that aspect. Users are given tokens of an unspecified form in order to give their feedback, which comes in the form of an encrypted 1 or 0. When every vote is combined, the blinding factors involved in each cancel out, and brute search can reveal the score from 0 to a maximum of the total number of votes.

ARS-PS [LAN+19] makes open use of a TTP, in order to help prevent Sybil attacks. The TTP is responsible for making sure users are only able to join the system once, and can be used later on to identify misbehaving users (although misbehaviour is detectable without identification). ARS-PS also employs an alternative underpinning for its blockchain relative to the other works in this approach. Namely, it relies on Proof of Stake instead of Proof of Work. The miners are votees, and their stake in the consensus protocol is directly tied to their own reputation scores. Voters submit votes homomorphically encrypted in such a manner that only all of a set of votees working together could decrypt it. That set adds all votes for a votee together, then decrypts the sum, and publishes it, using that to determine the stake for the next epoch in the blockchain.

3.7 Tradeoffs between Approaches

Naturally, different approaches have different strengths and weaknesses. When creating new reputation systems, discerning system designers should weigh their options with a specific mind for the goals of their particular system. We highlight some of the most notable of these tradeoffs with the aim of making more clear why certain approaches may be more or less desirable.

Coin-based systems were the first to implement Reputation-Usage Unlinkability and Exact Reputation Blinding. Though designers may desire these properties, they will find great difficulty in implementing many reputation functions with the repcoins inherent to coin-based

systems, particularly due to the difficulty in implementing negative coins. A similar problem occurs for signature-based reputation systems; due to their design around specific novel signature schemes, rescinding votes seems difficult to implement. In both cases, complications arise in implementation when designers desire votes to contain more nuanced information than merely affirming a positive (or negative) interaction.

Both signature-based and SMC-based reputation systems have a considerable amount of decentralization inherent to their architecture. In both, this comes with a cost that reputation functions must be relatively simple. Due to SMC-based systems' approach involving soliciting other users for their feedback for a votee directly, it would be difficult to receive correct feedback without divulging which votee is being inquired about. Thus, SMC-based systems have considerable difficulty providing votee privacy properties like Reputation-Usage Unlinkability and Exact Reputation Blinding. SMC-based systems also generally require that voters always be online to give their feedback, which is not necessarily the case for other approaches, and though improvements have been made over time, SMC still often involves significant calculation overheads.

Ticket-based reputation systems feature the most variety, due to the large amount of work done in the approach. Similar to SMC-based reputation systems, public log approaches to ticket-based systems have difficulty providing votee privacy properties; knowing who feedback is intended for is difficult in their public setting without direct identification. Public log approaches do feature significant decentralization. However, that decentralization is subject to familiar concerns around hijacking consensus in the blockchains used. Public log approaches also may have difficulty implementing voter-conscious reputation functions without sacrificing Voter-Vote or Two-Vote Unlinkability. SMC-based systems implement these functions without this sacrifice by asking voters to jointly calculate these functions before individual votes reach the requester. Where these functions have been implemented elsewhere, avoiding that sacrifice is typically accomplished by carefully relying on TTPs to perform the calculations.

This reliance on TTPs is the main drawback with TTP ticket-based systems. Though a significant variety of privacy properties and reputation functions is possible with ticket-based approaches, this has often been accomplished with increased reliance on TTPs. Where centralization is a concern, it may be difficult to justify using TTP approaches despite the breadth of privacy properties and reputation functions available.

3.8 AnonRep

This work is significantly influenced by AnonRep by Zhai *et al.* [ZWC+16]. As mentioned above, AnonRep is designed for use in forum settings, where users may make posts and tag said posts with their reputation. In particular, it was AnonRep that first incorporated mix-nets into a reputation system, in order to allow users to trust that their anonymity is preserved so long as any one server is honest. Using these mix-nets, AnonRep proceeds in epochs; within an epoch, users may post and vote under one pseudonym. At each epoch boundary, AnonRep undergoes a mix-net operation. The end result is that each user gets a new pseudonym, that they are still able to prove ownership of, and which is still associated with the same reputation as in the previous epoch. We also follow a similar approach, though we note that limitations in AnonRep's analysis mean that it underestimates the amount of trust users would need to place in their system.

In particular, AnonRep's basic construction does not provide Exact Reputation Blinding. As such, implementing its basic construction would likely not actually give users anonymity at all — in the worst case, one outlier user with unusually high or low reputation would always be clearly visible, and could be tracked by their reputation even across the generation of new pseudonyms. More generally, given that reputation can only shift so much between rounds, this property greatly limits their anonymity sets, and over several epochs, would likely identify them. Zhai *et al.* do provide a security-enhanced design, in which they attempt to address this shortcoming. Their solution consists of homomorphically encrypting scores, accomplishing two things. First, scores are encrypted, so that users cannot be tracked by their reputations across epochs. Second, scores are homomorphic, so feedback can still be provided (by adding or subtracting from this score).

Though their work is light on the details of how this is actually accomplished, up to and including basic details such as to whom scores are homomorphically encrypted, it is possible to work out a functioning system that matches what they broadly describe. This solution addresses the Exact Reputation Blinding issue, and we do something similar in our approach. However, there are other shortcomings that they do not address in full. In particular, users get no assurances that, once they have voted, it will be counted correctly, if at all. If there are errors with their score, AnonRep gestures at a blame protocol users can enter into in order to try to blame a server for tampering with their score. They acknowledge that a dishonest witness to this blame protocol may not actually acknowledge such an error. It is unclear, however, what happens if a dishonest server or set of servers choose not to respect an error acknowledged by another server. In our own approach, a robust system of zero-knowledge proofs is able to clearly establish blame and avoid this ambiguity.

In terms of more dramatic differences, the biggest lies in how users vote for one another.

There are two major differences here. First, to enable voting, AnonRep uses linkable ring signatures. These structures allow users to cast one legitimate vote for any post with an anonymity set of the entire group, but casting more votes would cause them to incriminate themselves. One might think of this as masking who had cast a vote, but not what their vote was. As such, depending on how exactly the homomorphically encrypted votes are structured, it may be possible for the servers to identify the amounts by which a user's reputation will change in any given epoch. Our approach, on the other hand, works in an opposite manner. Users cast votes associated with their new pseudonym (giving the same anonymity properties as posting a message), but which are encrypted in such a manner that users are able to prove that the votes are valid without revealing their exact contents to the server. One might think of this as masking what a vote is, but not who cast it (at least, within an epoch). This also takes away the ability for servers to track shifts in users' reputations.

The second major difference in voting between AnonRep and PRSONA involves reputation functions. AnonRep uses Accrue Stars as its reputation function. User reputations may go up or down without limitation. This is particularly why it is important for users to not be able to vote multiple times for the same post, and thus why they need linkable ring signatures in voting. PRSONA, however, uses Short-Term Memory Consensus — Iterated Weighting. As in Short-Term Memory Consensus, when a user gives a vote, they are in fact only updating their own previous vote (sometimes, with the same value), and thus scores are more limited. Further, a user's vote is weighted by their own score, so users that have a higher reputation are considered more trusted and are afforded a higher influence on the scores of others. These two factors have several impacts on reputation scores, which in turn may impact user behaviour. Unlike Accrue Stars, users in Short-Term Memory Consensus cannot bank goodwill by making high reputation posts and then abusing their high reputation without fear of repercussion. Negative reputation is much quicker to catch up with them. Additionally, in reputation extremes, Accrue Stars begins to lose clear meaning. When the average reputation is 50, what is the difference between 100 and 1000? Both are *very* large. Short-Term Memory Consensus does not face this problem; at any time, a score can clearly be correlated to the average opinion of other users in the system of a given user. Further, the weighting used in Short-Term Memory Consensus — Iterated Weighting specifically makes it more difficult for small, unpopular cliques to illicitly drive each other's scores up. Without positive feedback from outside the clique, the individual weights assigned to their votes will be low, and they will unable to increase their scores very far. The choice of different reputation functions is a particular driver of significant difference in design between AnonRep and PRSONA.

Chapter 4

Design

In this chapter, we lay out an understanding and rationale for our approach to designing PRSONA. As mentioned previously, we draw inspiration in part from a previous system, AnonRep [ZWC+16], but our design is novel, due to different goals we sought to achieve.

PRSONA is designed for communities where it is expected that any member could reasonably have interactions with and form opinions of every other member, a property we term "tight-knit". This property is related to Dunbar's number [Dun92], the number of people with whom one person can maintain stable social relationships; recent estimates for this number range up to about 500 [LWL21]. Tight-knittedness is directly modeled by the reputation function that PRSONA uses, Short Term Memory Consensus — Iterated Weighting; every voter provides exactly one input on every votee. We further expect that these communities may periodically have new members join, with the only method available to vet new members being participation in the community. The role Short Term Memory Consensus — Iterated Weighting plays in PRSONA's design to help tight-knit communities abate disruptive behaviour while being able to accomodate new members is discussed further in Section 4.4.

PRSONA is specifically intended for use in forum-like settings, where members of the underlying community interact regularly and form opinions of one another based on those interactions. The desire of PRSONA is to enable the ability to participate in such a setting anonymously, without jeopardizing the community's ability to jointly uphold and enforce a standard of behaviour. That being said, though PRSONA enables anonymous participation, it is reasonable to expect that users may still choose to participate pseudonymously; in such a case, PRSONA grants such users the ability to participate under multiple pseudonyms, while maintaining the integrity of the community's decisions about the appropriateness of said user's behaviour across all pseudonyms.

Table 4.1. PRSONA Attributes

Name	*Year*	*Centralization*	*Directionality*	*Rep. Scope*	*Rep. Ownership*	*Correctness*	*Untinkable to TTP*	*# TTP for Setup*	*# TTP Ongoing*	*More via Anytrust*	*V-V Unlinkability*	*2V Unlinkability*	*R-U Unlinkability*	*Exact Rep.*	*Blinding*	*Acc. Stars*	*Acc. Stars — Neg.*	*Avg. Stars*	*Gompertz*	*STMC*	*LTMC*
System		**Structure**				**Trust**					**Privacy**					**Rep.**					
PRSONA [this work]	2021	★	↔	∀	◗	●	●	2	2	●	●	●	●	●		-	-	-	-	●	●

Reputation Attributes

Centralization:	★ = Third-Party Mediation	☍ = Ephemeral Mesh Topology	∴ = Proofs of Validity
Directionality:	→ = Simplex	⇌ = Half-Duplex	↔ = Full-Duplex
Scope:	∀ = Global	∃ = Local	
Ownership:	⛛ = Votee-owned	◗ = TTP-owned	⊙ = Voter-owned
Correctness: via...	● = ...protocol guarantees	◐ = ...errors are traceable	- = ...TTP/miners
Trust Unlinkability: TTP can link...	● = ...nothing	◐ = ...misbehaviour	- = ...everything
Privacy Unlinkability:	◐ = Participants to a transaction can link each other		
Reputation:	* = This work considers reputation functions to be outside its scope.		

In Table 4.1, we reproduce the attributes of PRSONA that we previously discussed in Table 3.2. With respect to architecture, PRSONA's structure is broadly similar to other TTP ticket-based reputation systems, although it features key differences in terms of trust assumption. PRSONA provides all four of our recognized privacy properties, and in terms of reputation function, focuses on voter-conscious reputation functions. Our implementation specifically fulfills Short-Term Memory Consensus, but with only minor modification, it could alternatively support Long-Term Memory Consensus. In the following sections, we provide further detail on these features.

4.1 Architecture

In the first two sets of feature columns in Table 4.1, we identify features of PRSONA's structure and trust assumptions. Like all other TTP ticket-based reputation systems, PRSONA uses Third Party Mediation and the scope of its reputation is global. Further, like many such systems, it is full-duplex, and the TTP holds the reputation scores for requesters to view.

Where PRSONA differs more drastically from previous work is in its trust assumptions. In order to provide the security guarantees we desire (which are detailed further in Section 4.3), PRSONA relies on a multi-provider model. That is, at least two non-colluding servers must be operating for PRSONA to fulfill its privacy guarantees, which also include guarantees against the servers learning certain information. More servers can be added in an anytrust capacity to fulfill the privacy guarantees of the system, although a majority must behave honestly for

correctness to hold. These capabilities, along with those of user nodes, are detailed further in Section 4.2.

As a general overview, user nodes are completely untrusted. In different configurations of the protocol, server nodes behave in one of two settings (which are elaborated upon further in Section 4.2): covert or honest-but-curious. In a typical group, user nodes greatly outnumber server nodes. Server nodes are expected to be highly available and well-provisioned, while user nodes do not have such expectations.

Our assumption that server nodes do not collude mirrors that of AnonRep [ZWC+16]. That naturally implies that servers should be operated independently. Given the forum model that PRSONA is tailored to, as well as the idea that PRSONA is intended for use by communities, it would make particular sense for these servers to be operated by various stakeholders within said community. Such distribution of power across members allows for greater trust in the governance of said community and/or forum.

Users may communicate with as few as one server, if they so choose. All business they need to conduct to track their own reputation, to make and give votes, and to verify the reputations of other users, can be done while only interacting with one server. However, they are able to select their level of trust for the servers; users may query other servers to confirm the correctness of an (encrypted) value given by any individual server. Though servers could give invalid values (i.e., things that do not decrypt to anything), given a majority of honest servers, they would be caught doing so.

PRSONA proceeds in epochs, where each epoch represents some unit of time in which users may participate under a single identity. Servers need to be available and online at all times; either to other servers, when performing calculations between epochs to maintain user scores while generating new identities for them, or to users during epochs, to accept new votes and provide information as necessary. Users, however, do not need to be constantly online; they are able to check in as they desire, and re-establish a valid state of participation, no matter how many epochs they miss.

4.2 Threat Model

In considering potential designs for PRSONA, it is important to have an understanding of the threat we wish to protect against. To that end, we discuss the capabilities of users and servers for behaviour that may contravene our security goals.

We allow our users to behave however they wish. They may collude with one another; when they do so, the system is responsible to not leak any information beyond what colluding

users can conclude from their inputs and the output. Users may craft whatever sets of votes and reputation proofs that they like. There are no limitations on user behaviour, beyond the fact that servers and users will reject malformed votes and reputation proofs.

As for our servers, we consider two cases. First, in an honest-but-curious setting, we allow servers to investigate user input as much as they like, so long as they honestly follow the protocols of the system. This places a great deal of trust in our servers, but this trust comes with greater speed and efficiency of server operations. Second, in the covert setting, we allow servers to behave however they like, so long as they do not do anything that would cause them to be caught deviating from the protocol with non-negligible probability. Servers are barred from engaging in denial of service of the system they help to run, and if a malicious action they might take would be detectable by any other server or client in a way that could be traced to the server, said server will not engage in such an act. Beyond that, in this setting, servers are allowed to act as they will.

As a note for that covert setting, there are still limitations on how many servers may collude. By necessity, we assume that at any time, a majority of servers are honest. This assumes more than some previous work, where anytrust was the model at hand. As a counterpoint, though, our system does successfully protect user anonymity in an anytrust model — so long as one server is honest, anonymity is always protected. However, the accuracy of scores can only be guaranteed in the covert setting as long as a majority of servers behaves honestly.

Though we separate out these two cases in our analysis, design, and implementation, there is a method to take a hybrid approach between the covert and honest-but-curious settings. This hybrid approach provides security against a covert adversary, while only requiring an online cost in line with the honest-but-curious setting (along with an offline cost that is still in line with the covert setting). This hybrid approach is examined in greater detail in Section 5.3.

Additionally, although we seek to provide the above privacy properties for our users, even against our servers, there is still some leakage to the servers that is worth disclosing. Servers are allowed to learn the distribution of reputation scores in the system. They are not able to learn which user has what score, in terms of any pseudonym that they ever use in the system, but they do know which scores exist during any particular epoch. They also are allowed to learn which users vote during any given epoch. In certain extreme situations, this may combine in such a manner that servers know which score a voter affected, but again, they are not able to deduce which votee that voter voted for.

We feel that the covert setting, despite its disadvantages in speed compared to the honest-but-curious setting, represents a realistic setting for deployment, given the settings in which we foresee PRSONA being used.

4.3 Security Goals

Anonymity. The main goal of PRSONA is to, as widely as possible, allow a user to maintain ongoing participation in the system without having their actions linked. To that end, as discussed in Section 2.3, we seek to provide Voter-Vote Unlinkability, Two-Vote Unlinkability, Reputation-Usage Unlinkability, and Exact Reputation Blinding for our users from all other entities, server or otherwise.

Specifically, Reputation-Usage Unlinkability and Two-Vote Unlinkability applies between epochs; a user who participates in the system multiple times in one epoch will be identified as the same user each time within that epoch. However, those participations will not be linked in any way to any participation by that same user in any other epoch, no matter how many times they participated in either one. Voter-Vote Unlinkability and Exact Reputation Blinding always apply.

Users may choose to give up these properties for themselves, if they wish. A user is always capable of linking themselves between pseudonyms (such as by using the same external signature for both). In the same manner, a user is always capable of directly revealing their exact reputation score. However, no user has technological means to compel another user to link themselves or reveal their score. Social means to do so are out of scope for PRSONA.

Server misbehaviour detection. PRSONA intends to, as much as possible, make any server misbehaviour immediately detectable, as a means to dissuade servers from misbehaving. Put another way, given our covert security setting, whatever misbehaviour is not outright impossible given the system, will at least be detected so that misbehaving servers can be blamed.

Non-goals. PRSONA does not make any attempt to protect against Sybil attacks directly. This is considered out of scope for PRSONA; in order to participate, it is assumed that there is some mechanism to ensure that any given user is only registered in PRSONA once. PRSONA does not provide this mechanism, however, and such mechanisms are their own robust area of study. Further, PRSONA does not make any effort to prevent network-level Denial-of-Service attacks. Though it is not necessary for users to be constantly online and will do little more than frustrate users, issues can arise when servers are not online, and an attacker could specifically target servers to frustrate the system. This too is its own robust area of study, and many defences [ADT04, KN11, KLLA15, SBZD15, HK17, SBZD17, ZGD19] that have already been designed for such attacks could be deployed in concert with PRSONA to mitigate this risk.

4.4 Reputation Function

Keeping in mind PRSONA's intended use in forum-like settings, reputation in this setting can mean a variety of things. It could be used to measure popularity or adherence to virtues that the underlying community wishes to promote. PRSONA's design specifically intends reputation to represent a user's fit and behaviour within a community. A worse reputation should mean that that user misbehaves or is otherwise making themselves unwelcome in the community, while a better reputation should mean that a user is an upstanding examplar of the behaviour a community wishes to see. This is not the same as popularity; in an ideal setting, a person of modest popularity who behaves agreeably should have a higher reputation than a popular person who behaves in a polarizing manner. Whether this occurs in practice is dependent on user voting patterns.

This intention, however, serves the following purpose. As users find themselves dissatisfied with the behaviour of an individual, said individual will receive feedback that allows them to course-correct, rather than being castigated with no ability to learn from their mistakes. Due to the ability to participate as multiple pseudonyms, said individual might step into a new epoch under a new identity, and cast off the mistakes they made in the past. This would allow them to easily reset to a neutral or positive reputation in a short amount of time. This swift reset also means that even individuals with high reputation will quickly receive feedback about misbehaviour, and will not merely "coast" on the good graces of their past performances.

It is expected that users, or even servers, may choose a threshold reputation level, below which users face restrictions or an inability to participate. Significant enough misbehaviour, as recognized by a great enough number of users, would thus render an individual either in a purgatory state, or cast out of the community. This is intended specifically for the cases of trolling and abuse; an individual who comes to a community intending only to disrupt it must themselves face consequences, or else their mission will succeed. This is, after all, the entire point of using reputation in this setting. Choosing an appropriate threshold is a delicate manner, though; care must be given to choose it high enough that a truly disruptive person can consistently face punishment, but low enough that a temporarily misbehaved person can correct for their behaviour and rejoin the community proper.

It is natural that users participate as peers, equally able to rate one another. That is, PRSONA is full-duplex; anyone who participates in the system is capable of rating anyone else who participates. Servers do not participate in voting, either as voters or votees, although in practice the stakeholders who run servers may very well also participate separately as users.

In order to generate reputation scores in line with our expectations (that users cannot "coast" on previous good behaviour, and users can correct for misbehaviour relatively quickly),

we chose to use Short-Term Memory Consensus — Iterated Weighting as the reputation function for PRSONA. As discussed in Section 2.4.2, in Short-Term Memory Consensus, each voter gets one vote for every other user, modeling the tight-knit communities PRSONA is designed for. Voters may update their votes at any time, such that multiple votes for one votee merely replace each other, rather than stacking. Incidentally, this removes ballot-stuffing attacks entirely from the set of concerns PRSONA has to guard against. Due to this replacement of old votes, a votee can experience relatively quick swings in their score when several voters switch their votes from positive to negative (or vice versa). The "Weighting" portion of this function means that users who own high reputations are able to influence the reputations of other users more easily. It is expected that users who themselves behave well may be better judges of what constitutes good behaviour, while the opinions of misbehaving users may be less relevant. Finally, the "Iterated" portion means that the weighting does not progress by fully executing the PageRank algorithm every epoch on user scores, as is done in Short Term Memory Consensus — Weighted. Instead, an approximation is used; in each epoch, votes are weighted by the voters' current reputations. Over several epochs, this behaves the same as Short-Term Memory Consensus — Weighted. On the other hand, it also makes concessions to efficiency, as conducting the PageRank algorithm on encrypted data in our setting would prove challenging very quickly.

4.5 Cryptographic Tools

In order to accomplish its goals, PRSONA uses ElGamal and BGN encryptions in various places. For efficiency reasons, our implementation of PRSONA uses a prime-order version of BGN, and more detail on the exact libraries we build upon to do so is provided in Chapter 5. The differences between prime-order BGN and composite-order BGN also have minor implications for the designs of various zero knowledge proofs (ZKPs) that PRSONA uses.

4.5.1 ElGamal

The ElGamal cryptosystem was first proposed by Elgamal in 1985 [Elg85]. In PRSONA, we use a scheme containing two variations from the originally proposed design. The first change we make is to put our message in the exponent of the group, so that the message can be an integer rather than a group member. The second is less common: in our version of the scheme, ciphertexts swap where a user's public key (X) and the group's generator ($\mathfrak{g}$) are used from the more typical ElGamal construction. This is necessary to support the way that

PRSONA prevents servers from learning the linkage of a user's identity over time. More detail on this prevention is provided in Section 4.10.3. The full construction is as follows:

> **EGGroupKeyGen(1^λ):** Let $\mathscr{G}$ be a cyclic group generator. Compute $(\mathfrak{G}, \mathfrak{g}, q) \leftarrow \mathscr{G}(1^\lambda)$, where $\mathfrak{g}$ generates $\mathfrak{G}$, a cyclic group of order q. Choose $\mathfrak{h} \xleftarrow{\$} \mathfrak{G}$. Output the shared elements $E = (\mathfrak{G}, \mathfrak{g}, \mathfrak{h}, q)$.
>
> **EGUserKeyGen(E):** Choose $x \xleftarrow{\$} [1, q-1]$ and compute $X = \mathfrak{g}^x$. Output the public key $PK = (E, X)$ and the secret key $SK = x$.
>
> **EGEncrypt(PK, m):** Choose $r \xleftarrow{\$} [1, q-1]$. Output the ciphertext $C = (C_1, C_2) = (X^r, \mathfrak{g}^r \cdot \mathfrak{h}^m) \in \mathfrak{G}^2$.
>
> **EGDecrypt(SK, C):** Compute $y \in [1, q-1]$ such that $y = x^{-1} \bmod q$. Note that, as $C_1 = X^r$, $C_1^y = \mathfrak{g}^r$. Output $m = \log_{\mathfrak{h}}(\frac{C_2}{C_1^y})$.

Although **EGDecrypt(SK, C)** involves calculating a discrete log, the range of values in use in PRSONA that would be encrypted in this manner is very limited (falling within $[0, 2n]$, for n the number of users in the system). With such a small range of values, brute forcing this discrete log is not challenging.

In PRSONA, we additionally make one further, minor change from the approach outlined above. Instead of one constant $\mathfrak{g}$ used in perpetuity in the system, the PRSONA servers periodically calculate a new $\mathfrak{g}_t$, in such a way that all user public key Xs are updated as well (denoted $X_t = \mathfrak{g}_t^x$). This is detailed further in Section 4.10.

4.5.2 Prime-order BGN

While the above variant of ElGamal is additively homomorphic, our application additionally requires (depth-1) multiplication. This additional property is provided by the BGN [BGN05] cryptosystem. The prime-order BGN construction we use was first suggested by Freeman [Fre10]. First, we quote Freeman's definition of a bilinear group generator (Freeman's Definition 2.1 [Fre10, p. 48]):

> A *bilinear group generator* is an algorithm $\mathscr{G}$ that takes as input a security parameter λ and outputs a description of five abelian groups G, G_1, H, H_1, and G_T, with $G_1 \subset G$ and $H_1 \subset H$. We assume that this description permits efficient (*i.e.*, polynomial time in λ) group operations and random sampling in each group. The algorithm also outputs an efficiently computable map (or "pairing") $e : G \times H \to G_T$ that is

- Bilinear: $e(g_1 g_2, h_1 h_2) = e(g_1, h_1) e(g_1, h_2) e(g_2, h_1) e(g_2, h_2)$ for all $g_1, g_2 \in G$ and $h_1, h_2 \in H$; and
- Nondegenerate: for any $g \in G$, if $e(g, h) = 1$ for all $h \in H$, then $g = 1$ (and similarly with G, H reversed).

