# Ghost Recommendations

## A Protocol for Efficiently Enhancing User Privacy

by

Kritika Iyer

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, 2024

## Abstract

As the amount of online information accessible to users keeps increasing, we have come to rely more on services such as Netflix, Amazon, and eBay that are successful in recommending choices to users. The main goal of such services is to present the user with a more personalized set of choices or recommendations. The growing importance of recommendation systems that provide these services can be attested by the efforts the academic community is taking towards improving their performance. The quality of recommendation systems is primarily determined by the accuracy of the results they can provide to the users. To achieve high-accuracy results, these systems count on finding similarities between different users based on various features, such as the ratings the users provide for the items. Recommendation systems use different techniques, often harvesting private user information, to detect these similarities. Therefore, providing better recommendations frequently comes at the cost of user privacy and at the risk exposing the user's preferences. Owing to growing concerns about this risk, researchers started to investigate recommendation solutions with better assurances of privacy.

There is a growing body of work with respect to making recommendation systems more sensitive towards user privacy. The current solutions implemented use various methodologies like randomization of the dataset, anonymizing the identities of users, using data aggregation, obfuscating user data, using a trusted third party, and using cryptographic techniques. However, we are yet to have a solution that not only provides privacy guarantees, but is also a practical and efficient system, giving recommendations with high accuracy.

Our goal in this thesis is to implement a solution that enables high guarantees of user privacy, is practical and efficient, that scales well over a large dataset, and provides users with accurate recommendations. A common trend in the solutions mentioned before is to model a system around one or more *trusted third parties*. All the critical operations such as key generation or user authentication are delegated to these trusted third parties and combined with a threat model that restricts them from behaving in a malicious manner. We aim at implementing a system that is independent of such a trusted third party. We also desire a system that makes collusion among servers ineffective unless the number of corrupt servers exceeds a *threshold* value. We also want to make all computations independent of the availability of the participants, so that users would get recommendations even if other participants are offline. For our use case we have considered a scenario where users would like to get recommendations of movies that are based on ratings provided for other movies. To evaluate our system we have used the real world, publicly available "MovieLens" dataset. Our system consists of the following entities: a set of users or clients, a distributed set of servers, and a public bulletin board. Our scheme primarily focuses on maintaining the privacy of user preferences as well as the recommendations and it does not allow anyone other than the user herself to have access to the data.

## Acknowledgements

First, I would like to acknowledge my supervisor, Dr. Ian Goldberg. I am so thankful for your continuous support. This process was very challenging for me, so I am deeply appreciative of your guidance and your patience. Thank you, Ian.

I am also extremely grateful to Prof. Robin Cohen and Prof. Florian Kerschbaum. I am very thankful for Robin and Florian for being on my reading committee and for their mentorship. Their knowledge and experience has been invaluable.

I would also like to thank my lab-mates in the CrySP group, especially Erinn, Nik, Justin, and Chris. I appreciate the many discussions had in the lab. All of those discussions were fun, most of them made sense to me, and some of them even helped shape my thesis.

I also have to thank my family away from home. It has not been easy getting through this chapter of my life but I am forever indebted to the friends I have made who have encouraged me and supported me during this time. Niv, Arshee, Jacqueline, Namrah, and Maggie— I am so glad I met you wonderful women. Each one of you has a special place in my life. I will always cherish you.

And finally, thank you, Varuni Sakhalkar. You are a gem of a person. Your kindness and warmth kept me going during the toughest days.

## Dedication

*For my parents,*
*Dr. Rachna Vishwanathan and Mr. V. Vishwanathan.*

Thank you for your unconditional love and support, for being so inspiring and encouraging, and for your unwavering belief in me. Maa, you are my biggest cheerleader and my idol. Dad, your advice is invaluable to me and your faith in me makes me feel empowered.

Thank you, Anant Iyer. You are my biggest champion and my best defender. Your relentless drive to constantly do better and your ability to always take care of others is inspiring to me. Megha Joshi, I am so grateful for you always looking out for me.

Thank you, Sameer 'Galya' Jagdale. You mentioned it was your dream to be included in a dedication so here it is. And also thank you for being absolutely the best, I don't know how I would have made it without you.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

It is fairly common in today's society for people to have abundant choices, whether it is finding a good restaurant nearby or wanting to buy a new laptop. The problem they are then faced with is to make an informed decision without having enough knowledge about all the available options. The most intuitive solution for this problem would be to rely on recommendations from other people. In context of the aforementioned scenario, we would consider word of mouth, restaurant reviews, or ratings provided on laptop purchases by other customers, as data that helps us make a more informed decision.

Recommender systems (RS) are an automated extension of this intuitive process. RSs are a set of tools and techniques that provide suggestions of items to users [RV97]. Broadly, we can say these suggestions are predictions of items that the user would consider valuable. These predictions are made on the basis of *user preferences* such as information provided by the user or behavioural patterns of the user. RSs aim to provide a more personalized set of suggestions to the user, saving her the effort and resources spent to explore all the available options. Proving to be extremely convenient to users, over the past decade RSs have made significant progress in providing highly valuable and accurate recommendations to their users by using increasingly sophisticated algorithms. While RSs are highly effective in filtering recommendations that are most suitable for the user, they do raise crucial concerns with respect to users' privacy. The information collected by the RSs in order to provide accurate recommendations is at risk of being accessed by unauthorized or malicious entities. RSs inevitably risk mishandling private user information as a trade-off for providing highly accurate recommendations. This user information that is collected by RSs may include contact information such as addresses and telephone numbers, geographical location of the user, information shared on social media, email addresses, financial history, behavioural patterns of web surfing, and purchase history. Some of this information is characterized as

Personally Identifiable Information (PII). Leaking of such PII about users can be especially catastrophic when we consider RSs in use cases such as medical databases being used by doctors for finding similar diagnoses or financial databases used by insurance companies to determine users' credit scores. Even data that is considered harmless or not sensitive to user privacy can have undesirable implications when aggregated together.

Along with mishandling user information, another common risk that RSs take arises from misplaced trust. Such RSs often model their algorithms in an environment which is not adversarial and assume that all the entities involved in the system are honest. Another popular approach that RSs employ is to used a separate "trusted" entity to perform all the critical and secure computations. Such systems depend on this trusted third party to be *honest* to provide secure recommendations. But a malicious entity that can gain access to a trusted third party or collusion between the trusted third party and RS servers can lead to a single point of failure, resulting in exposing the user preferences.

Our goal in this work is to implement a RS which enables user privacy such that only the intended user herself has access to the provided preferences and the generated recommendations. Our protocol does not allow any other entity in the system to be able to view this information. We summarize our goal in the following thesis statement:

**It is possible to construct a recommender system that maintains the privacy of the users' preferences as well the recommendations received, by tolerating collusions among the servers up to a threshold limit, and generate recommendations in an efficient manner without using a trusted third party or relying on availability of other users.**

In Chapter 2 we discuss the working of RSs in general and study the available solutions for maintaining user privacy in RSs. In Chatper 3 we describe our protocol for a privacy-preserving RS, and in Chapter 4 we present the results of the tests we performed and evaluate how they compare to other solutions. Finally we provide a summary of our work and future prospects related to it in Chapter 5.

# Chapter 2

# Background

With the Internet becoming easily accessible to more parts of the world, and the increase in the amount of digital information available, we are faced with the *information overload problem*. Users have a daunting number of results to choose from when they are searching for a specific product or service, often not having enough expertise to make this decision. This challenge faced by users is what primarily led to the development of recommender systems, as people tend to rely on the recommendations of their peers to make routine decisions in their daily lives.

Amazon is one of the leading online retailers in the world. It provides a large inventory of 12 million products and services of various categories to its users.[1] It uses recommendation algorithms to provide the user with results based on the query the user makes. This is an example of how recommender systems assist the users in making better decisions while also reducing the costs for searching. In doing so the recommender systems are beneficial to both the users and the service providers, leading to a comprehensive body of work in improving recommender system techniques. Amazon and Netflix are among the most popular examples where recommender systems are used to provide new recommendations to their users based on the shopping and viewing habits.

In order to provide more personalized recommendations, recommender systems need information about the users. Friedman et al. [FKV+15] mention that the amount, the quality, and the freshness of user information impacts the caliber of the generated recommendations. However the same factors also impact the severity of a privacy breach on users. This is referred to as the privacy-personalization trade-off as noted by Awad and Krishnan [AK06] as well as by Li and Unger [LU12].

[1]https://www.repricerexpress.com/amazon-statistics/

The rest of the chapter is structured as follows. In Section 2.1 we will provide an overview of recommender systems and how they work. In Section 2.2 we discuss the risks faced by users while using services provided by recommender systems. In Section 2.3 we discuss the different types of techniques used to improve privacy guarantees in recommender systems. Finally, in Section 2.4 we survey the solutions available for providing more privacy-preserving recommendations.

## 2.1 Fundamentals of recommender systems

Recommender systems use multiple techniques and algorithms to generate the recommendations. These techniques have become increasingly nuanced as recommender systems became popular. Understanding the functioning of the these recommender systems requires the knowledge of certain core concepts, techniques, and similarity measures. We provide a brief summation of these topics in this section as discussed by Isinkaye et al. [IFO15].

### 2.1.1 Core concepts

Recommender systems offer recommendations of items or products to their users based on the preferences provided by the users. Users, items, and preferences can be considered as the inputs for recommender systems, while the recommendations generated are the desired outputs.

- **Users** of a recommender system are participants who have diverse interests. User profiles are key for providing accuracy in some recommendation algorithms and they consist of multiple parameters such as age, location, and purchase history. These parameters differ depending on the service the recommender system is providing.

- **Items** in a recommender system are objects or services that are desired by the users. The quality of the output of a recommendation algorithm depends on the item profiles in some cases. Considering an example of a movie dataset, the item profiles for the movies can consist of parameters such as movie genre, actors, and directors.

- Depending on the purpose of the recommender system, the **preferences** can be ratings of books, reviews of movies, or symptoms of medical conditions. Preferences can be represented in multiple formats such as: integer range [1–5] or binary value [like, dislike]. Any interaction between the users and items can be used to represent the preferences. The users and items are often represented by a *ratings matrix*, so that the correlation between them can be used to improve the final output, the **recommendations**.

The quality of the final recommendations depends on the quality of the user profiles, item profiles, and input preferences provided to the recommendation algorithm. For example, Rakesh et al. [RCR15] analyzed the crowdfunding platform Kickstarter in order to determine what led to some projects reaching their goal as compared to others. They built a framework to recommend investors to various Kickstarter projects in the crowdfunding domain. Instead of using just user or item profiles, they used a heterogeneous mix of project-based traits, personality traits of the investors, location-based traits, and network-based traits to develop a model for their recommendation algorithm. These included parameters such as duration of project, the goal amount, number of shares on Facebook, number of investors interested in the first three days of the project duration, amount pledged by the investors, number of updates on the project, comments, location, details of all the projects in the investor's history, number of projects hosted by the investor, and promotion on Twitter. The results collected from the first three days of their project proved to be an improvement over similar studies done.

### 2.1.2 Phases

The implementation of recommender systems can be divided into three phases:

1. **Data gathering:** In this phase the recommender systems collect as much information from the users as possible. This can consist of explicit preferences like user ratings as well as implicit information such as observing the behavioural patterns of users. These help the recommender system to model a profile for each user.

2. **Learning:** In the learning phase the recommender systems use the information collected in the first phase to train its prediction algorithm. The quality of the user profiles created can be directly associated with the success of the prediction algorithm.

3. **Prediction:** In this phase, the prediction algorithm generates recommendations for the users.

As shown in Figure 2.1 recommender system phases run iteratively, generating a feedback loop. This feedback loop helps the recommender system to refine the data gathering process and improve the prediction algorithm with every pass.

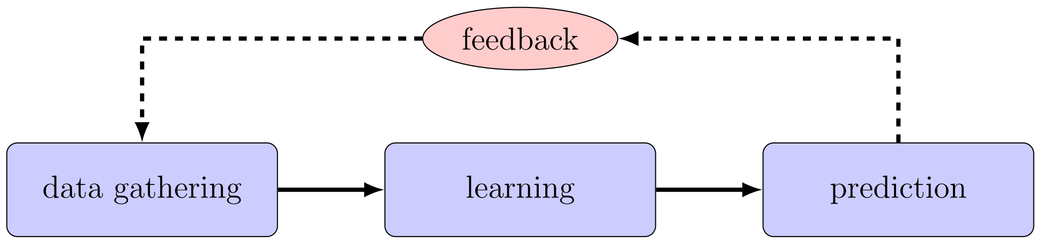

Figure 2.1: Phases in recommender system implementation

It is very common, for example, while using Amazon for users to put items in their carts but check out only at a later date. Smith and Linden [SL17] describe how information such as the number of times a user has viewed the item or compared similar items to it, helps the recommender system to recognize the user's interests and preferences as well as to broadly approximate the user's skills and abilities. Such implicit information helps recommender systems identify salient attributes of both users and the items. Based on the service provided by the recommender system, certain attributes are considered to be more influential to the prediction process than others. The performance of a recommender system largely depends on its ability to extract meaningful data from the information collected. Every interaction between the users and the items is useful in creating nuanced user profiles.

### 2.1.3 Techniques

There are three popular approaches to implement a recommender system based on the filtration design of the recommender model [IFO15].

1. **Content-based filtering:** Content-based filtering follows an item-centric approach. In this technique the recommender system relies on the items and their features to generate recommendations. The recommender system creates user profiles by extracting features of the items present in the user's history. The recommender system suggests new recommendations using these user profiles. This approach does not require the profiles of other users as they do not affect the recommendations. Instead, the quality of the items' metadata influences the quality of the recommendations. Recommendations for songs, videos, news articles, webpages, and scientific publications use content-based techniques. Deldjoo et al. [DEC+16] have developed a system that analyzes YouTube videos and extracts a set of stylistic features such as lighting,

colour, and motion. They use these features to obtain recommendations with higher accuracy as compared to genre-based recommendations.

2. **Collaborative filtering:** The collaborative filtering approach is mostly user-centric. In this technique the recommender system combines the profiles of multiple users, creating a database or matrix of users and items. Explicit user preferences for the items, the user's behavioural patterns, or their interactions with the items can be used to populate this matrix. Using this matrix, the recommender system then calculates the similarities between the users and provides recommendations that those users found relevant. Amazon uses this technique to recommend new products to users. It looks at the similarity of purchase history between users to recommend new items. Linden et al. [LSY03] describe the recommendation algorithm used by Amazon in 2003, an item-based collaborative filtering technique. In 2003 Amazon was primarily a book store and since then their approach to getting recommendations has changed along with the services they provide. Smith et al. [SL17] talk about how they refined their approach in the years since their original work was published. They emphasize the importance of nuanced preferences where every interaction between the users and items is used for providing the recommendations.

3. **Hybrid filtering:** As the name suggests, this approach combines content-based and collaborative filtering methods. The two techniques maybe applied consecutively or applied separately and their outputs are aggregated. Its implementation can be modified according to the needs of the system. Netflix is an example where hybrid filtering is used. It recommends movies to users using their watching habits, their searching habits, as well as other movies having similar characteristics to the ones the user has watched. Geetha et al. [GSFS18] use a hybrid system for generating movie recommendations. They first use a content-based algorithm to generate user-rating vectors for every user in the database using item attributes. Then using Pearson correlation between the rating vectors, all the users are compared with the active user and their similarity is measured. The users with the highest similarity form a neighbourhood. Finally recommendations are predicted from a weighted combination of the selected neighbour's ratings.

### 2.1.4 Similarity measures

Calculation of the similarity measures between users is critical in generating high-accuracy recommendations. Collaborative filtering is one of the most widely used techniques in recommender systems where recommendations are provided based on the previous preferences of users having similar interests. Our solution uses a collaborative filtering approach to generate recommendations for the users. We therefore now examine some ways of calculating similarity measures commonly utilized in collaborative filtering.

In collaborative filtering the user's preferences are stored as ratings in some kind of database like a user-item matrix where $r_{u,i}$ is the rating user $u$ has given item $i$ and $r_{v,i}$ is the rating user $v$ has given item $i$.

1. **Pearson correlation similarity:** The Pearson correlation similarity (PCS) is a popular metric used in collaborative filtering. It is used to find a linear correlation between the users and items based on the ratings provided. $\overline{r}_u$ and $\overline{r}_v$ represent the average rating of user $u$ and user $v$ respectively for items rated by both users. $I_{uv}$ is the set of items rated by both users. The PCS measure results in a value between -1 and +1, where a similarity measure of +1 indicates high similarity. It is calculated as follows:

   $$PCS(u,v) = \frac{\sum_{i \in I_{uv}} (r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{u,i} - \overline{r}_u)^2 + \sum_{i \in I_{uv}} (r_{v,i} - \overline{r}_v)^2}}$$

   Since Pearson correlation similarity considers only those items which have been rated by both users, it fails to provide an accurate result when two users have only one rating in common or in the case where users have only a single rated item.

2. **Cosine vector similarity:** Cosine vector similarity (CVS) is another commonly used similarity measure. The users are represented as vectors consisting of ratings the user provided for each item. Cosine vector similarity between two users is the measure of the cosine of the angle formed between the vectors representing the two users. A cosine similarity measure closer to 1 indicates high similarity between the users whereas 0 indicates no similarity. $r_{u,i}$ and $r_{v,i}$ are the vectors consisting of the list of ratings provided by users $u$ and $v$. $I_{uv}$ denotes the set of items rated by both users, $u$ and $v$. The similarity measure is computed as follows:

   $$CVS(u,v) = \frac{\sum_{i \in I_{uv}} r_{u,i} r_{v,i}}{\sqrt{\sum_{i \in I_{uv}} {r_{u,i}}^2} \sqrt{\sum_{i \in I_{uv}} {r_{v,i}}^2}}$$

   A major drawback of cosine vector similarity is that it does not take into account the difference in the scale of user ratings. For example, user $u$ has a set of item ratings $(1, 1, 2)$ and user $v$ has rated the same items as $(5, 5, 4)$. The similarity measure between these two users would be 0.9 indicating a high similarity in user preferences, even though their ratings indicate the exact opposite.

3. **Adjusted cosine vector similarity:** The Adjusted cosine vector similarity offsets the drawback of the cosine vector similarity by subtracting the corresponding user's average rating from each co-rated pair of ratings. It is calculated as:

$$AdjustedCVS(u,v) = \frac{\sum_{i \in I_{uv}} (r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{u,i} - \overline{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{v,i} - \overline{r}_v)^2}}$$

   Considering the same two users from the previous example, adjusted cosine vector similarity provides a similarity measure of -1. This value represents the user preferences more accurately.

4. **Jaccard similarity:** The Jaccard similarity index compares the similarity between two sets of user preferences. It records which values are common and which are distinct. The measure of similarity ranges from 0% to 100%, where a higher percentage indicates more similar users. $I_u$ and $I_v$ denote the set of items rated by user $u$ and $v$ respectively. Jaccard index is calculated as:

$$J(u,v) = \frac{|I_u \cap I_v|}{|I_u \cup I_v|}$$

   Jaccard similarity index considers which items the users have rated and not at the actual rating values. This leads to a loss of valuable information.

5. **Mean squared difference:** The Mean squared difference (MSD) considers absolute ratings instead of the number of common ratings. It measures the similarity between two users $u$ and $v$ as the inverse of the average squared difference between the ratings of the two users on co-rated items. $I_{uv}$ denotes the set of items rated by both users, $u$ and $v$. MSD is calculated as:

$$MSD(u,v) = \frac{|I_{uv}|}{\sum_{i \in I_{uv}} (r_{u,i} - r_{v,i})^2}$$

The mean of the squared differences provides the similarity measure where the lower the mean squared difference, the greater the similarity between the two users. Desrosiers et al. [DK11] have noted that finding negative correlation between users can improve prediction accuracy of collaborative filtering. The disadvantage of using mean squared difference is that it does not capture negative correlation between the users.

6. **Similarity measure for Ghost Recommendations:** The similarity measure we have used in our solution is relatively straightforward since our main goal is providing privacy preserving recommendations or **Ghost recommendations**. Our similarity measure (Ghost_Sim) uses the intersection and difference functions from set theory. Unlike the Jaccard similarity measure, we consider not just the item but also its rating, while checking for common values between two users.

   Consider $I_u$ to be a set of (item, rating) pairs indicating the preferences of a user $u$. The user $u$ is looking for new recommendations from the other $n$ users in the recommender system, based on the preferences provided by the user $u$.

   For our similarity measure, we first use the intersection function to find the items that have identical ratings for the users $u$ and $v$, where $v = 1, \cdots, n$. $I_v$ represents the preferences of user $v$. The intersection of the two users $u$ and $v$, is represented by $I_{uv}$, which includes the common values between them. Next, we check if the total number of common values between the two users is above a predefined threshold, $thr$. Once we find that the user $v$ meets this criteria, we use the set difference function to check how many dissimilar items are present in the set $I_v - I_u$. Of all the users that matched the threshold criteria, only three users having the most dissimilar items are recommended to user $u$. $argmax3_{w \in \{1,\ldots,n\}}$ returns the set of the three values of $w$ for which the argument is largest, thus selecting the users with the highest number of dissimilar items. Ghost_Sim$(u, v)$ is set to 1 if both the threshold and the $argmax3$ conditions are met, otherwise it is set to 0.

   Ghost_Sim$(u, v)$ is calculated as follows:

$$Ghost\_Sim(u,v) = \begin{cases} 0, & \text{if } |I_{uv}| \leq thr \\ 1, & \text{if } |I_{uv}| \geq thr \text{ and } v \in argmax3_{w \in \{1,\ldots,n\}}(|I_w - I_u|) \end{cases} \quad (2.1)$$

   The user $u$ will receive three users where Ghost_Sim$(u, v) = 1$. Having a high number dissimilar items rated by these similar users is useful as they are provided to the user $u$

as new recommendations. Desiring a larger set difference value is not required for the measure of similarity between the two users, but rather for the measure of usefulness. A high number of common rating between users $u$ and $v$ indicates common interests, but also a high number of uncommon ratings indicates the usefulness of the user $v$ for providing recommendations to the user $u$.

We have used movie ratings for implementing our solution which uses a "half-star" rating system having ratings from 1 to 5 with 0.5 increments. Following prior work, we supplement the input user's ratings with some extra values, by adding and subtracting 0.5 from each of the rating values. This will ensure that an exact match of ratings of a given movie between users $u$ and $v$ will result in three matches; ratings 0.5 apart will results in two matches; ratings 1.0 apart will result in one match.

Chapter 3 includes an in-depth description of how we calculate this measure and use it to acquire useful recommendations.

## 2.2 Privacy concerns in recommender systems

Friedman et al. [FKV+15] note that in today's information age, people regard their personal information as a commodity and that they are willing to give up some personal information in exchange for more refined services. RSs fit this use case perfectly. Gaining abundant user information is crucial for providing personalized recommendations to users. RSs thus benefit greatly by acquiring demographic knowledge provided by their users such as age, gender, ethnicity, nationality, and location.

Jeckmans et al. [JBE+13] have classified the different kinds of information that the RSs have access to. We have listed them below.

- **Behavioural information** consists of implicit data collected by the recommender system as the user interacts with various items. For example, the time the user spent viewing a particular item on Amazon before buying it can be considered as behavioural information.

- **Contextual information** relates to the context of the service provided by the recommender system. Location, time, and date of the user interaction are examples of such information. For example, location based information is typically used by travel recommender systems to make relevant predictions for their users.

- **Item metadata** includes in-depth knowledge of the item features. Genres of movies are examples of metadata for a movie recommender system.
- **Purchase or consumption history** includes list of items previously bought or services consumed by the user.
- **Feedback** is the information the recommender system gets from the users after they have received the recommendations. It is usually represented as a positive or negative value representing the quality of the recommendations.
- **Social information** relates to the data that the recommender systems can gather from social media. Websites like Twitter or Facebook can provide information about the user's relationship with other users to the recommender system.
- **User attributes** are used to represent characteristics of the user. These usually include demographic details of the users.
- **User preferences** are the explicitly provided opinions of the users and are represented in various ways depending on the service provided by the recommender system. For example a range of values from 1 to 5 or a binary set indicating "like" or "dislike".

As the algorithms used by RSs become increasingly sophisticated, they not only generate recommendations but they have the capability to draw new inferences about the users. They can correlate the information listed above with data publicly available, to generate previously unavailable data about the users. Additionally, since this newly extracted data was not originally provided by the users themselves, depending on the local privacy laws RSs could use this information however they see fit, without having the explicit consent of the users. This often leads to misuse of user information. We note examples of such unethical behavior below. Collecting data directly from users and using the newly generated inferences allows RSs to build a comprehensive model of the user. This allows RS to create fairly accurate profiles for users who have provided limited data. *Exposing, accessing, or using this inferred data, without the user's explicit consent*, could lead to a major privacy breach of the user's information. Some users might not be aware of the implications of these inferences, in which case they might unknowingly allow this violation of the privacy of their information.

RSs generally store the information collected from the users in some form of a dataset. Jeckmans et al. [JBE+13] have observed that privacy risks may also arise depending on *how securely the users' information was collected and stored in these datasets*. We have listed the privacy risks faced by user related to these datasets below.

- **Data collection:** While subscribing to a recommendation service, users mostly have no choice of opting out of data gathering performed by the recommender systems. Data collection relates to the amount of data collected by the recommender systems.

- **Data retention:** Information the user provides online to recommender systems is very hard to discard. Even if it is seemingly removed, a copy of it may still exist elsewhere in the system. The data retention concern is in context with the availability of the information for a duration longer than the intended duration.

- **Data sales:** A major concern for storing information online is the risk of it being sold to interested third parties. The information used by recommender systems is extremely valuable for marketing companies. Running advertisements generates income for service providers and information about users can be extremely useful for providing targeted advertisements to them.

- **Improper handling of private information:** The recommender system has complete access to the information collected about the users and their preferences. If the recommender system behaves in an adversarial manner or an adversary has breached the security of the system, they can leak users' private information.

- **Revealing information through inference:** Recommender systems have access to the user's input preferences as well as the recommendations provided to them. A lot of inference can be derived about the users from this data. Inference attacks by correlating various sets of data can risk the users' privacy.

Next we discuss examples that illustrate different kinds of privacy risks mentioned above.

Weinsberg et al. [WBIT12] developed multiple algorithms to infer the gender of a user using MovieLens and Flixster datasets using machine learning. A user's profile consists of a set of (movie, rating) pairs, such that the movies rated by the user are a subset of a universal set of movies. This user profile is submitted to a recommender mechanism which implements the gender inference module. This gender inference module is a classification technique used to profile and label the user's gender. The recommender and gender inference mechanisms are presumed to have access to a training dataset consisting of a set of $\mathcal{N} = \{1, \cdots, N\}$ users. Each of these users has rated a set of movies which are present in the universal set of movies, $\mathcal{M} = \{1, \cdots, M\}$. $\mathcal{S}_i \subseteq \mathcal{M}$ denotes the set of movies rated by user $i \in \mathcal{N}$. This training set also contains a binary variable $y_i \in \{0, 1\}$ indicating the user's gender, where 0 is mapped to male users. To train the classifiers, each user in the training set is associated with a characteristic vector $x_i \in \mathbb{R}^M$ such that $x_{ij} = r_{ij}$, if $j \in \mathcal{S}_i$ and $x_{ij} = 0$, otherwise. To check whether inferring a user's gender from their ratings is possible, Weinsberg et al. used three different types of classifiers: Bayesian classifiers, support vector machines (SVM), and logistic regression. Each of these classifiers were evaluated on both the Flixster and MovieLens datasets. They computed the average precision and recall for the two genders across a 10-fold cross validation. The class prior classification was used as a baseline method for evaluating the performance of the other classifiers. Using a training set of about 300 users was enough for the classifiers to get above 70% precision. Additionally, they used the logistic regression and SVM classifiers on a binary matrix, which is where a characteristic vector $x$ is defined as $\tilde{x} \in \mathbb{R}^M$, such that $\tilde{x}_j = \mathbf{1}_{x_j>0}$. With this they were able to capture the movies for which a rating is provided. This shows that the simple binary event "watched or not" is much more significant compared to having a rating value of a movie available. They found that using the regular matrix provided a less than 2% improvement, on all measures, as compared to using the binary matrix. This led Weinsberg et al. to make a key observation that whether the user watched the movie or not provides a strong correlation to the user's gender regardless of the rating given to it. This example illustrates how **behavioural information** can be used to predict the gender of other users, who had not originally provided that information. So regardless of the fact that some users may choose to limit the information they share about themselves with the RSs, this information is exposed, violating that user's privacy. Such inference attacks usually exploit the information provided by some users to derive sensitive and private information about other users.

Narayanan and Shmatikov [NS08] de-anonymized the Netflix Prize dataset by showing how an adversary having limited information about an individual user can use that information to identify that user's records in the anonymized dataset. This dataset consisted of just the user's ratings and dates when they watched the movies, of 500,000 Netflix subscribers. All remaining identifying information about the users was removed from the dataset and the dataset was subject to perturbation. With auxiliary information of just eight movie ratings, of which two may be wrong, and rating dates known within a 14-day error were used to uniquely re-identify 99% of the records in the dataset. The authors note that the auxiliary information can be easily obtained by a adversary from social media where users might mention their favourite artists or get the public ratings provided by users on IMDb. The de-anonymization algorithm demonstrates how using this easily acquired, limited information, an adversary can identify whether an individual user's records are present in the dataset.

Furthermore, by cross-correlating the anonymized records from the Netflix dataset with publicly available records from IMDb, the authors were able to learn sensitive information about a user's political affiliation and sexual preferences. This work effectively demonstrates how **user preferences** and **social information** of users can be used to violate the user's anonymity. It also shows how an adversary can acquire personal information about users from information gathered by RSs.

We have seen several examples of massive data breaches recently which led to sensitive information about the users being disclosed. The Facebook data breach of 2018 is one example where personal data on approximately 87 million users was collected without their consent. Using a third party app, Facebook users were paid to take personality tests where they consented to the collection of that data for academic purposes. However, the app also collected information on the test participants' Facebook friends, without their consent, resulting in a dataset of tens of millions of users. Cadwalladr and Graham-Harrison [CGH18] revealed how this data was used to create profiles of users based on their political leanings in order to target them with personalized political advertisements. This is an example of how **improper handling of data** and **data sales** was used for the unsolicited collection of user data, unlawful disclosure of that data, and revealing sensitive information about them. It also shows how users can be influenced by skewing the information they can view.

Calandrino et al. [CKN+11] show how any person can carry out an **inference attack** by using a limited amount of information about a user to surmise their private transactions. They have tested their solution using data publicly available from websites like Amazon. Calandrino et al. view the data a recommender system uses to make recommendations as a matrix with rows representing the users and columns the items. Instead of considering just the rating provided by a user on a particular item, the cells of this matrix represent transactions between the user and the item. Every time the user interacts with the system, the matrix is updated. They demonstrate three passive inference attacks using the public outputs of the recommender system; specifically they show an inference attack on related-items list, on the covariance matrix, and on the k-Nearest Neighbours recommender system. They also used different sources of auxiliary information to perform these inference attacks. Such sources include websites where users have publicly rated or commented on items, websites that explicitly mention a verified purchase of the user, user's sharing their feedback or reviews of the items on third-party websites, and mentions of a particular item on online forums.

In the first inference attack, the attacker uses the related-lists the recommender system outputs. An example of this is Amazon displaying the list, "Customers who bought this item also bought ..". Such lists appear on other websites as well. News websites such as CNN or New York Times provide users suggestions on what articles to read next. The attacker considers the target user's auxiliary information list, which consists of items the attacker knows to be associated with a target user. The attacker then monitors related-lists associated with each auxiliary item and records the movements of each target item in the these lists. The attacker then computes a score for each target item counting the number of auxiliary items in whose related-items lists it has appeared or moved up. Checking this over a period of time, if this score is more than a predefined value, the attacker concludes that the target item has been added to the user's record. This approach exploits the changes in covariance over time. It shows that information revealed by a recommender system, such as the item similarity lists and item-to-item covariances, are based on the user's public transactions. The algorithms leverage this to infer the users' non-public transactions, posing a threat to privacy. Although we discuss just one of the attacks mentioned in this solution to illustrate **inference attacks**, Calandrino et al. mention several other algorithms that can be performed to expose the privacy implications that arise in data collected by recommender systems.

## 2.3 Privacy-preserving techniques used in recommender systems

Being a tool that has proven to be extremely useful to its users and also profitable to the service providers, there has been substantial research conducted over the past 20 years towards improving the accuracy and performance of recommender systems. At the same time, the growing awareness of user privacy and the risks to it has led to advances in academic solutions for developing privacy-preserving recommender systems.

Providing high accuracy of the recommendations, improving the overall performance of the system, and providing user privacy are some of the goals these solutions aim for. But achieving all of these goals can be a challenge since these metrics are conflicting in nature. Owing to the trade-off between *accuracy*, *performance*, *utility*, and *privacy*, there are myriad solutions for privacy-preserving recommender systems. Many studies have been done to understand the different trends in privacy-preserving recommender systems in order to determine the best metrics of an ideal privacy-preserving recommender system.

In this section we discuss the different privacy-preserving techniques used by recommender systems as presented by Wang et al. [WZJR18] and Ogunseyi et al. [OY18] and how these techniques are used in various privacy-preserving solutions. The solutions can be broadly classified into the following categories:

1. Data Perturbation

This technique involves modifying the user input. This is achieved by introducing a measure of randomness or by masking the input data. The goal is to reduce the risk of exposure of an individual user's input by adding some noise to it such that the input is very different from the user's original input.

Polat and Du [PD03] used perturbation techniques to protect a user's data by adding random noise to it before sending it to the recommender system. The recommender system could provide accurate recommendations to the user as it used the aggregate of input values from multiple users. As long as the number of users participating is sufficiently large, the recommendations generated were of decent accuracy. This technique does trade accuracy for user privacy, but since computational overhead is low, it is much faster than other solutions that use cryptographic algorithms. However, this approach is not suitable for providing strong security for user data. Aggarwal [Agg06] exposed the vulnerabilities of this technique by demonstrating how it can leak information about the user.

Shokri et al. [SPTH09] proposed a technique to obfuscate the user-item connections from an untrusted server by ensuring that their users maintain separate online and offline profiles. While the untrusted server uses the online profiles to generate recommendations for the users, they do not have access to the offline profiles. Moreover, each user randomly chooses some other users to add items from their offline profiles to her own profile. This adds another layer of obfuscation for hiding the user's actual rated items. However, this system too trades off accuracy of recommendations for user privacy. It also leaks information about the user's interest on a broader scale.

Depending on the user's privacy requirement and sparsity of the input data there are different kinds of perturbation and obfuscation styles that are used, as mentioned below:

(a) Data obfuscation by substituting a default value: The user's data is substituted with a predefined value.
(b) Data obfuscation by substituting a uniform randomized value: The user's data is substituted by a random value chosen uniformly from a range of values.
(c) Data obfuscation by using values from a bell curve: The user generates a standard normal distribution of her input values. She then chooses a percentile and selects random numbers from this range to substitute values in her original input.
(d) Data perturbation using a values from a bell curve: Instead of substituting the user's data entirely, we augment the user's input with some random noise. The

random noise is generated in the same way using a standard normal distribution of the user's original input.

   (e) Data obfuscation by permutation: In this method, clusters of similar items remain consistent after applying the obfuscation to the user's input. The ratings of similar items is swapped between two users. This ensures that the aggregate of the data after applying permutation is very close to the aggregate of true data.

   (f) Randomized response: This is a fairly common technique used to suppress and generalize the user's input. Users can randomize their input using a known probability distribution, where they provide the true value of the data with a probability $p > 0.5$ and for a known probability $1 - p$, they send a dummy value for their data. Users can also split their data into groups, and apply randomized response for each set of the data.

   (g) Random filling: This technique deals with masking the unrated data cells. Recommender systems can populate unrated cells, uniformly selected at random, and fill them with some default values. The default value may be personalized depending on the user or item information available. This is used when the data provided by the user is sparse or incomplete.

2. Anonymization techniques

   Another technique for enhancing the user's privacy is by obscuring the link between the user and their input data, which can be achieved through anonymizing the input. In this approach the user's personally identifiable information (PII) is either removed from the input or transformed. This is done to protect the user's identity while maintaining the utility of their input. The recommender system can compute relevant recommendations using the aggregate of the input data. The anonymization method used to hide the user's PII will impact the security guarantees provided for the data.

3. Using a trusted third party

   Using a separate trusted third party is a technique that is widely used for providing privacy in recommender systems. The assumption in this case is that the trusted third party will always behave honestly. This trusted third party is used to perform computations such anonymizing the user's input and then sending the modified input to the recommender system. Then recommender system can generate private recommendations for the users without having access to the original user input.

   Both these techniques, anonymizing user input and using a trusted third party, are illustrated in the system designed by Guha et al. [GCF11]. Their system was developed for providing private advertising to users. Along with users and recommender servers, their system includes an additional participant called the dealer. The dealer anonymizes the interaction between the user and the servers by encrypting the user's input. However, availability of the dealer is critical for the implementation of this system, which is not ideal.

4. Using cryptographic techniques

   Cryptographic techniques are used by most of the latest solutions for privacy-preserving recommender systems. Compared to the methods discussed before, cryptographic techniques have proven to be extremely effective in addressing privacy challenges of recommender systems, although the computational overhead using these methods is significantly higher. These techniques use tools such as: public key encryption (PKE) schemes, secure multi-party computations (SMC), verified secret sharing (VSS), Homomorphic encryption (HE) schemes, partially homomorphic encryption (PHE) schemes, and garbled circuits (GC). One of the techniques used by recommender systems for getting high-accuracy recommendations is matrix factorization. This method is used frequently, along with the cryptographic solutions mentioned before. Matrix factorization uses the user-item interaction matrix. It represents this matrix into the product of two lower dimentionality matrices therefore generating a lower dimentionality latent space.

   In Section 2.4, we will discuss solutions using these techniques and then compare them.

## 2.4 Solutions for privacy-preserving recommender systems

There is a large body of work available for academic solutions that implement privacy-preserving recommender systems. These techniques provide varying levels of privacy guarantees that depend on the protocols used to design the system.

We have selected the following solutions firstly to illustrate a wide spectrum of different privacy-preserving approaches. Secondly, we classify them using five characteristics focused on privacy-preserving aspects. These characteristics are dependence on a trusted third party, computational overhead incurred, the reliance on the availability of other users, effectiveness against a malicious adversary, and how well they preserve the integrity and confidentiality of the users and their data. The solutions selected can each be represented using these classifiers.

We want to provide a comprehensive comparison of the different techniques available. This comparison also informed us of the different design choices we should implement in our solution so as to overcome any shortcomings we observed in the solutions noted below.

The selected solutions are presented below in order from least sophisticated to most sophisticated techniques used, followed by the comparison of the solutions.

Berkovsky et al. [BEKR07] consider a peer-to-peer (P2P) framework to represent their solution. In order to accommodate the privacy constraints, they initially use data obfuscation techniques performed by the users themselves, and then store the users' profiles in a distributed manner. One of the key benefits of implementing the privacy-preserving computation on the client side is that the users can have complete control over their data and decide exactly how to share their data. Since this solution is implemented in a P2P network, the connectivity of all the users allows them to directly communicate with each other.

The active user begins the process to get predictions for a specific item by broadcasting a request to the other users. This request includes the active user's profile, which represents the user's ratings for a set of items in a ratings vector. Here, the active user can partially obfuscate the ratings vector before broadcasting it to the other users. The active user can also select which of the other users to broadcast this request to. Once the active user's modified profile is sent out to the other chosen users, each of the chosen users have the option of responding to the active user's request. If the user decides to respond, they compute the cosine similarity measure of their profile with the active user's modified profile. This similarity measure and the user's rating for the specified item is sent back to the active user. The active user then selects users with the highest similarity measure, and performs a weighted average of the ratings depending on the similarity measure sent back by the chosen users.