Note that, in the above definition, G and H are not assumed to be prime order (in fact, they could not be, as described). This will be addressed shortly.

Next, we quote Freeman's definition of a projecting bilinear group generator (Freeman's Definition 3.1 [Fre10, p. 52]):

> Let $\mathscr{G}$ be a bilinear group generator. We say that $\mathscr{G}$ is *projecting* if it also outputs a group $G_T' \subset G_T$ and three group homomorphisms π_1, π_2, and π_T mapping G, H, and G_T to themselves such that
>
> 1. G_1, H_1, G_T' are contained in the kernels of π_1, π_2, π_T, respectively, and
> 2. $e(\pi_1(g), \pi_2(h)) = \pi_T(e(g, h))$ for all $g \in G$, $h \in H$.

The original BGN cryptosystem [BGN05] formed the hidden subgroups $G_1 \subset G$ and $H_1 \subset H$ by having $G = H$ be an elliptic curve group with a symmetric pairing, and whose order is an RSA number $N = p_1 p_2$, and $G_1 = H_1$ are the order-p_1 subgroups. (The factorization of N is part of the private key of the scheme.) The projections π_1 and π_2 are then exponentiations by p_1. Unfortunately, this requires the order of the group to be an RSA-sized integer (so thousands of bits for 128-bit security), which makes ciphertexts large and slow to process. Freeman's adaptation instead uses a standard prime-order asymmetric pairing setup, as can be seen in Freeman's Example 3.3 [Fre10, p. 53]:

> Let $\mathscr{P}$ be a prime-order bilinear group generator. Define $\mathscr{G}_{\mathscr{P}}$ to be a bilinear group generator that on input λ does the following:
>
> 1. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \xleftarrow{\$} \mathscr{P}(1^\lambda)$, and let $G = \mathbb{G}_1^2$, $H = \mathbb{G}_2^2$, $G_T = \mathbb{G}_T^4$.
> 2. Choose generators $g \xleftarrow{\$} \mathbb{G}_1$, $h \xleftarrow{\$} \mathbb{G}_2$, and let $\gamma = \hat{e}(g, h)$.
> 3. Choose random $a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2 \xleftarrow{\$} \mathbb{F}_p$ such that $a_1 d_1 - b_1 c_1 = a_2 d_2 - b_2 c_2 = 1$.

4. Let G_1 be the subgroup of G generated by $g' = (g^{a_1}, g^{b_1})$, let H_1 be the subgroup of H generated by $h' = (h^{a_2}, h^{b_2})$, and let G_T' be the subgroup of G_T generated by

$$\{\gamma^{(a_1a_2, a_1b_2, b_1a_2, b_1b_2)}, \gamma^{(a_1c_2, a_1d_2, b_1c_2, b_1d_2)}, \gamma^{(c_1a_2, d_1b_2, c_1a_2, d_1b_2)}\}.$$

5. Define $e : G \times H \to G_T$ by

$$e((g_1, g_2), (h_1, h_2)) := (\hat{e}(g_1, h_1), \hat{e}(g_1, h_2), \hat{e}(g_2, h_1), \hat{e}(g_2, h_2)).$$

6. Let $A = \begin{pmatrix} -b_1c_1 & -b_1d_1 \\ a_1c_1 & a_1d_1 \end{pmatrix}$, $B = \begin{pmatrix} -b_2c_2 & -b_2d_2 \\ a_2c_2 & a_2d_2 \end{pmatrix}$, and define

$$\pi_1((g_1, g_2)) := (g_1, g_2)^A = (g_1^{-b_1c_1} g_2^{a_1c_1}, g_1^{-b_1d_1} g_2^{a_1d_1})$$

$$\pi_2((h_1, h_2)) := (h_1, h_2)^B = (h_1^{-b_2c_2} h_2^{a_2c_2}, h_1^{-b_2d_2} h_2^{a_2d_2})$$

$$\pi_T((\gamma_1, \gamma_2, \gamma_3, \gamma_4)) := (\gamma_1, \gamma_2, \gamma_3, \gamma_4)^{A \otimes B}$$

7. Output the tuple $(G, G_1, H, H_1, G_T, G_T', e, \pi_1, \pi_2, \pi_T)$.

Due to the prime-order assymetric pairing setup described above, the ciphertexts in use are smaller and faster to operate on, while still admitting depth-1 multiplication, as can be seen in Freeman's full construction (the names of the functions have been altered to specify they are for BGN, and the outputs of the encryption function has been altered to include details used in our algorithms, but these are otherwise from Freeman's Section 5 [Fre10, p. 57]):

BGNKeyGen(1^λ): Compute $(G, G_1, H, H_1, G_T, G_T', e, \pi_1, \pi_2, \pi_T) \leftarrow \mathcal{G}_{\mathcal{P}}(1^\lambda)$. Choose $g \xleftarrow{\$} G, h \xleftarrow{\$} H$, and output the public key $PK = (G, G_1, H, H_1, G_T, e, g, h)$ and the secret key $SK = (\pi_1, \pi_2, \pi_T)$.

BGNEncrypt(PK, m): Choose $r_1 \xleftarrow{\$} [1, |G_1| - 1]$ and $r_2 \xleftarrow{\$} [1, |G_1| - 1]$. Compute $g_1 = g'^{r_1}$ and $h_1 = h'^{r_2}$. (Recall that G_1 and H_1 are defined by the respective generators $g' = (g^{a_1}, g^{b_1})$ and $h' = (h^{a_2}, h^{b_2})$.) Output the ciphertext $(C_A, C_B) = (g^m \cdot g_1, h^m \cdot h_1) \in G \times H$.

BGNMultiply(PK, C_A, C_B): This algorithm takes as inputs two ciphertexts $C_A \in G$ and $C_B \in H$. Choose $g_1 \xleftarrow{\$} G_1$ and $h_1 \xleftarrow{\$} H_1$, and output $C = e(C_A, C_B) \cdot e(g, h_1) \cdot e(g_1, h) \in G_T$.

BGNAdd(PK, C, C'): This algorithm takes as input two ciphertexts C, C' in one of G, H, or G_T. Choose $g_1 \xleftarrow{\$} G_1$ and $h_1 \xleftarrow{\$} H_1$, and do the following:

1. If $C, C' \in G$, output $C \cdot C' \cdot g_1 \in G$.
2. If $C, C' \in H$, output $C \cdot C' \cdot g_2 \in H$.
3. If $C, C' \in G_T$, output $C \cdot C' \cdot e(g, h_1) \cdot e(g_1, h) \in G_T$.

BGNDecrypt(SK, C): The input ciphertext can be an element of G, H, or G_T.

1. If $C \in G$, output $m \leftarrow \log_{\pi_1(g)}(\pi_1(C))$.
2. If $C \in H$, output $m \leftarrow \log_{\pi_2(h)}(\pi_2(C))$.
3. If $C \in G_T$, output $m \leftarrow \log_{\pi_T(e(g,h))}(\pi_T(C))$.

We further define two derivative functions that are helpful for our purposes:

BGNEncrypt$_G$(PK, m): Compute and output only the first component $C_A \in G$ of **BGNEncrypt(PK, m)**.

BGNEncrypt$_H$(PK, m): Compute and output only the second component $C_B \in H$ of **BGNEncrypt(PK, m)**.

Although **BGNDecrypt(SK, C)** involves calculating a discrete log, the range of values in use in PRSONA that would be encrypted in this manner is limited (albeit a different range from **EGDecrypt()**; these values fall within $[0, 4n^2]$, for n the number of users in the system). Though this range is larger than that of **EGDecrypt()**, brute forcing this discrete log is still within reason. A discrete log of this range can be calculated in time $O(n)$ with the Pollard kangaroo method [Pol78].

We assume that it is possible to distribute the group homomorphisms π_1, π_2, and π_T such that multiple entities must jointly calculate them. (As defined in Freeman's Example 3.3 above, these homomorphisms involve raising group elements to specific combinations of integer factors. By distributing c_1, c_2, d_1, and d_2 (or similar) across multiple parties, this is straightforward to calculate in a distributed manner.)

Thorough details on the prime-order pairings $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \xleftarrow{\$} \mathscr{P}(1^\lambda)$ used in this work can be found in the discussion by Naehrig et al. [NNS10]. The implementation of PRSONA uses the documented Barreto-Naehrig curves and pairing of this work directly. Barreto-Naehrig curves are pairing-friendly elliptic curves generated by a specific method devised by Barreto and Naehrig [BN06].

4.6 Data Types

During setup, the PRSONA servers run **EGGroupKeyGen(1^λ)** and **BGNKeyGen(1^λ)** to obtain and make public $(\mathfrak{G}, \mathfrak{g}, \mathfrak{h}, q)$ and $(G, G_1, H, H_1, G_T, e, g, h)$, respectively. They simultaneously distribute (π_1, π_2, π_T) to each other such that a threshold of servers must cooperate to execute **BGNDecrypt()**. (In practice, this means sharing a_1, b_1, a_2, and b_2 directly, and distributing additive shares of c_1, c_2, d_1 and d_2, as well as additive shares of the products c_1c_2, c_1d_2, d_1c_2, and d_1d_2, across servers. With this setup, decryption reduces to a secure sum operation carried out by all the servers together.) In our instantiation, we require said threshold to be all servers, to maintain our privacy guarantees.

PRSONA operates in rounds or *epochs*. The PRSONA servers collaboratively maintain an **Epoch Generator** (that is, the group generator corresponding to the epoch) $\mathfrak{g}_t \in \mathfrak{G}$, where t represents the current epoch. The PRSONA servers collaboratively alter $\mathfrak{g}_t$ once at the beginning of each epoch. When at least one server behaves honestly (altering $\mathfrak{g}_t$ randomly), the output $\mathfrak{g}_t$ itself is random.

PRSONA users have a variety of data that corresponds to them during their participation in the system, some of which they hold, and some of which is held by the servers. A brief overview of this data follows; how it is actually maintained and used is elaborated in Section 4.7:

- **Long-Term Secret Key:** When users register in the system, they must choose a long-term secret key $x \xleftarrow{\$} [1, q-1]$.
- **Fresh Pseudonym:** At all times, users have a pseudonym $X_t = \mathfrak{g}_t^x$ corresponding to the epoch t.
- **Plaintext Votes:** Users are enabled to cast votes for other users, reflecting the voter's opinion of the votee. These votes, when in plaintext, must be one of the following values: $\{0, 1, 2\}$ (which is intended to represent {"negative", "neutral", "positive"}, respectively). By default, the system assumes a user casts a vote of 1 for each other user, until they update that particular vote.
- **Ciphertext Votes:** PRSONA servers are only given individual votes as ciphertexts, outputs of **BGNEncrypt$_H$()**. Once encrypted, not even a voter can decrypt the ciphertext vote; only a threshold of servers cooperating with one another could decrypt them.
- **Vote Matrix:** Ciphertext votes are stored in parallel by each PRSONA server. The matrix is n-by-n, for n users in the system. Each row represents all the votes cast by one

voter, and each column represents all the votes received by one votee. When changing between epochs, this vote matrix is permuted randomly.

- **Vote Rows:** Each user can request their BGN-encrypted vote row out of the vote matrix. When changing between epochs, the order of votes in each row is permuted randomly, and each vote is rerandomized. This has the effect that users will not know which encrypted votes in the vote row correspond to which prior votes they made, nor even the contents of any of those votes. Users will, however, still know the current fresh pseudonym corresponding to each encrypted vote. Users may request their vote row specifically to update their votes; by rerandomizing each ciphertext vote a voter does not intend to update (*i.e.*, encrypting 0 and adding it to the ciphertext vote), PRSONA servers are prevented from knowing which and how many votes were updated.

- **Server-Encrypted Reputation List:** In order to calculate Short-Term Memory Consensus — Iterated Weighting, the previously calculated reputation score for each user is required. This is stored by the PRSONA servers when changing between epochs as the output of $\textbf{BGNEncrypt}_G\textbf{()}$. Specifically, PRSONA servers are only intended to store ciphertext reputation scores (they can observe plaintext scores, but cannot associate them with any users in any form, fresh pseudonym or otherwise).

- **User-Encrypted Reputation List:** In order for users to know and be able to form proofs based on their own reputation scores, they have to have access to them in some form. This is stored by the PRSONA servers when changing between epochs as the output of **EGEncrypt()**. PRSONA servers store each user's reputation score encrypted to said user's new fresh pseudonym for the upcoming epoch; as the epoch generator is public knowledge, users can decrypt this ciphertext in the same way they would otherwise decrypt ElGamal ciphertexts.

4.7 Workflow

PRSONA operates in rounds or *epochs*. At the beginning of each epoch, users are assigned a new "fresh pseudonym", which is unlinkable to any previous or future fresh pseudonyms they have held. In practice, this fresh pseudonym is a public key, to which users are able to prove ownership of a corresponding long-term secret key with a straightforward ZKP: $\{(x) : X_t = \mathfrak{g}_t^x\}$.

With this fresh pseudonym, users are able to participate in the forum setting, by both posting messages signed by (and labelled with) their fresh pseudonym and evaluating other

users according to their fresh pseudonyms. If a user posts multiple messages within an epoch, all such messages will be clearly associated with their fresh pseudonym for that epoch. However, messages posted by the same user in different epochs (and thus under different fresh pseudonyms) are not able to be linked, unless the user specifically chooses to link those messages through some external mechanism. If a user votes within an epoch, an adversary can learn that they updated their vote row, but will gain no information on whom they gave a vote for (if anyone), nor what the content of that vote was.

At the end of each epoch, the servers collaboratively recalculate each user's reputation level. In this process, the servers are able to learn the distribution of scores (*i.e.*, what the multiset of all user scores is), but gain no information about which user has which score. At the same time, they also collaboratively generate new fresh pseudonyms for each user, before beginning a new epoch.

Servers are required to always be online, in order to synchronize updates (*e.g.*, for their views of encrypted votes to remain synchronized). Users, however, have no such requirements. A user can go offline and completely ignore epochs as they wish. When they come back online, they are able to asynchronously update their view of the system state, learning their new fresh pseudonym for the current epoch, as well as their current score (if it has changed). Whenever they choose to update any of their votes, they can obtain a list of all current fresh pseudonyms, each paired with an encrypted version of the most recent vote the user had cast for the underlying votee. Users are **not** able to decrypt these encrypted votes. This is to prevent a user being able to track another user and their fresh pseudonyms across epochs by virtue of an outlier vote cast for them. As such, a user's only option when deciding how to handle such votes is either to replace them or to keep (and rerandomize) them.

4.8 User Registration

Naturally, a new user to PRSONA must register with the servers in order to participate. This is done as follows.

Each epoch, as part of generating fresh pseudonyms for each user (explained in more detail in Section 4.10), the servers collaboratively calculate an epoch generator ($\mathfrak{g}_t \in \mathfrak{G}$, corresponding to epoch t) for the group the PRSONA servers generated at setup time via **EGGroupGenKey(**1^λ**)**. In order to register, a user first receives the epoch generator and a signature from all servers that attests to it being correctly created ($\pi_{\mathfrak{g}_t}$). Then, the user i chooses $x_i \xleftarrow{\$} [1, q-1]$ and generates a key pair $\langle X_{t,i} = (\mathfrak{g}_t^{x_i}, x_i)\rangle$. $X_{t,i}$ is user i's fresh pseudonym for epoch t, and x_i is their long-term private key. The client then uploads $X_{t,i}$ to

a randomly selected server, along with the following ZKP: $\{(x_i) : X_{t,i} = \mathfrak{g}_t^{x_i}\}$. This process can be observed in Algorithm 1.

Algorithm 1: createPseudonym

Input: $\mathfrak{g}_t$: The current epoch generator,
$\pi_{\mathfrak{g}_t}$: A proof of correctness for the current epoch generator,
q: The order of the ElGamal group
Output: X_t: A user's current fresh pseudonym,
x: A user's long-term secret key,
π_{X_t}: A proof of correctness for the current fresh pseudonym

if `verifySignature`$(\mathfrak{g}_t, \pi_{\mathfrak{g}_t}) = \perp$ **then**
  return $\perp$;
end
$x \xleftarrow{\$} [1, q-1]$;
$X_t \leftarrow \mathfrak{g}_t^x$;
$\pi_{X_t} \leftarrow$ `generateProof`$(\mathfrak{g}_t, X_t, x, \{(x) : X_t = \mathfrak{g}_t^x\})$;
return X_t, x, π_{X_t};

Said server encrypts default values for the user's votes on all existing users (typically, neutral (1)), all other existing users' votes for said new user (typically, neutral (1)), and the user's initial reputation level. We use 1 for the initial reputation, but any number other than 0 is appropriate. This score will stabilize towards a "correct" value (one that reflects the votes towards the user) quickly, but if all users ever have a reputation of 0, the reputation function will cause all scores to permanently stay at 0. This process of encrypting default values can be observed in Algorithm 2.

These encrypted defaults do not need to be secret. It is expected that a new user will have a default score until a new epoch, and default votes until either a new epoch or they cast their own votes. As such, these encrypted defaults will need to match the form of other encrypted values (that is, they should look like the outputs of $\textbf{BGNEncrypt}_G()$ and $\textbf{BGNEncrypt}_H()$), but do not need to use the random elements typically generated as part of $\textbf{BGNEncrypt}_G()$ and $\textbf{BGNEncrypt}_H()$. Recall that the outputs of $\textbf{BGNEncrypt}_G()$ have form $g^m \cdot g'^{r_1}$ where $r_1 \xleftarrow{\$} [1, |G_1| - 1]$, and the outputs of $\textbf{BGNEncrypt}_H()$ have form $h^m \cdot h'^{r_2}$ where $r_2 \xleftarrow{\$} [1, |H_1| - 1]$. As these values do not need to be secret, and m will always be 1, servers can choose $r_1 = r_2 = 1$, such that a default vote is always $h \cdot h'$, and a default score is always $g \cdot g'$. It is trivial for other entities to verify that a default value matches these forms, and thus confirm that the encrypted values are the default values a server claims. The other servers,

Algorithm 2: acceptNewUser

Input: X_t: A user's current fresh pseudonym,
π_{X_t}: A proof of correctness for the current fresh pseudonym,
PK_{BGN}: The servers' collective BGN public key

Output: voteMatrix: A vote matrix with added default entries for votes by and for the new user,
serverEncryptedScores: All reputation scores, including the new user's default reputation score

Result: A server adds a valid new user. The number of users (n) is updated.

---

if `verifyProof`$(X_t, \pi_{X_t}, \{(x) : X_t = \mathfrak{g}_t^x\}) = \perp$ **then**
  return $\perp$;
end
/* The `obtainGeneratorsFromBGNPK` function here returns the generators of G, G_1, H, and H_1 given the BGN public key as input. */
$g, g', h, h' \leftarrow$ `obtainGeneratorsFromBGNPK`(PK_{BGN});
for $i \leftarrow 1 \ldots n$ **do**
  newVoteRow$[i] \leftarrow h \cdot h'$;
  voteMatrix$[i, n+1] \leftarrow h \cdot h'$;
end
newVoteRow$[n+1] \leftarrow h \cdot h'$;
voteMatrix$[n+1] \leftarrow$ newVoteRow;
serverEncryptedScores$[n+1] \leftarrow g \cdot g'$;
$n \leftarrow n+1$;
return voteMatrix, serverEncryptedScores;

upon seeing the initial proof of a valid fresh pseudonym, and confirming that the votes and score are correct, update their own data stores to add the new user.

A server is not able to cheat and encrypt non-default values for any of the relevant data, or else the other servers will be able to detect it instantly. It is, however, possible for a malicious server to instead choose to ignore a user's request to be added. Such a case is easily detected by a user (who can simply request information from other servers to see if their data stores have been updated with the relevant information for them). A user in such a case would not be able to incriminate said server, but would not lose any privacy through this action, and would be able to try to register again with a new server as they wish.

4.9 User Participation

Once a user is registered, their participation is relatively straightforward. Users participate in PRSONA in two ways: posting (which requires sufficient **Reputation**), and **Voting**.

4.9.1 Reputation

During the calculations that occur between each pair of epochs, a record of a user's reputation score is encrypted to their fresh pseudonym for the epoch. This encrypted score is the output of **EGEncrypt()**, and takes form $(C_1 = X_t^r, C_2 = \mathfrak{g}_t^r \cdot \mathfrak{h}^s)$, for $r \in [1, q-1]$ a random blinding value and s the user's score. They may request this record, and then use it to create a ZKP that their reputation score is above a given threshold: $\{(x,s) : X_t = \mathfrak{g}_t^x \wedge s \in [\lambda, 2n] \wedge C_2 = C_1^{x^{-1}} \cdot \mathfrak{h}^s\}$, for λ the publicly communicated threshold the user is above, and n the number of users. Note that $2n$ is the greatest possible score a user can have in PRSONA. In practice, either λ is chosen such that the size of $[\lambda, 2n]$ is a power of 2, or the proof actually proves that $s \in [\lambda, m]$, for m the smallest integer greater than $2n$ that would make the size of this range a power of 2. In the latter case, it is assumed that s is not larger than $2n$ under the assumption that the servers have followed the protocol honestly to keep s no greater than $2n$. That being said, as $n \ll q$, even if s were greater than $2n$, this would not jeopardize the proof through overflow. The size and complexity of this ZKP is logarithmic in the size of the range between λ and $2n$ (and for fixed λ, this would mean logarithmic in the number of users).

A verifier of this proof would request $(X_t, (C_1, C_2))$ directly from the servers, who would additionally reply with a signature (signed by a threshold of servers) to prove the correctness of the record. Here, generally a majority of servers would be sufficient to prove correctness, rather than all.

In practice, it would likely make sense to have the verifiers of this proof be whichever entities run the forum itself. If a user can prove their reputation is above some satisfactory threshold, their posts can be accepted without any special notation. If the user can only prove their reputation is above a lesser threshold, their posts may be marked as coming from a user with low reputation, and if the user cannot prove their reputation is above even this lesser threshold, their posts may be rejected entirely. Naturally, choosing appropriate values for this satisfactory and lesser threshold is a challenging social endeavour.

4.9.2 Voting

Each post that a user makes will be tagged in this way and will also be marked with their current fresh pseudonym. This allows other users to, in turn, evaluate their behaviour, and give feedback if said behaviour is inappropriate. These evaluations come in the form of votes, which are sent to servers and stored as the output of $\textbf{BGNEncrypt}_H()$ (such values having form $V = h^v \cdot h'^r$, for h the generator of H, h' the generator of H_1, $v \in [0,2]$ the plaintext vote, and $r \in [1, |H_1| - 1]$ a random blinding factor).

When a user votes in PRSONA, they touch each of the votes they currently have for all other users. For any users that a voter wishes to evaluate based on the most recent evidence, they may replace existing votes. Although users cannot decrypt such votes that were originally cast in previous epochs (as, in our usage, $\textbf{BGNEncrypt}_H()$ encrypts values that can only be opened by a secret key distributed across the servers), servers provide voters encryptions of their previous votes tagged with which current fresh pseudonym each vote applies to. For all other votes, they rerandomize the existing vote as follows. First, they choose a random factor $r' \xleftarrow{\$} [1, |H_1| - 1]$. They then calculate $V' = V \cdot h'^{r'}$, which, assuming V is well formed, is equivalent to $h^v \cdot h'^{r+r'}$, for their previous vote v for this votee, and for some unknown $r \in [0, |H_1| - 1]$. The assumption that V is well-formed is sound: in the case that a server is initially accepting a vote, or is accepting its rerandomization, said server first verifies the correctness of the vote via a ZKP that the voter provides. In the case that a server is rerandomizing a vote as part of epoch changeovers, for other servers to accept the validity of the epoch calculations, they too first verify a vote's correctness via a ZKP that the server provides (when in the covert setting; in the honest-but-curious setting, we are already assuming that a server is carrying out this protocol correctly without causing V to become malformed). Due to the nature of **BGNDecrypt()**, namely that the h' term is removed via the secret key projection π_2, this forms a new, still well-formed vote without changing the value of said vote. **BGNDecrypt()** and its effect on V will be discussed further in Section 4.10.2.

Once this has been done, the voter generates a ZKP over all their votes to prove that each is either a new encryption of a valid value (within the range of accepted votes) or a rerandomization of the vote that was already there: $\bigwedge_{i=1}^{n} \{(x, v'_i, r') : X_t = \mathfrak{g}_t^x \wedge ((V' = h^{v'_i} h'^{r'} \wedge v' \in [0,2]) \vee V' = V h'^{r'})\}$. The $v'_i \in [0,2]$ portion of the proof is a straightforward extension of a well-known Σ-protocol for knowledge that a committed value is 0 or 1, such as can be seen in Figure 1 of Groth and Kohlweiss [GK14]. This ZKP is proportional in size and complexity to the number of users in PRSONA.

The voter submits this proof and their votes (new and rerandomized) to a randomly selected server. Upon verification of this proof, this server forwards the votes and the proof to

the other servers, who verify the proof and update their data stores with the new set of votes. As with user registration, a server that ignores a user's vote will be quickly detected by said user, without loss of privacy to the user, and said user can merely submit their vote to a new server if necessary.