Although this approach requires sending the active user's profile over the network, one of the main contributions of this work is that they have demonstrated that it is possible to get predictions from a recommender system while using only a small portion of the modified user profile. They have shown this by using three data obfuscating techniques: default obfuscation, uniform random obfuscation, and bell curved random obfuscation. These obfuscation policies were applied to two cases: (1) **obfuscate all the ratings**, and (2) **obfuscate only extreme ratings**. The authors test these polices on (1) **predictions of all ratings**, and (2) **predictions of extreme ratings**. The experiments were conducted on the MovieLens dataset, where they considered two datasets: full, which consisted of all the available ratings, and extreme, which consisted of only those ratings where more than a third of a user's ratings had the variance farther than the average by more than 50%. Their results showed that the mean square error of the predictions increases linearly with the data obfuscation rate and approaches the accuracy of non-personalized predictions. With their results, the authors were successful in showing that the impact of obfuscating extreme ratings is stronger than of obfuscating moderate ratings. This allowed them to conclude that the extreme ratings are more important for the accuracy of CF recommendations.

While this solution does not rely on a central server to store user profiles and hence avoid it being accessed for malicious purposes, the solution offered is not feasible to use in the real world. Obfuscating the user's profile does not ensure that integrity or confidentiality of the user's data will be maintained. Additionally, depending on the size of the network, there will be significant costs in communication if a user needs to broadcast the profile to all the other users in her network, in order to get a prediction value. This is also not desirable since this would cause a dependency on other users being available at the time.

Next we look at the solution presented by Weinsberg et al. [WBIT12] which uses matrix factorization for generating recommendations. Matrix factorization is a commonly used method in collaborative filtering, where both users and items are mapped to a joint latent factor space of a set dimensionality, such that user-item interactions are modeled as inner products in that space. It is an effective way of generating latent features between two different kinds of entities. This solution uses a recommender mechanism that implements matrix factorization.

As discussed in Section 2.2, Weinsberg et al. demonstrated how easily the gender of a user could be inferred from their user profile. They used a training dataset containing users that have rated a set of movies, and where each user is associated with a binary value indicating their gender. Using a training dataset of 300 users, the machine learning classifiers were able to predict the gender of new users with an accuracy above 70%. They also found that whether or not a user has watched a movie has more impact on the gender-correlation than the rating of the movies. Now we discuss how they used an obfuscation mechanism to diminish this inference.

The obfuscation mechanism alters the user's input profile $S_i$ for the user $i$ by adding certain (movie, rating) values to it before sending it to the recommender mechanism for generating recommendations using matrix factorization. For choosing which movies to add to the user's profile, it uses two ordered lists $L_M$ and $L_F$, generated by the training set, for storing male and female correlated movies in decreasing order of the scoring function $w : L_M \cup L_F \to \mathbb{R}$ and $w(j)$ indicates how closely associated a movie $j \in L_M \cup L_F$ is with the user's gender. $k$ is the number of values added to the input set $S_i$ where $k < \min(|L_M|, |L_F|) - |S_i|$ and $L_M \cap L_F = \emptyset$. For any given female (or male) user $i$, they set $S_i' = S_i$. Then a movie $j$ is picked from $L_M$ (or $L_F$), and added to $S_i'$, if $j \notin S_i'$ until

$k$ values have been added. They used three strategies for selecting the movies to add to the user's profile: random strategy, sampled strategy, and greedy strategy. Considering a female (or male) user $i$, the random strategy picks any movie $j$ from the list of the opposite gender $L_M$ (or $L_F$), regardless of the movie scores. The sampled strategy uses the distribution of the movie scores in the opposite gender's list. Consider movies $j_1, j_2, j_3$ are present in $L_M$ having scores of 0.5, 0.4, 0.3 respectively. Then for a given female user $i$, $j_1$ will be picked with a probability of 0.5, $j_2$ with a probability of 0.4, and $j_3$ with a probability of 0.3. In the greedy strategy, the movie having the highest score in the opposite gender's list is selected.

The next step of the obfuscation mechanism is to assign ratings to the newly added movie values. As the ratings have low impact on the gender correlation, they were able set it to whatever value they needed to maintain the quality of the recommendations. They used two approaches for assigning the rating value. The first approach used an average rating value of all the movies $j \in {S_i}' - S_i$ and added those to the user's altered profile ${S_i}'$. The second approach computes the latent factors of movies by performing matrix factorization on the training dataset, and uses that to predict the ratings. The predicted ratings for all movies $j \in S_i' - S_i$ are added to the altered profile.

The gender-inference accuracy was computed using 10 fold cross validation, where the model is trained on unadulterated data, and tested on the obfuscated data. They found that on adding 1% of extra ratings to the user's profile, the accuracy of predicting the gender drops to 15% from around 75% and by adding 10% of extra ratings, this is close to 0.1%. These results were obtained using the logistic regression classifier on the Flixster dataset while implementing a greedy strategy for selecting the extra movies to be added. They also evaluated the recommendation quality that the user will observe if they obfuscate their gender. They measured this impact by computing the RMSE of matrix factorization. The change in RMSE was not significant with a maximum of 0.015 for Flixster and 0.058 for MovieLens. This method achieves the desired result of decreasing the accuracy of predicting a user's gender by obfuscating their rating profile, thus maintaining their privacy. However, since the output of predicting the user's gender is considered as a binary space, the predicted gender will almost always be just the opposite of the user's actual gender, rendering this result useless. This solution does mention that adding 5% of extra ratings to the user's profile while implementing the sampled strategy to select the movies, results in the accuracy of gender prediction to be approximately 36%, whereas adding 1% of extra produces an accuracy of approximately 60%. Using these settings would prove to be a better strategy while considering this particular use case where the output is binary.

This solution used matrix factorization to obfuscate the users' data, while still preserving the quality of recommendations provided to the users. However, it relies on a training set to provide key computations, and therefore assumes that this data is not tampered with. This approach would not be able provide strong guarantees of confidentiality and integrity for the user's data. It also relies on separate entities performing the recommendation and obfuscation mechanisms. Using this setup again relies on these entities acting honestly, and not being able to defend against adversarial behaviors.

Ying [Yin20] proposes a shared matrix factorization solution for a distributed recommender system, which uses secret sharing. This solution considers federated learning to protect the user data in the distributed setting. Federated learning is used to improve the performance of a recommender system by combining user data distributed across multiple data sources. It enables multiple participants to work together without sharing data, thus maintaining the users' privacy. The users' data is considered to be distributed across different data sources. For example users can buy an iPhone from Amazon or from Apple. This solution considers the scenario where the items are shared across multiple data sources. It also considers that different users rate the same item but on different data sources.

This solution considers $n$ users and $m$ items where each user has rated a subset of the $m$ items. This gives a $n \times m$ rating matrix $M$, where $r_{ij} \in M$ denotes the rating of item $j$ by user $i$. To resolve the possibility of a sparse matrix where not every $r_{ij}$ is available, matrix factorization is used by fitting a bilinear model $M \approx UV$ on the rating matrix. The user profile matrix is denoted as $U \in R^{n \times d}$ and the item profile matrix is denoted as $V \in R^{d \times m}$. $U_t$ denotes the user profile for a data source $t$, where $t = 1, \cdots, T$ and $T$ is the total number of data sources.

Each data source has its own user profile, $U_t$, which is private. Each data source uses their respective rating matrix to update these user profiles. The updated $U_t$ is used to compute the item matrix gradient, $g_t^{plain}$. Then each data source performs secret sharing to split $g_t^{plain}$ into shares $g_t^{sub_1}, \cdots, g_t^{sub_T}$. Each data source keeps $g_t^{sub_t}$ and sends the other shares to the other respective data sources. Then it computes an interpolation of the shares it received from the other data sources to compute $g_t^{hybrid} = \sum_{i=1}^{T} g_i^{sub_t}$. This hybrid gradient value is sent to the central server. The central server computes an interpolation of the hybrid values received from each data source to get $G = \sum_{t=1}^{T} g_t^{hybrid}$. As the central server stores the public item profiles, it uses this interpolated gradient value to update the item profiles.

This algorithm was tested using the MovieLens dataset. To verify the improvement of using a distributed recommendation system, the number of data sources used was varied using one, three, and five data sources. For each additional data source, the number of users increased by 200, while the number of movies was kept constant at 500. The results show that with the increase of data sources, the loss of the matrix factorization model goes down. This is because with an increase in data sources and the number of users, the rating matrix is more perfect, making the item vector fitting better.

They have also compared the performance between the shared matrix factorization algorithm that uses secret sharing against an algorithm that does not use secret sharing. They find that the communication cost caused by secret sharing is less than the computation cost by regular matrix factorization. They evaluated this by checking the time it took for each algorithm to perform for a varying number of data sources.

This method uses matrix factorization and secret sharing techniques which have considerably lower computational overhead. It also relies on a distributed set of servers and thus does not need a third party that is assumed to act honestly. Using secret sharing ensures that the confidentiality of the user's profile is maintained. However, this approach does not consider that an adversarial user can modify or inject data being sent to the central server, influencing the value of item profiles, and thus cannot provide integrity guarantees for the users' data.

Artail and Farhat [AF14] provide a solution for a system that provides ads to users based on their preferences. Their system reads the user's location, the time the request was made, and considers the usage context to provide these personalized ads, while maintaining the confidentiality of this data. They use data aggregation and anonymization techniques to achieve this goal. To deliver personalized ads to users, mobile advertising relies on content providers like webpages users may visit and applications they may use. These content providers subscribe to ad servers, where service providers can register their ads. Every time the user accesses the webpage or the application, an ad request is generated that includes the user's location and her preferences. The ad server then uses this information to deliver targeted ads to the user. Artail and Farhat have designed their system to aggregate the requests of multiple users in order to hide their identities from the ad-server. Their design requires co-operation between a set of users that are requesting the ads. A user can initiate the process by broadcasting an 'ad announcement'. The users who respond to this broadcast form a group. The user's response includes their address and their public key. The initial user then broadcasts a message nominating a 'primary peer' and the list of the users' addresses and public keys. All the users generate their requests using their preferences, location, and billing reports which capture their clicks on previous ads, and encrypt the request using the primary's public key. Additionally, each user randomly selects another user's public key to encrypt their encrypted requests before broadcasting them. This ensures that only a particular user, other than the primary, can decrypt the message before sending it to the primary. Additionally, that particular user who receives this request will know the identity of the sender but will not be able to decrypt the request since it has been encrypted using the primary's public key. This 'shuffling' process hides the identity of the users from the primary when the primary finally decrypts the requests using her private key. The primary then aggregates these requests and sends them to the ad-server. The ad-server replies back to the primary with ads, who then broadcasts them to the entire group. In order to improve the privacy further, they implemented their system by running the shuffling steps multiple times.

They have implemented their solution using Java. They considered a class of users where some could be marked as malicious. Their solution formed random groups of users where the primary user is selected and the shuffling process is conducted to generate aggregated requests. For testing their solution, they designated a 'key user' in the group and after each implementation checked whether the requests of this key user were discovered or not. They kept a tally of instances where the requests were discovered while repeating the implementation one billion times. They considered three scenarios where privacy of the user could be jeopardized: 1) attacks by colluding malicious users, 2) attacks by a malicious ad-server, and 3) attacks by colluding malicious users with the ad-server. They concluded that the probability of the key user's requests being discovered follows the formula: Probability (discovering a user interest) $=$ (% malicious users)$^{n+1} \times$ (% of colluding malicious users)$^{n}$ where $n$ is the number times of shuffling is done. They obtained this formula by using a default value of 20 users per group for $n = 1$, with 10% colluding malicious users. Their tests showed that the probability of tracking requests increases from 0 to 1 per 1000 requests as the percentage of malicious users increases from 0 to 10.

There are a few drawbacks of using this solution. The primary user receives the requests from the users in the group and is able to decrypt them using her private key. This solution assumes that the primary user will not behave in a malicious manner. Without this key assumption, the primary user has the opportunity to alter user requests, impacting data confidentiality of users' requests. An adversary could use this vulnerability to distort the responses received from the ad-server to favour a particular service-provider. Additionally, even though the identity of the users is kept anonymous while generating the aggregated requests, the responses received from the ad-server are broadcasted to all members of the group, diminishing the level of privacy assigned to them.

Samanthula et al. [SCJS15] designed a system for privacy-preserving friend recommendation in online social networks. Other users with similar interests are recommended as new friends, using a privacy-preserving approach. Their solution considers the topology of the social network to check the closeness between users where common neighbours is the measure selected to compute this closeness.

This design uses secure multi-party computation to generate recommendations of a "friend" or another user that has similar interests. Their solution provides two protocols for performing friend recommendation. Each protocol includes a trade-off between security, accuracy, and efficiency.

This solution considers $A$ to be the target user that wants to get recommendations for new friends and $Fr(A)$ denotes the friend list for user $A$ such that $Fr(A) =< B_1, \cdots, Bm >$ where $B_i$ is a friend and $1 \leq i \leq m$. Then the privacy-preserving friend recommendation (PPFR) protocol can be defined as: PPFR$(A, Fr(A), Fr(B_1), \cdots, Fr(B_m), t) \rightarrow S$. Here, $t$ denotes a threshold value and $S$ denotes the list of recommended friends whose common neighbours scores with $A$ are greater than or equal to $t$. The common neighbours score for any two users is defined as the number of common friends between them. The PPFR protocol aims to never disclose the $Fr(A)$ to users other than $A$, keep friend lists of all users private from the network administrator, and finally to make sure that $S$ is known only to $A$.

In the first protocol presented by Samanthula et al., $PPFR_h$, they use a probabilistic additive homomorphic scheme (HEnc) for providing the security guarantee as well as a universal hash function. The protocol also assumes that there is a party $T$, such as the network administrator, which generates the key pair $(pk, pr)$ using the Paillier encryption scheme. $T$ publishes the public key $pk$ and threshold $t$ to the users in the network. Whenever $A$ wants new recommendations for friends, it shares the parameters of the a universal hash function with each $B_i$ in $Fr(A)$. This hash function is used to hash the IDs of each $C_{ij}$ in $Fr(B_i)$ into integers less than $s$, which is the domain size predefined in the hash function. This creates a pseudo and oblivious candidate space. $B_i$ then generates a matrix $M_i$ of size $s \times 2$. $B_i$ hashes each user $C_{ij}$ in $Fr(B_i)$ and sets the corresponding row entries to $Fr(B_i)[j]$ and 1 (indicating $Fr(B_i)[j]$ being $B_i$'s friend). Then each row of $M_i$ is encrypted using the public key $pk$. The encrypted matrix $M'_i$ is sent back to $A$. Upon receiving the encrypted matrices from each $B_i$, $A$ performs component-wise homomorphic additions to get a new matrix $Z$. In matrix $Z$, the second column of each row in $Z$ now contains a frequency of each ID stored in the first column, while all the data is encrypted under $pk$. Next, $A$ permutes each row in $Z$ with a random permutation function to get $\hat{Z} \leftarrow \pi(Z)$ before sending it to $T$. This is done to ensure that no information is leaked to $T$. $T$ decrypts the second component of each row of $\hat{Z}$ using the private key $pr$ which gives the approximate frequency, $freq$, of the corresponding hashed user. If $freq \geq t$, then $T$ decrypts the first component of $\hat{Z}$. Then $T$ sends the decrypted component $\beta$ back to $A$. $A$ applies the inverse of the random permutation $\pi$ on $\beta$ to receive a list of recommended friend IDs.

$PPFR_h$ is able to generate a list of friends for $A$ without disclosing their IDs to the other users or the network administrator, $T$. Using additive homomorphic encryption ensures that

$A$ and $T$ can both perform computations on encrypted data without incurring significant overhead. Additionally, $A$ and $T$ are able to securely perform the decryption of the data in $Z$ without revealing their private inputs. However, the $freq$, which denotes the score of each common neighbour, is leaked to $T$. It also needs to be noted that since the scores are hashed and a random permutation is performed on it, $T$ will not be able to identify the source of this data. Although the $PPFR_h$ protocol can produce friend recommendations without disclosing the friend list to any other entity in the system, the computation costs are dependent on the domain size $s$. The second protocol presented by Samanthula et al. is aimed to overcome this shortcoming.

The second protocol, $PPFR_{sp}$, uses anonymous message routing. In this, the potential candidates for new friends are allowed to introduce themselves. The list of potential candidates includes all the friends included in friend lists of every member of $Fr(A)$. $B_i$ creates a random path for a candidate to use to introduce themselves to $A$. Initially, every potential candidate, $C_{ij}$ uses as AES encryption algorithm to encrypt their data and the secret key is split into $|Fr(C_{ij})|$ shares such that at least $t$ shares are required to reconstruct the secret AES key. $A$ informs all $B_i$ that they want new friends and generates a key pair using the RSA public key system, $(pu_A, pr_A)$. Once $B_i$ receives this message from $A$, it randomly select two users, $X_{ij}$ and $Y_{ij}$, from the network for each friend $C_{ij} \in Fr(B_i)$. Then $B_i$ sends the path $M_{ij} = X_{ij} \| Y_{ij} \| E_{pu_{Y_{ij}}}(A)$ to $C_{ij}$. $B_i$ then sends $M_{ij}$ to $C_{ij}$. $C_{ij}$ receives this value and checks if $|Fr(C_{ij})| \geq t$. If this is true, $C_{ij}$ generates a share of the key $k_{C_{ij}}$ corresponding to $B_i$: $k_{C_{ij}}^{B_i}$. It also generates the encrypted $ID$ of $C_{ij}$: $AES_{k_{C_{ij}}}(C_{ij})$. Then it sends the following value to $X_{ij}$: $Y_{ij} \| E_{pu_{Y_{ij}}}(A) \| k_{C_{ij}}^{B_i} \| AES_{k_{C_{ij}}}(C_{ij})$. After $X_{ij}$ receives this value, it sends $E_{pu_{Y_{ij}}}(A) \| k_{C_{ij}}^{B_i} \| AES_{k_{C_{ij}}}(C_{ij})$ to $Y_{ij}$. Now $Y_{ij}$ can decrypt the first part using their private key $pr_{Y_{ij}}$ to get $A$. Then it forwards the remaining message $k_{C_{ij}}^{B_i} \| AES_{k_{C_{ij}}}(C_{ij})$ to $A$. Finally $A$ waits and collects the returned messages from each $Y_{ij}$ and groups them based on $AES_{k_{C_{ij}}}(C_{ij})$. For each group $G_{ij}$, $A$ performs the following steps. If $|G_{ij}| \geq t$, then $A$ can generate the corresponding key $k_{C_{ij}}$ from any $t$ different keys shares in $G_{ij}$ using polynomial interpolation. Using key $k_{C_{ij}}$, $A$ can successfully decrypt the message $AES_{k_{C_{ij}}}(C_{ij})$ to get the user $ID$: $C_{ij}$ of a newly recommended friend. Otherwise, $A$ dumps the group $G_{ij}$ since the number of partial keys shares are less than the threshold $t$ and $A$ cannot generate the corresponding key from those.

$PPFR_{sp}$ uses AES encryption scheme and secret sharing to generate friend recommendations for the user $A$. However, this approach also leaks some user information. Some common neighbour scores, that are above a threshold value, are revealed to the primary user. However, since the protocol provides source privacy, the primary user is not able to identify the users corresponding to those scores. Therefore, $A$ cannot trace back to the source $C_{ij}$. Although the two random users are necessary for performing this protocol, to increase security guarantees, there can more than two random users selected. $PPFR_{sp}$ is more efficient in providing recommendations to $A$ but it comes at the expense of weaker security guarantees compared to $PPFR_h$.

$PPRF_h$ and $PPRF_{sp}$ are the two options that this solution provides, giving the user the flexibility to adjust the trade-off between security, accuracy, and efficiency depending on the particular scenario. Samanthula et al. define the privacy guarantees for their solution as preserving the privacy of each user's friend list. While both the protocols have accomplished this goal, they assume that the users and the network administrator operate in a semi-honest adversarial behavior.