As a practical note, a user may choose to intentionally reveal some information about their votes in exchange for a smaller ZKP. Namely, if a user consents to revealing that they did *not* vote for certain other users in a given update, they may choose not to rerandomize those votes at all, which would result in cheaper proofs (both in size and in efficiency, as in both cases, the proofs are linear with respect to the number of votees for which the user's encrypted vote has changed). However, this does restrict the anonymity set of who users *may* have voted for, and complicates PRSONA's guarantee of Voter-Vote Unlinkability. This is a tradeoff the user should very carefully consider before weakening their privacy guarantees.

4.10 Epoch Changeover

The most computationally intensive work that is done as part of PRSONA happens between each pair of epochs. During this time, the servers must recalculate users' reputation levels, generate fresh pseudonyms for each user, and associate those reputations with the correct fresh pseudonym, without allowing fresh pseudonyms to be linked between epochs.

While servers are computing the epoch changeover, certain functionalities of PRSONA are limited. Users can still make ZKPs that their reputation is above given thresholds, but new votes cannot be submitted, nor can new users be added. As will be seen in Chapter 5, computing the epoch changeover can take significant amounts of time, so care must be taken in choosing when epoch changeovers are computed, in order to minimize downtime for users during periods of high activity. This, in turn, must be balanced with the need to compute epoch changeovers in order for users to obtain new fresh pseudonyms, and the privacy benefits those carry.

As discussed in Section 4.6, each server holds several pieces of information. They track all of the fresh pseudonyms in a given epoch, along with the epoch generator that the fresh pseudonyms are all based from. (The servers keep track of these fresh pseudonyms, but cannot know which fresh pseudonym in this epoch corresponds to which fresh pseudonym from any previous or future epoch, so long as any one server behaves honestly.) They additionally track the matrix of encrypted votes and the previous epoch's calculated scores for each user, both of which are encrypted to the shared server BGN secret key (*i.e.*, they are the output of $\mathbf{BGNEncrypt}_H()$ and $\mathbf{BGNEncrypt}_G()$, respectively). Finally, the servers also hold a list of the

previous epoch's calculated scores for each user, where each encrypted score is encrypted to the user's long-term secret key, using the current epoch generator (via **EGEncrypt()**).

In order to achieve the necessary goals, the inter-epoch calculation proceeds in four rounds, as follows:

1. **Build-up Phase:** The servers collaboratively choose the next epoch's epoch generator, and raise each user's fresh pseudonym to new values, putting the fresh pseudonyms into a transitional state between the epochs (which cannot be linked to the ordinary state in either the previous or following epoch).

2. **Decryption Phase:** The servers calculate the reputation scores for the next epoch and collaboratively decrypt them (as they will need their plaintext values to re-encrypt them). These reputation scores will then be seen by the servers in plaintext form, associated with the transitional state fresh pseudonyms.

3. **Re-encryption Phase:** The servers collaboratively re-encrypt the plaintext reputation scores into two forms. First, they encrypt as the output of **BGNEncrypt$_G$()**, which will be used during the next Decryption Phase. Second, they encrypt as the output of **EGEncrypt()**. During this phase, these encryptions will use the transitional state fresh pseudonyms generated in the Build-up Phase, and over the course of the Break-down Phase, these encrypted values will be altered so that they are correctly encrypted to the user's long-term secret key using the next epoch generator. This somewhat complex arrangement is necessary to avoid linking new epoch fresh pseudonyms to their users' raw scores, which are observed during this and the Decryption Phase.

4. **Break-down Phase:** The servers raise each user's fresh pseudonym (and part of their encrypted reputation score) to the inverse of the factors added during the previous inter-epoch calculations, leaving only the factors they used during *this* set of inter-epoch calculations (in the Build-up Phase). Once this is done, the fresh pseudonyms (and encrypted reputation scores) are in their final state for the new epoch.

When describing each of these phases, we include pseudocode to describe what is happening in each phase. This pseudocode makes use of a number of helper functions. Algorithm 3 is a general helper function that calculates the product of several sets of values at once. Algorithm 4, Algorithm 5, Algorithm 6, Algorithm 7, and Algorithm 8 are helper functions specific to shuffling the data held by servers in the covert setting, which happens during the Build-up and Break-down Phases. Each of those functions handles shuffling a specific type of data. Algorithm 9 is the specific function that does the shuffling in the covert setting.

Algorithm 10 is the specific function that does the shuffling in the honest-but-curious setting. Algorithm 11 and Algorithm 12 are helper functions specific to verifying the ZKPs used to prove shuffles are correct in the covert setting, which also happens during the Build-up and Break-down Phases; Algorithm 13 is the specific function that does the verification (again, in the covert setting).

Algorithm 3: compressMatrix. Calculates $\prod_{x \in \mathbf{X}[i,\dots]}$ for all i.

Input: **X**: A matrix of partial data values. Each row is constructed such that the product of its elements produces one randomized and/or shuffled data value.
Output: $\vec{x}$ (retval): The output vector of randomized and/or shuffled data.

```
for i ← 1...n do
    retval[i] ← X[i, 1];
    for j ← 2...n do
        retval[i] ← retval[i] × X[i, j];
    end
end
return retval;
```

In the covert setting, the strategy for shuffling a vector is as follows. First, we create a matrix **P** and commit to it with a matrix **B**. We prove that **P** is a permutation matrix, and form the matrix whose rows are the componentwise products of the vector with each row of **P**. With that in hand, we use Algorithm 3 to compress the rows into single elements by taking the product of the elements in the row. That is, Algorithm 3 is used with every vector that needs to be shuffled, each shuffle. The same process is also done for the vote matrix, row-by-row. (Though, in order to permute both the rows and columns correctly, it must be performed twice on the vote matrix, the second time being column-by-column.) All of these cases are further explored in the following algorithms.

To help illuminate both how shuffles function, as well as the specific role of Algorithm 3, we will work a toy example. Consider a scenario where we have three users, $A, B,$ and C, and two servers I and II. Suppose we are at server I, executing the Break-down Phase of the epoch changeover between epochs ℓ and $\ell+1$. We have a vector of server-encrypted reputation scores as follows: $\vec{S} = \langle S_a = g^{s_a} g'^{r_a}, S_b = g^{s_b} g'^{r_b}, S_c = g^{s_c} g'^{r_c} \rangle$. We have a vector of part-way pseudonyms as follows: $\vec{X} = \langle A_{\ell/\ell+1} = \mathfrak{g}^{\mathfrak{r}_{\mathrm{I},\ell}\mathfrak{r}_{\mathrm{II},\ell}\mathfrak{r}_{\mathrm{I},\ell+1}\mathfrak{r}_{\mathrm{II},\ell+1}x_a}, B_{\ell/\ell+1} = \mathfrak{g}^{\mathfrak{r}_{\mathrm{I},\ell}\mathfrak{r}_{\mathrm{II},\ell}\mathfrak{r}_{\mathrm{I},\ell+1}\mathfrak{r}_{\mathrm{II},\ell+1}x_b}, C_{\ell/\ell+1} = \mathfrak{g}^{\mathfrak{r}_{\mathrm{I},\ell}\mathfrak{r}_{\mathrm{II},\ell}\mathfrak{r}_{\mathrm{I},\ell+1}\mathfrak{r}_{\mathrm{II},\ell+1}x_c} \rangle$. Finally, we have a vector of user-encrypted reputation scores as follows: $\vec{U} = \langle U_a = (A^{r'_a}_{\ell/\ell+1}, \mathfrak{g}^{r'_a}_{\ell+1} \mathfrak{h}^{s_a}), U_b = (B^{r'_b}_{\ell/\ell+1}, \mathfrak{g}^{r'_b}_{\ell+1} \mathfrak{h}^{s_b}), U_c = (C^{r'_c}_{\ell/\ell+1}, \mathfrak{g}^{r'_c}_{\ell+1} \mathfrak{h}^{s_c}) \rangle$

Suppose we generate a permutation matrix **P**:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

We must commit to this matrix, and so do so with **B**:

$$\mathbf{B} = \begin{bmatrix} \mathfrak{g}\mathfrak{h}^{r_1} & \mathfrak{h}^{r_2} & \mathfrak{h}^{r_3} \\ \mathfrak{h}^{r_4} & \mathfrak{h}^{r_5} & \mathfrak{g}\mathfrak{h}^{r_6} \\ \mathfrak{h}^{r_7} & \mathfrak{g}\mathfrak{h}^{r_8} & \mathfrak{h}^{r_9} \end{bmatrix}$$

Note that in the above, $r_1 + r_2 + r_3 = r_4 + r_5 + r_6 = r_7 + r_8 + r_9 = 0 \bmod q$.

In the process of shuffling and rerandomizing $\vec{S}$, we calculate a new matrix **C** and prove its consistency with **B** and $\vec{S}$ using a zero-knowledge proof. **C** is calculated as follows:

$$\mathbf{C} = \begin{bmatrix} S_a^1 g'^{s'_1} & S_b^0 g'^{s'_2} & S_c^0 g'^{s'_3} \\ S_a^0 g'^{s'_4} & S_b^0 g'^{s'_5} & S_c^1 g'^{s'_6} \\ S_a^0 g'^{s'_7} & S_b^1 g'^{s'_8} & S_c^0 g'^{s'_9} \end{bmatrix}$$

We follow the same process to shuffle the vote matrix **V**, making a new such matrix for each row in the vote matrix (and executing the shuffle twice in order to compute $\mathbf{P}^T\mathbf{V}\mathbf{P}$). Elements of **V** are in H instead of G, but there are no other differences in the approach.

In the process of shuffling $\vec{X}$, we calculate a new matrix **D** and prove its consistency with **B** and $\vec{X}$ using a zero-knowledge proof. **D** is calculated as follows:

$$\mathbf{D} = \begin{bmatrix} A_{\ell/\ell+1}^{1\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_1} & B_{\ell/\ell+1}^{0\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_2} & C_{\ell/\ell+1}^{0\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_3} \\ A_{\ell/\ell+1}^{0\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_4} & B_{\ell/\ell+1}^{0\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_5} & C_{\ell/\ell+1}^{1\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_6} \\ A_{\ell/\ell+1}^{0\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_7} & B_{\ell/\ell+1}^{1\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_8} & C_{\ell/\ell+1}^{0\tau_{I,\ell}^{-1}} \mathfrak{h}^{s''_9} \end{bmatrix}$$

Note that in the above, $s''_1 + s''_2 + s''_3 = s''_4 + s''_5 + s''_6 = s''_7 + s''_8 + s''_9 = 0 \bmod q$.

In the process of shuffling and rerandomizing $\vec{U}$, we calculate two new matrices **F** and **H** and prove their consistency with **B** and $\vec{U}$ using a zero-knowledge proof. **F** and **H** are calculated as follows:

$$\mathbf{F} = \begin{bmatrix} (A_{\ell/\ell+1}^{r'_a})^{1\mathfrak{r}_\ell^{-1}} A_{\ell/\ell+1}^{1\mathfrak{r}_\ell^{-1}s'''_1} \mathfrak{h}^{s'''_1} & (B_{\ell/\ell+1}^{r'_b})^{0\mathfrak{r}_\ell^{-1}} B_{\ell/\ell+1}^{0\mathfrak{r}_\ell^{-1}s'''_2} \mathfrak{h}^{s'''_2} & (C_{\ell/\ell+1}^{r'_c})^{0\mathfrak{r}_\ell^{-1}} C_{\ell/\ell+1}^{0\mathfrak{r}_\ell^{-1}s'''_3} \mathfrak{h}^{s'''_3} \\ (A_{\ell/\ell+1}^{r'_a})^{0\mathfrak{r}_\ell^{-1}} A_{\ell/\ell+1}^{0\mathfrak{r}_\ell^{-1}s'''_4} \mathfrak{h}^{s'''_4} & (B_{\ell/\ell+1}^{r'_b})^{0\mathfrak{r}_\ell^{-1}} B_{\ell/\ell+1}^{0\mathfrak{r}_\ell^{-1}s'''_5} \mathfrak{h}^{s'''_5} & (C_{\ell/\ell+1}^{r'_c})^{1\mathfrak{r}_\ell^{-1}} C_{\ell/\ell+1}^{1\mathfrak{r}_\ell^{-1}s'''_6} \mathfrak{h}^{s'''_6} \\ (A_{\ell/\ell+1}^{r'_a})^{0\mathfrak{r}_\ell^{-1}} A_{\ell/\ell+1}^{0\mathfrak{r}_\ell^{-1}s'''_7} \mathfrak{h}^{s'''_7} & (B_{\ell/\ell+1}^{r'_b})^{1\mathfrak{r}_\ell^{-1}} B_{\ell/\ell+1}^{1\mathfrak{r}_\ell^{-1}s'''_8} \mathfrak{h}^{s'''_8} & (C_{\ell/\ell+1}^{r'_c})^{0\mathfrak{r}_\ell^{-1}} C_{\ell/\ell+1}^{0\mathfrak{r}_\ell^{-1}s'''_9} \mathfrak{h}^{s'''_9} \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} (\mathfrak{g}_{\ell+1}^{r'_a} \mathfrak{h}^{s_a})^1 \mathfrak{g}_{\ell+1}^{1s'''_1} \mathfrak{h}^{s'''_1} & (\mathfrak{g}_{\ell+1}^{r'_b} \mathfrak{h}^{s_b})^0 \mathfrak{g}_{\ell+1}^{0s'''_2} \mathfrak{h}^{s'''_2} & (\mathfrak{g}_{\ell+1}^{r'_c} \mathfrak{h}^{s_c})^0 \mathfrak{g}_{\ell+1}^{0s'''_3} \mathfrak{h}^{s'''_3} \\ (\mathfrak{g}_{\ell+1}^{r'_a} \mathfrak{h}^{s_a})^0 \mathfrak{g}_{\ell+1}^{0s'''_4} \mathfrak{h}^{s'''_4} & (\mathfrak{g}_{\ell+1}^{r'_b} \mathfrak{h}^{s_b})^0 \mathfrak{g}_{\ell+1}^{0s'''_5} \mathfrak{h}^{s'''_5} & (\mathfrak{g}_{\ell+1}^{r'_c} \mathfrak{h}^{s_c})^1 \mathfrak{g}_{\ell+1}^{1s'''_6} \mathfrak{h}^{s'''_6} \\ (\mathfrak{g}_{\ell+1}^{r'_a} \mathfrak{h}^{s_a})^0 \mathfrak{g}_{\ell+1}^{0s'''_7} \mathfrak{h}^{s'''_7} & (\mathfrak{g}_{\ell+1}^{r'_b} \mathfrak{h}^{s_b})^1 \mathfrak{g}_{\ell+1}^{1s'''_8} \mathfrak{h}^{s'''_8} & (\mathfrak{g}_{\ell+1}^{r'_c} \mathfrak{h}^{s_c})^0 \mathfrak{g}_{\ell+1}^{0s'''_9} \mathfrak{h}^{s'''_9} \end{bmatrix}$$

Note that in the above, $s'''_1 + s'''_2 + s'''_3 = s'''_4 + s'''_5 + s'''_6 = s'''_7 + s'''_8 + s'''_9 = 0 \bmod q$.

When Algorithm 3 is run on **C**, the output will be the following vector: $\vec{S'} = \langle S_a g'^{s'_1+s'_2+s'_3}, S_c g'^{s'_4+s'_5+s'_6}, S_b g'^{s'_7+s'_8+s'_9} \rangle$, a shuffled and rerandomized server-encrypted reputation score vector. When Algorithm 3 is run on **D**, the output will be the following vector: $\vec{X'} = \langle A_{\ell/\ell+1}^{\mathfrak{r}_{I,\ell}^{-1}}, C_{\ell/\ell+1}^{\mathfrak{r}_{I,\ell}^{-1}}, B_{\ell/\ell+1}^{\mathfrak{r}_{I,\ell}^{-1}} \rangle$, a shuffled vector of partway pseudonyms raised to the proper inverse epoch factor for server I. When Algorithm 3 is run on **F**, the output will be the following vector: $\vec{U'_1} = \langle A_{\ell/\ell+1}^{\mathfrak{r}_\ell^{-1}(r'_a+s'''_1)}, C_{\ell/\ell+1}^{\mathfrak{r}_\ell^{-1}(r'_c+s'''_6)}, B_{\ell/\ell+1}^{\mathfrak{r}_\ell^{-1}(r'_b+s'''_8)} \rangle$, a shuffled and rerandomized vector of the first elements of user-encrypted reputation scores. When Algorithm 3 is run on **H**, the output will be the following vector: $\vec{U'_2} = \langle \mathfrak{g}_{\ell+1}^{r'_a+s'''_1} \mathfrak{h}^{s_a}, \mathfrak{g}_{\ell+1}^{r'_c+s'''_6} \mathfrak{h}^{s_c}, \mathfrak{g}_{\ell+1}^{r'_b+s'''_8} \mathfrak{h}^{s_b} \rangle$, a shuffled and rerandomized vector of

the second elements of user-encrypted reputation scores.

Algorithm 4: generatePermutationCommitmentAndProofIsBinary. Creates a matrix of Pedersen commitments to a permutation matrix and proves that this commitment matrix is a binary matrix.

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
P (permutationMatrix): The permutation matrix we are committing to

Output: **B** (permutationCommitment): A matrix of Pedersen commitments to elements of a permutation matrix,
S (permutationSeeds): A matrix of blinding factors used in those Pedersen commitments,
π_{perm}: A ZKP of the correctness of this commitment matrix

---

for $i \leftarrow 1 \dots n$ **do**
  for $j \leftarrow 1 \dots n$ **do**
    if $j \neq n$ **then**
      permutationSeeds$[i, j] \xleftarrow{\$} [1, q-1]$;
    else
      permutationSeeds$[i, j] \leftarrow -\sum_{k=1}^{n-1}$ permutationSeeds$[i, k]$;
    end
    $p_{i,j} \leftarrow$ permutationMatrix$[i, j]$;
    $s_{i,j} \leftarrow$ permutationSeeds$[i, j]$;
    permutationCommitment$[i, j] \leftarrow \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}}$;
  end
end
$\pi_{\text{perm}} \leftarrow$ `generateProof`($\mathfrak{g}$, $\mathfrak{h}$, **B** $\leftarrow$permutationCommitment, **P** $\leftarrow$permutationMatrix, **S** $\leftarrow$permutationSeeds, $\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(p_{i,j}, s_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge p_{i,j} \in [0, 1]\}$);
return permutationCommitment, permutationSeeds, π_{perm};

Algorithm 4 takes as input a permutation matrix — a matrix whose elements are all exclusively 0 or 1, and for which it is true that both every row and every column sums to 1 — and generates a matrix of Pedersen commitments to its elements. Further, in the covert setting, it generates a ZKP that these commitments are correct and that this matrix is a binary matrix. Extra steps are taken to prove that the matrix is a permutation matrix from there, which are

detailed further (along with the exact ZKP referenced by this algorithm) in Section 4.10.1.

Algorithm 5 is the first subroutine that explicitly involves shuffling data. This subroutine is responsible for shuffling fresh pseudonyms in each server's turn. As will be detailed in the explanation of the Build-up Phase below, in addition to being shuffled, fresh pseudonyms are also all exponentiated by a common epoch factor $\mathfrak{r}$; this subroutine does both actions simultaneously. Its output is a matrix of elements, which, when Algorithm 3 is run on this matrix, the output will be the shuffled fresh pseudonyms with this (random) epoch factor applied to them. Further detail, particularly on the ZKP used to prove the correctness of this matrix in the covert setting, is provided in Section 4.10.1.

Algorithm 6 is the subroutine responsible for shuffling data that needs to be rerandomized. This subroutine is used to shuffle the elements of the vote matrix, as well as the server-encrypted reputation scores, during each server's turn. This subroutine handles both shuffling and rerandomization simultaneously. Its output is a matrix of elements; when Algorithm 3 is run on this matrix, the output of that subroutine will be the shuffled, rerandomized encryptions. Algorithm 6 expects a vector as input for the data to be shuffled; shuffling the vote matrix therefore involves shuffling each row/column individually (though still with the same permutation matrix used everywhere else), then putting the results back together into one matrix. Further detail, particularly on the ZKP used to prove the correctness of the output matrix in the covert setting, is provided in Section 4.10.1.

Algorithm 7 is a specific subroutine responsible for shuffling the vote matrix $\mathbf{V}$ using a permutation matrix $\mathbf{P}$. It uses the previously discussed Algorithm 6 extensively to accomplish this. The main purpose of Algorithm 7 beyond that subroutine is to organize the vote matrix to be input into Algorithm 6 correctly. As rows and columns must be consistently permuted for the votes to stay in an order congruous with the shuffle dictated by $\mathbf{P}$ for both voter and votee, we specifically want to compute $\mathbf{P}^T\mathbf{V}\mathbf{P}$; thus, there are two matrix-matrix products to compute. Because of this, there are two series of invocations to `generateReorderedMatrixAndProof` (Algorithm 6). We term the series of outputs from that subroutine a "vote tensor", as it is a series of matrices, each of which can be compressed (via Algorithm 3) into a row of the shuffled vote matrix.

Algorithm 8 is a subroutine that is responsible for shuffling the user-encrypted reputation scores. This subroutine is only used during the Break-down Phase, as user-encrypted reputation scores are ignored during the Build-up Phase. These reputation scores are encrypted as the output of **EGEncrypt()** (*i.e.*, a pair of elements of the ElGamal group), so each of the two parts of the ciphertext has a different operation done to it. More detail is provided in Section 4.10.2 and Section 4.10.3, but the first elements of the ciphertexts have a per-epoch factor applied to them in addition to being invidually rerandomized, while the second elements of the

Algorithm 5: generatePseudonymMatrixAndProof. Handles creating the matrix that represents the shuffled fresh pseudonyms, with a server's random factor applied to them, as well as a ZKP of the correctness of this matrix.

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
$\mathfrak{r}$: The server's random factor for the new epoch,
currentPseudonyms: User's fresh pseudonyms from the previous epoch,
P (permutationMatrix): The permutation matrix we are committing to,
B (permutationCommitment): A matrix of Pedersen commitments to elements of a permutation matrix,
S (permutationSeeds): A matrix of blinding factors used in those Pedersen commitments

Output: **D** (pseudonymMatrix): A matrix whose elements are the products of elements from currentPseudonyms and elements from **P**, raised to the power $\mathfrak{r}$,
E (pseudonymSeedMatrix): A matrix committing to the random values used to mask the elements of **D**,
π_{pseud}: A ZKP of the correctness of these matrices

for $i \leftarrow 1 \ldots n$ **do**
  for $j \leftarrow 1 \ldots n$ **do**
    if $j \neq n$ **then**
      pseudonymSeeds$[i,j] \xleftarrow{\$} [1, q-1]$;
    else
      pseudonymSeeds$[i,j] \leftarrow -\sum_{k=1}^{n-1}$ pseudonymSeeds$[i,k]$;
    end
    $p_{i,j} \leftarrow$ permutationMatrix$[i,j]$;
    $s'_{i,j} \leftarrow$ pseudonymSeeds$[i,j]$;
    pseudonymMatrix$[i,j] \leftarrow$ currentPseudonyms$[j]^{p_{i,j}\mathfrak{r}} \mathfrak{h}^{s'_{i,j}}$;
    pseudonymSeedMatrix$[i,j] \leftarrow \mathfrak{g}^{s'_{i,j}}$;
  end
end
$\pi_{\text{pseud}} \leftarrow$ `generateProof`($\mathfrak{g}$, $\mathfrak{h}$, $\vec{a} \leftarrow$currentPseudonyms, **B** $\leftarrow$ permutationCommitment, **D** $\leftarrow$pseudonymMatrix, **E** $\leftarrow$pseudonymSeedMatrix, r, **P** $\leftarrow$permutationMatrix, **S** $\leftarrow$permutationSeeds, **S**$'$ $\leftarrow$pseudonymSeeds, $\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(\mathfrak{r}, p_{i,j}, s_{i,j}, s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge D_{i,j} = A_j^{p_{i,j}\mathfrak{r}} \mathfrak{h}^{s'_{i,j}} \wedge E_{i,j} = \mathfrak{g}^{s'_{i,j}}\}$);
return pseudonymMatrix, pseudonymSeedMatrix, π_{pseud};

Algorithm 6: generateReorderedMatrixAndProof. Creates the matrix that represents shuffled and rerandomized data, as well as a ZKP of the correctness of this matrix.

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
$\vec{a}$: The data to be reordered,
P (permutationMatrix): The permutation matrix we are committing to,
B (permutationCommitment): A matrix of Pedersen commitments to elements of a permutation matrix,
S (permutationSeeds): A matrix of blinding factors used in those Pedersen commitments

Output: **C** (reorderedMatrix): A matrix whose elements are the products of elements from $\vec{a}$ and elements from **P**,
π_{reorder}: A ZKP of the correctness of **C**

---

for $i \leftarrow 1 \dots n$ **do**
  for $j \leftarrow 1 \dots n$ **do**
    reorderedSeeds$[i,j] \xleftarrow{\$} [1, q-1]$;
    $p_{i,j} \leftarrow$ permutationMatrix$[i,j]$;
    $s'_{i,j} \leftarrow$ reorderedSeeds$[i,j]$;
    /* Elements of $\vec{a}$ are group elements of the BGN groups; the j-th element of $\vec{a}$ is thus notated as A_j */
    reorderedMatrix$[i,j] \leftarrow A_j^{p_{i,j}} \mathfrak{h}^{s'_{i,j}}$;
  end
end
$\pi_{\text{reorder}} \leftarrow$ `generateProof`$(\mathfrak{g}, \mathfrak{h}, \langle A \rangle, \vec{B} \leftarrow$ permutationCommitment, $\vec{C} \leftarrow$reorderedMatrix, $\vec{P} \leftarrow$permutationMatrix, $\vec{S} \leftarrow$permutationSeeds, $\vec{S'} \leftarrow$reorderedSeeds, $\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(p_{i,j}, s_{i,j}, s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge C_{i,j} = A_j^{p_{i,j}} h'^{s'_{i,j}}\})$;
return reorderedMatrix, π_{reorder};

Algorithm 7: generateVoteTensorAndProof. Creates a series of matrices that represent the rows of a shuffled vote matrix, and collates the ZKPs for reordering each of these rows.

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
V (voteMatrix): The matrix containing encrypted votes users cast for one another,
P (permutationMatrix): The permutation matrix we are committing to,
B (permutationCommitment): A matrix of Pedersen commitments to elements of a permutation matrix,
S (permutationSeeds): A matrix of blinding factors used in those Pedersen commitments

Output: voteTensorPartA: The vote tensor for $\mathbf{P}^T\mathbf{V}$,
voteTensorPartB: The vote tensor for $(\mathbf{P}^T\mathbf{V})\mathbf{P}$,
π_{voteA}: A ZKP of the correctness of the first vote tensor,
π_{voteB}: A ZKP of the correctness of the second vote tensor

---

for $k \leftarrow 1 \dots n$ **do**
  voteTensorPartA[k], $\pi_{\text{voteA}}[k] \leftarrow$ `generateReorderedMatrixAndProof`($\mathfrak{g}$, $\mathfrak{h}$, $\vec{a} \leftarrow$ voteMatrix$[k, \dots]$, $\mathbf{B} \leftarrow$ permutationCommitment, permutationMatrix, permutationSeeds);
  partwayVoteMatrix$[k, \dots] \leftarrow$ `compressMatrix`($\mathbf{X} \leftarrow$ voteTensorPartA$[k]$);
end
for $k \leftarrow 1 \dots n$ **do**
  voteTensorPartB[k], $\pi_{\text{voteB}}[k] \leftarrow$ `generateReorderedMatrixAndProof`($\mathfrak{g}$, $\mathfrak{h}$, $\langle A \rangle \leftarrow$ partwayVoteMatrix$[k, \dots]$, $\vec{B} \leftarrow$ permutationCommitment, permutationMatrix, permutationSeeds);
end
return voteTensorPartA, voteTensorPartB, π_{voteA}, π_{voteB};

Algorithm 8: generateUserEncryptedMatricesAndProof. Shuffles, rerandomizes, and proves correctness of user-encrypted reputation scores.

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
$\mathfrak{g}_{\ell+1}$: The epoch generator for the next epoch $(\ell + 1)$,
$\mathfrak{r}$: The server's random factor for the new epoch,
userEncryptedScores: The reputation scores from the previous epoch, in the form encrypted to users' long-term secret keys,
$\vec{x}_{\ell/\ell+1}$ (currentPseudonyms): The partway pseudonyms of users used during the inter-epoch calculation between epochs ℓ and $\ell + 1$,
P (permutationMatrix): The permutation matrix we are committing to,
B (permutationCommitment): A matrix of Pedersen commitments to elements of a permutation matrix,
S (permutationSeeds): A matrix of blinding factors used in those Pedersen commitments

Output: **E** (userEncryptedScoreMaskSeedMatrix): A matrix committing to the random values used to mask the elements of **F**,
F (userEncryptedScoreMaskMatrix): A matrix whose elements are the products of the mask portions of encrypted scores in userEncryptedScores and elements from **P**, raised to the power $\mathfrak{r}$,
H (userEncryptedScoreMessageMatrix): A matrix whose elements are the products of the message portions of encrypted scores in userEncryptedScores and elements from **P**,
π_{pseud}: A ZKP of the correctness of these matrices

$\vec{a} \leftarrow \langle C_1 : (C_1, C_2) \in \text{ userEncryptedScores} \rangle$;
$\vec{a'} \leftarrow \langle C_2 : (C_1, C_2) \in \text{ userEncryptedScores} \rangle$;
for $i \leftarrow 1 \ldots n$ **do**
  for $j \leftarrow 1 \ldots n$ **do**
    if $j \neq n$ **then**
      userEncryptedScoreSeeds$[i, j] \xleftarrow{\$} [1, q-1]$;
    else
      userEncryptedScoreSeeds$[i, j] \leftarrow -\sum_{k=1}^{n-1}$ userEncryptedScoreSeeds$[i, k]$;
    end
    $p_{i,j} \leftarrow$ permutationMatrix$[i, j]$;
    $s'_{i,j} \leftarrow$ userEncryptedScoreSeeds$[i, j]$;
    $X_j \leftarrow$ currentPseudonyms$[j]$;
    userEncryptedScoreMaskMatrix$[i, j] \leftarrow A_j^{p_{i,j}\mathfrak{r}} X_j^{p_{i,j}\mathfrak{r}s'_{i,j}} \mathfrak{h}^{s'_{i,j}}$;
    userEncryptedScoreMaskSeedMatrix$[i, j] \leftarrow \mathfrak{g}^{s'_{i,j}}$;
    userEncryptedScoreMessageMatrix$[i, j] \leftarrow A_j'^{p_{i,j}} \mathfrak{g}_{\ell+1}^{p_{i,j}s'_{i,j}} \mathfrak{h}^{s'_{i,j}}$;
  end
end
$\pi_{\text{pseud}} \leftarrow$ `generateProof`($\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{g}_{\ell+1}$, $\vec{a}$, $\vec{a'}$, $\vec{x}_{\ell/\ell+1} \leftarrow$ currentPseudonyms, **B** $\leftarrow$ permutationCommitment, **E** $\leftarrow$userEncryptedScoreMaskSeedMatrix, **F** $\leftarrow$userEncryptedScoreMaskMatrix, **H** $\leftarrow$userEncryptedScoreMessageMatrix, $\mathfrak{r}$, **P** $\leftarrow$permutationMatrix, **S** $\leftarrow$permutationSeeds, **S**$'$ $\leftarrow$userEncryptedScoreSeeds, $\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(\mathfrak{r}, p_{i,j}, s_{i,j}, s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge E_{i,j} = \mathfrak{g}^{s'_{i,j}} \wedge F_{i,j} = A_j^{p_{i,j}\mathfrak{r}} X_{j,\ell/\ell+1}^{p_{i,j}\mathfrak{r}s'_{i,j}} \mathfrak{h}^{s'_{i,j}} \wedge H_{i,j} = A_j'^{p_{i,j}} \mathfrak{g}_{\ell+1}^{p_{i,j}s'_{i,j}} \mathfrak{h}^{s'_{i,j}}\}$);
return userEncryptedScoreMaskSeedMatrix, userEncryptedScoreMaskMatrix, userEncryptedScoreMessageMatrix, $\pi_{\text{userEncrypted}}$;

ciphertexts must be rerandomized with the same new blinding factor used to rerandomize the first element. The output of this function is two matrices, which can be compressed (via Algorithm 3) into a vector representing each of the first and second elements of the shuffled and rerandomized ciphertexts. More detail on the ZKP used for this function in the covert setting is provided in Section 4.10.3.

Algorithm 9 is the core subroutine used in both the Build-up and Break-down Phases in the covert setting. It calls the various shuffling subroutines with the correct inputs, and marshals the outputs together to be accepted by other servers. The Build-up and Break-down Phases have slight variations in how they use this function, between the value of $\mathfrak{r}$ (a factor the servers apply to certain data elements; during Build-up, it is a random value (the "epoch factor"), and during Break-down, it is the inverse of a previous value of $\mathfrak{r}$), and whether or not Algorithm 8 is invoked. Its output includes the various matrices that can be compressed into shuffled data, and ZKPs for all of these matrices.

Algorithm 10 is the core subroutine used in both the Build-up and Break-down Phases in the honest-but-curious setting. It generates a random order to shuffle the data into, then shuffles all data vectors into the specified new order (while also applying the server's epoch factor to the appropriate values). The Build-up and Break-down Phases have slight variations in how they use this function, between the value of $\mathfrak{r}$ (with the same variations as in Algorithm 9), and whether or not user-encrypted reputation scores are shuffled as well. Its output is the shuffled data, to be directly accepted by the next server.

Algorithm 11 is the counterpart to Algorithm 7. It is used to organize verifying the ZKPs created during its counterpart subroutine, which must be done before a server can accept

Algorithm 9: covertShuffleAndApplyFactor. The main subroutine for both the Build-up and Break-down Phases in the covert setting.

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
$\mathfrak{g}_{\ell+1}$: The epoch generator for the next epoch $(\ell + 1)$,
$\mathfrak{r}$: Either the server's random factor for the next epoch $(\mathfrak{r}_{k,\ell+1})$, or the inverse of its random factor for the previous $(\mathfrak{r}_{k,\ell}^{-1})$,
isBreakdown: Boolean value representing whether the server is executing the Break-down Phase or Build-up Phase,
currentPseudonyms: Either the previous epoch's fresh pseudonyms, the partway pseudonyms between epochs, or the output shuffled pseudonyms of the previous server, depending on whether this server is the first to execute in a phase (and which phase),
serverEncryptedScores: The scores as they are encrypted to the servers' shared secret key (or the output shuffled such scores of the previous server),
voteMatrix: The matrix containing encrypted votes users cast for one another (or the reordered version output by the previous server),
userEncryptedScores: The scores as they are encrypted to the users' long-term secret keys (or the output shuffled such scores of the previous server)

Output: outputData: The shuffled data,
π_{epoch}: A ZKP that the data was shuffled correctly

/* `shuffle()` returns a random permutation matrix that, when used to calculate a matrix-vector product with a data vector, results in a shuffled data vector */
permutationMatrix ← `shuffle()`;

permutationCommitment, permutationSeeds, π_{perm} ← `generatePermutationCommitmentAndProof`($\mathfrak{g}$, $\mathfrak{h}$, permutationMatrix);

pseudonymMatrix, pseudonymSeedMatrix, π_{pseud} ← `generatePseudonymMatrixAndProof`($\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{r}$, currentPseudonyms, $\vec{B}$ ← permutationCommitment, permutationMatrix, permutationSeeds);

serverEncryptedScoreMatrix, $\pi_{\text{serverEncrypted}}$ ← `generateReorderedMatrixAndProof`($\mathfrak{g}$, $\mathfrak{h}$, $\langle A \rangle$ ← serverEncryptedScores, $\vec{B}$ ← permutationCommitment, permutationMatrix, permutationSeeds);

/* The voteTensor here is a series of matrices, where each matrix compresses into one output vote row. Part A calculates $\vec{P}^T\vec{V}$, and part B calculates $(\vec{P}^T\vec{V})\vec{P}$. */

voteTensorPartA, voteTensorPartB, π_{voteA}, π_{voteB} ← `generateVoteTensorAndProof`($\mathfrak{g}$, $\mathfrak{h}$, voteMatrix, $\vec{B}$ ← permutationCommitment, permutationMatrix, permutationSeeds);

if *isBreakdown* $= \top$ **then**

  userEncryptedScoreMaskSeedMatrix, userEncryptedScoreMaskMatrix, userEncryptedScoreMessageMatrix, $\pi_{\text{userEncrypted}}$ ← `generateUserEncryptedMatricesAndProof`($\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{g}_{\ell+1}$, $\mathfrak{r}$, userEncryptedScores, currentPseudonyms, $\vec{B}$ ← permutationCommitment, permutationMatrix, permutationSeeds);

else

  userEncryptedScoreMaskSeedMatrix, userEncryptedScoreMaskMatrix, userEncryptedScoreMessageMatrix, $\pi_{\text{userEncrypted}} \leftarrow \perp$;

end

outputData ← (permutationCommitment, pseudonymMatrix, pseudonymSeedMatrix, serverEncryptedScoreMatrix, voteTensorPartA, voteTensorPartB, userEncryptedScoreMaskMatrix, userEncryptedScoreMaskSeedMatrix, userEncryptedScoreMessageMatrix);

$\pi_{\text{epoch}} \leftarrow (\pi_{\text{perm}}, \pi_{\text{pseud}}, \pi_{\text{serverEncrypted}}, \pi_{\text{voteA}}, \pi_{\text{voteB}}, \pi_{\text{userEncrypted}})$;

return outputData, π_{epoch};

Algorithm 10: hbcShuffleAndApplyFactor. The main subroutine for both the Build-up and Break-down Phases in the honest-but-curious setting.

Input: PK_{BGN}: The BGN public key,
$\mathfrak{g}_{\ell+1}$: The epoch generator for the next epoch $(\ell + 1)$,
$\mathfrak{r}$: Either the server's random factor for the next epoch $(\mathfrak{r}_{k,\ell+1})$, or the inverse of its random factor for the previous $(\mathfrak{r}_{k,\ell}^{-1})$,
isBreakdown: Boolean value representing whether the server is executing the Break-down Phase or Build-up Phase,
currentPseudonyms: Either the previous epoch's fresh pseudonyms, the partway pseudonyms between epochs, or the output shuffled pseudonyms of the previous server, depending on whether this server is the first to execute in a phase (and which phase),
serverEncryptedScores: The scores as they are encrypted to the servers' shared secret key (or the output shuffled such scores of the previous server),
voteMatrix: The matrix containing encrypted votes users cast for one another (or the reordered version output by the previous server),
userEncryptedScores: The scores as they are encrypted to the users' long-term secret keys (or the output shuffled such scores of the previous server)

Output: outputData: The shuffled data

```
/* shuffleOrder() returns a random order to permute the data with */
order ← shuffleOrder();
for i ← 1...n do
    shuffledPseudonyms[i] ← currentPseudonyms[order[i]]^r;
    shuffledServerEncryptedScores[i] ← BGNAdd(PK_BGN,
      serverEncryptedScores[order[i]], BGNEncrypt_G(PK_BGN, 0));
    for j ← 1...n do
        shuffledVoteMatrix[i][j] ←
          BGNAdd(PK_BGN, voteMatrix[order[i]][order[j]], BGNEncrypt_H(PK_BGN, 0));
    end
    if isBreakdown = ⊤ then
        s ←$ [1, q − 1];
        (currMask, currMessage) ← userEncryptedScores[order[i]];
        currMask ← currMask^r · currentPseudonyms[order[i]]^(rs);
        currMessage ← currMessage · g_{ℓ+1}^s;
        shuffledUserEncryptedScores[i] ← (currMask, currMessage);
    end
end
return (shuffledPseudonyms, shuffledServerEncryptedScores, shuffledVoteMatrix,
  shuffledUserEncryptedScores);
```

Algorithm 11: verifyVoteTensorProof. Organizes verifying the ZKPs created during generateVoteTensorAndProof.

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
B (permutationCommitment): A matrix of Pedersen commitments to elements of a permutation matrix,
voteTensorPartA: The vote tensor for $\mathbf{P}^T\mathbf{V}$,
voteTensorPartB: The vote tensor for $(\mathbf{P}^T\mathbf{V})\mathbf{P}$,
π_{voteA}: A ZKP of the correctness of the first vote tensor,
π_{voteB}: A ZKP of the correctness of the second vote tensor,
voteMatrix: The matrix containing encrypted votes users cast for one another

Output: A boolean value representing whether the proofs are accepted or not

---

for $k \leftarrow 1 \dots n$ **do**
  partwayVoteMatrix$[k, \dots] \leftarrow$ `compressMatrix`(voteTensorPartA$[k]$);
end
verifyVoteTensors $\leftarrow \top$;
for $k \leftarrow 1 \dots n$ **do**
  verifyVoteTensors $\leftarrow$ verifyVoteTensors $\wedge$`verifyProof`($\mathfrak{g}$, $\mathfrak{h}$, $\vec{a} \leftarrow$ voteMatrix$[k, \dots]$, **B** $\leftarrow$ permutationCommitment, **C** $\leftarrow$voteTensorPartA$[k]$, $\pi_{\text{voteA}}[k]$, $\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\{(p_{i,j}, s_{i,j}, s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}}\mathfrak{h}^{s_{i,j}} \wedge C_{i,j} = A_j^{p_{i,j}}h'^{s'_{i,j}}\}$);
  verifyVoteTensors $\leftarrow$ verifyVoteTensors $\wedge$`verifyProof`($\mathfrak{g}$, $\mathfrak{h}$, $\vec{a} \leftarrow$ partwayVoteMatrix$[k, \dots]$, **B** $\leftarrow$ permutationCommitment, **C** $\leftarrow$voteTensorPartB$[k]$, $\pi_{\text{voteB}}[k]$, $\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\{(p_{i,j}, s_{i,j}, s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}}\mathfrak{h}^{s_{i,j}} \wedge C_{i,j} = A_j^{p_{i,j}}h'^{s'_{i,j}}\}$);
end
return verifyVoteTensors;

shuffled data as correct in the covert setting.

Algorithm 12: compressAllData. Invokes compressMatrix several times to obtain the shuffled data output by the previous server.

Input: outputData: The previous server's output from `covertShuffleAndApplyFactor`

Output: $\text{outputPseudonyms}_{k-1}$: The shuffled fresh pseudonyms produced by the previous server,
$\text{outputServerEncryptedScores}_{k-1}$: The shuffled reputation scores encrypted to the servers' shared secret key, as produced by the previous server,
$\text{outputVoteMatrix}_{k-1}$: The shuffled encrypted votes, as produced by the previous server,
$\text{outputUserEncryptedScores}_{k-1}$: The shuffled reputation scores encrypted to the users' long-term secret keys, as produced by the previous server

$\text{outputPseudonyms}_{k-1} \leftarrow$ `compressMatrix`(pseudonymMatrix);
$\text{outputServerEncryptedScores}_{k-1} \leftarrow$ `compressMatrix`(serverEncryptedScoreMatrix);
for $\ell \leftarrow 1 \ldots n$ **do**
  $\text{outputVoteMatrix}_{k-1}[\ell, \ldots] \leftarrow$ `compressMatrix`(voteTensorPartB$[k]$);
end
$\text{outputUserEncryptedScores}_{k-1} \leftarrow \perp$;
if *userEncryptedScoreMaskMatrix* $\neq \perp \wedge$ *userEncryptedScoreMessageMatrix* $\neq \perp$ **then**
  $\vec{a} \leftarrow$ `compressMatrix`(userEncryptedScoreMaskMatrix);
  $\vec{a'} \leftarrow$ `compressMatrix`(userEncryptedScoreMessageMatrix);
  $\text{outputUserEncryptedScores}_{k-1} \leftarrow \langle (A_j, A'_j) : j \in [1, n] \rangle$;
end
return $\text{outputPseudonyms}_{k-1}$, $\text{outputServerEncryptedScores}_{k-1}$, $\text{outputVoteMatrix}_{k-1}$, $\text{outputUserEncryptedScores}_{k-1}$;

Algorithm 12 invokes Algorithm 3 repeatedly on the output matrices from Algorithm 9, so that it can output the shuffled (and rerandomized, and/or with the server's per-epoch factor applied) data. This subroutine is used after a server has verified the ZKPs of correct shuffle of another server in the covert setting, or upon receiving the shuffle output in the honest-but-curious setting.

Algorithm 13 is the counterpart to Algorithm 9. In the covert setting, every server must verify and accept the shuffled data of every other server after that server's turn, so that ZKPs can be correctly generated and verified for a server during its own turn to shuffle data. In the honest-but-curious setting, this subroutine is still necessary to compress the output matrices into their useful form as shuffled vectors.

Algorithm 13: acceptShuffledData

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
$\mathfrak{g}_{\ell+1}$: The epoch generator for the next epoch $(\ell + 1)$,
isBreakdown: Boolean value representing whether the server is executing the Break-down Phase or Build-up Phase,
outputData: The previous server's output from `covertShuffleAndApplyFactor`,
π_{epoch}: A ZKP that the data was shuffled correctly,
currentPseudonyms: Either the previous epoch's fresh pseudonyms, the partway pseudonyms between epochs, or the output shuffled pseudonyms of the server preceding the one whose shuffles are being evaluated,
serverEncryptedScores: The scores as they are encrypted to the servers' shared secret key (or the output shuffled such scores of the server preceding the one whose shuffles are being evaluated),
voteMatrix: The matrix containing encrypted votes users cast for one another (or the reordered version output by the server preceding the one whose shuffles are being evaluated),
userEncryptedScores: The scores as they are encrypted to the users' long-term secret keys (or the output shuffled such scores of the server preceding the one whose shuffles are being evaluated)

Output: $\text{outputPseudonyms}_{k-1}$: The shuffled fresh pseudonyms produced by the previous server,
$\text{outputServerEncryptedScores}_{k-1}$: The shuffled reputation scores encrypted to the servers' shared secret key, as produced by the previous server,
$\text{outputVoteMatrix}_{k-1}$: The shuffled encrypted votes, as produced by the previous server,
$\text{outputUserEncryptedScores}_{k-1}$: The shuffled reputation scores encrypted to the users' long-term secret keys, as produced by the previous server

/* outputData and π_{epoch} are tuples of the variables output by `covertShuffleAndApplyFactor()` */

$\vec{B} \leftarrow$ permutationCommitment;

verifyPermutation $\leftarrow$ `verifyProof`$(\mathfrak{g}, \mathfrak{h}, \vec{B}, \pi_{\text{perm}}, \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(p_{i,j}, s_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge p_{i,j} \in [0,1]\})$;

verifyPseudonym $\leftarrow$ `verifyProof`$(\mathfrak{g}, \mathfrak{h}, \langle A \rangle \leftarrow$currentPseudonyms$, \vec{B}, \vec{D} \leftarrow$pseudonymMatrix$, \vec{E} \leftarrow$pseudonymSeedMatrix$, \pi_{\text{pseud}}, \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(r, p_{i,j}, s_{i,j}, s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge D_{i,j} = A_j^{p_{i,j} r} \mathfrak{h}^{s'_{i,j}} \wedge E_{i,j} = \mathfrak{g}^{s'_{i,j}}\})$;

verifyServerEncrypted $\leftarrow$ `verifyProof`$(\mathfrak{g}, \mathfrak{h}, \langle A \rangle \leftarrow$ serverEncryptedScores$, \vec{B}, \vec{C} \leftarrow$serverEncryptedScoreMatrix$, \pi_{\text{serverEncrypted}}, \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(p_{i,j}, s_{i,j}, s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge C_{i,j} = A_j^{p_{i,j}} h'^{s'_{i,j}}\})$;

verifyVoteTensors $\leftarrow$ `verifyVoteTensorProof`$(\mathfrak{g}, \mathfrak{h},$ permutationCommitment, voteTensorPartA, voteTensorPartB, $\pi_{\text{voteA}}, \pi_{\text{voteB}},$ voteMatrix);

verifyUserEncrypted $\leftarrow \top$;

if *isBreakdown* $= \top$ **then**

  verifyUserEncrypted $\leftarrow$ `verifyProof`$(\mathfrak{g}, \mathfrak{h}, \mathfrak{g}_{\ell+1},$
  $\langle A \rangle \leftarrow \langle C_1 : (C_1, C_2) \in$ userEncryptedScores$\rangle$,
  $\langle A' \rangle \leftarrow \langle C_2 : (C_1, C_2) \in$ userEncryptedScores$\rangle, \langle X_{\ell/\ell+1} \rangle \leftarrow$ currentPseudonyms$, \vec{B},$
  $\vec{E} \leftarrow$userEncryptedScoreMaskSeedMatrix$, \vec{F} \leftarrow$userEncryptedScoreMaskMatrix,
  $\vec{H} \leftarrow$userEncryptedScoreMessageMatrix$, \pi_{\text{userEncrypted}}, \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(r, p_{i,j}, s_{i,j}, s'_{i,j}) :$
  $B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge E_{i,j} = \mathfrak{g}^{s'_{i,j}} \wedge F_{i,j} = A_j^{p_{i,j} r} X_{j,\ell/\ell+1}^{p_{i,j} r s'_{i,j}} \mathfrak{h}^{s'_{i,j}} \wedge H_{i,j} = A_j'^{p_{i,j}} \mathfrak{g}_{\ell+1}^{p_{i,j} s'_{i,j}} \mathfrak{h}^{s'_{i,j}}\})$;

end

(outputPseudonyms$_{k-1}$, outputServerEncryptedScores$_{k-1}$, outputVoteMatrix$_{k-1}$, outputUserEncryptedScores$_{k-1}$) $\leftarrow$ `compressAllData`(outputData);

/* `checkUnique()` returns $\top$ if every element in a vector is unique, and $\bot$ otherwise */

verifyPermutation $\leftarrow$ verifyPermutation $\wedge$ `checkUnique`(outputPseudonyms$_{k-1}$);

if verifyPermutation $= \bot \vee$ verifyPseudonym $= \bot \vee$ verifyServerEncrypted $= \bot \vee$ verifyVoteTensors $= \bot \vee$ verifyUserEncrypted $= \bot$ **then**

  return $\bot$;

end

return (outputPseudonyms$_{k-1}$, outputServerEncryptedScores$_{k-1}$, outputVoteMatrix$_{k-1}$, outputUserEncryptedScores$_{k-1}$);

In addition to this pseudocode, we also include diagrams to help describe what each phase does. The visual notation that we use in these diagrams is outlined in Figure 4.1; in general, single-lined circles indicate elements of the ElGamal group (the group output by **EGGroupGenKey()**), double-lined circles designate ElGamal ciphertexts (as output by **EGEncrypt()**, circular sectors designate BGN ciphertexts in G or H (as output by **BGNEncrypt()**), and semicircles designate BGN ciphertexts in G_T (as output by **BGNMultiply()**. Further, rotation of the exterior portion of double-lined circles, as well as of circular sectors and semicircles indicates the randomization currently being used as a blinding factor in that ciphertext.