This design was tested on a set of Facebook users consisting of 11,500 users' friend lists. Using a crawler on Facebook's AJAX API, the friend lists were extracted using regular expressions. This was using only those profiles whose privacy settings allows the friend list to be public. Using this as as dataset, the experiments were implemented in C to evaluate the performance of the solution. Samanthula et al. observed that $PPFR_{sp}$ was more efficient than $PPFR_h$. For a user $A$ having 447 friends and $s = 5000$, total running time for $PPFR_h$ protocol increased from 80.853 seconds to 553.459 seconds when the key size was changed from 1,024 to 2,048 bits. Whereas, for the same key size variation, $PPFR_{sp}$ required less than 50 seconds of runtime. They also observed that while running the $PPFR_h$ protocol, most of the computation time was incurred by the steps performed by $B_i$ and $T$.

Now we look at a solution for privacy preserving collaborative filtering provided by Badsha et al. [BYKB17] that uses a technique based on homomorphic encryption. Their system design includes a recommender server which generates recommendations and a decryption server which performs the privacy and decryption computations. Their solution used the Boneh Goh Nissim (BGN) cryptosystem defined by Boneh et al. [BGN05] where secure multiplications can be computed by a single server. They consider a rating matrix $M_R$ consisting of ratings provided by all $n$ users for $m$ total items, where $r_{ij}$ is the rating provided by user $u_i$ for item $i_j$. They measure cosine distance between two users, $u_i$ and $u_k$, to determine the similarity measure. It is represented as $S(u_i, u_k) = \sum_{j=1}^{m} R_{i.j} \cdot R_{k.j}$.

Their design consists of two stages, initialization and recommendation generation. In the initialization stage, each user $u_i$ first performs the following computation locally.

$$R_{i,j} = \left\lfloor \frac{r_{i,j}}{\sqrt{{r^2}_{i,1} + \cdots + {r^2}_{i,m}}} \cdot t \right\rfloor \tag{2.2}$$

Here, $t$ is a positive integer such as 10 or 100, to accommodate the limitation that BGN system has for being unable to work with fractions. Then the decryption server generates a tuple $(q_1, q_2, G, G_1, e)$, where $q_1$ and $q_2$ are two large prime numbers, $G$ is cyclic group with generator $g$ and $N = q_1 q_2$, and $e$ is pairing map $e : G \times G \to G_1$. The decryption server picks two random generators $g, u$ from $G$ and sets $h = u^{q_2}$. Then $h$ is a random generator of the subgroup of $G$ of order $q_1$. The public key sent to all users is $PK = \{N, G, G_1, e, g, h\}$ while the private key $SK = q_1$ is kept with the decryption server. Each user then encrypts their ratings $r_{i,j}$ and $R_{i,j}$ using this public key as $A_{i,j} = E(R_{i,j}) = g^{R_{i,j}} h^{w_i}$ and $B_{i,j} = E(r_{i,j}) = g^{r_{i,j}} h^{x_i}$ where $r_{i,j}$ and $R_{i,j}$ are integers in the set $\{1, \cdots, T\}$ and $T \ll q_2$. $w_i$ and $x_i$ are the random values selected to generate the different encryptions. Once these encryptions are performed, each user $u_i$ sends the message $M_{1,i}$ to the recommender server, containing the ciphertext: $M_{1,i} = \{A_{i,j}, B_{i,j}\}_{j=1,\cdots,m}$

The recommender servers stores these encrypted values in its storage. Next is the recommendation generation stage where only one user, $u_i$, participates. The recommender server sends $E(R_{i,j})$ of the other users $u_k$ to the user $u_i$ as follows: $M_2 = \{A_{k,j}\}_{1 \le k \le n; j=1,2,\cdots,m}$. The primary user, $u_i$, does a similarity check on these values by performing the following computations locally.

$$\begin{aligned} C_{i,k,j} &= (A_{k,j})^{R_{i,j}} = E(R_{k,j})^{R_{i,j}} \\ D_{i,k} = \prod_{j=1}^{m} C_{i,k,j} &= \prod_{j=1}^{m} E(R_{i,j} \cdot R_{k,j}) = E(s(u_i, u_k)) \end{aligned} \tag{2.3}$$

$D_{i,k}$ denotes the similarity between the ciphertexts of users $u_i$ and $u_k$. User $u_i$ returns these ciphertexts to the recommender server. The recommender server generates recommendations for $u_i$ using the encrypted similarity values and the other user's encrypted ratings. The recommender server uses the bilinear pairing of the BGN cryptosystem as follows:

$$\begin{aligned} F_{i,j} &= \prod_{1 \le k \le n, k \ne i} e(B_{k,j}, D_{i,k}) h^{z_k}{}_1 \\ &= E(\sum_{1 \le k \le n, k \ne i} r_{k,j} \cdot s(u_i, u_k)) \end{aligned} \tag{2.4}$$

where $F_{i,j}$ denotes the encrypted recommendation, $z_k \in \{1, 2, \cdots, N-1\}$ are random numbers, and $h_1 = e(g, h)$. After this the recommender server permutes the encrypted recommendations in a random order.

$$\{H_{i,j}\}_{j=1,2,\cdots,m} = Perm\{F_{i,j}\}_{j=1,2,\cdots,m} \tag{2.5}$$

Then the recommender server signs the message $M_3$ which contains this permuted list of recommendations using a secret key $SK_1$ as $\delta = Sign(M_3, SK_1)$. The recommender server sends the signed message, its public key $PK_1$, and the permutation order to the user $u_i$ as $M_4 = \{\delta, PK_1, permutation\_order\}$. Once the user $u_i$ receives this message from the recommender server, they remove the permutation order and then pass the rest of the information to the decryption server. The decryption server first verifies the signature as:

$$Verify(\delta, PK_1) = True|False \tag{2.6}$$

After verifying the signature, the decryption server decrypts the ciphertexts of the encrypted recommendations as follows:

$$\begin{gathered} d_{i,j} = (H_{i,j})^{q_1} \\ P_{i,j} = log_{g^{q_1}}(d_{i,j}) = \sum_{1 \leq k \leq n, k \neq i} r_{k,j} \cdot s(u_i, u_k) \end{gathered} \tag{2.7}$$

$P_{i,j}$ represents the recommendation prediction of item $i_j$ for user $u_i$. The decryption server performs these computations and finds the maximum score among all the values in the permutation list and sends the corresponding index to the user $u_i$. Finally the user $u_i$ can retrieve the actual item by using the *permutation_order*.

This design was tested on the GroupLens data, using 200 users and 500 items. Compared to the previous examples we have discussed, this design is able to provide higher standards for maintaining the integrity of the user's input rating as well as the output recommendations. It does however rely on the servers being semi-honest and the assumption that no collusion takes place between any participants of the system. Additionally, this design requires significant computations to be performed on the user's end. The initialization stage takes 0.02 seconds for all users to encrypt their ratings whereas a single user $u_i$ needs 2.5 seconds to compute the similarity check with the other 199 users. The recommendation server and the decryption server need 52 and 63 seconds respectively to perform their tasks.

Erkin et al. [EVTL12] propose encrypting the private data and performing computations on them using collaborative filtering techniques to generate recommendations. They aim to keep the user's data private and preserve the functionality of the system while protecting the data from an adversarial service provider. This is made possible by introducing a semi-trusted third party called the privacy service provider (PSP). The users are upload their encrypted data to the service provider (SP). The service provider and the PSP generate recommendations for the user by running a cryptographic protocol between them. Since this protocol needs to work with encrypted data, this system uses homomorphic cryptosystems, Paillier [Pai99] and DGK [DGK07, DGK09]. The PSP has the private keys for the Paillier and DGK cryptosystems. The system considers a set of $N$ users and $M$ items and each user is represented by a preference vector consisting of the user's ratings. They use cosine similarity measure to generate the similarity measure between two users. The generated recommendations are kept secret from the SP, the PSP, and all other users. They are only revealed to the user who asks for the recommendations along with the number of users $L$ whose ratings were considered for the output.

The system assumes a set of $R$ items out of $M$, which consists of the most rated items. For every user $i$, the user preference vector is split into two parts: $V_i^d = (v_{(i,0)}^d, \ldots, v_{(i,R-1)}^d)$ that consists of the densely rated items and $V_i^p = (v_{(i,0)}^p, \ldots, v_{(i,M-R-1)}^p)$ that consists of $M - R$ partly rated items. $V_i^d$ is used to compute the similarity measure whereas $V_i^p$ is used to get the recommendations. The protocol consists of two phases: (1) constructing the encrypted database and (2) generating the recommendations.

In the first phase, two inputs $V_i^d$ and $V_i^p$ are needed from each user. The similarity measure is then computed over the entire item set using the subset which consists of the $R$ densely rated items. The vector elements after computing the cosine similarity are scaled with a constant parameter $s$ and rounded to the nearest integer. The user vectors are now $k$-bit integer values: $\hat{V}_i^d = (\hat{v}_{(i,0)}^d, \ldots, \hat{v}_{(i,R-1)}^d)$. The similarity value computed over $R$ items is now: $sim(A,B) = \sum_{r=0}^{R-1} \hat{v}_{(A,r)}^d \cdot \hat{v}_{(B,r)}^d$, with each similarity scaled by a factor of $s^2$. Each user then encrypts her vectors $\hat{V}_i^d$ and $\hat{V}_i^p$ with PSP's Paillier public key to get $[V_i^d]$ and $[V_i^p]$. (In this solution, all encrypted values are represented within [ ].) These encrypted vectors are sent to the SP.

In the second phase, to provide recommendations to a specific user $A$, the PSP and the SP undertake a cryptographic protocol using just the encrypted vectors sent by users to the SP in the first phase. The SP obtains the encrypted products of the vector elements $\hat{v}_{(A,i)}^d$ and $\hat{v}_{(A,j)}^d$, and adds them in the encrypted domain. This step is performed $R$ times for a single similarity computation and the results are multiplied to obtain the encrypted sum. Doing this same operation for each user excluding $A$, the SP will have a vector of $N-1$ records for user $A$: $SIM_A = ([sim_{(A,0)}], \ldots, [sim_{(A,N-2)}])$. Next, the most similar users to $A$ are found by comparing each similarity value with a publicly known threshold, $\delta$. Using the comparison protocol mentioned by Cramer et al. [CDN01], the SP and the PSP compare each similarity value with $\delta$ and get the results as: $[\Gamma_A] = ([\gamma_{(A,0)}], [\gamma_{(A,1)}], \ldots, [\gamma_{(A,N-2)}])$, where $[\gamma_{(A,i)}]$ is the results of comparing $sim_{(A,i)}$ and the threshold $d$. Then the SP finds the number of users with a similarity value above the threshold and computes the encrypted sums of the ratings of these users for each item. This is performed by adding all the $[\gamma_{(A,i)}]$'s and is denoted as $L$. Here, to find the encrypted sum of the ratings the SP first runs the secure multiplication protocol with the PSP to multiply the $[\gamma_{(A,i)}]$ and the partly rated elements of user $i$ as follows:

$$\begin{aligned}
[UR_i] &= ([\gamma_{(A,i)}] \otimes [v^p_{(i,0)}], \ldots, [\gamma_{(A,i)}] \otimes [v^p_{(i,M-R-1)}]) \\
&= \left( \left[ \gamma_{(A,i)} \cdot v^p_{(i,0)} \right], \ldots, \left[ \gamma_{(A,i)} \cdot v^p_{(i,M-R-1)} \right] \right) \\
&= [UR_{(i,0)}], \ldots, [UR_{(i,M-R-1)}]
\end{aligned} \tag{2.8}$$

where, $[UR_i] = \left[ V^p_{(i,j)} \right]$ if $\gamma_{(A,i)}$ is 1, and $[UR_i] = ([0], \cdots, [0])$ if $\gamma_{(A,i)}$ is 0.

Then to get the sum of the ratings of the $L$ most similar users, the SP multiplies all of the encrypted $UR_{(i,j)}$ to obtain $UR_{sum}$. The SP sends $UR_{sum}$ and $L$ to the user $A$. The user $A$ needs the decryptions of $UR_{sum}$ and $L$. This can be done by applying a mask to these values and sending them to the PSP via the SP for decryptions. This ensures that user $A$ has the decrypted values and can obtain $UR_{(i,j)}$ and $L$ for all $j$. The final step involves dividing each $UR_{(i,j)}$ by $L$.

By using secure multiplication and decryption protocol on encrypted data, Erkin et al. reduced the computation overhead significantly as compared to using homomorphic computations. They also used data packing before encrypting the input values. They evaluated their system by running tests on a dataset with 10000 users and 1000 items, in terms of bandwidth and runtime. They do not have a solution for a dynamic protocol yet, as this system works with a static dataset of users and items. If new values need to the added to the datasets or current values need to be updated, the SP and PSP need to run the protocol from the beginning. Additionally, the number of users whose similarity measure exceeds $\delta$ is available to the user A. This system does not provide a solution for this information leak. They also mention that their design still has scalability challenges. The recommendation generation step accounts for 76% of the entire computational effort.

Nikolaenko et al. [NIW+13] proposed a design for a privacy preserving recommender system that learns item profiles from user ratings without accessing the ratings provided by the users or even learning which items they have rated. Their system uses a 'crypto-service provider' (CSP), a trusted third party, to implement the privacy-preserving computations. It also includes a recommender system RecSys, that uses a matrix factorization protocol to generate item profiles, which are then used to provide recommendations to the user. The CSP prepares a garbled circuit that is used to run the matrix factorization and provides it to the RecSys. The garbled inputs for this circuit are obtained from the users and oblivious transfer is used to ensure that neither the RecSys nor the CSP can learn this input.

As garbled circuits can only be used once, this approach uses a partially homomorphic encryption scheme with the garbled circuits to overcome this shortfall as described below.

Each user $i$ encrypts her inputs $(j, r_{ij})$ under the CSP's public key $\mathsf{pk}_{csp}$. For each item $j$ she rates, she submits a pair $(i, \mathsf{Enc}_{\mathsf{pk}_{csp}}(j, r_{ij}))$ to the RecSys, with a total of $M$ ratings. The algorithm used by $\mathsf{Enc}$ is partially homomorphic and hence after receiving $M$ ratings from the user, the RecSys can add randomly generated values $\mu$ to obscure them before sending these values to the CSP: $\mathsf{Enc}'_{\mathsf{pk}_{csp}} = \mathsf{Enc}_{\mathsf{pk}_{csp}} \oplus \mu$. The RecSys also sends the specifications to build a garbled circuit which includes the total number of user and item profiles, total number of ratings, total number of users and items, and the bits that represent the integer and fraction of a real number in the garbled circuits. On receiving these values from the RecSys, the CSP decrypts these values to get the masked values: $(i, (j, r_{ij}) \oplus \mu)$. The CSP then uses the matrix factorization circuit provided by the RecSys as a blueprint and builds a garbled circuit that takes as input the garbled values corresponding to the masks. Inside the circuit, by removing the masks $\mu$, the corresponding tuple $(i, j, r_{ij})$ is recovered. Next matrix factorization is performed to generate item profiles $V$. The CSP then makes the garbled circuits available to the RecSys. By using oblivious transfer, the RecSys is able to receive the garbled values of the masks from CSP. Once the RecSys evaluates the circuits, it has the ungarbled item profiles $V$. These item profiles can be used to generate private recommendations. The RecSys sends the item profiles to a user who can solve an optimization problem to recover her own profile. Once the user has her own profile, she can use it with the received item profiles to get local recommendations of unrated items.

This design was tested on the MovieLens ML-100K dataset. They compared the relative error of their approach against that of a system that operates with floating point representation. The metric used to capture the execution time included time to garble and evaluate the circuit. They excluded the encryption and decryption times performed by the users and the CSP, since they were short in comparison to the circuit processing time. They also calculated the number of bytes that are transmitted between the CSP and RecSys to capture the size of the circuit. They observe that one iteration of gradient descent with parameters set to achieve error of $10^{-4}$ took 2.9 hours.

This solution assumes that the RecSys and the CSP do not collude. They also assume that both the RecSys and the CSP work under the honest but curious threat model. Extensions to this design include a case considering a malicious RecSys, but not a malicious CSP.

The last solution we discuss is presented by Vadapalli et al. [VBH21] where they present a digital content delivery system called PIRSONA. The authors provide users with personalized content recommendations based on their consumption patterns, using a highly efficient and practical solution that also keeps those user consumption patterns private.

The architecture of PIRSONA includes several content distributors where at least four content distributor servers each have identical copies of records. Users fetch records of interest from the servers by using an efficient PIR protocol. Along with fetching results for the the users, the servers also store per-user, secret-shared consumption histories. These histories are extracted directly from the users' incoming PIR queries. Here, to maintain the privacy of the users' consumption patterns, the PIR protocol used is the Hafiz-Henry 1-private PIR protocol [HH19]. The servers transform these consumption histories into collaborative filtering item and user profiles using a 4PC variation of Boolean matrix factorization. These newly generated user and item profiles continue to remain secret-shared and these profiles are used to generate oblivious, yet personalized recommendations for the users.

This solution considers a database $D$ modelled by a $r \cdot s$ matrix over a binary field $GF(2^w)$. Each of the $r$ rows has a distinct record $D$ such as a video, an e-book, or an app, composed of $s$ $w$-bit words, for some chosen bit length $w$. A copy of record $D$ resides at each of the $s+1$ pairwise non-colluding database servers denoted by $P_0, \cdots, P_s$.

To fetch a record $\vec{D}_j$ from $D$, the user constructs a query template by sampling a length-$r$ vector $\vec{q} \in_R [0 \cdots s]^r$ uniformly at random such that $\vec{q}[j] = 0$, where $\vec{q}[j] \in [0 \cdots s]$ represents the $j^{th}$ component of vector $\vec{q}$. Then the user selects a permutation $\sigma : [0 \cdots s] \to [0 \cdots s]$ uniformly at random. Then for each $k \in [0 \cdots s]$, the user computes and sends the vector $query_k := \vec{q} + \sigma(k) \cdot \vec{e}_j$ to each $P_k$. Here $\vec{e}_j := < 0\, 0 \cdots 1 \cdots 0 > \in \mathbb{Z}^r$ denotes the $j^{th}$ length-$r$ standard basis vector. When $P_k$ receives $query_k$ from the user, it computes and responds with $response_k := \oplus_{i=1}^{r} \vec{D}_i(query_k[i])$, a scalar in $GF(2^w)$. Here, $query_k[i]$ refers to the $i^{th}$ component of $query_k$ and $\vec{D}_i(x) := \vec{D}_i[x]$, which is the $x^{th}$ word of $\vec{D}_i$. The user then computes $\sigma(k)^{th}$ word of $\vec{D}_j$ from the pair $(response_k, response_0)$ using $\vec{D}_j[\sigma(k)] = response_k \oplus response_0$.

Next, the authors illustrate how collaborative-filtering was performed on the PIR protocol. For this computation, a semi-honest 4-party computation is used. PIRSONA leverages a latent-factor collaborative filtering algorithm based on Boolean matrix factorization as described by Koren et al. [KBV09]. In this system there are $m$ distinct users fetching items from a database comprising $r$ distinct records and let $M \in \{0,1\}^{m \times r}$ denote the $(0,1)$-matrix having one row per user and one column per record and having $M_{ij} = 1$ if and only if the $i^{th}$ user has fetched the $j^{th}$ record. This recommender system aims to recommend $(i,j)$ pairs for which $M_{ij} = 0$. For this, the recommender system implements a user profile matrix $U := [\vec{u}_1; \cdots; \vec{u}_m] \in \mathbb{R}^{m \times d}$ and item profile matrix $V := [\vec{v}_1; \cdots; \vec{v}_r] \in \mathbb{R}^{r \times d}$ to associate a profile from $(\mathbb{R} \cap [0,1])^d$ to each of the $m$ users and to each of the $r$ records in

$M$. Here $d \in \mathbb{N}$ is the dimensionality parameter representing the number of latent features needed to capture user preferences. Then using the gradient descent method, PIRSONA performs several iterations of the following computations, where each iteration provides a progressively "better" approximation of $M$. After several iterations of gradient descent, the resulting $U$ and $V$ are used to generate recommendations for user $i$ by computing length-$r$ vector $\vec{R}_i := \vec{u}_i V^T - \vec{M}_i$, where $\vec{M}_i$ represents the $i^{th}$ row of $M$. Here the largest components of $\vec{R}_i$ are the top recommendations for user $i$.