4.10.1 Build-up Phase

In the first phase, shown visually in Figure 4.2, the servers "build up" on top of users' fresh pseudonyms for the previous epoch (which we will call ℓ), in addition to building up a new fresh generator for the next epoch ($\ell + 1$).

To do so, the servers begin with the current fresh pseudonyms and $\mathfrak{g}$, the generator of the group $\mathfrak{G}$ that was output by **EGGroupGenKey()**. Each server operates in turns; the order does not strictly matter, but must be agreed upon ahead of time for purposes of synchonization. The pseudocode for the algorithm used during server k's turn can be seen in Algorithm 14; in the covert setting, during other servers' turns, server k executes Algorithm 13 to accept the shuffled data (which is necessary to do in order to correctly verify later ZKPs; servers in the honest-but-curious setting do nothing during other servers' turns). During server k's turn, first, it chooses a random epoch factor $\mathfrak{r}_{k,\ell+1} \xleftarrow{\$} [1, q-1]$. Then, if k is first, it applies its (random) epoch factor to $\mathfrak{g}$; otherwise, it applies the epoch factor to the output partial epoch generator from the previous server. The epoch generator for the next epoch is thus the result of each server applying their epoch factor to $\mathfrak{g}$ in turn.

The server also applies its epoch factor to each of the fresh pseudonyms (or the output partway fresh pseudonyms from the previous server, as appropriate). It then randomly shuffles the partway fresh pseudonyms (along with every piece of data tagged by the fresh pseudonyms, with the same random shuffle). In the honest-but-curious setting, the server passes this shuffled data on to the next server. By this, we expect the honest-but-curious setting to have quadratic complexity, as the honest-but-curious algorithm is dominated by its need to rerandomize $O(n^2)$ elements of the vote matrix, and rerandomization is an $O(1)$ operation. In the covert setting, server k generates a ZKP that it applied the same shuffle to all the data, and passes the data along with the ZKP to all other servers, who verify the ZKP before the next server begins their turn. Discussion of the covert setting's computational complexity requires more information about the ZKP.

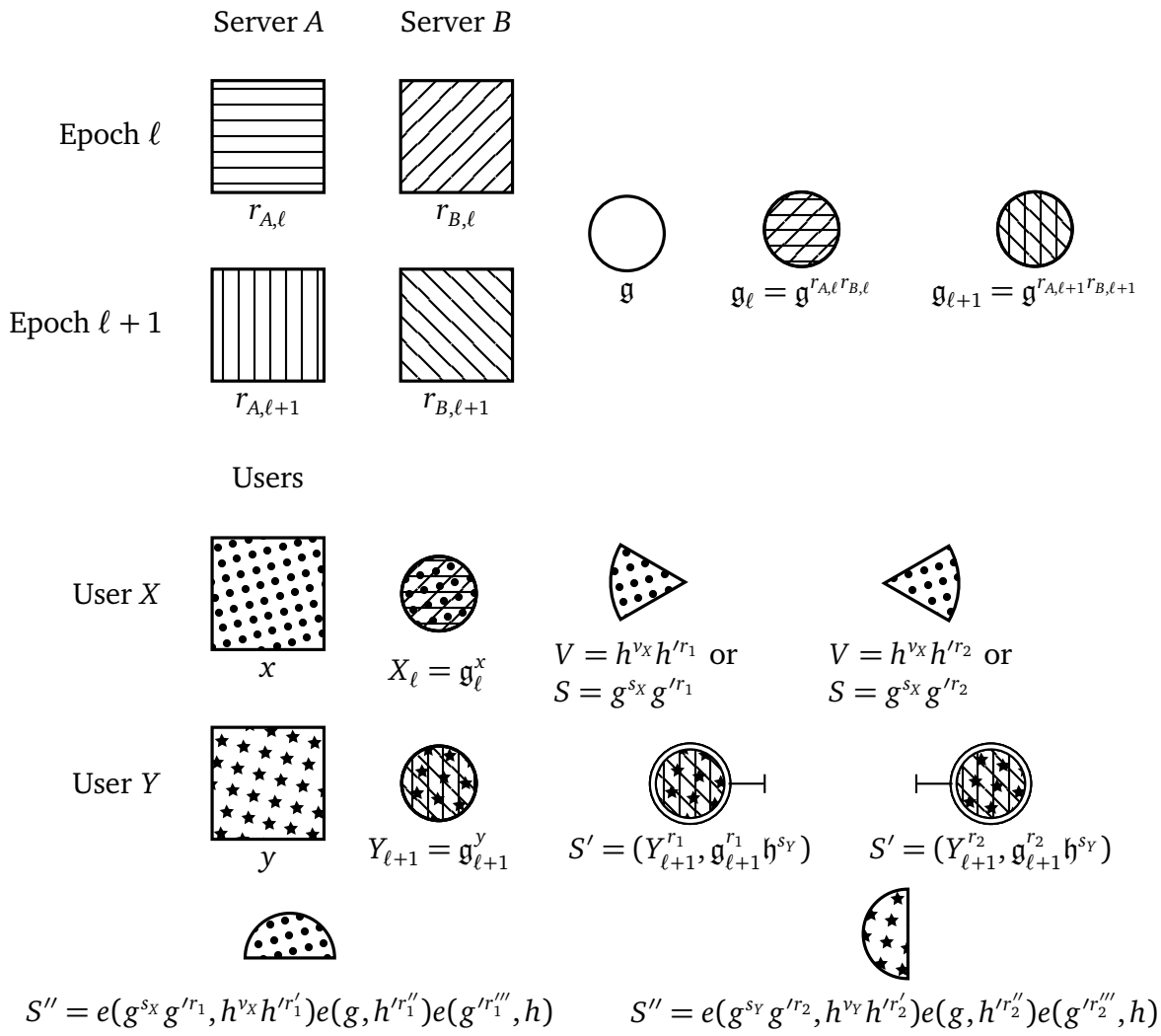


Figure 4.1. A legend to describe the visual notation observed in the diagrams for each phase of the inter-epoch calculations. Single-outlined circles designate singular elements of the ElGamal group (potentially with patterns/factors applied to them). Double-outlined circles designate ElGamal ciphertexts. Circular sectors designate BGN ciphertexts in G or H (*i.e.*, those that have not yet been multiplied). Semicircles designate BGN ciphertexts in G_T (*i.e.*, those that have been multiplied). Rotation of the whiskers of double-lined circles and of circular sectors and semicircles indicates randomization.

This ZKP is more elaborate than the previous ZKPs used here. The intuition for this ZKP is as follows. Imagine that the list of fresh pseudonyms is a vector. Next, imagine a permutation matrix — a matrix whose elements are all exclusively 0 or 1, and for which it is true that both every row and every column sums to 1 — for which the matrix-vector product with the list of fresh pseudonyms is the shuffled list of fresh pseudonyms. In the ZKP, a server commits to such a permutation matrix, then proves that the output fresh pseudonyms are the matrix-vector product of that permutation matrix raised to a new, secret factor. Further, the server proves how the other reordered datasets can be generated as its own matrix-vector product with the same permutation matrix, which is homomorphically added to by a new, secret vector of encryptions of the value 0.

We show the notation for the ZKP in parts. First, we must prove that we have a valid permutation matrix ($\mathbf{P}$). Let us call each element of $\mathbf{P}$ $p_{i,j}$. Our Pedersen commitment permutation matrix is $\mathbf{B}$, and its elements are $B_{i,j} = \mathfrak{g}^{p_{i,j}}\mathfrak{h}^{s_{i,j}}$. We choose the $s_{i,j}$ values carefully such that $\sum_{i=1}^{n} s_{i,j} = 0 \bmod q$ for each j. We prove that each $B_{i,j}$ is a commitment to 0 or 1: $\{(p_{i,j}, s_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}}\mathfrak{h}^{s_{i,j}} \wedge p_{i,j} \in [0,1]\}$. With that, assuming that no server can know $d \in \mathbb{Z}_q$ such that $\mathfrak{g} = \mathfrak{h}^d$, a verifier can calculate $B_j = \prod_{i=1}^{n} B_{i,j} \stackrel{?}{=} \mathfrak{g}$. If this holds, this suffices to show that each row of the permutation matrix has only one 1 in it. The proof that each column of the permutation matrix has only one 1 in it follows implicitly from a check that will happen later on as part of verifying this proof: if the permutation matrix is correctly applied to the fresh pseudonyms, and there are no duplicates among the fresh pseudonyms, there can only be one 1 in each column of the permutation matrix. We know this is true because instead of randomizing fresh pseudonyms, they all have the same random factor applied to them, so taking the same pseudonym as input twice will necessarily result in duplicate outputs. A verifier need only confirm that there are no such duplicates to complete the proof that $\mathbf{P}$ is a valid permutation matrix, and this is present in Algorithm 13, but is not otherwise marked in the ZKP parts that follow.

Next, some stored data only needs to be reordered and rerandomized by the permutation matrix. This applies to the vote matrix ($\mathbf{V}$) and to the server-encrypted reputation scores. In order to correctly permute both the rows and the columns of $\mathbf{V}$, we actually calculate $\mathbf{P}^T\mathbf{V}\mathbf{P}$, and we do so by each row (or column, as appropriate). Now, suppose $\vec{a} = (A_1, \ldots, A_j, \ldots A_n)$ is such a vector we want to reorder. (The elements of the vote matrix and the server-encrypted reputation scores will be in G and H (two of the BGN groups)). In addition to the Pedersen commitment permutation matrix from before, there is additionally an intermediary matrix $\mathbf{C}$, whose elements are $C_{i,j} = A_j^{p_{i,j}}h'^{s'_{i,j}}$. In this case, $\forall s'_{i,j}, s'_{i,j} \stackrel{\$}{\leftarrow} [1, |H_1|]$ (that is, all $s'_{i,j}$s are chosen completely at random). With this in place, we include the following ZKP part:

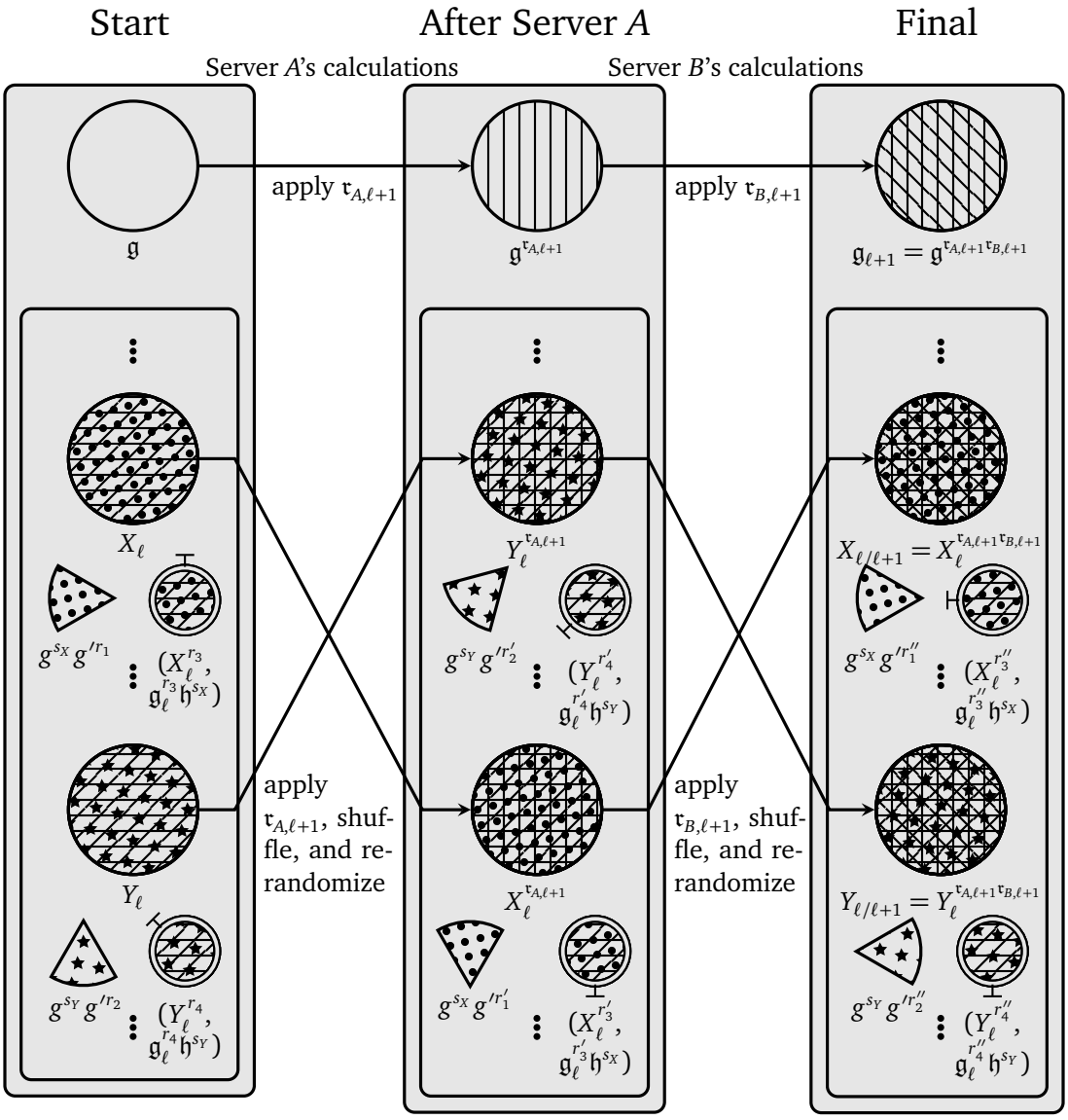


Figure 4.2. A diagram representing the Build-up Phase of the inter-epoch calculations between Epochs ℓ and $\ell + 1$. A 2-server arrangement is shown for simplicity; more servers can be added by repeating the actions. Note that all data that is tagged by user's fresh pseudonyms (*e.g.*, the vote matrix and users' encrypted scores) is also shuffled with the same swaps as the fresh pseudonyms, at the same time. Further, this associated data is also re-randomized during the shuffle, to avoid leaking the permutation of the shuffle itself.

Algorithm 14: buildUp

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
PK_{BGN}: The BGN public key,
k: Which server this is (in the order of precedence for conducting the Build-up Phase),
partialEpochGenerator: The previous server's output partially constructed epoch generator for the next epoch (or $\mathfrak{g}$ if $k = 1$),
outputData$_{k-1}$: The output data from the previous server (or $\perp$ if $k = 1$),
$\pi_{\text{epoch},k-1}$: The previous server's ZKP that their shuffle was done correctly (or $\perp$ if $k = 1$),
previousPseudonyms: The previous epoch's fresh pseudonyms ($\vec{x_\ell}$) if $k \leq 2$, or the output shuffled pseudonyms of the server preceding the one whose shuffles are being evaluated if $k > 2$,
previousServerEncryptedScores: The previous epoch's server encrypted scores if $k \leq 2$, or the output shuffled server encrypted scores of the server preceding the one whose shuffles are being evaluated if $k > 2$,
previousVoteMatrix: The previous epoch's vote matrix if $k \leq 2$, or the output shuffled vote matrix of the server preceding the one whose shuffles are being evaluated if $k > 2$

Output: partialEpochGenerator: The partially calculated epoch generator for the next epoch,
outputData$_k$: The shuffled data,
$\pi_{\text{epoch},k}$: A ZKP that the data was shuffled correctly

if $k = 1$ **then**
  currentPseudonyms ← previousPseudonyms;
  serverEncryptedScores ← previousServerEncryptedScores;
  voteMatrix ← previousVoteMatrix;
else
  if *isCovert* **then**
    currentPseudonyms, serverEncryptedScores, voteMatrix,
     $\perp$ ← `acceptShuffledData`($\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{g}_{\ell+1} \leftarrow \perp$, isBreakdown ← $\perp$,
     outputData_{k-1}, $\pi_{\text{epoch},k-1}$, previousPseudonyms,
     previousServerEncryptedScores, previousVoteMatrix,
     previousUserEncryptedScores);
  else
    (currentPseudonyms, serverEncryptedScores, voteMatrix,
     userEncryptedScores) ← outputData_{k-1};
  end
  if currentPseudonyms = $\perp \vee$ serverEncryptedScores = $\perp \vee$ voteMatrix = $\perp$ **then**
    return $\perp$;
  end
end
$\mathfrak{r}_{k,\ell+1} \xleftarrow{\$} [1, q-1]$;
partialEpochGenerator ← $\text{partialEpochGenerator}^{\mathfrak{r}_{k,\ell+1}}$;
if *isCovert* **then**
  return partialEpochGenerator, `covertShuffleAndApplyFactor`($\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{g}_{\ell+1} \leftarrow \perp$,
   $\mathfrak{r} \leftarrow \mathfrak{r}_{k,\ell+1}$, isBreakdown ← $\perp$, currentPseudonyms, serverEncryptedScores,
   voteMatrix, userEncryptedScores ← $\perp$);
else
  return partialEpochGenerator, `hbcShuffleAndApplyFactor`(PK_{BGN}, $\mathfrak{g}_{\ell+1} \leftarrow \perp$,
   $\mathfrak{r} \leftarrow \mathfrak{r}_{k,\ell+1}$, isBreakdown ← $\perp$, currentPseudonyms, serverEncryptedScores,
   voteMatrix, userEncryptedScores ← $\perp$);
end

$$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\{(p_{i,j},s_{i,j},s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}}\mathfrak{h}^{s_{i,j}} \wedge C_{i,j} = A_j^{p_{i,j}}h'^{s'_{i,j}}\}$$

Similar to the permutation matrix before, the verifier then calculates $\vec{c_i} = \langle\prod_{j=1}^{n} C_{i,j}\rangle_{i\in[1,n]}$. As there is only one 1 in each row of the permutation matrix, most of the $A_j^{p_{i,j}}$ terms drop away, and the remaining $h'^{s'_{i,j}}$ terms rerandomize the element. The resulting c_i values are the re-ordered and rerandomized vector we sought.

The size and complexity of this ZKP scales quadratically with the number of elements being reordered. This is true of all the ZKP parts we describe, as they all involve transforming an n-element vector into an n-by-n element matrix (*i.e.*, to touch every element in the resulting matrix, it is necessarily the case that there are $O(n^2)$ operations). However, the overall size and complexity of the ZKP for a correct shuffle is actually cubic — this is because in order to reorder a vote matrix specifically, there are n different n-element vectors that must be transformed into n-by-n element matrices (that is, there are $O(n^2)$ operations per matrix, and n matrices, resulting in an $O(n^3)$ complexity).

Some stored data needs to be reordered by the permutation matrix while also applying the server's epoch factor $\mathfrak{r}_{k,\ell+1}$ to them, without doing any form of rerandomization. This is true of the fresh pseudonyms. As with before, we will make use of the Pedersen commitment permutation matrix. We also use two new intermediary matrices, $\mathbf{D}$ (whose elements are $D_{i,j} = A_j^{p_{i,j}\mathfrak{r}_{k,\ell+1}}\mathfrak{h}^{s'_{i,j}}$) and $\mathbf{E}$ (whose elements are $E_{i,j} = \mathfrak{g}^{s'_{i,j}}$). In this case, the $s'_{i,j}$s are chosen such that $\sum_{j=1}^{n} s'_{i,j} = 0 \bmod q$ for each i. With this in place, we include the following ZKP part:

$$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\{(\mathfrak{r}_{k,\ell+1},p_{i,j},s_{i,j},s'_{i,j}) : B_{i,j} = \mathfrak{g}^{p_{i,j}}\mathfrak{h}^{s_{i,j}} \wedge D_{i,j} = A_j^{p_{i,j}\mathfrak{r}_{k,\ell+1}}\mathfrak{h}^{s'_{i,j}} \wedge E_{i,j} = \mathfrak{g}^{s'_{i,j}}\}$$

As with before, the verifier then calculates $D_i = \prod_{j=1}^{n} D_{i,j}$ and $\prod_{j=1}^{n} E_{i,j} \stackrel{?}{=} \mathfrak{g}^0$. Assuming the $E_{i,j}$ check passes (confirming that the $s'_{i,j}$s canceled out), then the D_is are the reordered fresh pseudonyms with the $\mathfrak{r}_{k,\ell+1}$ factor applied. This is also the point when the verifier would confirm that there are no duplicate fresh pseudonyms, finishing the confirmation that the permutation matrix was valid.

In practice, the user-encrypted reputation scores are not reordered during this phase, but are instead just ignored entirely. This is because they will be recalculated and replaced in the next two phases.

For all of these ZKP parts, proof batching techniques, such as those suggested by Henry and Goldberg [HG13], can be used to make the proofs more efficient, at the expense of introducing a potential soundness error (that is, a small chance that an incorrect proof will verify when it should not). Proof batching involves choosing a batch parameter λ, and the chance of proof failure occurring is typically something along the lines of $2^{-\lambda}$. As such, $\lambda = 50$ would indicate that a proof that is generated once per second would not be expected to incorrectly verify for 2^{24} years. In our implementation, we also implement proof batching, and test with $\lambda = 50$; this choice is appropriate for the usage expected with PRSONA in practical settings.

At the end of this process, the servers will have collaboratively generated the epoch generator for epoch $\ell + 1$, and a set of pseudonyms that is related to the fresh pseudonyms for both the previous and next epochs, without being linkable to either (we call these "partway pseudonyms", and use the notation $X_{\mathrm{I/II}}$ to refer to such a pseudonym for user X between Epochs I and II). In this state, none of the users and none of the servers individually are able to determine which of the partway pseudonyms apply to which user.

4.10.2 Decryption and Re-encryption Phases

The next two phases, shown in Figure 4.3, are closely related to each other. In both phases, there is no difference between the covert setting and the honest-but-curious setting. Once the partway pseudonyms have been "built up", the servers collaboratively work to calculate the new reputation scores for each user. Votes in the vote matrix are encrypted as elements in H, and associated with both the voter's and votee's partway pseudonyms. Reputation scores from Epoch ℓ are encrypted as elements in G, and associated with the partway pseudonym of the user they apply to; we term the vector form of these scores $\vec{s}$. Each server has their own copy of this data.

First, as a setup step, servers independently compute (in encrypted form) the matrix-vector product of the vote matrix and the reputation scores from Epoch ℓ by using **BGNMultiply()** and **BGNAdd()** as appropriate (suitable values for the g_1 and h_1 values described in those functions can be agreed upon by the servers in advance, to ensure that they all calculate the same output values correctly). Specifically, each element in the vote matrix is multiplied by the reputation score of its voter, then added with the other votes for the same votee, to achieve the weighting of Short-Term Memory Consensus — Iterated Weighting. Because each row in the vote matrix is associated with one voter, and each column with one votee, this is simple to do. The resulting vector consists of elements of G_T, in the same order as $\vec{s}$ (and therefore still associated with the correct partway pseudonyms).

PRSONA reputation scores are limited to the range $[0, 2n]$, for n the number of users.

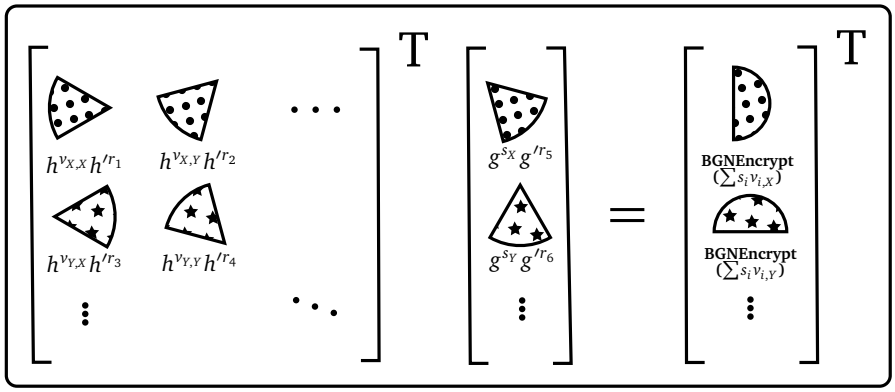
However, these output values can be in range $[0, 4n^2]$ (the maximum being the case when every voter has reputation score $2n$ and gives a positive vote (*i.e.*, 2), to the same votee). The range of reputation scores PRSONA displays is intended to represent sentiment at time of vote, so output values are scaled according to the maximum score that was possible to achieve during the epoch. That is, servers also independently calculate the sum of all elements of $\vec{s}$ (again using **BGNAdd()**), then add that value to itself to calculate $m = 2 \sum_{S_i \in \vec{s}} S_i$, which is the value that one votee could have achieved if every voter rated them positively. Because this calculation involves no multiplications, the resulting value is an element of G, just as the members of $\vec{s}$ are.