This design was implemented in C++ using the open-source dpf++ library for (2, 2)-DPFs. PIRSONA was deployed on Amazon EC2 servers present in four geographically distant regions. They used the MovieLens ML-100K and ML-Latest datasets to evaluate the performance of their solution. They compared the MSE between user and item profiles produced by PIRSONA on the ML-100K dataset with those produced by a non-private collaborative filtering implementation that uses IEEE double-precision floating-point arithmetic. The difference was statistically insignificant and the MSE was minuscule with $p = 14$ or more bits, where $p$ denotes the fractional precision. This work also shows a proof-of-concept implementation of PIRSONA for generating recommendations for a large-scale streaming service. It discusses a hypothetical deployment modeled on a Netflix-like system and shows that PIRSONA can easily scale to such a system, serving thousands of users per second. They also estimated the cost of performing this deployment, producing an upper bound of about US\$ 0.022 per subscriber per month, where all subscribers are assumed to streams HD videos 24 hours a day, 7 days a week.

They ran three sets of experiments respectively measuring the impact on training time of varying the number of queries being trained on, the number of items $r$ in the database, and the dimension $d$ of the profiles. They also measured how many recommendations were generated while varying the number of items $r$. First, for $m = 943$ users, $r = 2^{12}$ items, profiles of dimension $d = 8$, and varying the number of queries ranges from $q = 2^{10}$ up to $2^{16}$, they had a linear regression that gave $R^2 = 0.9972$. Here, the training times scaled from about $2.3 \pm 0.1\,\mathrm{s}$ for $q = 2^{10}$ queries up to $21 \pm 3\,\mathrm{s}$ for $q = 2^{16}$ queries, an effective rate of about 3500 queries/s. Second, for $m = 943$ users, $q = 2^{14}$, profiles of dimension $d = 8$, and varying the number of items ranges from $r = 2^8$ to $2^{14}$, they had a linear regression that gave $R^2 = 0.9889$. The training time was $8 \pm 1\,\mathrm{s}$ for $r = 28$ items through $28 \pm 3\,\mathrm{s}$ for r = 214 items, giving approximately 1 s slowdown per 1000 items. And third, for $m = 943$ users, $q = 2^{16}$, $r = 2^{10}$, and profiles of dimensions ranging from $d = 4$ to 32, linear regression gives $R^2 = 0.9978$. The training time varied from about $20 \pm 3\,\mathrm{s}$ for d = 8 up to $41 \pm 3\,\mathrm{s}$ for $d = 32$, giving an approximate 0.9 s slowdown per profile component.

Another experiment compared the training and recommendation times for performing PIRSONA on the MovieLens ML-100K and ML-Latest datasets. Both datasets included

100000 queries and the number of users in ML-100K dataset is $m = 943$ and in ML-Latest is $m = 610$. In the ML-100K dataset, the the mean time for providing one recommendation to each user was $11 \pm 4\,\mathrm{s}$ for $d = 4$, and $13 \pm 3\,\mathrm{s}$ for $d = 8$. Whereas, for the ML-Latest dataset, the mean time for providing one recommendation to each user was $21 \pm 3\,\mathrm{s}$ for $d = 4$ and $22 \pm 4\,\mathrm{s}$ for $d = 8$.

In Table 2.1 we have summarized all the solutions that were just discussed and compared them. Specifically, we compared them with respect to five characteristics, mentioned below. We also provide a list of techniques mentioned in the table.

- **Independent of a trusted third party:**

| | |
|---|---|
| Independent: ● | if there is no dependency on a trusted third party |
| Partially dependent: ◐ | if the solution has two servers to provide recommendations and both servers are assumed to not collude with each other |
| Dependent: ○ | if there is a dependency on a trusted third party |

- **Low computational overhead:**

| | |
|---|---|
| Low: ● | if no computationally expensive techniques are used such as homomorphic encryption schemes or secure multiparty computations |
| High: ○ | otherwise |

- **No need for availability of other users:**

| | |
|---|---|
| Yes: ● | it is independent |
| No: ○ | if it needs interaction with other users to get output |

- **Works against a malicious adversary:**

| | |
|---|---|
| Yes ● | solution provides defense against a strong adversarial model |
| No ○ | otherwise |

- **Data protection:**

| | |
|---|---|
| Strong: ● | solution provides data integrity as well as confidentiality |
| Somewhat strong: ◐ | solution provides either data integrity or confidentiality, not both |
| Weak: ○ | solution does not provide data integrity or confidentiality |

- **Techniques**:
  1. MF: Matrix factorization
  2. VSS: Verified secret sharing
  3. PKE: Public key encryption
  4. PHE: Partially homomorphic encryption
  5. TTP: Trusted third party
  6. SMC: Secure multi-party computation

7. GC: Garbled circuits
8. PIR: Private information retrieval
9. OPRF: Oblivious pseudo-random functions

Table 2.1: Comparison of privacy-preserving recommender system solutions

| **SOLUTIONS** | **Independent of trusted third party** | **Low computational overhead** | **No need for availability of other users** | **Works against a malicious adversary** | **Data protection** | **TECHNIQUES USED** |
|---|---|---|---|---|---|---|
| Berkovsky et al. [BEKR07] | ● | ○ | ○ | ○ | ○ | Data obfuscation |
| Weinsberg et al. [WBIT12] | ○ | ● | ● | ○ | ○ | Data obfuscation, MF |
| Ying [Yin20] | ● | ● | ● | ○ | ◐ | MF, VSS |
| Artail and Farhat [AF14] | ● | ● | ○ | ● | ◐ | Data aggregation, PKE |
| $PPFR_h$ Samanthula et al. [SCJS15] | ● | ○ | ○ | ○ | ◐ | SMC, HE |
| $PPFR_{sp}$ Samanthula et al. [SCJS15] | ● | ● | ○ | ○ | ◐ | PKE, VSS |
| Badsha et al. [BYKB17] | ◐ | ○ | ● | ○ | ● | PHE, TTP |
| Erkin et al. [EVTL12] | ◐ | ○ | ● | ○ | ● | PHE, SMC |
| Nikolaenko et al. [NIW$^+$13] | ◐ | ○ | ● | ○ | ● | MF, PHE, GC |
| Vadapalli et al. [VBH21] | ● | ● | ● | ◐ | ● | PIR, 4PC Boolean MF |
| Ghost Recommendation Protocol | ● | ● | ● | ● | ● | VSS, OPRF, modified ElGamal encryption |

Reviewing all the techniques we discussed in Section 2.3 and the solutions we studied in Section 2.4, we can identify some of the key features that are considered while designing a privacy-preserving recommender algorithm.

User profiles and/or item profiles are modified by applying obfuscation techniques in solutions provided by Berkovsky et al. [BEKR07] and Weinsberg et al. [WBIT12]. These solutions are useful in suppressing the correlation between the user's private information and their preferences, while providing high accuracy recommendations. However, without the strong security guarantees provided by cryptographic techniques, the users are still vulnerable to privacy risks mentioned in Section 2.2. We will be considering modifying the user's input along with other primitives such as hashing in order to improve this approach.

Artail and Farhat [AF14] provides a solution using data aggregation by including other users in the request generation process. This solution leverages user-user interaction to reduce the risk of exposure of user data by interacting with the main recommender server. It uses the aggregate of multiple user requests to generate recommendations, while keeping the user identities private. While such techniques were successful in maintaining the accuracy of output recommendations, the requirement for multiple users to participate every time a request is made is a major drawback. This is an issue with solutions such as Samanthula et al. [SCJS15] as it too relies on its neighbors being available to get accurate recommendations. However, we aim to develop a solution that will perform even if other users are unavailable.

Multiple solutions we looked at use trusted third parties such as Badsha et al. [BYKB17], Nikolaenko et al. [NIW$^+$13], and Erkin et al. [EVTL12]. This is done to limit direct interaction between the user and the recommender system. Each of these solutions has split the functionality of the recommendation system to two servers; recommendation server and decryption server in Badsha et al., RecSys and CSP in Nikolaenko et al., and SP and PSP in Erkin et al. Each server performs a part of the computation and the solution then assumes that these separate entities are semi-honest and do not collude with any other participant in the system. This assumption is hard to deliver in the real world setting. Taking this into account, we will be designing our solution to provide recommendations even if a threshold number of servers in our solution operate in an adversarial manner.

Another characteristic we have evaluated for each solution is whether their implementation is efficient and has low computational overhead. Discussing the different techniques used for enhancing user privacy, we noticed that solutions that do not use cryptographic protocols tend to be faster. This is an obvious conclusion that has led to every new solution to consider the privacy-efficiency trade-off. Most cryptographic protocols require higher computational costs. Even within this category of solutions, the efficiency of each solution varies depending on what protocols were exactly used. Using encryption schemes and secret sharing have comparably lesser overhead to using homomorphic encryption schemes.

The solutions we have mentioned each use a combination of multiple techniques. The techniques used in each solution defines the type of data protection it can provide. This is evident from comparison shown in Table 2.1 where solutions using secure multi-party computation, homomorphic encryption schemes, and private information retrieval are able to offer stronger privacy and security guarantees.

Studying these solutions and how they all compare against the five characteristics mentioned in Table 2.1, has helped us in understanding what techniques we would like to use for developing our own solution for a privacy-preserving recommender system.

In Chapter 3, we will discuss our design for providing users with recommendations while maintaining their privacy, in an efficient way and without relying on any external trusted service.

# Chapter 3

# Methodology

In the previous chapter, we discussed some solutions for privacy-preserving recommendation systems. In this chapter we will discuss a new scheme that aims to perform without relying on a trusted third party, is available to users at all times, resists collusion by malicious entities up to a threshold, and is efficient.

## 3.1 Ghost Recommendation Protocol

We have designed a protocol for providing recommendations to users based on their preferences. The goal of this design is to perform this computation on the preferences without revealing them to any other participant of the system. This can be achieved by encrypting these values. Recommendation systems can use homomorphic computations on encrypted data to generate recommendations. However, as we have seen in the previous chapter, working with homomorphic operations leads to high computational overhead. Our protocol uses alternate ways of processing encrypted data, without introducing significant amounts of computational and communication costs. Our protocol allows only the user herself to read the recommendations in plaintext form. Even if this information is acquired by a server or other malicious entity, they will not be able to unmask it to reveal the true value of the recommendations, prompting us to title them **"Ghost Recommendations"**.

The participants of the Ghost Recommendation Protocol are the users who provide input preferences, a set of distributed servers, and a public bulletin-board server (PBS). These users can all run this protocol independently.

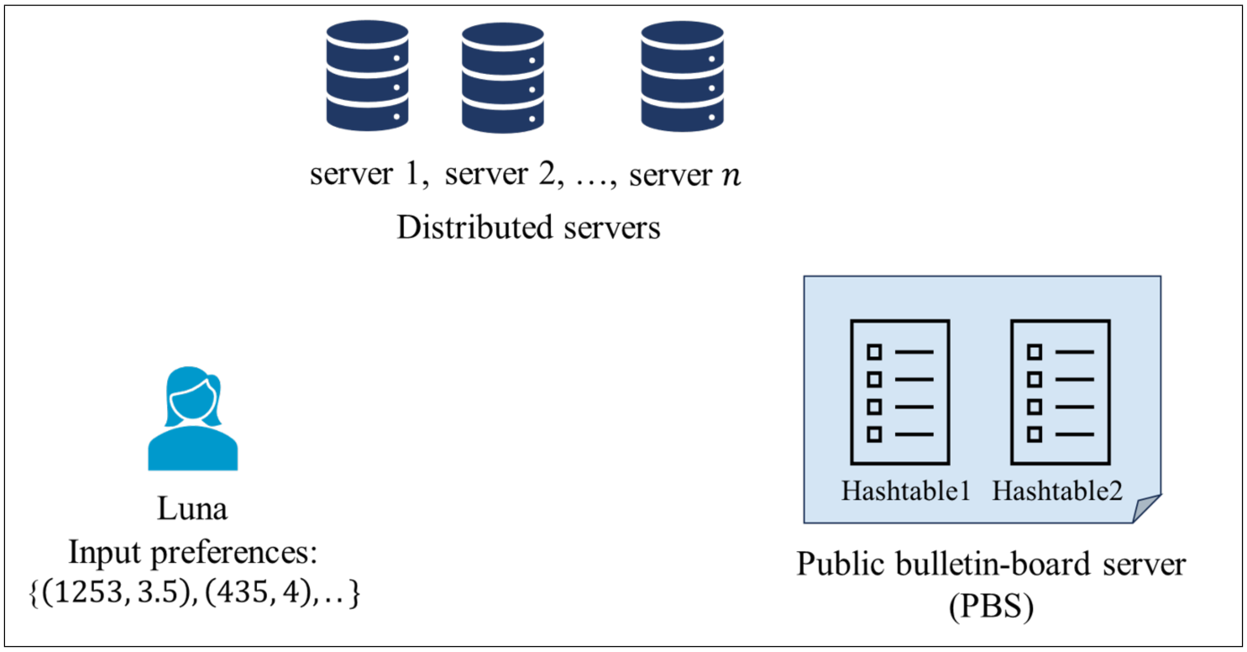

Figure 3.1: Participating entities of Ghost Recommendation Protocol

We demonstrate our design using a movie recommendation system. In our system the user's preferences are each represented as a tuple: (movie ID, movie rating). Figure 3.1 depicts the participants of our system. Consider a user Luna, who has certain input preferences and she would like to get recommendations from the Ghost Recommendation Protocol without risking the confidentiality or integrity of her preferences. In the rest of the chapter, we will demonstrate how Luna interacts with our system to get recommendations based on her preferences. The protocol we have implemented requires the use of a cryptographic group. For efficiency reasons, we select the Curve25519 group introduced by Bernstein [Ber06].

## 3.2 Notation

We now list the nomenclature we have used throughout our protocol.

Table 3.1: Notation used in Ghost Recommendation Protocol

| **Notation** | **Definition** |
|---|---|
| $m$ | Total number of preferences uploaded by Luna where $j = 1, 2, \ldots, m$ |
| $n$ | Total number of distributed servers where $i = 1, 2, \ldots, n$ |
| $k$ | Private OPRF key which is distributed among the servers such that each server has a $k_i$ share |
| $K$ | Public key corresponding to the private OPRF key $k$, which is distributed among the servers such that each server has $K_i$ share |
| $thr$ | Similarity parameter enforced for the entire system, which is a percentage value |
| $thr_{count}$ | Calculated as: $m \times thr \div 100$ |
| $t$ | Server threshold value set for secret sharing between distributed servers, available to everyone |
| $W$ | Public key for Double ElGamal encryption which is distributed among the servers such that each server has $W_i$ share |
| $w$ | Private key for Double ElGamal decryption which is distributed among the servers such that each server has $w_i$ share |
| $B$ | Generator for Curve25519, available to everyone |
| $d_j = (ID_j, r_j)$ | User preference for a single movie where $ID$ represents the movie ID and $r$ is the rating given to that movie |
| $F(d_j)$ | Output of SHA3 hash function applied on $d_j$ such that it is a point on Curve25519 |
| $b$ | OPRF blinding factor |
| $H()$ | SHA3 hash function used in NIZK proofs |
| $T(d_j)$ | Interpolated OPRF output |
| $G(d_j)$ | Output of SHA3 hash function applied on $T(d_j)$ |
| $A_j$ | Double ElGamal encryption of $d_j$ |
| $Q_{master}$ | 32-byte value selected by Luna to split into $m$ shares: $Q_1, \ldots, Q_m$ |
| $X_j$ | AES encryption output of $A_j$ under key $Q_{master}$ |
| $Y_j$ | AES encryption output of $Q_{master}$ under key $T(d_j)$ |

## 3.3 Threat model

Our goal is to provide the user Luna with relevant recommendations based on her input preferences while maintaining the confidentiality and integrity of the preferences as well as the recommendations. Our solution is designed to ensure that only Luna can access these preferences and recommendations in plaintext form.

As depicted in Figure 3.1, there are three entities participating in our recommendation protocol: the user Luna, a set of distributed servers, and the public bulletin-board server.

Our protocol is designed in a way that the distributed servers interact with just the user and not the PBS. The distributed servers are used to perform threshold encryption of the user input and threshold decryption of the recommendations. The private key used for this encryption is split using secret sharing and the distributed servers are each given a share of this key. A distributed key generation algorithm such as the one defined by Kate et al. [KHG12] can be used to generate and distribute this private key without relying on a trusted third party. We use an oblivious pseudorandom function protocol while performing the threshold encryption steps where the user masks her input with a blinding factor before interacting with the servers. The blinding factor ensures that the servers cannot retrieve the user's input, thus maintaining the confidentiality of the input. Using secret sharing provides a guarantee that at least a threshold number of servers need to be honest to get results. Thus we can be sure that as long as a threshold number of servers act honestly, the integrity of the computations is maintained.

Additionally, the input preferences are also encrypted using a secret value $Q_{master}$, known only to the user. This value is split into a number of shares, using secret sharing and a threshold value is set to indicate the minimum number of shares needed to recreate this secret.

Once the encrypted input is generated, it is sent to the PBS. The PBS stores the encrypted inputs of each user, creating a database of encrypted records. The PBS performs computations on this encrypted database to implement a similarity check against Luna's input. Then the PBS sends back the encrypted recommendations to the user which she decrypts using the threshold decryption scheme mentioned above. As these computations are performed on encrypted values, we can preserve the integrity of the user input as well as the generated recommendations by using the double ElGamal encryption scheme [FKMV12], which provides IND-CCA2 confidentiality and integrity while allowing for threshold decryption functionality.

The users already present in the PBS have each encrypted their records using a secret sharing scheme, with a pre-defined threshold value, before uploading their records to the

PBS. While performing a similarity check for Luna, the PBS finds other users having values in common with Luna's encrypted records. If the number of common values is greater than the threshold value set, we use Lagrange interpolation to recreate the secret and thus decrypt the other user's records. In this scenario, Luna has the ability to verify whether the response records interpolate and decrypt properly. With these computations, Luna will be able to identify and prove if the user, whose records she has received from the PBS, has misbehaved. This characteristic of our protocol provides an incentive for users to not misbehave while using the protocol.

Using the proof systems defined by Faust et al. [FKMV12], we can provide CCA2-secure encryption for our scheme. A non-interactive zero-knowledge (NIZK) proof is required to be attached to the input provided by the distributed servers during the OPRF computations. The user can verify this proof before proceeding with the rest of the protocol. Similarly, while performing the threshold encryption of the input preferences, we attach a proof to the input which can be verified during the decryption of the recommendations.

Luna has access to her own input preferences and the public values mentioned in Table 3.1. We use NIZK proofs while interacting with the distributed servers to ensure that the confidentiality and integrity of the preferences and recommendations is maintained. The last participant of our protocol, the PBS, is a bulletin board that is available to everyone. This is done so that PBS cannot violate the correctness of the user's data since any user can cross check if their data was entered incorrectly or not entered at all. The computations performed at the PBS are simply to optimize the search functionality needed for finding similar records. The computations done here are all performed on encrypted values, hence there is no possibility of leaking user information other than the user's ID, which is just a numeric value.

With Ghost Recommendation Protocol, we provide a system that can defend against up to a threshold number of malicious servers and also against covert users.

In the rest of the chapter we will be defining the NIZK proofs used as well as detail the encryption scheme we have implemented.

## 3.4 Preliminary definitions

We now provide definitions for of all the schemes we have used for implementing Ghost Recommendation Protocol.

### 3.4.1 Shamir's secret sharing protocol

Shamir [Sha79] developed a secret sharing (SSS) protocol to implement secure multi-party computation. In SSS the dealer divides a secret $S$ between $n$ participants such that each participant is given a share: $S_1, .., S_n$. The objective in doing so is to ensure:

1. **Recovery of secret is possible**: It must be possible to reconstruct the secret $S$ if we have a *threshold* number, $t$ or more participants combine their shares.

2. **Secrecy is maintained**: Combining $t-1$ or fewer shares should reveal no information about the original secret $S$.

Such a scheme is called a $(t, n)$ threshold scheme. They are essential in enabling robust key management where we can split the key among a group of mutually suspicious entities. For a finite field $\mathbb{F}_P$, where $P$ is a large prime, we can assume $S$ to be the secret we want to share, such that $1 \leq t \leq n$ and $S \leq P$. We can formalize this using a polynomial function for a $(t, n)$ scheme where the degree of the polynomial is $t$–1. We then need to build the polynomial by choosing random coefficients $a_1$, $a_2$, $\ldots$, $a_{t-1}$ and assigning the secret $S$ to $a_0$, giving us:

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{t\text{-}1}x^{t\text{-}1} \tag{3.1}$$

Once we have formed the polynomial, we can provide each participant with an index and the output of polynomial for the respective point. So for points $x = 1, \ldots, n$ we will have corresponding pairs $(x, f(x))$.

**Procedure 1** Secret Sharing

```
Input: $S, n, t$
Output: $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$
Dealer
1 :   Assign $S$ to $a_0$: $a_0 = S$
2 :   Select random values for $a_1, a_2, \ldots, a_{t-1}$
3 :   Create polynomial $f(x)$: $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$
4 :   Select points $x = 1, \ldots, n$ and generate $f(x)$ for each point
5 :   Send $(x, f(x))$ pairs back as shares of secret $S$
```

### 3.4.2 Lagrange interpolation

As seen in Section 3.4.1, implementing Shamir's Secret Sharing protocol requires the secret $S$ to be represented as a polynomial. Lagrange interpolation method is used to form a polynomial from a given set of data points. We use this interpolation method in our solution for reconstructing the secret $S$ needed in Shamir's Secret Sharing protocol, provided we have more than the threshold number of shares of the secret $S$.

In order to recover the secret $S = a_0$, we will need a subset of $t$ participants to *interpolate* these pairs. We can reconstruct the secret using Lagrange interpolation by calculating only the free coefficient, i.e. secret $S = f(0)$. We can recover the value of $S$ from any $t$ shares of $(x, f(x))$ by:

$$S = f(0) = \sum_{p=1}^{t} f(x_p) \prod_{q=1, q\neq p}^{t} \frac{x_q}{x_q - x_p} \tag{3.2}$$

**Procedure 2** Lagrange Interpolation

**Input:** $(x_1, f(x_1)), ..., (x_t, f(x_t))$

**Output:** $S = f(0)$

**Dealer**

```
1 :   Perform: $f(0) = \sum_{p=1}^{t} f(x_p) \prod_{q=1, q\neq p}^{t} \frac{x_q}{x_q - x_p}$
2 :   Send $f(0)$ back as reconstructed secret $S$
```

### 3.4.3 Oblivious pseudorandom functions

An oblivious pseudorandom function (OPRF) is a secure two-party protocol that is used in applications such as dynamic hashing as shown by Goldreich et al. [GGM84] and for the construction of deterministic memoryless authentication schemes as shown by Freedman et al. [FIPR05]. OPRFs compute the output of a pseudorandom function (PRF) $f_k(x)$ on some input $x$, where the PRF key $k$ and the input $x$ are each provided by a different participant. Using OPRF ensures that neither participant learns the other's contribution.

In our scheme, we use the OPRF protocol between a user and the distributed servers. The user provides the input, a string $d_j$ representing her preferences: $(ID_j, r_j)$, where $j = 1, \ldots, m$. This string is hashed using the SHA3 hash algorithm, $F()$ such that the hashed string is a point on the Curve25519 group. The user selects the OPRF blinding factor by randomly generating a scalar value $b \leftarrow\!\!\$\ Z_q^*$, where q is the order of the Curve25519 group. The hashed string is masked with the OPRF blinding factor $b$ by scalar multiplying the two. The user then sends this value to each of the $n$ servers.

Each server has its own share of the OPRF key $k_i$ where $i = 1, \ldots, n$. Every server then performs a scalar multiplication between its key share $k_i$ and the value sent by the user. Additionally, each server is required to provide a proof attached with its output that can be verified by the user. The definition for the proof and its verification is provided in Section 3.4.4. The servers then send back these values to the user.

Once the user has verified the proof provided by each server, she can remove the blinding factor applied to mask the original input by multiplying the inverse of the OPRF blinding factor $b$ to the values received from the servers to obtain $T_i(d_j)$. The user then performs an interpolation of the $T_i(d_j)$ values using the procedure mentioned in Section 3.4.3 to get the final desired output of the OPRF procedure, $T(d_j)$.

By concealing a part of the information needed from each participant to compute the final output, the OPRF ensures that the user cannot independently compute the final output of the PRF, and at the same time even if all the servers work together, they learn nothing about the user's input or final output.

**Procedure 3** OPRF

```
Input: User provides $d_j$, servers provide key share $k_i$
Output: $T(d_j)$
User                                                          Servers: $i, \ldots, n$
1: Hash $d_j$ to get: $F(d_j)$
2: Select blinding factor: $b \leftarrow\!\!\$\ Z_q^*$
3: Scalar multiply: $b \cdot F(d_j)$        $\xrightarrow{b \cdot F(d_j)}$        4: Receive $b \cdot F(d_j)$ from the user
                                                              5: Scalar multiply with OPRF key $k_i$
                                                              6: Attach proof for $k_i \cdot b \cdot F(d_j)$
                        $\xleftarrow[\text{proof}(k_i \cdot b \cdot F(d_j))]{k_i \cdot b \cdot F(d_j)}$        7: Send output back to user
8: Verify proof: $(k_i \cdot b \cdot F(d_j))$
9: Scalar multiply with $b^{-1}$ : $T_i(d_j) = b^{-1} \cdot\ k_i \cdot b \cdot F(d_j)$
10: $T(d_j) =$ Lagrange Interpolation$(T_1(d_j), \ldots, T_n(d_j))$
```

### 3.4.4 NIZK proof and verification

We use efficient and non-malleable non-interactive zero-knowledge proofs in our solution. These proofs are variants of a proof of equality of discrete logarithms. This proof is applied in three components of our protocol. The computations performed to build the proof and to perform its verification varies for each instance. We have defined each instance below.

1. Proof used in OPRF transaction:

   This proof is built by each distributed server while interacting with the user during the OPRF protocol described in Section 3.4.3. The servers need to apply their share of the key, $k_i$, to the input sent by the user. Then the servers build a proof using the values sent by the user, the $k_i$ share, a random value $c$, and $B$, the generator of Curve25519. With this proof we are showing $DL_B(K_i) = DL_{F(d_j)\cdot b}(F(d_j) \cdot b \cdot k_i)$. The steps in building the proof and verifying it are listed below.

   **Procedure 4** Building OPRF proof

   ```
   Input: $F(d_j) \cdot b, k_i$
   Output: $(F(d_j) \cdot b \cdot k_i), u, v$
   Servers: $i = 1, \ldots, n$
   1 :  Read input received from user: $F(d_j) \cdot b$
   2 :  Select a random value: $c \leftarrow\!\$\ Z_q^*$
   3 :  Hash together values: $u = H(B, (k_i \cdot B), (F(d_j) \cdot b), (F(d_j) \cdot b \cdot k_i), c \cdot B, c \cdot F(d_j) \cdot b)$
   4 :  Compute: $v = c - u \cdot k_i$
   5 :  Send to user: $(F(d_j) \cdot b \cdot k_i), u, v$
   ```

   The verification of this value is done in the next step when the user receives these values from the servers. She can verify the proof by performing the following steps.

**Procedure 5** Verification of OPRF proof

```
Input: $K_i, (F(d_j) \cdot b), (F(d_j) \cdot b \cdot k_i), u, v$
Output: $0|1$
User:
1 :   Read input received from server: $(F(d_j) \cdot b \cdot k_i), u, v$
2 :   Read each server's public key share $K_i$
3 :   Compute values: $T_0 = u \cdot K_i + v \cdot B$ and $T_1 = u \cdot (F(d_j) \cdot b \cdot k_i) + v \cdot (F(d_j) \cdot b)$
4 :   Hash together values: $u_{verify} = H(B, K_i, (k_i \cdot B), (F(d_j) \cdot b \cdot k_i), T_0, T_1)$
5 :   Check if: $u == u_{verify}$
6 :   Return: 1 if True, 0 if False
```

2. Proof used in Double ElGamal encryption:

   The user builds this proof while encrypting her input using a Double ElGamal encryption scheme. The user needs the public key $W$, her input $d_j$, and three random values: $r_0, r_1, e$. The output is a set of ciphertexts and the proof. We explain why we have used a Double ElGamal encryption scheme instead of an ordinary public key encryption (PKE) scheme in Section 3.5.2. With this proof we are showing $DL_B(W) = DL_{CT1-CT3}(CT2-CT4)$. We define the computations performed below.

**Procedure 6** Building Double ElGamal encryption proof

```
Input: Public key $W$, $d_j$
Output: Ciphertexts: $(CT1, CT2, CT3, CT4)$, NIZK proof: $u, v$
User:
1 :  Reads public values: public key $W$, generator of Curve25519 $B$
2 :  Convert input string to a point on Curve25519: $P = string\_to\_point(d_j)$
3 :  Select random values: $r_0, r_1, e \leftarrow\!\$\; Z_q^*$
4 :  Hash together values: $u = H(B, B(r_0 - r_1), W, W(r_0 - r_1), e \cdot B, e \cdot W)$
5 :  Perform the following computations:
         $CT1 = r_0 \cdot B$
         $CT2 = r_0 \cdot W + P$
         $CT3 = r_1 \cdot B$
         $CT4 = r_1 \cdot W + P$
6 :  Compute: $v = e - u(r_0 - r_1)$
7 :  User has ciphertext, proof for her input $d_j$ : $(CT1, CT2, CT3, CT4), u, v$
```

The verification of this proof is performed by each of the distributed servers using the following steps.

**Procedure 7** Verification of Double Elgamal encryption proof

```
Input: Cipher texts: $(CT1, CT2, CT3, CT4)$, NIZK proof: $u, v$
Output: $0|1$
Servers: $i = 1, \ldots, n$
1 :  Read input: $CT1, CT2, CT3, CT4, u, v$
2 :  Read public values: public key $W$, generator of Curve25519 $B$
3 :  Perform following computations:
         $T_0 = u \cdot (CT1 - CT3) + v \cdot B$
         $T_1 = u \cdot (CT2 - CT4) + v \cdot W$
4 :  Hash together values: $u_{verify} = H(B, (CT1 - CT3), W, (CT2 - CT4), T_0, T_1)$
5 :  Check if: $u == u_{verify}$
6 :  Return: 1 if True, 0 if False
```

3. Proof used in Double ElGamal decryption:

   This proof is built by each distributed server while performing the Double Elgamal decryption. Since the user needs the server's private key share $w_i$ to decrypt the ciphertext value, she sends this ciphertext to each of the distributed servers. Each of the servers uses its private key share $w_i$, a random value $s$, and the ciphertext $CT1$, received from the user, to build a proof. The servers then send this share of the decrypted value back to the user, who computes the final decrypted value after verifying the proof attached to validate each server's response. With this proof we are showing: $DL_B(W) = DL_{CT1}(CT1 \cdot w_i)$. The following steps are implemented for building and verifying this proof.

**Procedure 8** Building proof for Double ElGamal decryption

```
Input: $w_i$,$(CT1, CT2, CT3, CT4, u, v)$
Output: $(CT1 \cdot w_i), u, v$
Servers: $i = 1, \dots, n$
 1 :  Read input received from user: $CT1, CT2, CT3, CT4, u, v$
 2 :  Read private key share $w_i$, generator of Curve25519 $B$
 3 :  Select random values: $s \leftarrow\$ Z_q^*$
 4 :  Hash together values: $u = H(B, (w_i \cdot B), CT1, (w_i \cdot CT1), (s \cdot B), (s \cdot CT1))$
 5 :  Compute: $v = s - u \cdot w_i$
 6 :  Server has ciphertext, proof for key share $w_i$ : $(CT1 \cdot w_i), u, v$
```

Once this value is sent back to the user, the user performs a verification to check the response she received from each distributed server, using the following steps.

**Procedure 9** Verification of Double ElGamal decryption proof

```
Input:$(CT1 \cdot w_i), u, v, W_i$
Output: $0|1$
User:
 1 :  Read input read from server:$(CT1 \cdot w_i), u, v$
 2 :  Read public key share $W_i$, generator of Curve25519 $B$
 3 :  Performs following computations:
          $T_0 = u \cdot W_i + v \cdot B$
          $T_1 = u \cdot (CT1 \cdot w_i) + v \cdot CT1$
 4 :  Hash together values: $u_{verify} = H(B, W_i, CT1, (CT1 \cdot w_i), T_0, T_1)$
 5 :  Check if $u == u_{verify}$
 6 :  Return 1 if True, 0 if False
```

## 3.5 Construction of Ghost Recommendation Protocol

In this section we will describe the implementation of our protocol.

### 3.5.1 Initialization

Before we proceed with the protocol, our design requires a one-time initialization, where we set up the following values.

We split the private key $k$ and its corresponding public key $K$ into $n$ shares. $k_i$ and $K_i$ shares are used in the OPRF procedure.

Similarly, we split the private key $w$ and its corresponding public key $W$, each into $n$ shares. Here $W$ is a point on Curve25519 and $W = w \cdot B$ where $B$ is the generator for the Curve25519. $W$ and $W_i$ shares are used in the Double Elgamal encryption scheme.

As mentioned in Section 3.3, in practice, a distributed key generation (DKG) protocol would be used as a one-time setup to distribute $k_i, K_i, w_i, W_i$ shares to the correct parties.

Our protocol is divided into the following three phases:

1. **Secure request generation phase:** We use oblivious pseudorandom functions and Double ElGamal encryption on Luna's input preferences to transform them to secure requests in the first phase. These requests are then uploaded to the PBS.
2. **Similarity ranking phase:** The PBS stores Luna's encrypted requests as records in two hash tables. Based on the requests sent, the PBS performs a similarity ranking to generate recommendations for Luna.
3. **Response decryption phase:** In this phase, the Luna decrypts the recommendations sent by the PBS using Double ElGamal decryption to get the plaintext-form recommendations.

### 3.5.2 Secure request generation phase

Consider Luna, having $CID$ as her client ID, has rated a set of $e$ movies. Each movie has an identifier $ID$ and a rating value $r$, that ranges from 1 to 5, in increments of 0.5. Luna's input preferences would consist of a set of $(ID, r)$ pairs.

We begin with **pre-processing user input** in which we augment Luna's input ratings by adding and subtracting 0.5 from each of the $r$ values. This will ensure that an exact match of ratings of a given movie between Luna and some other user will result in three matches for that movie in their rating sets; ratings 0.5 apart will result in two matches; ratings 1.0 apart will result in one match. This helps in calculating the similarity between users. Performing this computation will provide matches that are close to Luna's input and not just exact matches, thus still useful as they will contribute to the similarity score. Luna now has three times the number of input preferences, $m = 3e$.

The next step is to perform the **OPRF** procedure on each of the $d_j = (ID_j, r_j)$ input pairs, where $j = 1, \ldots, m$. As shown in Section 3.4.3, in the OPRF procedure Luna hashes her input $d_j$ to a Curve25519 point to get $F(d_j)$ and then applies a random blinding factor $b$ to mask the hashed input string. She then sends this value, $b \cdot F(d_j)$ to each of the distributed servers so that they can apply their $k_i$ share to it. Here, we require the servers to attach a proof of correctness using the steps mentioned in Procedure 4 of Section 3.4.4. Each server sends $k_i \cdot b \cdot F(d_j)$ and the proof back to Luna. Luna uses the steps mentioned in Procedure 5 of Section 3.4.4 to verify the proof attached. If she is able to correctly verify the proof sent by the servers, she proceeds with the next steps of the OPRF procedure. She removes the blinding factor masking her initial input by multiplying each of the values received from the servers with an inverse of the blinding factor $b$ to get $T_i(d_j) = k_i \cdot F(d_j)$ values. Then she uses the Lagrange interpolation procedure shown in Section 3.4.2 to interpolate $T_i(d_j)$ values providing her with $T(d_j)$. She hashes $T(d_j)$ using SHA3 to get $G(d_j)$.

Next, Luna builds the **Double ElGamal encryption** proof. We use this encryption scheme instead of regular ElGamal protocol or a simple PKE to provide CCA2-security guarantees for our protocol while allowing for threshold decryption. Regular ElGamal encryption is not secure under a chosen ciphertext attack and it needs to be modified to achieve this security. As shown in Procedure 6, 7, 8, and 9 mentioned in Section 3.4.4, we use a modified Double ElGamal encryption scheme to maintain the integrity of the input preferences and the output recommendations. Luna uses this Double ElGamal encryption to encrypt her input $d_j$ using public key $W$ and the basepoint of Curve25519, $B$. The output of this proof gives Luna a set of encrypted values:

$$A_j = [CT1, CT2, CT3, CT4, u, v] \tag{3.3}$$

We essentially compute two ElGamal encryptions of the input $d_j$: $(CT1, CT2)$ and $(CT3, CT4)$ in this step. The values $u$ and $v$ are the zero-knowledge proof that show the plaintexts in the two ElGamal encryptions are the same.

After this Luna needs to select a random value $Q_{master}$ and split it into $m$ shares using the secret sharing procedure explained in Section 3.4.1 giving Luna $Q_1, \ldots, Q_m$ shares of $Q_{master}$. The threshold value used for this procedure is the predefined value, $thr_{count}$. For example, if $thr$ is set to 10, it means that at least 10% of Luna's preferences need to match with another user Ginny's preferences, for Ginny to be considered as a similar user. Setting a low $thr$ value ideally should give Luna a higher number of recommendations as a higher number of users would be considered similar to Luna. However, this would also mean that the threshold number of records needed to reconstruct Luna's secret, $Q_{master}$ will be lower, increasing the potential of breach in confidentiality of her input preferences. We call this value the 'similarity parameter' which brings on a trade-off between receiving more recommendations and risking the confidentiality of Luna's input. In Chapter 4, we see if modifying this value impacts the recommendations received by the user.

The next step in this phase includes performing two **AES encryption** computations. The first is to encrypt $A_j$ with encryption key $Q_{master}$ and the second is to encrypt $Q_j$ using encryption key $T(d_j)$, for all $j = 1, \ldots, m$. Luna now has $X_j, Y_j$:

$$X_j = AESenc_{Q_{master}}(A_j) \tag{3.4}$$

$$Y_j = AESenc_{T(d_j)}(Q_j) \tag{3.5}$$

for all inputs where $j = 1, \ldots, m$.

Finally Luna needs to upload a set of the following encrypted values as to the PBS:

$$CID, G(d_j), X_j, Y_j$$

where $CID$ is the client ID and $j = 1, \ldots, m$. The other generated values: $d_j, T(d_j), Q_j$ are kept private and are only available to the user, Luna.

These steps from the secure request generation phase are mentioned below in Algorithm 1.

**Algorithm 1** Secure request generation algorithm

```
1:  Luna enters her preferences $d_j = (ID_j, r_j)$
2:  Luna augments input preferences: $(ID_j, r_j + 0.5), (ID_j, r_j - 0.5)$    ▷ Pre-processing user input
3:  procedure OPRF($d_j, k_i$)
4:      Luna performs OPRF with each distributed server
5:      Each server provides proof along with $k_i \cdot b \cdot F(d_j)$ value
6:      Luna interpolates values to get $T(d_j)$
7:  Luna does $G(d_j) = SHA3(T(d_j))$
8:  procedure Double ElGamal Encryption($W, d_j$)
9:      Luna does two ElGamal encryptions: $(CT1, CT2)$ and $(CT3, CT3)$
10:     Luna attaches proof $u, v$
11: User has $A_j = (CT1, CT2, CT3, CT4, u, v)$
12: procedure Secret Sharing($Q_{master}, m, thr_{count}$)
13:     Luna splits a random value $Q_{master}$ into $m$ shares using security parameter $thr_{count}$
14: Luna computes AES encryptions: $X_j = AESenc_{Q_{master}}(A_j)$ and $Y_j = AESenc_{T(d_j)}(Q_j)$
15: Luna now has: $d_j, T(d_j), G(d_j), A_j, Q_j, X_j, Y_j$ for all inputs $j = 1, \ldots, m$
16: Luna sends to the PBS: $CID, G(d_j), X_j, Y_j$
17: Luna keeps in her private records: $d_j, T(d_j), Q_j$
```

### 3.5.3 Similarity ranking phase

This phase is executed entirely by the PBS. Here, we formalize the Ghost_Sim similarity measure discussed in Section 2.1.4 to find other users similar to Luna.