Next, the Decryption Phase begins. Servers confirm that they all hold the same values in the output vector, as well as the same value for the maximum possible score, then collaboratively execute **BGNDecrypt()** on these. The result is that each server holds a plaintext version of the output vector, where each element is associated with the partway pseudonym of the user it applies to, and a plaintext version of the maximum possible score. Servers then independently calculate $s'_i = \left\lfloor \frac{2ns_i}{m} \right\rfloor$, for s_i the plaintext element of the output vector corresponding to user i. (If every user had a reputation of 0, m would also be 0, leading to undefined behaviour in this calculation. This is why the default reputation of a new user is deliberately chosen to be nonzero.) The resulting $\vec{s'}$ represents the plaintext reputation scores that will apply to each user in the next epoch, which are correctly scaled down to the range $[0, 2n]$, for the minimum of that range representing those votees who received the most negative ratings, and the maximum of that range representing those votees who received the most positive ratings, up to $2n$ in cases where all voters rated them positively.

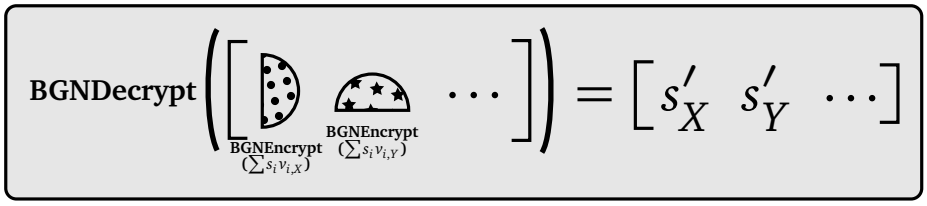
Once these scaled plaintext values have been calculated, servers move forward to the Re-encryption Phase. In this phase, servers encrypt the plaintext scores in two ways. First, they encrypt them as elements of G using **BGNEncrypt()**, to serve as the weights in the Short-Term Memory Consensus — Iterated Weighting calculation during the next inter-epoch calculations. In doing so, servers either agree at the time or pre-emptively on appropriate values of g_1 to use, so that all servers have identical copies of these newly encrypted values. Second, servers encrypt the scores as elements of $\mathfrak{G}^2$ using a variation of **EGEncrypt()**.

Ordinarily, the output of **EGEncrypt()** is encrypted to a pseudonym $X_{i,\ell}$ with form $(X^r_{i,\ell}, \mathfrak{g}^r_\ell \mathfrak{h}^{s'_i})$, for $r \in [1, q-1]$ some random blinding factor and s'_i the scaled score of the user in plaintext. However, the servers have not yet calculated the fresh pseudonyms $X_{\ell+1}$. They did calculate $\mathfrak{g}_{\ell+1}$ during the Build-up Phase, however. With this, the servers encrypt a score s_i like so: $(X^r_{i,\ell/\ell+1}, \mathfrak{g}^r_{\ell+1} \mathfrak{h}^{s'_i})$. Any individual server can compute this encryption, but a majority of servers must agree that it represents the correct score.

These values can be decrypted by any individual server during this Re-encryption Phase,



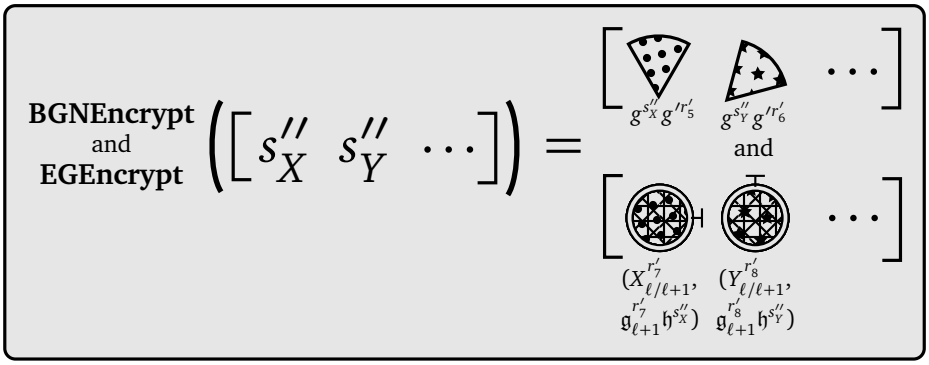
Servers calculate $\vec{S}$ individually



Servers calculate **BGNDecrypt($\vec{S}$)** together

$$\left\lfloor \frac{2n}{m} \begin{bmatrix} s'_X & s'_Y & \cdots \end{bmatrix} \right\rfloor = \begin{bmatrix} s''_X & s''_Y & \cdots \end{bmatrix}$$

Servers calculate $\vec{s''}$ individually



Servers agree upon values for server-encrypted and user-encrypted reputation scores

Figure 4.3. A diagram representing the Decryption and Re-encryption Phases of the inter-epoch calculations. Shaded boxes indicate calculations that servers do collaboratively, while unshaded boxes indicate calculations that servers do individually.

as the servers will know the r value used to blind the score. This does not represent a leakage, as these servers already saw these values in plaintext during the Decryption and Re-encryption Phases before this point. These values cannot be decrypted by a user unless the r values are leaked by a covert server; a covert server could already leak the distribution of plaintext scores that it already knew, which represents the same leakage to users, as these scores are not connected to any recognizable pseudonym for any user. Other than the covert server leak scenario, users cannot normally decrypt these values because $X_{i,\ell/\ell+1} \neq \mathfrak{g}_{\ell+1}^{x}$. Importantly, though, during the Break-down Phase, the servers will manipulate these encrypted values to put them back into a form that users will be able to correctly decrypt, while also rerandomizing the encryptions with r' blinding factors that the other servers do *not* know. Due to this rerandomization aspect, these encryptions will not be possible for other servers to decrypt when associated with users' fresh pseudonyms for the new epoch, as long as any one server honestly rerandomized the scores. As with the outputs of **BGNEncrypt()**, appropriate values of r can be agreed upon at the time or pre-emptively by the servers, so that each server has an identical copy of the outputs of **EGEncrypt()**. In both cases, this is specifically necessary so that ZKPs can be verified by servers accurately.

4.10.3 Break-down Phase

In the final phase, shown in Figure 4.4, the servers "break down" partway pseudonyms into users' fresh pseudonyms for the next epoch (as well as the user-encrypted reputation scores, in the same manner).

During Epoch ℓ, servers each chose a (random) epoch factor $\mathfrak{r}_{k,\ell}$ to apply to $\mathfrak{g}$ so that they could collaboratively construct $\mathfrak{g}_\ell$. The partway pseudonyms are related to the fresh pseudonyms for Epoch ℓ because they still have those epoch factors from Epoch ℓ applied to them. As in the Build-up Phase, each server operates in turns, and again the order does not strictly matter, but must be agreed upon ahead of time for purposes of synchonization. The pseudocode for the algorithm used during server k's turn can be seen in Algorithm 15; in the covert setting, during other servers' turns, server k again executes Algorithm 13 to accept the shuffled data (which is necessary to do in order to correctly verify later ZKPs; servers in the honest-but-curious setting do nothing during other servers' turns). During server k's turn, first, it calculates $\mathfrak{r}_{k,\ell}^{-1} \bmod q$. Then, k applies this inverse epoch factor to each of the partway pseudonyms (or the output partway pseudonyms from the previous server, as appropriate). Like in the Build-up Phase, the server then randomly shuffles the partway pseudonyms (along with every piece of data tagged by the pseudonyms, with the same random shuffle).

In the honest-but-curious setting, the server passes this shuffled data on to the next server.

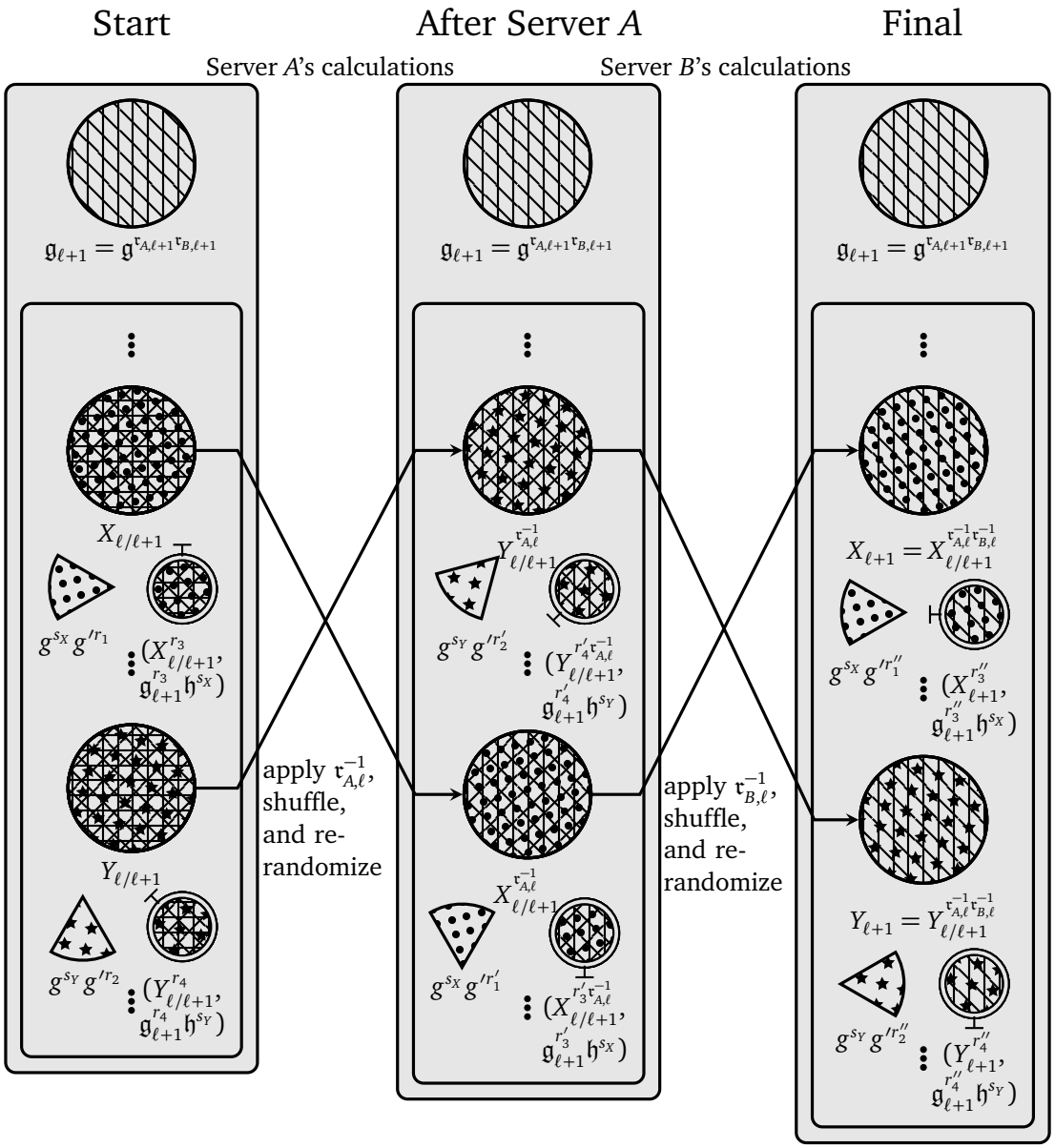


Figure 4.4. A diagram representing the Break-down Phase of the inter-epoch calculations between Epochs ℓ and $\ell+1$. A 2-server arrangement is shown for simplicity; more servers can be added by repeating the actions. Note that all data that is tagged by user's fresh pseudonyms (*e.g.*, the vote matrix and users' encrypted scores) is also shuffled with the same swaps as the fresh pseudonyms, at the same time. Further, this associated data is also re-randomized during the shuffle, to avoid leaking the permutation of the shuffle itself.

As with the Build-up Phase, rerandomizing the vote matrix dominates the honest-but-curious Break-down Phase algorithm, resulting in an expected quadratic computational complexity. Across all phases, the greatest complexity any phase faces in the honest-but-curious setting is a quadratic complexity, and each phase executes independently. This leads us to conclude that the honest-but-curious setting should expect quadratic complexity for the epoch changeover calculations as a whole. In the covert setting, server k generates a ZKP that it applied the same shuffle to all the data, and passes the data along with the ZKP to all other servers, who verify the ZKP before the next server begins their turn. As with the Build-up Phase, discussion of the covert setting Break-down Phase algorithm's computational complexity requires more knowledge of this specific ZKP.

This ZKP is almost identical to that of the Build-up Phase. Wherever the Build-up Phase refers to $\mathfrak{r}_{k,\ell+1}$, it can simply be replaced with $\mathfrak{r}_{k,\ell}^{-1} \bmod q$ without issue. The ZKP for the Break-down Phase has one notable addition on top of the ZKP for the Build-up Phase. During the Build-up Phase, user-encrypted reputation scores were ignored, because they are recalculated during the Decryption and Re-encryption Phases. During the Break-down Phase, however, user-encrypted reputation scores cannot be ignored. However, their shuffle requires a slightly altered ZKP from the two types of data that do have ZKPs during the Build-up Phase.

Algorithm 15: breakDown

Input: $\mathfrak{g}$: One of the generators of the ElGamal group,
$\mathfrak{h}$: Another generator of the ElGamal group, chosen in a manner where no entity knows x such that $\mathfrak{g}^x = \mathfrak{h}$,
PK_{BGN}: The BGN public key,
k: Which server this is (in the order of precedence for conducting the Break-down Phase),
$\mathfrak{g}_{\ell+1}$: The next epoch's epoch generator,
$\mathfrak{r}_{k,\ell}$: The per-epoch factor this server used in the previous epoch's epoch generator,
outputData$_{k-1}$: The output data from the previous server (or $\perp$ if $k = 1$),
$\pi_{\text{epoch},k-1}$: The previous server's ZKP that their shuffle was done correctly (or $\perp$ if $k = 1$),
previousPseudonyms: The previous epoch's fresh pseudonyms ($\vec{x_\ell}$) if $k \leq 2$, or the output shuffled pseudonyms of the server preceding the one whose shuffles are being evaluated if $k > 2$,
previousServerEncryptedScores: The previous epoch's server encrypted scores if $k \leq 2$, or the output shuffled server encrypted scores of the server preceding the one whose shuffles are being evaluated if $k > 2$,
previousVoteMatrix: The previous epoch's vote matrix if $k \leq 2$, or the output shuffled vote matrix of the server preceding the one whose shuffles are being evaluated if $k > 2$previousUserEncryptedScores: The previous epoch's iser encrypted scores if $k \leq 2$, or the output shuffled user encrypted scores of the server preceding the one whose shuffles are being evaluated if $k > 2$,

Output: outputData$_k$: The shuffled data,
$\pi_{\text{epoch},k}$: A ZKP that the data was shuffled correctly

if $k = 1$ **then**
  currentPseudonyms ← previousPseudonyms;
  serverEncryptedScores ← previousServerEncryptedScores;
  voteMatrix ← previousVoteMatrix;
  userEncryptedScores ← previousUserEncryptedScores;
else
  if *isCovert* **then**
    currentPseudonyms, serverEncryptedScores, voteMatrix, userEncryptedScores
     ← `acceptShuffledData`($\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{g}_{\ell+1}$, isBreakdown ← $\top$, outputData_{k-1},
     $\pi_{\text{epoch},k-1}$, previousPseudonyms, previousServerEncryptedScores,
     previousVoteMatrix, previousUserEncryptedScores);
  else
    (currentPseudonyms, serverEncryptedScores, voteMatrix,
     userEncryptedScores) ← outputData_{k-1};
  end
  if currentPseudonyms $= \perp \vee$ serverEncryptedScores $= \perp \vee$ voteMatrix $= \perp \vee$
   userEncryptedScores $= \perp$ **then**
    return $\perp$;
  end
end
if *isCovert* **then**
  return `covertShuffleAndApplyFactor`($\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{g}_{\ell+1}$, $\mathfrak{r} \leftarrow \mathfrak{r}_{k,\ell}^{-1}$, isBreakdown ← $\top$,
   currentPseudonyms, serverEncryptedScores, voteMatrix, userEncryptedScores);
else
  return `hbcShuffleAndApplyFactor`(PK_{BGN}, $\mathfrak{g}_{\ell+1}$, $\mathfrak{r} \leftarrow \mathfrak{r}_{k,\ell}^{-1}$, isBreakdown ← $\top$,
   currentPseudonyms, serverEncryptedScores, voteMatrix, userEncryptedScores);
end

Recall that user-encrypted reputation scores are (usually) the output of **EGEncrypt()** with form $(X_\ell^r, \mathfrak{g}_\ell^r \mathfrak{h}^s)$, for X_ℓ the fresh pseudonym of some user during Epoch ℓ, r a random blinding factor, and s the user's reputation score. Recall also that the "user-encrypted reputation scores" obtained during the Re-encryption Phase actually have form $(X_{\ell/\ell+1}^r, \mathfrak{g}_{\ell+1}^r \mathfrak{h}^s)$, because servers did not have access to the fresh pseudonyms for Epoch $\ell+1$ during the Re-encryption Phase. As the ciphertext is a tuple, there are two different operations that need to be done on each piece of the tuple. The first part, much like the fresh and partway pseudonyms during the Build-up Phase, need to have a factor applied to them in addition to being reordered (specifically, $\mathfrak{r}_{k,\ell}^{-1} \bmod q$). The second part, much like the vote matrix and server-encrypted reputation scores, needs to be reordered and rerandomized — and importantly, the same rerandomization needs to *also* be applied to the first part of the tuple, so that the encrypted value can still be properly decrypted after the shuffle operation. If not for this requirement, each part of the tuple could be reordered separately using pieces of the ZKP we have already defined.

As with the ZKP parts defined during the Build-up Phase, for this ZKP part we will make use of the Pedersen commitment permutation matrix (**B**, with elements $B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}}$, where $\sum_{i=1}^{n} s_{i,j} = 0 \bmod q$ for each j). Previously, $\vec{a}$ described the data to be reordered in vector form; now, $\vec{a}$ will describe the first elements of the tuples in the encrypted values arranged as a vector, and $\vec{a'}$ will describe the second elements of the tuples arranged accordingly. $X_{i,\ell/\ell+1}$ refers to user i's partway pseudonym (or the output partway pseudonym of the previous server corresponding to user i as appropriate). We make use of intermediate vectors **E** from the Build-up Phase (whose elements are $E_{i,j} = \mathfrak{g}^{s'_{i,j}}$), the new **F** (whose elements are $F_{i,j} = A_j^{p_{i,j}\mathfrak{r}_{k,\ell}^{-1}} X_{j,\ell/\ell+1}^{p_{i,j}\mathfrak{r}_{k,\ell}^{-1}s'_{i,j}} \mathfrak{h}^{s'_{i,j}}$), and the new **H** (whose elements are $H_{i,j} = A_j'^{p_{i,j}} \mathfrak{g}_{\ell+1}^{p_{i,j}s'_{i,j}} \mathfrak{h}^{s'_{i,j}}$). As with the Build-up Phase, the $s'_{i,j}$ values are chosen such that $\sum_{j=1}^{n} s'_{i,j} = 0 \bmod q$ for each i. With this in place, we include the following ZKP part:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \{(\mathfrak{r}_{k,\ell}^{-1}, p_{i,j}, s_{i,j}, s'_{i,j}) :$$

$$B_{i,j} = \mathfrak{g}^{p_{i,j}} \mathfrak{h}^{s_{i,j}} \wedge E_{i,j} = \mathfrak{g}^{s'_{i,j}} \wedge F_{i,j} = A_j^{p_{i,j}\mathfrak{r}_{k,\ell}^{-1}} X_{j,\ell/\ell+1}^{p_{i,j}\mathfrak{r}_{k,\ell}^{-1}s'_{i,j}} \mathfrak{h}^{s'_{i,j}} \wedge H_{i,j} = A_j'^{p_{i,j}} \mathfrak{g}_{\ell+1}^{p_{i,j}s'_{i,j}} \mathfrak{h}^{s'_{i,j}}\}$$

Like what was done in the Build-up Phase, the verifier then calculates $\prod_{j=1}^{n} E_{i,j} \stackrel{?}{=} \mathfrak{g}^0$, $\vec{f}_i =$

$\langle \prod_{j=1}^{n} F_{i,j} \rangle_i$, and $\vec{h}_i = \langle \prod_{j=1}^{n} H_{i,j} \rangle_{i \in [1,n]}$. In the F_i cases, the $h^{s'_{i,j}}$ terms drop away as expected, all but one of the $A_j^{p_{i,j} \tau_{k,\ell}^{-1}}$ terms drop away, and the one $X_{j,\ell/\ell+1}^{p_{i,j} \tau_{k,\ell}^{-1} s'_{i,j}}$ term that does not drop away rerandomizes the first part of the tuple appropriately. In the H_i cases, the $h^{s'_{i,j}}$ terms again drop away as expected, all but one of the $A'^{p_{i,j}}_{j}$ terms drop away, and the one $\mathfrak{g}_{\ell+1}^{p_{i,j} s'_{i,j}}$ term that does not drop away rerandomizes the second part of the tuple (with the same random factor as was applied to the first part of the tuple). The resulting F_i and H_i values are the re-ordered and rerandomized elements of the tuple we sought.

Because the vote matrix portion of the ZKP still dominates in the Break-down Phase, as it did in the Build-up Phase, we expect the overall size and complexity of the ZKP for a correct shuffle in this phase to be cubic. As each phase operates independently, this leads us to conclude that the covert setting should expect cubic complexity for the epoch changeover calculations as a whole. As with the Build-up Phase ZKP parts, proof batching techniques can also be applied to this ZKP part. Once each server has participated, the inter-epoch calculation is finished, and fresh pseudonyms can be distributed to users for the new epoch.

4.11 Security Analysis

In both the honest-but-curious and covert settings, user inputs are not trusted. Users must provide ZKPs that their submitted values (new user registrations, votes, and reputation attestations) adhere to the proper format and values allowed in the system. The vote ZKPs are tailored such that they exclusively allow voters to alter the votes associated with fresh pseudonyms for which said voters can prove knowledge of a long-term secret key. Reputation attestations include the same requirement; a user can only form a valid reputation attestation proof with a reputation score assigned to a fresh pseudonym for which said user can prove knowledge of a long-term secret key. So long as users are able to protect their own long-term secret key, no other user (nor any server) can impersonate them. Further, users are not able to submit values that would break system assumptions (such as that votes are always 0, 1, or 2), because they are not able to form ZKPs that servers would accept for such assumption-breaking values. Finally, voters are not able to ballot-stuff in PRSONA. The reputation function that PRSONA utilizes allows voters to provide feedback as often as they like, while only allowing it to count once for or against any votee.

In the honest-but-curious setting, servers are trusted to faithfully and accurately carry out the prescribed calculations during epoch changeovers. Thus, by assumption, in that setting,

PRSONA maintains the integrity of votes and scores across epoch boundaries. In the covert setting, however, we do not assume that servers maintain these values. Instead, servers must provide their own ZKPs that they have executed the calculations correctly and accurately. These ZKPs guarantee that the integrity of votes and scores are maintained across epoch boundaries, which is the only time servers change such data independently. By the nature of these protocols for users and servers, PRSONA is a secure reputation system, correctly preserving the integrity of reputations.

4.12 Summary

Throughout this chapter, we have illustrated the system design of PRSONA, as well as provided discussion for the various choices we made throughout this design. Though PRSONA's architecture is quite similar to that of AnonRep [ZWC+16], our threat model and security goals are different in subtle ways. Whereas AnonRep relies only on an anytrust model, we split our threat model between preserving our privacy properties (which still relies on anytrust) and ensuring correctness of the system (which relies on a majority of honest servers). AnonRep also differentiates between its "security-enhanced" version, which provides Exact Reputation Blinding, a key privacy property, and its normal operation; PRSONA does not have such a split, ensuring that all important privacy properties are always provided. Further, our reputation function is quite different. The reputation function directly models tight-knit communities, and it addresses issues such as bad actors coasting on accumulated good will or small cliques abusing the reputation function's system to keep themselves in good reputation. Beyond this, we discussed the various cryptographic tools we use, and the assorted data types that appear throughout PRSONA. We provided thorough detail on the operations of PRSONA in its various phases, and how servers and clients interact to carry out the system's goals. We also discussed specifically how these design choices and interactions together make PRSONA a secure reputation system.

In addition to describing our system's designs in terms of mathematics and pseudocode, it is valuable to instantiate the system in functioning code. Chapter 5 discusses our instantiation, as well as providing further insight into the real-world viability of the operations and interactions within PRSONA that we described in this chapter.

Chapter 5

Implementation

In this chapter, we discuss our implementation of PRSONA and evaluate it on suitable benchmarks.

5.1 Implementation

We have implemented a functional prototype of PRSONA, consisting of approximately 11400 lines of C++ code (as measured by `cloc`; this count measures only lines of code in PRSONA itself, not any libraries that it uses). Our implementation makes use of an open-source C++ library implementing prime-order BGN from Herbert et al. [HBF19], upon which we have made significant extensions (totalling approximately 2700 lines of C++ code, again as measured by `cloc`). That library in turn uses an open-source C library implementing various elliptic curve and pairing primitives from Naehrig et al. [NNS10]. Our prototype specifically implements the complete design of PRSONA, including all relevant zero-knowledge proofs; servers and clients communicate via TCP. The source code of our prototype is available at `https://git-crysp.uwaterloo.ca/tmgurtler/PRSONA`, and the source code of the modifications we made to the BGN library is available at `https://git-crysp.uwaterloo.ca/tmgurtler/BGN2/`.