Once the PBS receives the set of encrypted values from Luna, they are added to two hashtables: $Hashtable1$ is indexed by the uploading user's $CID$ and $Hashtable2$ is indexed by the $G(d_j)$ value in each encrypted record. In our protocol, all the records having the same key are concatenated at a single index in the hash tables in order to access the records in $\mathcal{O}(1)$ time.

First the PBS performs the **hits tallying** procedure. Here, for every $G(d_j)$ uploaded by Luna, the PBS finds records in $Hashtable2$ using $G(d_j)$ as a key. It then reads the $CID$ of the records it retrieved, maintaining a counter for every $CID$ it comes across while performing this step. At the end of this procedure the PBS has a list of 'selected CIDs', and a *hits_count* representing the total number of times each $CID$ appeared in the retrieved records.

Next the PBS needs to perform a **threshold verification** on these selected CIDs. Consider that the similarity parameter $thr$ set is 10. The PBS checks if the $hits\_count$ for each selected CID is more than the $thr_{count}$ for that respective CID and only those selected CIDs are used for the next step.

For example, suppose Ginny and Ron each have a total of 60 records available in the PBS and they both have passed the threshold verification. Now if Ron's $hits\_count$ is 12 whereas Ginny's $hits\_count$ is 7, the PBS is more inclined to choose Ginny over Ron, since Ginny has more dissimilar records and could potentially provide more recommendations to Luna. So after verifying the threshold limit, the PBS selects the three CIDs with the **highest number of dissimilar matches**. These users are denoted as 'similar users'. The records uploaded by these users to the PBS will be sent back to Luna as recommendations. Let us consider for our example that Ginny, Neville, and Hermione are similar users to Luna.

The final step of this phase includes putting together the set of encrypted records to send back to Luna, as her encrypted recommendations. PBS creates two hash tables, $ResponseHT1$ and $ResponseHT2$. PBS populates these two hash tables with the records of Luna's similar users. We use $simCID$ to denote the CID of a similar user and $\ell$ represents the record for a single preference of the similar user.

The PBS finds records from $Hashtable2$ using Luna's $G(d_j)$ values as hash keys. Then it checks the $CID$ of the retrieved records and adds the entire record to $ResponseHT1$ if $CID = simCID$. For populating $ResponseHT2$, the PBS finds all the records in $Hashtable1$ using $simCID$ as the hash key and then adds each those records to $ResponseHT2$. This is done for each of the three similar users.

These two hash tables are sent back to Luna.

The entire procedure of the similarity ranking phase is detailed in Algorithm 2.

**Algorithm 2** Similarity Ranking algorithm

```
1: PBS receives $m$ records from Luna: $CID, G(d_j), X_j, Y_j$ where $j = 1, \ldots, m$
2: procedure POPULATE HASHTABLE($CID, G(d_j), X_j, Y_j$)
3:     Add Luna's ($CID, G(d_j), X_j, Y_j$) to $Hashtable1$ using $CID$ as the hash key
4:     Add Luna's ($CID, G(d_j), X_j, Y_j$) to $Hashtable2$ using $G(d_j)$ as the hash key
5: procedure HITS TALLYING
6:     Read $G(d_j)$ from records uploaded by Luna
7:     Find the records in $Hashtable2$ using $G(d_j)$ as hash key
8:     Keep counter for every $CID$ found in the retrieved records    ▷ These are 'selected
   CIDs' each having $hits\_count$
9: procedure THRESHOLD VERIFICATION
10:     Check if $hits\_count$ is more than $thr_{count}$ for each selected CID
11:     Sort selected users by descending number of dissimilar records available
12: Select the 3 $CID$s having the largest number of dissimilar records as similar users
13: procedure POPULATE $ResponseHT1$
14:     Find records in $Hashtable2$ using each $G(d_j)$ as hash key
15:     if $CID = simCID$ then                                  ▷ For each record retrieved
16:         Add ($simCID, G(d_\ell), X_\ell, Y_\ell$) to $ResponseHT1$
17: procedure POPULATE $ResponseHT2$
18:     Find records in $Hashtable1$ using each $simCID$ as hash key
19:     Add ($simCID, G(d_\ell), X_\ell, Y_\ell$) to $ResponseHT2$            ▷ For each record retrieved
20: $ResponseHT1$ and $ResponseHT2$ are sent back to the user.
```

### 3.5.4 Response decryption phase

Once Luna receives the two response hash tables, she performs AES decryption on all the $Y_\ell$ values from $ResponseHT1$. Since all the records in $ResponseHT1$ are those that have been selected using Luna's $G(d_j)$ values as hash keys, she can use her private set of corresponding $T(d_j)$ values to decrypt the $Y_\ell$ values from $ResponseHT1$, as seen in Equation 3.5. This will give Luna a set of $Q_\ell$ shares for each similar user selected by the PBS: Ginny, Neville, and Hermione in our example.

After getting these shares, Luna can obtain $Q_{master}$ corresponding to each of the three similar users using the **Lagrange interpolation** procedure mentioned in Section 3.4.2. We can be sure the threshold number of shares required for interpolation will be present as the PBS performed a threshold verification on the records in the similarity ranking phase.

Luna would now have: $Q_{master\_Ginny}, Q_{master\_Neville}$, and $Q_{master\_Hermione}$.

Using $Q_{master\_Ginny}, Q_{master\_Neville}$, and $Q_{master\_Hermione}$ values Luna can perform the second AES decryption on the $X_\ell$ values present in $ResponseHT2$. As seen in Equation 3.4 Luna would then have all the $A_\ell$ values corresponding to the three similar users. Each $A_\ell$ should contain a set of values: $CT1_\ell$, $CT2_\ell$, $CT3_\ell$, $CT4_\ell$, $u_\ell$, $v_\ell$.

Next, Luna needs to decrypt $CT1_\ell$ from each $A_\ell$. So she sends $A_\ell$ to each distributed server where they can apply their $w_i$ share to it. Once the servers receive this input, they first check the proof attached to it using the steps shown in Procedure 7 in Section 3.4.4. This proof was built by each similar user when they uploaded their preferences to the PBS. After the servers verify the proof, they apply their share of the **Double ElGamal decryption** key $w_i$ to $CT1_\ell$. Along with this the servers attach their own proof that the user can authenticate using the steps from Procedure 8 in Section 3.4.4. The server sends back $(CT1_\ell \cdot w_i), u_{\ell\_dec}, v_{\ell\_dec}$ to the user.

Luna can authenticate the proof attached in $(CT1_\ell \cdot w_i)$, $u_{\ell\_dec}$, $v_{\ell\_dec}$ by performing the steps shown in Procedure 9 in Section 3.4.4. After the verification of these proofs, Luna can use the $(CT1_\ell \cdot w_i)$ shares sent by the servers to perform **Lagrange interpolation** to get $(CT1_\ell \cdot w)$ for each record present in $ResponseHT2$.

For each $(CT1_\ell \cdot w)$ Luna has a corresponding $CT2_\ell$ in $ResponseHT2$. Using this, Luna can perform the computations mentioned in Procedure 6 in Section 3.4.4 to get the final recommendations, $P_\ell$ in plaintext form.

$P_\ell$ is a point on Curve25519 representing the string containing plaintext values: $(ID_\ell, r_\ell)$. Using a *point_to_string* function Luna can retrieve this tuple which represents a similar user's preference now provided to Luna as a recommendation. The output recommendations would be a list of movie $(ID, r)$ pairs, for example: $\{(3114, 4.5), (480, 3.5)..\}$. Thus, Luna can retrieve the recommendations from each of the similar users: Ginny, Neville, and Hermione. Algorithm 3 has the steps we have just discussed.

**Algorithm 3** Response Decryption algorithm

```
1: Read $ResponseHT1$ and $ResponseHT2$ received from PBS
2: Luna performs AES decryption: $Q_\ell = AESdec_{T(d_j)}(Y_\ell)$    ▷ Using $T(d_j)$ corresponding to $G(d_j)$
3: procedure LAGRANGE INTERPOLATION($Q_1, Q_2, \cdots, Q_{thr\_count}$)
4:     $Q_{master} = Lagrange\ Interpolation(Q_1, Q_2, \cdots, Q_{thr\_count})$    ▷ For all three similar users
5: Luna performs AES decryption: $A_\ell = AESdec_{Q_{master}}(X_\ell)$ ▷ For all three similar users
6: Luna has a set of values:
7: $A_\ell = CT1_\ell$, $CT2_\ell$, $CT3_\ell$, $CT4_\ell$, $u_\ell$, $v_\ell$
8: procedure DOUBLE ELGAMAL DECRYPTION($A_\ell, w_i$)
9:     Luna sends $A_\ell$ to each distributed server
10:     Each server performs verification of Double ElGamal encryption proof attached within $A_\ell$
11:     If the proof is verified, each server sends back $CT1_\ell \cdot w_i$
12:     Each server attaches a proof $u_{\ell\_dec}$, $v_{\ell\_dec}$ as well
13: Luna verifies the proof attached with each $CT1_\ell \cdot w_i$ received from the servers
14: procedure LAGRANGE INTERPOLATION($CT1_\ell \cdot w_i$ shares)
15:     $CT1_\ell \cdot w = Lagrange\ Interpolation(CT1_\ell \cdot w_i$ shares)
16: Luna can perform the following computation to get recommendations from similar users: $P_\ell = CT2_\ell - CT1_\ell \cdot w$
17: Luna gets the recommendation in plaintext format by: $d_\ell = point\_to\_string(P_\ell)$
18: Luna has recommendation: $(id_\ell, r_\ell) = d_\ell$
```

# Chapter 4

# Evaluation

In this chapter we discuss in detail the performance of our solution. First, in Section 4.1, we describe the system specifications and tools used for implementing our protocol. Then, in Section 4.2, we report the results of our experiments. In Section 4.3, we analyze the bandwidth consumption of our protocols. Finally, in Section 4.4, we compare our protocol with other contemporary solutions mentioned in Chapter 2.

## 4.1 Experiment specifications

The details of the different components used for implementing our scheme are mentioned below.

### 4.1.1 Hardware setup

For conducting our experiments, we needed a system configuration where we could host multiple participants; *i.e.*, the set of distributed servers, the PBS, and the user. We used two **tick** machines from the CrySP RIPPLE Facility [Uni13]. Each of these has 8 Intel Xeon E7-8870 CPUs with 10 cores each, 1 TB of RAM, and runs Ubuntu 14.04.

We simulated a network consisting of distributed servers and the PBS, where a user uploads her preferences and receives personalized recommendations. In order to recreate a client-server interaction that closely resembles the real-world environment, we ensured that the memory allocated to each participant was independent to it and not shared by any other participant. The results obtained in this setting, with respect to the bandwidth required and wall-clock runtime, should hence be comparable to real-world settings.

### 4.1.2 Data sets

We required a comprehensive data set of user preferences to examine the results of our protocol. We used the data sets available on the MovieLens [HK15] research platform. These include public data sets of multiple sizes that have been widely used in research studies, education, and the industry. For our protocol we have used the MovieLens 1M (ML-1m) data set which contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users and 3,952 movies. Each user in this data set has rated at least 20 movies. The anonymous ratings in the MovieLens 1M data set are in the format: (UserID, MovieID, Rating, Timestamp). The ratings provided by the users are on a 5-star scale. Table 4.1 presents sample entries from the ML-1m data set.

Table 4.1: Sample of ML-1m data set

| UserID | MovieID | Rating | Timestamp |
|---|---|---|---|
| 23 | 2117 | 2 | 978464541 |
| 23 | 2118 | 3 | 978466204 |
| 23 | 1243 | 3 | 978465537 |
| 23 | 2119 | 3 | 993707016 |
| 24 | 3354 | 1 | 978132906 |
| 24 | 2628 | 3 | 978135358 |
| 24 | 1259 | 4 | 978131980 |
| 24 | 3361 | 4 | 978132232 |
| 825 | 1225 | 4 | 975376263 |
| 825 | 1230 | 4 | 975376069 |
| 825 | 2971 | 3 | 975376317 |
| 825 | 1244 | 4 | 975376263 |
| 825 | 1247 | 5 | 975376317 |
| 826 | 3935 | 3 | 975373258 |
| 826 | 581 | 4 | 975373420 |
| 826 | 3944 | 2 | 975373226 |
| 826 | 3948 | 1 | 975373226 |
| 826 | 2064 | 4 | 975373457 |
| 826 | 756 | 3 | 975373636 |
| 826 | 3077 | 4 | 975373610 |

We created multiple data sets using ML-1m, where each data set represents a PBS with varying dimensions. The dimensions used for our data sets were:

1. Server threshold
2. Similarity parameter
3. PBS size

Once these dimensions were defined, we added UserIDs from ML-1m to an 'upload list' and ran Algorithm 1 for each user in the list. Tables 4.2, 4.3, and 4.4 show the specifications we used. Then we added 10 userIDs from ML-1m to a 'query list', such that the query list was distinct from the upload list used to create each data set. We uploaded the preferences of each user in the query list by running Algorithm 1. Once their preferences were uploaded, we ran Algorithms 2 and 3, 100 times for each userID. All the results we report in this chapter sample the mean running timings across these 100 trials, noted along with the standard deviation from the sample mean.

Table 4.2: Data sets for checking PBS size impact

| Data set | Server threshold | Similarity parameter | PBS size |
|---|---|---|---|
| PBS 1 | (2,5) | 25% | **296250** |
| PBS 2 | (2,5) | 25% | **551191** |
| PBS 3 | (2,5) | 25% | **1022072** |
| PBS 4 | (2,5) | 25% | **1264334** |
| PBS 5 | (2,5) | 25% | **1545020** |

Table 4.3: Data sets for checking similarity parameter impact

| Data Set | Server threshold | Similarity parameter | PBS size |
|---|---|---|---|
| SP 1 | (2,5) | **10%** | 1025396 |
| SP 2 | (2,5) | **15%** | 1004025 |
| SP 3 | (2,5) | **20%** | 1026667 |
| SP 4 | (2,5) | **25%** | 1141543 |
| SP 5 | (2,5) | **30%** | 1012678 |

## 4.2 Results

We developed the Ghost Recommendation Protocol, shown in Chapter 3, to provide a user with recommendations based on her preferences, while maintaining the integrity and

Table 4.4: Data sets for checking server threshold impact

| Data Set | Server threshold | Similarity parameter | PBS size |
|---|---|---|---|
| THR 1 | **(3,7)** | 20% | 1019475 |
| THR 2 | **(4,7)** | 20% | 1017742 |

confidentiality of her data. We now present the results of the experiments we conducted using the data sets mentioned above.

### 4.2.1 Regression analysis

In our first set of experiments, we generated a regression model for each, the total upload time and the total query time. For this experiment we have used the data from all the data sets mentioned above. We have used multiple linear regression on this data for creating our predictive models. This method of analysis lets us see the relationship between the various parameters and how they impact the upload time and query time. In this section, we present the results of these regression models.

#### Regression model for upload time

The first regression model is for the total upload time. The total upload time was calculated as total runtime for performing the steps in Algorithm 1. It includes the runtime for the pre-processing steps, the OPRF steps where the user interacts with the distributed servers, the Double ElGamal encryption steps, the secret sharing steps, the two AES encryption steps, and sending the encrypted user records to the PBS. Specifically, the explanatory variables we considered for this model were:

- $n$: total number of distributed servers
- $t$: threshold number for the distributed server
- $m$: total preferences uploaded
- $sp$: $m \times$ similarity parameter /100
- $PBS_{size}$: PBS size

The function for total upload time's regression model $lm()$ is as follows:

$$\text{lm}(\text{upload\_time} \sim n + t + t^2 + sp + sp^2 + m + PBS_{size}) \tag{4.1}$$

Table 4.5: Summary of regression analysis model for upload time

| **Variable** | **Coefficient** | **Standard Error** | **t-value** | **p-value** |
|---|---|---|---|---|
| $n$ | 0.604 | 0.005 | 100 | 2e-16 |
| $m$ | 0.0319 | 0.0002 | 200 | 2e-16 |
| $t$ | 0.030 | 0.006 | 5 | 0.0000006 |
| $sp^2$ | 0.00058 | 0.00004 | 10 | 2e-16 |
| $PBS_{size}$ | -0.000000006 | 0.000000004 | -2 | 0.1 |
| $sp$ | -0.022 | 0.002 | -10 | 2e-16 |
| Intercept | -2.56 | 0.04 | -70 | 2e-16 |
| $t^2$ | NA | NA | NA | NA |

The summary of this regression model is presented in Table 4.5. It presents the variables from largest coefficients to smallest since the coefficients tell us how strongly a variable is associated to the upload time. In this case, the total number of distributed servers participating provides the most impact on the upload time. These servers perform the OPRF operation with the user to generate the secure requests in Algorithm 1. From the summary we can deduce that for every one unit increase in $n$, the upload time increases by 0.604 seconds. As $m$ has a coefficient of 0.0319, we can say that even if a user uploads a large number of input ratings, the upload time increases by just 0.0319 seconds per unit increase in the number of ratings.

The t-value displays the test statistic which measures how many standard errors the coefficient is away from zero. Generally, any t-value greater than $+2$ or less than $-2$ is acceptable and the higher the t-value, the greater the confidence we have in the coefficient as a predictor. This allows us to consider $n$ and $m$ as the most significant variables for predicting the upload time, from among the variables we selected.

The p-value indicates whether the relationship between each of these variables and the upload time is statistically significant. A p-value of 0.05 or lower is generally considered statistically significant. This means that all the variables we considered except the $PBS_{size}$ have a significant impact on the upload time of Ghost recommendation protocol.

We note that the values for $t^2$ are not defined by our model. As compared to $sp$, $t$ has has a limited range of values ($t = 3$ and $t = 4$). For this reason, we have regression coefficients available for $sp^2$ but not for $t^2$.

Lastly, we show how well this model fits our data. The overall quality of the regression fit can be assessed using the three quantities, the residual standard error (RSE), the multiple R-squared value ($R^2$), and the F-statistic value. RSE represents the average variation of the observation points around the fitted regression line. This is the standard deviation of residual errors. Dividing the RSE by the average value of the outcome variable gives us the prediction error rate, which should be as small as possible. RSE in our upload time regression model is 0.1358, meaning that the observed upload time deviates from the true regression line by approximately 0.1358 seconds in average. In our data set, the mean value of upload time is 4.054 seconds, and so the percentage error is $0.1358/4.054 * 100 = 3.35\%$. $R^2$ measures the variation of a regression model and the adjusted $R^2$ value shows how much of the variation in the dependent variable can be explained by the variation in the independent variables. So an adjusted $R^2$ that is close to 1 indicates that a large proportion of the variability in the outcome has been explained by the regression model. In our upload time regression model, $R^2$ is 0.8692 and adjusted $R^2$ is 0.8691. The F-statistic provides a way for globally testing if any of the explanatory variables are related to the upload time. A large F-statistic will corresponds to a statistically significant p-value ($p < 0.05$). In our example, the F-statistic is $1.328e + 04$ producing a p-value of less than $2.2e - 16$, which is highly significant.

The regression model for the total upload time for Ghost recommendation protocol is shown in Figure 4.1.