The prime-order BGN library that we use implements prime-order BGN (as described in Section 4.5) over a Barreto-Naehrig curve — the library specifically builds upon a previous implementation by Naehrig et al. [NNS10]. We also implement ElGamal encryption (also described in Section 4.5) over this same curve.

In Section 4.10, we mention that proof batching techniques (such as those described by Henry and Goldberg [HG13]) can be applied to the ZKPs used in our proofs of correct shuffles.

In our implementation, we do implement such techniques. Our benchmarks in this chapter are measured with a suitable value for the batch soundness parameter λ (the negative log of the soundness error induced by batching).

5.2 Evaluation

We deployed servers and clients on the CrySP RIPPLE Facility; each individual machine had 1 TiB of RAM and 80 cores and was used for multiple clients and servers. The maximum RAM usage for servers in our largest experiments never exceeded 30 GiB, and the clients' cryptographic calculations use no more than 3 MiB of RAM. In all timing calculations we perform, the effect of bandwidth is small (each machine is connected to the others through a 1 Gbps network). Timings accounting for lower bandwidth may be simulated via the recorded data for how much bandwidth was used, but as will be seen, CPU usage has a much more significant impact on the time any operation takes than the time necessary to transmit the relatively small amounts of data PRSONA requires during these operations.

Throughout our experiments, we benchmarked the system with $n = \{5, 10, 15, 20, 25, 30, 40, 50\}$ users connected. These benchmarks were run as a proof of concept and to confirm our analysis of PRSONA's asymptotic behaviour; with the optimizations (the hybrid honest-but-curious and covert approach, as well as probabilistic proofs) that will be discussed in Section 5.3, we expect that PRSONA could reasonably support several hundred users. We specifically benchmark three operations: the calculations involved in turning over to a new epoch (done by servers), making votes (done by clients), and making proofs of reputation above a reasonable threshold (done by clients). For each operation, we measure three values: CPU time, wall clock time, and bandwidth used (both up and down).

5.2.1 Epoch Calculations (server side)

In order to benchmark PRSONA, it was necessary both to run the system with suitable numbers of clients and servers, as well as to choose an appropriate workload to benchmark the system with. That is to say, PRSONA operates in two broad phases. There is the client participation phase, where clients interact, give reputation proofs, and make votes for each other, and there is the epoch changeover phase, where servers perform the necessary calculations to change to a new epoch. The epoch changeover phase is straightforward, but during the client participation phase, there is more latitude for differing behaviour. In real usage, different numbers of voters

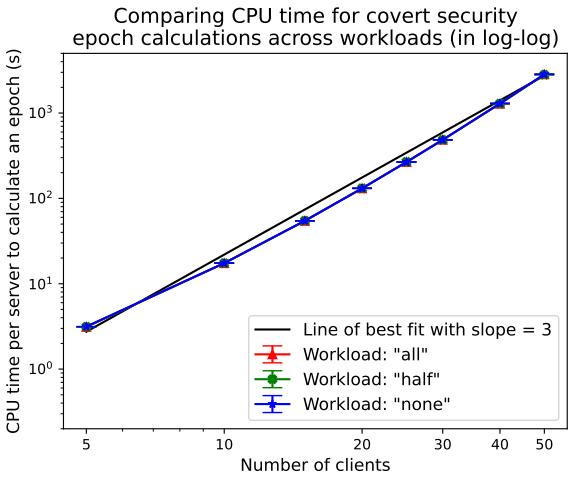
may make votes during any given epoch. For the epoch changeover calculations in particular, it is important that we verify that the way we benchmark our system is appropriate.

To this end, when benchmarking epoch changeover calculations, we tested three workloads, all with differing numbers of voters per each client participation phase. In the first, the "no voter" workload, no clients voted during the client participation phase, an epoch changeover was calculated, then one reputation proof was made. In another, a randomly selected set of clients, half the size of the full set of clients, voted during the client participation phase, but the other two steps stayed the same. In the final epoch benchmarking workload, all of the clients voted during the client participation phase, and again the other two steps stayed the same. We repeated each experiment 25 times. When comparing these workloads under covert security (Figures 5.1a and 5.1c), the proof batching techniques discussed in Section 4.10 were used, with a consistent batch soundness parameter $\lambda = 50$. When comparing these workloads in the honest-but-curious setting (HbC) (Figures 5.1b and 5.1d), proof batching is not relevant, as no proofs are generated or verified.

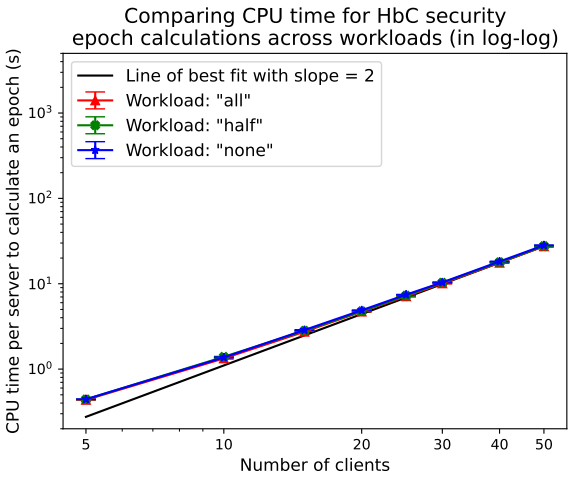
As can be seen in Figures 5.1a–5.1d, epoch calculations in both covert and honest-but-curious security had virtually no difference in either CPU time or server-to-server outgoing bandwidth regardless of the workload chosen. There is some slight variation, but all workload curves take the same shape, with only very minor differences between them; for most values of the number of clients observed, their 95% confidence intervals overlap. This is expected, as the servers perform their epoch calculations obliviously to the data they hold for each user's votes. Where not specified, later graphs measuring epoch changeover values use either the "all voters" workload or a mix between multiple workloads.

As expected, the honest-but-curious cases (Figures 5.1b and 5.1d) have significant performance improvements over the covert cases (Figures 5.1a and 5.1c) in both CPU time and server-to-server outgoing bandwidth. These lower costs may make the system more attractive to deploy, at the cost of requiring a greater amount of trust in the central servers.

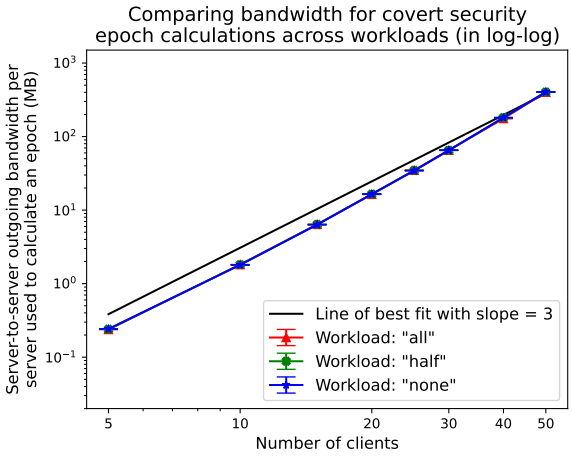
In all of Figures 5.1a–5.1d, we compare the graphs with a simple cubic curve in the covert setting, and a simple quadratic curve in the honest-but-curious setting. For both CPU time and server-to-server outgoing bandwidth, in the covert setting, the graphs are roughly straight lines with slope ≈ 3; in the honest-but-curious setting, the graphs are roughly straight lines with slope ≈ 2. This indicates, as expected in Section 4.10, that the epoch calculation has approximately cubic complexity in the covert setting, and approximately quadratic complexity in the honest-but-curious setting. Cubic growth in particular is difficult to scale to very large numbers of users. However, even with this cubic growth, in the small community setting that PRSONA targets, there are reasonable numbers of clients for which epoch calculations finish in a quick enough fashion to still be useful for a forum setting, where responses frequently



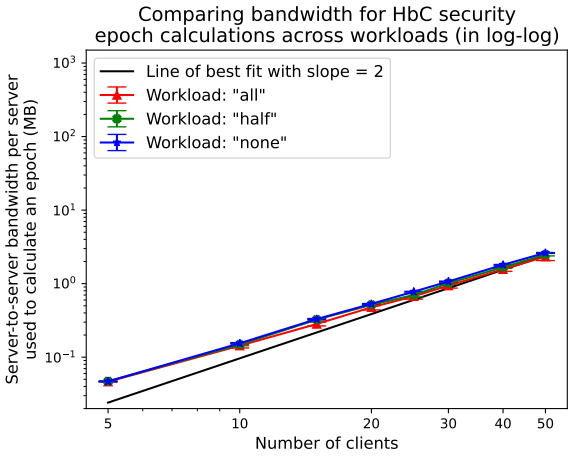
(a) Evaluating covert setting epoch workloads by CPU time, log-log



(b) Evaluating HbC setting epoch workloads by CPU time, log-log

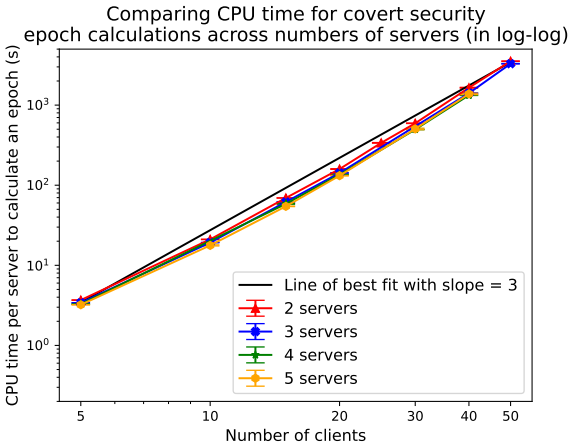


(c) Evaluating covert setting epoch workloads by server-server bandwidth, log-log

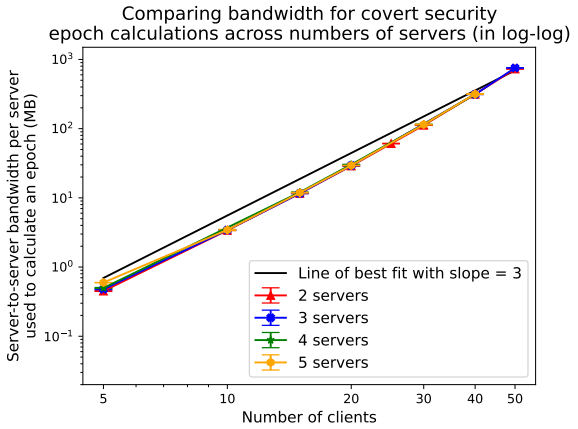


(d) Evaluating HbC setting epoch workloads by server-server bandwidth, log-log

Figure 5.1. Comparing the CPU times and server-to-server outgoing bandwidths required to perform epoch calculations with covert and honest-but-curious security to a cubic (covert) or quadratic (honest-but-curious) curve. In all experiments, a 2-server setup was used, and in the covert security experiments, proof batching was used, with $\lambda = 50$.



(a) Evaluating covert security epochs with different servers by CPU time, log-log



(b) Evaluating covert security epochs with different servers by server-server bandwidth, log-log

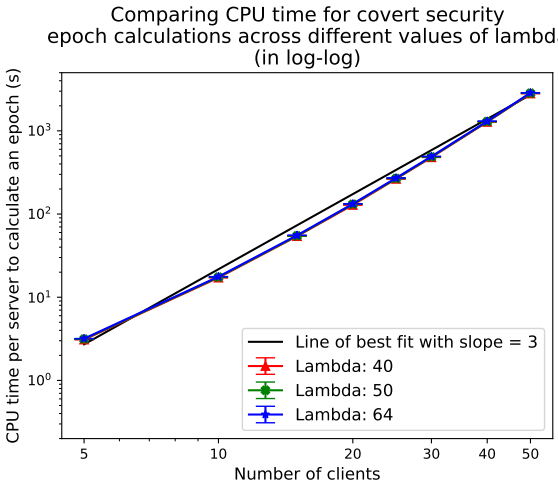
Figure 5.2. Comparing the covert security epoch calculation CPU times and server-to-server outgoing bandwidth of different numbers of servers to simple cubic curves. Datapoints represent a mix of workloads. Proof batching was used in these experiments, with $\lambda = 50$.

come with relatively high latency.

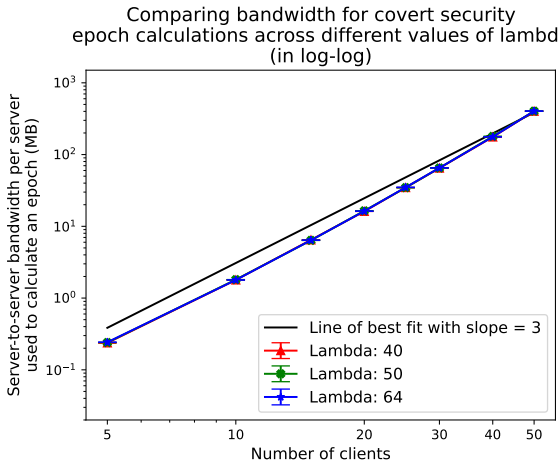
With appropriate workloads for benchmarking epoch changeover calculations set, we begin by considering cases with different number of servers, as in Figures 5.2a–5.2b. In Figure 5.2a specifically, we observe that the per-server CPU time to calculate an epoch changeover has little discernable difference between different number of servers. That is, there is no significant overhead in terms of CPU time for adding additional servers. Further, we compare this CPU usage to a cubic curve on a log-log graph. As in previous cases in the covert setting, graphs are roughly straight lines with slope ≈ 3, indicating approximately cubic complexity. Importantly, all lines have approximately the same slope; this indicates that the relative overhead increases with the same complexity.

Figure 5.2b depicts the relationship between server-to-server outgoing bandwidth per server and number of clients in the covert setting. As with per-server CPU time, we observe little discernable difference between different number of servers in per-server bandwidth. Put another way, there is no significant overhead in terms of bandwidth for adding additional servers. Figure 5.2b also compares these graphs to a cubic curve on a log-log graph — again, all lines have approximately the same slope (≈ 3), indicating that they have the same cubic complexity.

In Section 4.10, we discussed proof batching, an optimization technique for the large



(a) Evaluating covert security epochs with different lambdas by CPU time, log-log



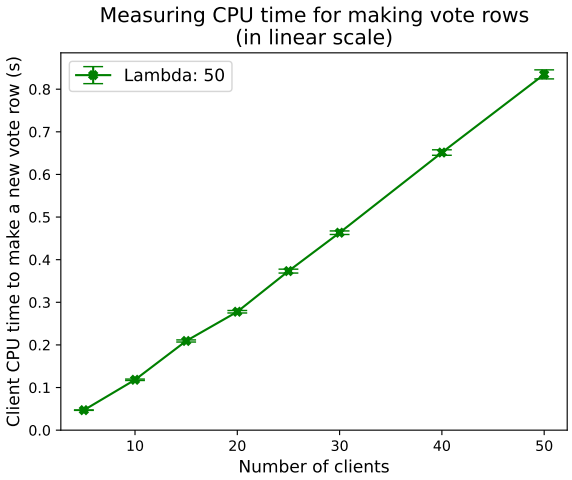
(b) Evaluating covert security epochs with different lambdas by server-server bandwidth, log-log

Figure 5.3. Evaluating using different values of lambda by the CPU time and server-to-server outgoing bandwidth required to perform epoch calculations in covert security. Due to underlying library implementation details, the curves are very similar. Datapoints are pulled from the "all voters" workload. A 2 server setup was used.

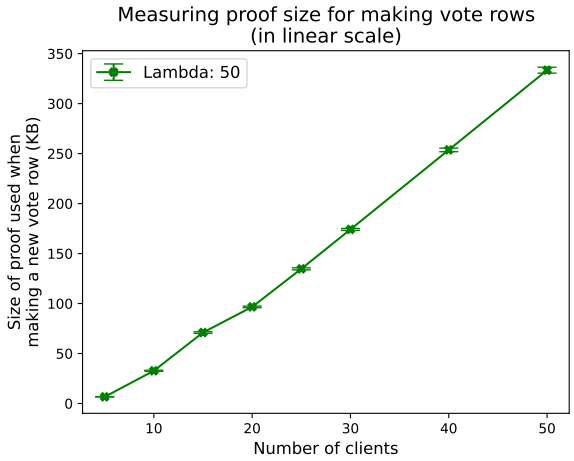
numbers of proofs we conduct. Due to implementation details of curvepoint-scalar multiplication in the elliptic curve library we use, minor variation in lambda will not significantly impact our computations. This can be observed in Figures 5.3a–5.3b, where we vary this choice. Throughout, the shape of the graphs are dominated by the number of clients, with differences so small as to not be visible on our graphs. This being noted, we recommend that implementers likely should base their choice of lambda more on the desired chance for an erroneous proof to be improperly accepted, rather than on efficiency concerns.

5.2.2 New Votes (client side)

For benchmarking vote making, our approach differed from the epoch changeover benchmarks. As new votes depend only on the number of potential votees for any voter, we merely cause a random voter make a new vote, repeatedly. Importantly, however, we only triggered a new voter after the previous voter had completed its vote. This is because, in our implementation, servers are only able to accept one new vote at a time, in order to stay synchronized among one another (this is not an inherent issue with the protocol, merely a limitation of the implementation). If multiple voters submitted new vote rows at the same time, they would



(a) Measuring making a new vote by CPU time



(b) Measuring making a new vote by proof size

Figure 5.4. Measuring the CPU time and proof size required to make a new vote. The curve grows linearly with the number of clients. A 2-server setup was used. Note the linear scale of the x and y-axes.

frequently be forced to wait and resubmit these vote rows only after other clients get through. We do not wish to measure the effects of competing for the limited resource of server availability, only the actual cost of making a vote itself.

With our approach set, we benchmark making new votes. Figures 5.4a–5.4b examine the relationship between making new votes with different numbers of clients. Proofs need to account for a voter's potential vote for each votee, and with more votees, these proofs are therefore bigger. Our expectation from the underlying algorithm is that this relationship would be linear with respect to the number of clients, and this is what we observe. While we ran this measurement with the system set to use a lambda of 50, between the underlying implementation detail that elides small changes in lambda and the fact that proof batching is only conducted for epoch changeover calculations, there is no reason to expect different behaviour for different values of lambda.

Importantly, even in our largest cases, making new vote rows can be completed on the order of seconds, and proof sizes are no larger than 1 MiB. We may not expect users of the system to have as powerful machines as are necessary to carry out epoch changeover calculations. Even so, for any reasonably sized group, making new vote rows will be more or less unnoticeable to the average user; the CPU time to do so will not be more distracting than latency to send the vote to the servers, and the bandwidth required to do so is well within reason.

5.2.3 Reputation Proofs (client side)

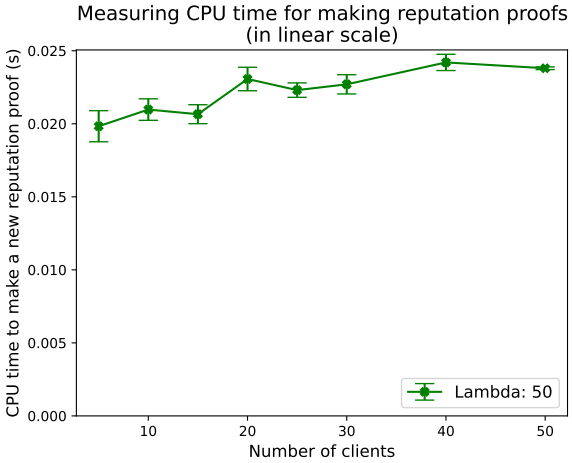
Finally, we benchmark forming the proofs of reputation score that a user can display. As mentioned in Section 4.9, the size and time needed to create reputation proofs is logarithmic in the difference between the chosen threshold and $2n$, for n the number of users in the system. For the sake of consistency, we set all reputation proofs within one parameter set to have the same threshold, and therefore the same size. To achieve this, we have no new votes made, so that all voters retain their default votes. Next, we trigger one epoch changeover, so that all votees shift out of their default reputation score (which is the same regardless of number of votees) to one that is determined by the number of votees (note, however, that all votees have the same score, as all votes are the same). Lastly, we cause a random votee to make a reputation proof to a random voter, repeatedly.

For different numbers of clients, when making reputation proofs, as in Figure 5.5a, the variation is affected by noise introduced by the library used to manage TCP connections between clients. This noise is significant and makes analysis of the complexity of the operation difficult, but even with this noise, making a reputation proof takes a very small amount of time (less than 0.025 seconds in all cases). Recall that the size of the reputation proof is directly tied to the number of bits in the size of the range that the threshold represents. We can see this directly in Figure 5.5b; at each power of two in the number of clients, the proof size steps up. This stepwise behaviour is an artifact of our experiments using only default votes; normally the size of these proofs increases when the maximum possible reputation minus the threshold a user is trying to prove they are above crosses powers of two. Proof sizes overall are very small, along with the previously discussed time required to make proofs, even before the expected logarithmic growth is accounted for, so these operations are very reasonable for users to engage with regularly as needed.

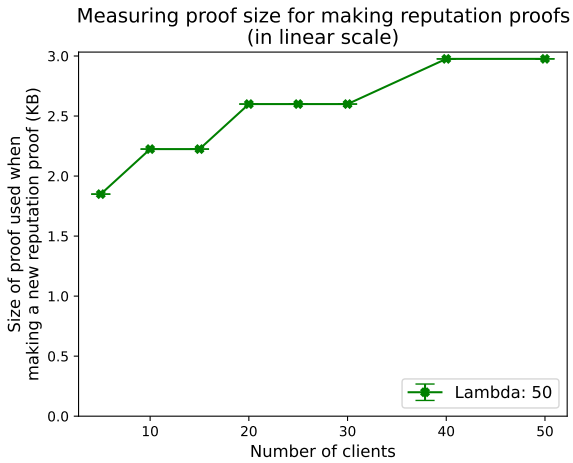
Figure 5.5c depicts the relationship between verifying reputation proofs and CPU time (the related relationship to client-to-client bandwidth is not depicted, as it would be identical to Figure 5.5b). The same noise that impacts Figure 5.5a also impacts Figure 5.5c, and again makes analysis of verification's complexity difficult. Despite this noise, as in Figure 5.5c, proof verification takes a very small amount of time (less than 0.025 seconds in all cases).

5.3 Discussion

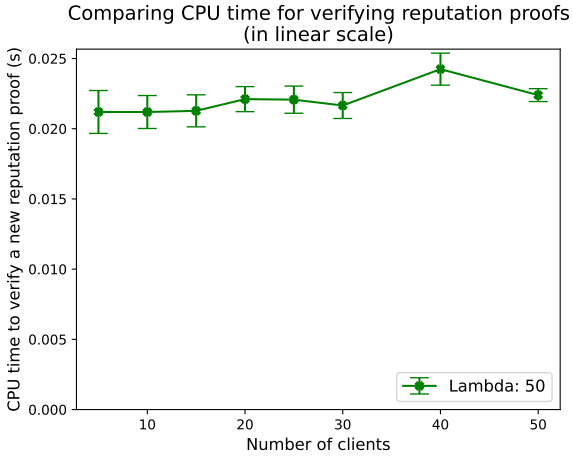
With these benchmarks, making new votes and reputation proofs is not particularly intensive no matter the number of clients we selected. Our expectations for users are fairly reasonable in all cases, and things are relatively simple.



(a) Measuring making a new reputation proof by CPU time



(b) Measuring making a new reputation proof by proof size



(c) Measuring verifying a reputation proof by CPU time

Figure 5.5. Measuring the CPU time and proof size required to make a new reputation proof. CPU time curves are impacted by noise introduced by a networking library. Proof size curves display a clear stepwise approximation of logarithmic growth, owing to the logarithmic growth of the reputation proof size itself. A 2-server setup was used.

Conducting the epoch changeover calculations is more intensive. We require higher bandwidth usage for servers than we do clients, and the times involved in calculating these epoch changeovers can be relatively high. As mentioned at the beginning of Chapter 4, we target PRSONA for usage in tight-knit communities. Our definition of tight-knit is related to Dunbar's number [Dun92], recent estimates for which range up to about 500 [LWL21]. As such, we expect sizes of the tight-knit groups we consider to not exceed some several hundred members. For the largest of our proof-of-concept test cases, which are much smaller than these upper bound expectations, PRSONA epoch changes already take significant amounts of time in the covert setting (several hours of total CPU time). These results are unoptimized — there are a considerable number of matrix operations that could be parallelized, for example. In contrast, in the honest-but-curious setting, epoch changes take only a few minutes of total CPU time.

Though we consider the honest-but-curious setting and the covert setting for epoch calculations separately, there is a way to combine the two modes. If a server conducts their shuffle in the honest-but-curious setting, but holds on to the random order they used in the Build-up and Break-down Phases, as well as the blinding factors used to rerandomize elements, they would be able to retroactively produce the proof used in the covert setting. Because of this, servers can conduct the shuffle using the honest-but-curious setting in real time, which would prevent the system from being blocked for long periods of time. Afterwards, they could retroactively produce the proofs used in the covert setting, and verify them amongst each other, to confirm that all servers carried out their shuffle correctly. Because a server who did not conform to the protocol would be eventually identified, a covert server would not be able to act maliciously in this case. Additionally, when done this way, servers would be able to generate their covert setting proofs in parallel, which is otherwise not possible. It is even within reason that servers could be asked for their proofs only probabilistically. An implementer could choose a parameter for how likely it would be to ask a server for their covert setting proof. Then, at the end of the shuffle, data shared between the servers could be hashed to generate a random value that would require servers to provide their covert setting proofs with some non-negligible probability. A covert server, in this setting, would still be prevented from cheating, because of the non-negligible probability that they would be caught if they did so.

With this hybrid approach, an implementer gains the security against a covert adversary while only requiring the online cost of the honest-but-curious setting. Though there is still an offline cost in line with the covert setting, the system would not block for long periods of time, meaning that users would be unable to vote on each other for only a few minutes each epoch change. If the epoch is set to change every day, or every other day, users would receive the benefits of fresh pseudonyms on a timescale appropriate for the speed of forum

conversations, and the system would be able to support a reasonable number of users. Given what we have observed from these benchmarks, we conclude that PRSONA is appropriate for use in such settings.

Chapter 6

Conclusion

In this work, we have demonstrated that the following statement is true:

It is possible to build a secure reputation system that preserves the privacy of its participants and enables the implementation of multiple complex reputation functions.

In Chapter 2, we discussed various details about reputation systems in a generalized form, including what makes them secure, how they can preserve the privacy of their participants, and what constitutes multiple complex reputation functions.

In Chapter 3, we discussed and analyzed previous work on privacy-preserving reputation systems. We found that no previous work has already proven that the thesis statement we evaluate in this work is true.

In Chapter 4, we took this information and used it to design a reputation system. This reputation system is secure. Even in a covert security setting, so long as a majority of servers are honest, the reputation system will continue functioning while maintaining the integrity of votes and scores and the availability of the system as a whole. Further, as long as even just one server is honest, the confidentiality of votes and scores is preserved. This reputation system preserves the privacy of its participants. This system provides Voter-Vote Unlinkability, Two-Vote Unlinkability, Reputation-Usage Unlinkability, and Exact Reputation Blinding, all of the privacy properties we identify with reputation systems. This reputation system implements complex reputation functions. We specifically chose to implement a version of Short-Term Memory Consensus, but only minor modification would be necessary to implement Long-Term Memory Consensus (taking a votee's previous round score and performing some averaging operation with their new calculated score each epoch), and implementing less complex reputation functions is much simpler.

In Chapter 5, we made a proof-of-concept implementation of our design. After benchmarking our system, we conclude that, for the setting in which we anticipate it to be used (smaller communities with self-protection interests), the cost of deployment is acceptable.

We have therefore proven our thesis statement by construction.

6.1 Future Work

There remains room for improvement in the design of reputation systems. Our approach focuses on tight-knit communities of up to several hundred members, but aspects of our design can still readily be applied to larger contexts. In particular, larger contexts do not often employ privacy properties such as Reputation-Usage Unlinkability, which would provide a great deal of opportunity for users in the right contexts.

More importantly, in the literature at large, reputation is limited to applying by and to individual users, and our system is no exception. Individual votees still receive scores, and those scores still reflect the opinions of all voters in the system. However, our approach does have natural in-roads towards selecting the opinions of a subset of voters, such that reputation could apply by groups. And it also has natural ways to incorporate collecting the scores of multiple users and reducing them to one visible score, such that reputation could apply to groups. Further work would be necessary to fully specify and implement these aspects but represents multiple exciting new directions for research.

Reference

[AAG09] Carlos Aguilar Melchor, Boussad Ait-Salem, and Philippe Gaborit. A collusion-resistant distributed scalar product protocol with application to privacy-preserving computation of trust. In *2009 Eighth IEEE International Symposium on Network Computing and Applications*, pages 140–147, July 2009.

[ABH18] Muhammad Ajmal Azad, Samiran Bag, and Feng Hao. PrivBox: Verifiable decentralized reputation system for online marketplaces. *Future Generation Computer Systems*, 89:44–57, 2018.

[ABHS18] Muhammad Ajmal Azad, Samiran Bag, Feng Hao, and Khaled Salah. M2M-REP: Reputation system for machines in the internet of things. *Computers and Security*, 79:1–16, 2018.

[ACBM08] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies*, pages 202–218. Springer Berlin Heidelberg, 2008.

[ADT04] Sharad Agarwal, Travis Dawson, and Christos Tryfonas. DDoS mitigation via regional cleaning centers. Technical report, Sprint ATL, January 2004.

[AG06] Mohd Anwar and Jim Greer. Reputation management in privacy-enhanced e-learning. In *The Proceedings of the 3rd Annual Scientific Conference of the LORNET Research Network (I2LOR 2006)*, November 2006.

[All15] Jay Allen. The invasion boards that set out to ruin lives. https://boingboing.net/2015/01/19/invasion-boards-set-out-to-rui.html, January 2015.

[Alt16] Larry Alton. How Purple, Uber and Airbnb are disrupting and redefining old industries. https://www.entrepreneur.com/article/273650, April 2016.

[BAH18] Samiran Bag, Muhammad Ajmal Azad, and Feng Hao. A privacy-aware decentralized and personalized reputation system. *Computers and Security*, 77:514–530, 2018.

[BBB+18] Kai Bemmann, Johannes Blömer, Jan Bobolz, Henrik Bröcher, Denis Diemert, Fabian Eidens, Lukas Eilers, Jan Haltermann, Jakob Juhnke, Burhan Otour, Laurens Porzenheim, Simon Pukrop, Erik Schilling, Michael Schlichtig, and Marcel Stienemeier. Fully-featured anonymous credentials with reputation system. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, pages 42:1–42:10, New York, NY, USA, 2018. ACM.

[BEJ18] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. Practical, anonymous, and publicly linkable universally-composable reputation systems. In Nigel P. Smart, editor, *Topics in Cryptology — CT-RSA 2018*, pages 470–490. Springer International Publishing, 2018.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BIJR04] Colin Boyd, Roslan Ismail, Audun Jøsang, and Selwyn Russell. Private reputation schemes for P2P systems. In Fernandex-Medina, Castro, and Villalba, editors, *Proceedings of the 2nd International Workshop on Security In Information Systems, WOSIS 2004*, pages 196–206, Porto, Portugal, 2004. INSTICC Press.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, pages 319–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BPS+17] Núria Busom, Ronald Petrlic, Francesc Sebé, Christoph Sorge, and Magda Valls. A privacy-preserving reputation system with user rewards. *Journal of Network and Computer Applications*, 80:58–66, 2017.

[BSHB16] Rémi Bazin, Alexander Schaub, Omar Hasan, and Lionel Brunie. A decentralized anonymity-preserving reputation system with constant-time score retrieval. Cryptology ePrint Archive, Report 2016/416, 2016. https://eprint.iacr.org/2016/416.

[BSS10] John Bethencourt, Elaine Shi, and Dawn Song. Signatures of reputation. In Radu Sion, editor, *Financial Cryptography and Data Security*, pages 400–407, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CBC18] Facebook makes changes to 'real names' policy after complaints. https://www.cbc.ca/news/technology/facebook-real-names-1.3367403, September 2018.

[Cha83] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[Cha85] David Chaum. Security without identification: Transaction systems to make Big Brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.

[Cho19a] Niraj Chokshi. The rise and fall of the Jeremy Renner app, which was a real thing. https://www.nytimes.com/2019/09/05/style/jeremy-renner-app.html, September 2019.

[Cho19b] Hasan Chowdhury. Facebook to unveil new cryptocurrency in bid to disrupt online payments. https://www.telegraph.co.uk/technology/2019/06/17/facebook-unveil-new-cryptocurrency-bid-disrupt-online-payments/, June 2019.

[CRH+13] Delphine Christin, Christian Roßkopf, Matthias Hollick, Leonardo A. Martucci, and Salil S. Kanhere. IncogniSense: An anonymity-preserving reputation framework for participatory sensing applications. *Pervasive and Mobile Computing*, 9(3):353–371, 2013. Special Issue: Selected Papers from the 2012 IEEE International Conference on Pervasive Computing and Communications (PerCom 2012).

[CSH17] Michael R. Clark, Kyle Stewart, and Kenneth M. Hopkinson. Dynamic, privacy-preserving decentralized reputation systems. *IEEE Transactions on Mobile Computing*, 16(9):2506–2517, September 2017.

[CSK13] Sebastian Clauß, Stefan Schiffner, and Florian Kerschbaum. k-anonymous reputation. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 359–368, New York, NY, USA, 2013. ACM.

[Cut19] Anthony Cuthbertson. Tumblr defends controversial porn ban despite 20 per cent drop in traffic. https://www.independent.co.uk/life-style/gadgets-and-tech/news/tumblr-porn-ban-nsfw-blog-sex-photos-videos-gifs-a8824536.html, March 2019.

[DC21] Tim De Chant. Catholic priest quits after "anonymized" data revealed alleged use of Grindr. https://arstechnica.com/tech-policy/2021/07/catholic-priest-quits-after-anonymized-data-revealed-alleged-use-of-grindr/, July 2021.

[DdVPS03] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Managing and sharing servents' reputations in P2P systems. *IEEE Transactions on Data and Knowledge Engineering*, 15(4):840–854, 2003.

[Dic19] EJ Dickson. Furries got an alt-right troll banned from their convention. https://www.rollingstone.com/culture/culture-news/milo-yiannopolous-furry-convention-884960/, September 2019.

[Dim21] Tassos Dimitriou. Decentralized reputation. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, CODASPY '21, pages 119–130, New York, NY, USA, 2021. Association for Computing Machinery.

[Dun92] Robin Ian MacDonald Dunbar. Neocortex size as a constraint on group size in primates. *Journal of Human Evolution*, 22(6):469–493, 1992.

[EKKS18] Ali El Kaafarani, Shuichi Katsumata, and Ravital Solomon. Anonymous reputation systems achieving full dynamicity from lattices. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security*, pages 388–406, Berlin, Heidelberg, 2018. Springer Berlin Heidelberg.

[Elg85] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Elg19] Mike Elgin. Uh-oh: Silicon Valley is building a Chinese-style social credit system. https://www.fastcompany.com/90394048/uh-oh-silicon-valley-is-building-a-chinese-style-social-credit-system, August 2019.

[Ell19] Emma Grey Ellis. YouTube continues to fail its queer creators. https://www.wired.com/story/youtube-carlos-maza/, June 2019.

[FGL20] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. Influence function based data poisoning attacks to top-n recommender systems. In *Proceedings of The Web Conference 2020*, WWW '20, pages 3019–3025, New York, NY, USA, 2020. Association for Computing Machinery.

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In Henri Gilbert, editor, *Advances in Cryptology — EUROCRYPT 2010*, pages 44–61, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[FYGL18] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. Poisoning attacks to graph-based recommender systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, pages 381–392, New York, NY, USA, 2018. Association for Computing Machinery.

[GFM14] Neil Zhenqiang Gong, Mario Frank, and Prateek Mittal. SybilBelief: A semi-supervised learning approach for structure-based Sybil detection. *IEEE Transactions on Information Forensics and Security*, 9(6):976–987, 2014.

[GK14] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. Cryptology ePrint Archive, Report 2014/764, 2014. https://ia.cr/2014/764.

[GMN17] Lydia Garms, Keith Martin, and Siaw-Lynn Ng. Reputation schemes for pervasive social networks with anonymity (short paper). In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 311–316, August 2017.

[GNQT20] Lydia Garms, Siaw-Lynn Ng, Elizabeth A. Quaglia, and Giulia Traverso. Anonymity and rewards in peer rating systems. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks*, pages 277–297, Cham, 2020. Springer International Publishing.

[Gua19] TikTok's local moderation guidelines ban pro-LGBT content. https://www.theguardian.com/technology/2019/sep/26/tiktoks-local-moderation-guidelines-ban-pro-lgbt-content, September 2019.

[HBB10a] Omar Hasan, Elisa Bertino, and Lionel Brunie. Efficient privacy preserving reputation protocols inspired by secure sum. In *2010 Eighth International Conference on Privacy, Security and Trust*, pages 126–133, August 2010.

[HBB10b] Omar Hasan, Lionel Brunie, and Elisa Bertino. k-Shares: A privacy preserving reputation protocol for decentralized environments. In Kai Rannenberg, Vijay Varadharajan, and Christian Weber, editors, *Security and Privacy—Silver Linings in the Cloud*, pages 253–264, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[HBB12] Omar Hasan, Lionel Brunie, and Elisa Bertino. Preserving privacy of feedback providers in decentralized reputation systems. *Computers and Security*, 31(7):816–826, 2012. IFIP/SEC 2010 "Security and Privacy—Silver Linings in the Cloud".

[HBBS13] Omar Hasan, Lionel Brunie, Elisa Bertino, and Ning Shang. A decentralized privacy preserving reputation protocol for the malicious adversarial model. *IEEE Transactions on Information Forensics and Security*, 8(6):949–962, June 2013.

[HBF19] Vincent Herbert, Bhaskar Biswas, and Caroline Fontaine. Design and implementation of low-depth pairing-based homomorphic encryption scheme. *Journal of Cryptographic Engineering*, 9(2):185–201, June 2019.

[HG13] Ryan Henry and Ian Goldberg. Batch proofs of partial knowledge. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, pages 502–517, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[HK17] Sufian Hameed and Hassan Ahmed Khan. Leveraging SDN for collaborative DDoS mitigation. In *2017 International Conference on Networked Systems (NetSys)*, pages 1–6, 2017.

[HKH10] Kuan Lun Huang, Salil S. Kanhere, and Wen Hu. Are you contributing trustworthy data? The case for a reputation system in participatory sensing. In *Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, MSWIM '10, pages 14–22, New York, NY, USA, 2010. Association for Computing Machinery.

[HKH12] Kuan Lun Huang, Salil S. Kanhere, and Wen Hu. A privacy-preserving reputation system for participatory sensing. In *37th Annual IEEE Conference on Local Computer Networks*, pages 10–18, October 2012.

[HLTZ08] Liming Hao, Songnian Lu, Junhua Tang, and Aixin Zhang. A low cost and reliable anonymity scheme in P2P reputation systems with trusted third parties. In *IEEE GLOBECOM 2008 — 2008 IEEE Global Telecommunications Conference*, pages 1–5, November 2008.

[HS11] Mohammed Hussain and David B. Skillicorn. Mitigating the linkability problem in anonymous reputation management. *Journal of Internet Services and Applications*, 2(1):47–65, July 2011.

[HYLC07] Liming Hao, Shutang Yang, Songnian Lu, and Gongliang Chen. A dynamic anonymous P2P reputation system based on trusted computing technology. In *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, pages 332–337, November 2007.

[IBJR03] Roslan Ismail, Colin Boyd, Audun Jøsang, and Selywn Russel. Strong privacy in reputation systems. In *Proceedings of the 4th International Workshop on Information Security Applications (WISA)*, August 2003.

[JC19] Hyo Jin Jo and Wonsuk Choi. BPRF: Blockchain-based privacy-preserving reputation framework for participatory sensing systems. *PLoS ONE*, 14, 2019.

[JWZG17] Jinyuan Jia, Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. AttriInfer: Inferring user attributes in online social networks using Markov random fields. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1561–1569, 2017.

[KBS+19] Christiane Kuhn, Martin Beck, Stefan Schiffner, Eduard Jorswieck, and Thorsten Strufe. On privacy notions in anonymous communication. *Proceedings on Privacy Enhancing Technologies*, 2019(2):105–125, 2019.

[Ker09] Florian Kerschbaum. A verifiable, centralized, coercion-free reputation system. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, WPES '09, pages 61–70. ACM, 2009.

[Kil17] Eric Killelea. Does the furry community have a Nazi problem? https://www.rollingstone.com/culture/culture-features/does-the-furry-community-have-a-nazi-problem-194282/, April 2017.

[KLLA15] Aapo Kalliola, Kiryong Lee, Heejo Lee, and Tuomas Aura. Flooding DDoS mitigation and traffic management with software defined networking. In *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pages 248–254, 2015.

[KN11] Soon Hin Khor and Akihiro Nakao. DaaS: DDoS mitigation-as-a-service. In *2011 IEEE/IPSJ International Symposium on Applications and the Internet*, pages 160–171, 2011.

[KP03] Michael Kinateder and Siani Pearson. A privacy-enhanced peer-to-peer reputation system. In Kurt Bauknecht, A. Min Tjoa, and Gerald Quirchmayr, editors, *E-Commerce and Web Technologies*, pages 206–215, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[KYMX20] Vishnu Teja Kilari, Ruozhou Yu, Satyajayant Misra, and Guoliang Xue. EARS: Enabling private feedback updates in anonymous reputation systems. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2020.

[LAN+19] Dongxiao Liu, Amal Alahmadi, Jianbing Ni, Xiaodong Lin, and Xuemin Shen. Anonymous reputation system for IIoT-enabled retail marketing atop PoS blockchain. *IEEE Transactions on Industrial Informatics*, 15(6):3527–3537, June 2019.

[LBW16] Barry Libert, Megan Beck, and Jerry Wind. How platforms will disrupt the future of media and entertainment. https://knowledge.wharton.upenn.edu/misc/platforms-will-disrupt-future-media-entertainment/, November 2016.

[Lee18] Dave Lee. The tactics of a Russian troll farm. https://www.bbc.com/news/technology-43093390, February 2018.

[Lee19] Timothy B. Lee. Report: Facebook looking to disrupt credit cards with cryptocurrency. https://arstechnica.com/tech-policy/2019/05/report-facebook-looking-to-disrupt-credit-cards-with-cryptocurrency/, May 2019.

[LM19] Jia Liu and Mark Manulis. pRate: Anonymous star rating with rating secrecy. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 550–570. Springer International Publishing, 2019.

[LR04] Shyong K. Lam and John Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 393–402, New York, NY, USA, 2004. Association for Computing Machinery.

[LWL21] Patrick Lindenfors, Andreas Wartel, and Johan Lind. 'Dunbar's number' deconstructed. *Biology Letters*, 17(5):1–4, April 2021.

[MPRS08] Wolf Müller, Henryk Plötz, Jens-Peter Redlich, and Takashi Shiraki. Sybil proof anonymous reputation management. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, SecureComm '08, pages 7:1–7:10, New York, NY, USA, 2008. ACM.

[MR06] Hugo Miranda and Luis Rodrigues. A framework to provide anonymity in reputation systems. In *2006 Third Annual International Conference on Mobile and Ubiquitous Systems: Networking Services*, pages 1–4, July 2006.

[NNS10] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *Proceedings of the First International Conference on Progress in Cryptology: Cryptology and Information Security in Latin America*, LATINCRYPT'10, pages 109–123, Berlin, Heidelberg, 2010. Springer-Verlag.

[NR09] Rishab Nithyanand and Karthik Raman. Fuzzy privacy preserving peer-to-peer reputation management. Cryptology ePrint Archive, Report 2009/442, January 2009. https://eprint.iacr.org/2009/442.

[Ple16] Margaret Pless. Kiwi Farms, the web's biggest community of stalkers. https://nymag.com/intelligencer/2016/07/kiwi-farms-the-webs-biggest-community-of-stalkers.html, July 2016.

[PLS14] Ronald Petrlic, Sascha Lutters, and Christoph Sorge. Privacy-preserving reputation management. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1712–1718. ACM, 2014.

[PLZZ10] Hao Peng, Song-nian Lu, Dan-dan Zhao, and Ai-xin Zhang. Low cost and reliable anonymity protocols in P2P reputation systems. *Journal of Shanghai Jiaotong University (Science)*, 15(2):207–212, April 2010.

[Pol78] John M Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[PRT04] Elan Pavlov, Jeffrey S. Rosenschein, and Zvi Topol. Supporting privacy in decentralized additive reputation systems. In Christian Jensen, Stefan Poslad, and Theo Dimitrakos, editors, *Trust Management*, pages 108–119, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Rob16] Rob. What I learned selling my Reddit accounts. https://medium.com/@Rob79/what-i-learned-selling-my-reddit-accounts-c5e9f6348005, April 2016.

[Rug13] Mike Rugnetta. Mike Rugnetta, Idea Channel - XOXO Festival (2013). https://www.youtube.com/watch?v=-D9Xq3Xr8aE, October 2013.

[SBHB16] Alexander Schaub, Rémi Bazin, Omar Hasan, and Lionel Brunie. A trustless privacy-preserving reputation system. In Jaap-Henk Hoepman and Stefan Katzenbeisser, editors, *ICT Systems Security and Privacy Protection*, pages 398–411. Springer International Publishing, 2016.

[SBZD15] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, and Hervé Debar. Towards autonomic DDoS mitigation using software defined networking. In *SENT 2015: NDSS Workshop on Security of Emerging Networking Technologies*, San Diego, Ca, United States, February 2015. Internet Society.

[SBZD17] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, and Hervé Debar. ArOMA: An SDN based autonomic DDoS mitigation framework. *Computers & Security*, 70:482–499, 2017.

[SC17] Ian Sherr and Erin Carson. GamerGate to Trump: How video game culture blew everything up. https://www.cnet.com/news/gamergate-donald-trump-american-nazis-how-video-game-culture-blew-everything-up/, November 2017.

[SKCD16] Kyle Soska, Albert Kwon, Nicolas Christin, and Srinivas Devadas. Beaver: A decentralized anonymous marketplace with secure reputation. Cryptology ePrint Archive, Report 2016/464, 2016. https://eprint.iacr.org/2016/464.

[SL03] Aameek Singh and Ling Liu. TrustMe: anonymous management of trust relationships in decentralized P2P systems. In *Proceedings Third International Conference on Peer-to-Peer Computing (P2P 2003)*, pages 142–149, September 2003.

[SPT11] Stefan Schiffner, Andreas Pashalidis, and Elmar Tischhauser. On the limits of privacy in reputation systems. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, pages 33–42, New York, NY, USA, 2011. ACM.

[Sta18] Nick Statt. Reddit CEO says racism is permitted on the platform, and users are up in arms. https://www.theverge.com/2018/4/11/17226416/reddit-ceo-steve-huffman-racism-racist-slurs-are-okay, April 2018.

[Ste15] Adam Steinbaugh. Kevin Bollaert sentenced to 18 years over revenge porn site "You Got Posted". http://adamsteinbaugh.com/2015/04/03/kevin-bollaert-sentenced-to-years-over-revenge-porn-site-you-got-posted/, April 2015.

[TMLS09] Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient online content voting. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '09, pages 15–28, USA, 2009. USENIX Association.

[UNE19] From Internet Universality to ROAM-X indicators. https://en.unesco.org/themes/internet-universality-indicators/background, 2019.

[VHM05] Marco Voss, Andreas Heinemann, and Max Muhlhauser. A privacy preserving reputation system for mobile information dissemination networks. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm '05)*, pages 171–181, September 2005.

[Vos04] Marco Voss. Privacy preserving online reputation systems. In Yves Deswarte, Frédéric Cuppens, Sushil Jajodia, and Lingyu Wang, editors, *Information Security Management, Education and Privacy*, pages 249–264, Boston, MA, 2004. Springer US.

[WCMA13] Xinlei (Oscar) Wang, Wei Cheng, Prasant Mohapatra, and Tarek Abdelzaher. ARTSense: Anonymous reputation and trust in participatory sensing. In *2013 Proceedings IEEE INFOCOM*, pages 2517–2525, April 2013.

[Wel15] Jonathan Wells. Tyler Oakley: How the internet revolutionised LGBT life. https://www.telegraph.co.uk/men/thinking-man/tyler-oakley-how-the-internet-revolutionised-lgbt-life/, November 2015.

[WH09] Yunzhao Wei and YanXiang He. A pseudonym changing-based anonymity protocol for P2P reputation systems. In *2009 First International Workshop on Education Technology and Computer Science*, volume 3, pages 975–980, March 2009.

[WMF19] Daniel Wood, Sean McMinn, and Emily Feng. China used Twitter to disrupt Hong Kong protests, but efforts began years earlier. https://www.npr.org/2019/09/17/758146019/china-used-twitter-to-disrupt-hong-kong-protests-but-efforts-began-years-earlier, September 2019.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard: Defending against Sybil attacks via social networks. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 267–278, New York, NY, USA, 2006. Association for Computing Machinery.

[YTP07] Danfeng Yao, Roberto Tamassia, and Seth Proctor. Private distributed scalar product protocol with application to privacy-preserving computation of trust. In Sandro Etalle and Stephen Marsh, editors, *Trust Management*, pages 1–16, Boston, MA, 2007. Springer US.

[ZGD19] Luying Zhou, Huaqun Guo, and Gelei Deng. A fog computing based approach to DDoS mitigation in IIoT systems. *Computers & Security*, 85:51–62, 2019.

[ZWC+16] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. AnonRep: Towards tracking-resistant anonymous reputation. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 583–596. USENIX Association, March 2016.

[ZXYM16] Mingwu Zhang, Yong Xia, Ou Yuan, and Kirill Morozov. Privacy-friendly weighted-reputation aggregation protocols against malicious adversaries in cloud services. *International Journal of Communication Systems*, 29(12):1863–1872, 2016.