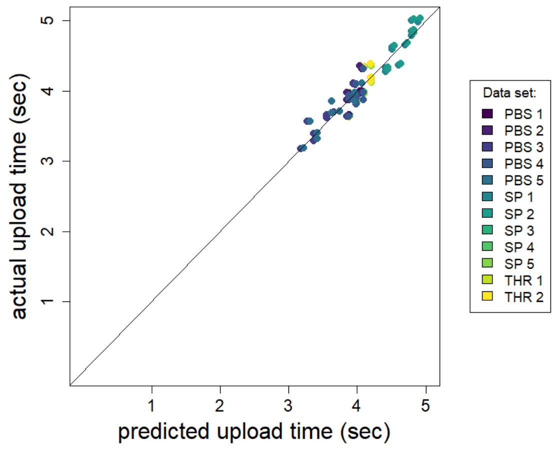

Figure 4.1: Upload time regression analysis

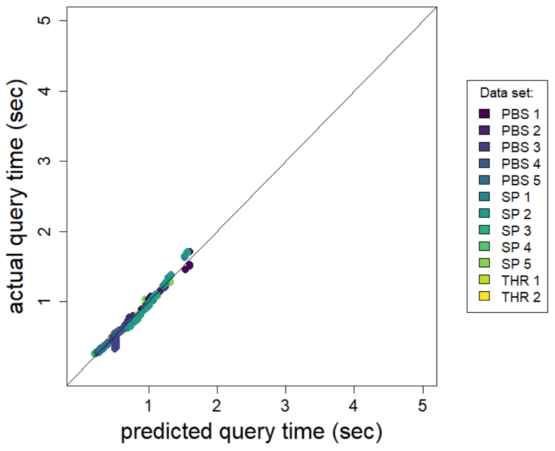

Figure 4.2: Query time regression analysis

### Regression model for query time

Now we present the regression model for the total query time. The total query time was calculated as total runtime for performing the steps included in Algorithms 2 and 3. It includes the runtime for the PBS to perform similarity ranking steps, for the PBS to send the data back to the user, and for the user to perform the response decryption steps along with the distributed servers. The explanatory variables we considered for building the regression model for total query time were:

- $n$: total number of distributed servers
- $t$: threshold number for the distributed server
- $m$: total preferences uploaded
- $sp$: $m \times$ similarity parameter /100
- $PBS_{size}$: PBS size
- $decA$: number of $A_\ell$ strings decrypted by the user
- $recs$: number of recommendations

The function for total query time's regression model $lm()$ is as follows:

$$\text{lm}(\text{query\_time} \sim n + t + t^2 + sp + sp^2 + m + PBS_{size} + decA + recs) \tag{4.2}$$

Table 4.6: Summary of regression analysis model for query time

| **Variable** | **Coefficient** | **Standard Error** | **t-value** | **p-value** |
|---|---|---|---|---|
| $n$ | 0.105 | 0.001 | 70 | 2e-16 |
| $t$ | 0.040 | 0.002 | 20 | 2e-16 |
| $sp$ | 0.0058 | 0.0006 | 10 | 2e-16 |
| $decA$ | 0.00409 | 0.00001 | 300 | 2e-16 |
| $m$ | 0.00092 | 0.00001 | 20 | 2e-16 |
| $PBS_{size}$ | 0.000000037 | 0.000000001 | 30 | 2e-16 |
| $sp^2$ | -0.00011 | 0.00001 | -9 | 2e-16 |
| $recs$ | -0.00145 | 0.00006 | -20 | 2e-16 |
| Intercept | -0.8 | 0.01 | -70 | 2e-16 |
| $t^2$ | NA | NA | NA | NA |

Table 4.6 summarizes the regression model for query time. The variable with the highest coefficient is once again $n$, the total number of distributed servers. The distributed servers are used in Algorithm 3 to perform the Double ElGamal decryption of $A_\ell$. Specifically, each distributed server performs the verification of NIZK proof for the Double ElGamal encryption attached within $A_\ell$. Then each server builds a proof for $CT1_\ell{\cdot}w_i$ before returning this value back to the user. These steps account for the most runtime within the total query time and as seen in the summary table, $n$ has the most impact on the total query time. The t-value for $decA$ is 300, meaning that this variable's coefficient is a good predictor for our model as well. We can see that all the variable in our regression model are statistically significant since they all have a p-value much less than 0.05. As with the upload time model, the coefficient for $t^2$ was not available in the query time model.

Now we look at the RSE, $R^2$, adjusted $R^2$, and F-statistic values for the query time regression model. The RSE provides us with the average variation of the observation points around the fitted regression line. RSE for this regression model is 0.03768. The mean query time for our entire data is 0.708 seconds. So the percentage error is $0.03768/0.708 \times 100 = 5.32\%$. $R^2$ value is 0.9876 and the adjusted $R^2$ value is 0.9876. The F-statistic for this regression model is $1.197e + 05$ with a p-value of less than $2.2e - 16$, which is highly significant. The regression model for total query time for Ghost recommendation protocol is shown in Figure 4.2.

### Validation of the regression models

We have shown the significance of the explanatory variable with the coefficients, the t-value, and the p-value. We have also analyzed the goodness of fit of the regression models by measuring the RSE, $R^2$, and F-statistic values. We now perform statistical analysis on an independent data set. Over-fitting a model occurs when the we get good accuracy for the training data set but poor results on new data sets. Such models cannot be of much use in the real world as it is not able to predict outcomes for new cases. By performing cross-validation, we can train our model with a subset of our data set and then evaluate using the complementary subset of the data set.

We have used a 10-fold cross validation for checking the effectiveness of our Ghost recommendation protocol for each, total upload time and total query time. After performing a 10-fold cross validation, the statistical metrics that we used for evaluating the accuracy of our regression models are the root mean squared error (RMSE), the mean absolute error (MAE), and the $R^2$ error. RMSE is the square root of the averaged squared difference between the actual value and the predicted value of the upload time and the query time.

It gives the average prediction error made by the model. MAE gives the absolute difference between the actual values and the values predicted by the model for the upload time and the query time. The values for the intercepts and the coefficients for the 10-fold cross validation is almost identical to the values presented above. We have presented the RMSE, MAE, and $R^2$ values in Table 4.7 for the total upload time and the total query time.

Table 4.7: Summary of 10-fold cross validation on upload time and query time

| **Target Variable** | **RMSE** | **MAE** | $R^2$ |
|---|---|---|---|
| total upload time | 0.1358 | 0.1115 | 0.869 |
| total query time | 0.0377 | 0.0269 | 0.9876 |

### 4.2.2 Scalability benchmarks

In this set of experiments, we selected three parameters to test their impact in our protocol. With the goal of isolating the effect of varying each of these parameters, we compared the total upload time and total query time, when all the other parameters were fixed. The data sets used for these experiments are shown in Tables 4.2, 4.3, and 4.4.

The first parameter we evaluated was the **PBS size**. We built five data sets (PBS 1–5) of varying PBS sizes, ranging from ≈300,000 records to ≈1,500,000 records as seen in Table 4.2. To analyze if the size of the PBS influences the output of our protocol, we kept the other parameters constant. The distributed servers had a $(2, 5)$ threshold set and the similarity parameter was set to 25%, for this experiment. Then we added 10 userIDs from ML-1m to a 'query list', such that the query list was distinct from the upload lists used to create the PBS data sets. We used the same query list for each PBS data set. We uploaded the preferences of each user in the query list by running Algorithm 1. Once their preferences were uploaded, we ran Algorithms 2 and 3, 100 times for each userID. Table 4.8 shows that for the ten users in the dataset, the mean rate at which the preferences are uploaded to the PBS is $\approx$ 28 preferences/second and this does not vary much, since it is independent of the PBS size. Whereas, for the ten users in the query list, the mean rate of receiving recommendations varies slightly from $\approx$ 39 recommendations/second for PBS having $\approx$ 300,000 records to $\approx$ 42 recommendations/second for PBS having $\approx$ 1,500,000 records. The number of recommendations provided to the users also increases as the PBS size increases. We note that these PBS data sets were not built cumulatively, and the benchmarks we have presented here depend on the records present in each PBS.

The second value we analyzed was the **similarity parameter** that was predefined while a user uploaded her preferences. We built five data sets (SP 1–5) where the distributed

Table 4.8: Scalability benchmarks for PBS size

| Data set | Preference upload rate (per sec) | Mean upload time (secs) | Recommendation rate (per sec) | Mean query time (secs) | Mean number of recommendations |
|---|---|---|---|---|---|
| PBS 1 | ≈28 | $4.1 \pm 0.2$ | ≈39 | $0.5 \pm 0.1$ | ≈21 |
| PBS 2 | ≈28 | $4.0 \pm 0.2$ | ≈44 | $0.7 \pm 0.2$ | ≈29 |
| PBS 3 | ≈28 | $4.0 \pm 0.2$ | ≈43 | $0.7 \pm 0.3$ | ≈28 |
| PBS 4 | ≈28 | $4.0 \pm 0.2$ | ≈43 | $0.6 \pm 0.2$ | ≈28 |
| PBS 5 | ≈28 | $4.0 \pm 0.2$ | ≈42 | $0.7 \pm 0.3$ | ≈28 |

servers had a $(2,5)$ threshold and each PBS size was set to ≈100,000 records. However when each data set was built, we changed the similarity parameter of the users in the upload list as well as the PBS from 10% to 30%, as seen in Table 4.3. Then as before, we selected a query list that was run on each SP data set 100 times. Table 4.9 shows how varying the similarity parameter impacted the mean upload time, mean query time, and the recommendation rate, for the ten users in the query list. We can see the recommendation rate decreases from $\approx 63$ recommendations/second to $\approx 40$ recommendations/second as the similarity parameter varies from 10% to 30%. The mean query time to receive these recommendations also reduces from $1.1 \pm 0.2$ seconds to $0.4 \pm 0.1$ seconds as the similarity parameter increases from 10% to 30%. This same pattern is observed for the number of recommendations received when the similarity parameter increases. This supports our hypothesis that if the threshold set for the similarity measure is kept low, it will provide the user with a higher number of recommendations. Adjusting the value for the similarity parameter is also key for getting faster recommendations.

Table 4.9: Scalability benchmarks for similarity parameter

| Data set | Preference upload rate (per sec) | Mean upload time (secs) | Recommendation rate (per sec) | Mean query time (secs) | Mean number of recommendations |
|---|---|---|---|---|---|
| SP 1 | ≈ 28 | $4.0 \pm 0.2$ | 63 | $1.1 \pm 0.2$ | ≈ 67 |
| SP 2 | ≈ 28 | $4.0 \pm 0.2$ | 51 | $0.7 \pm 0.3$ | ≈ 37 |
| SP 3 | ≈ 28 | $3.6 \pm 0.3$ | 51 | $0.6 \pm 0.2$ | ≈ 28 |
| SP 4 | ≈ 29 | $3.9 \pm 0.2$ | 38 | $0.5 \pm 0.1$ | ≈ 18 |
| SP 5 | ≈ 27 | $3.7 \pm 0.3$ | 40 | $0.4 \pm 0.1$ | ≈ 14 |

The last parameter we analyzed was the **server threshold**. This threshold value was set for implementing secret sharing among the distributed servers. As seen in Table 4.4, we created two data sets, THR 1 and THR 2, where the similarity parameter of the users in the upload list was kept constant but the server threshold was changed from $(3,7)$ to $(4,7)$. We generated a query list of 10 users which was tested against each THR data set 100 times. In Table 4.10, we can see that varying the server threshold does not significantly change the mean upload time as it changes from $4.6 \pm 0.3$ seconds to $4.7 \pm 0.3$ seconds as the server threshold increases from $(3, 7)$ to $(4, 7)$. The recommendation rate varies from $\approx 35$ recommendations/second to $\approx 33$ recommendations/second as the server threshold increases whereas the change in mean query time again remains insignificant as it goes from $1.1 \pm 0.3$ seconds to $1.1 \pm 0.4$ seconds.

Table 4.10: Scalability benchmarks for server threshold

| **Data set** | **Preference upload rate (per sec)** | **Mean upload time (secs)** | **Recommendation rate (per sec)** | **Mean query time (secs)** | **Mean number of recommendations** |
|---|---|---|---|---|---|
| THR 1 | $\approx 20$ | $4.6 \pm 0.3$ | 35 | $1.1 \pm 0.3$ | $\approx 37$ |
| THR 2 | $\approx 20$ | $4.7 \pm 0.3$ | 33 | $1.1 \pm 0.4$ | $\approx 37$ |

The mean upload time calculated for all the data sets together was $4.1 \pm 0.4$ seconds and the mean query time was $0.7 \pm 0.3$ seconds.

## 4.3 Bandwidth consumption

Having discussed the computation time in the previous section, in this section we look at the bandwidth consumption of Ghost recommendation protocol. The architecture of our protocol has multiple entities, specifically, the user, the distributed servers, and the PBS. To simulate the conditions of a real world application, we ensured that each entity was hosted on an independent core and thus not sharing the same memory or logical processor. The network speed in our simulated environment was 128 Gbps, but using the bandwidths we report in this section, one can compute the contribution of data transfer on upload and query time for any desired bandwidth.

In Algorithm 1, Ghost recommendation protocol performs secure request generation. The operations performed in this algorithm solely by the user are the pre-processing operations, the Lagrange interpolation, the Double ElGamal encryption, a secret sharing computation, and two AES encryption operations. The user only interacts with the $n$ distributed servers during the OPRF computation where for each of the $m$ preferences, the user sends 197 bytes to each of the $n$ servers. In their response, each server sends back 97 bytes to the user. At the end of this algorithm, the user uploads the secure requests to the PBS. Each of these is 357 bytes in size. Table 4.11 summarizes the bandwidth consumption for Algorithm 1.

The PBS performs all the operations in Algorithm 2 for computing the similarity ranking. At the end of this algorithm the PBS sends back two hashtables which have encrypted

Table 4.11: Bandwidth consumption for Algorithm 1 in bytes

| **Sender\Receiver** | **User** | **Distributed servers** | **PBS** |
|---|---|---|---|
| **User** | | $197 \cdot m \cdot n$ | $357 \cdot m$ |
| **Distributed servers** | $97 \cdot m \cdot n$ | | |
| **PBS** | | | |

records of similar users. The size of each record is 356 bytes in $ResponseHT1$ and 360 bytes in $ResponseHT2$. The number of records present in each hashtable would depend on the similar users found by the PBS. For the purpose of presenting the bandwidth data, we are considering $rec1$ and $rec2$ as the size of each hashtable. Table 4.12 shows the bandwidth consumption for Algorithm 2.

Table 4.12: Bandwidth consumption for Algorithm 2 in bytes

| **Sender\Receiver** | **User** | **Distributed servers** | **PBS** |
|---|---|---|---|
| **User** | | | |
| **Distributed servers** | | | |
| **PBS** | $356 \cdot rec1 + 360 \cdot rec2$ | | |

In Algorithm 3, the user performs the response decryption steps. Here, the user computes two AES decryptions, two Lagrange interpolations, and the Double ElGamal decryption with the help of the $n$ servers. During the Double ElGamal decryption, for every decryption of $A_\ell$, the user sends 197 bytes to each of the distributed servers. The response sent back by each distributed server consists of 102 bytes. The number of $A_\ell$ decryptions depends on the quality of the similar users selected by the PBS. For providing an estimate of the bandwidth consumption, we consider this number to be $DE\_dec$. Table 4.13 shows the bandwidth consumption for Algorithm 3.

Table 4.13: Bandwidth consumption for Algorithm 3 in bytes

| **Sender\Receiver** | **User** | **Distributed servers** | **PBS** |
|---|---|---|---|
| **User** | | $197 \cdot n \cdot DE\_dec$ | |
| **Distributed servers** | $102 \cdot n \cdot DE\_dec$ | | |
| **PBS** | | | |

Consider an example where a user, Severus, has 50 preferences and $n = 5$. Severus would need $(197 \times 50 \times 5) + (357 \times 50) = 67.1$ KB bandwidth to encrypt the preferences and upload them to the PBS. Considering $DE\_dec = 200$, the bandwidth required by Severus to perform the response decryption to get the recommendations would be $102 \times 5 \times 200 = 197$ KB.

## 4.4 Comparison with other solutions

In this section, we evaluate the Ghost recommendation protocol under the same categories we used to compare solutions in Table 2.1 mentioned in Chapter 2. To recollect, we compared privacy-preserving solutions offered by several recommender systems against five characteristics: dependency on a trusted third party, computational overhead incurred, dependency on the availability of other users in the system, ability to defend against a strong adversarial model, and the privacy of user's data.

As discussed in Chapter 3, the design of our system does not include any entity that acts as a trusted third party. The distributed servers behave according to a $(t, n)$ threshold encryption scheme, where we consider that up to $t-1$ participants can act in an untrustworthy manner. Additionally, the PBS is a public database of encrypted records. Any addition or modification to it can be easily recorded. So we can state that our system does not consider these participants to be a trusted third party, and consequently, our solution performs without relying on one.

The second metric we considered was computational overhead. Ghost recommendation protocol requires an average runtime of $4.1 \pm 0.4$ seconds to upload the preferences of a single user to the PBS and an average runtime of $0.7 \pm 0.3$ seconds to receive new recommendations for a single user. Our regression model shows that the upload runtime increases by 0.604 seconds for every unit increase in the number of distributed servers and by 0.0319 seconds for every unit increase in the number of preferences uploaded. The regression model for total query time allows us to summarize that a unit increase in the number of distributed servers, increases the total query time by at most 0.105 seconds, whereas a unit increase in the threshold number of servers increases the total query time by 0.040 seconds. This indicates that our solution is scalable and will perform well even as the number of servers or input preferences increases. As mentioned in Section 2.4, in the solution presented by Vadapalli et al. [VBH21], using the MovieLens-100K dataset, the mean time for providing one recommendation to each user was $11 \pm 4$ seconds for a profile with dimension $d = 4$, and $13 \pm 3$ seconds for $d = 8$. The mean time for providing one recommendation using the MovieLens-Latest dataset was $21 \pm 3$ seconds for $d = 4$ and $22 \pm 4$ seconds for $d = 8$. For Ghost recommendation protocol, using the MovieLens-1M dataset, the mean time for providing a single recommendation to a user is $0.245 \pm 0.007$ seconds.

The communication cost with respect to the bandwidth requirements is also fairly low as shown in the example above where a user requires 67.1 KB to upload 50 preferences and 197 KB to receive new recommendations. We can conclude that the runtimes and the bandwidth requirement of Ghost recommendation protocol are fairly low and stay comparatively low as the size of the system increases.

Now we measure our solution for the next characteristic considered, having a dependency on the availability other users. Our solution does not require other users to actively participate in the protocol to generate recommendations. This metric was included since solutions using garbled circuits and $k$-anonymity clusters rely heavily on the availability of other users, so as to remove the risk of exposing user data to the recommender servers.

The fourth characteristic we considered while comparing various privacy-preserving solutions was its ability to perform against a malicious adversary. As seen in Chapter 2, most of the solutions consider a semi-honest model where the participants in the system do not deviate from the defined protocol. However, as shown in Section 3.3, Ghost recommendation protocol is designed to defend against up to a threshold number of malicious servers. Additionally, it also offers covert defense for the users, deterring them from acting in a adversarial manner.

The final characteristic considered for comparison is level of data protection provided by a solution. Using Ghost recommendation protocol, we are able to maintain the confidentiality and integrity of the user's input preferences as well as the output recommendations. We consider providing both of these security guarantees as a strong level of data privacy for the user.

So we conclude that we have designed and implemented a recommender system that maintains the confidentiality and integrity of a user's input as well as the recommendations she receives, that can operate in an adversarial model with a $(t, n)$ threshold, and performs efficiently, without using a trusted third party or relying on the availability of other users.

# Chapter 5

# Conclusion

With this work we have presented Ghost recommendation protocol, a recommender scheme that maintains the confidentiality and integrity of users' preferences as well as the generated recommendations. It ensures that only the user themselves can access the input preferences and the received recommendations in plaintext form. It also guarantees that as long as a threshold number of servers act honestly, the encrypted requests and recommendations cannot be modified or decrypted. The computations needed for conducting similarity measure are performed on the encrypted records as well. The protocol design additionally ensures that all computations are performed efficiently, without requiring significant runtime or bandwidth. We have tested our protocol using the MovieLens 1 million dataset and have successfully demonstrated our goals.

In future studies, certain parts of our design can be improved upon. Instead of the similarity measure currently implemented, a more sophisticated technique such as matrix factorization can be used. Using matrix factorization can help in capturing the latent factors of the users and the items, which can in turn help in refining the quality of the recommendations generated. Additionally, the underlying framework of our design can be drawn and used in other applications where it is crucial to maintain the privacy of users' information, such as secure targeted advertisements, healthcare, defense, and banking services. Services in these fields require working with intellectual property, or processing of sensitive customer data, or even performing privacy-preserving data analytics. These services can be handled by modifying the Ghost recommendation protocol as it allows performing computations on encrypted data.

In this work, we have designed, implemented, and tested a new protocol to provide users with recommendations while maintaining the integrity and confidentiality of their data. Our protocol does not use a trusted third party, is efficient, and is secure against up to a threshold number of malicious servers. We have also reviewed several other solutions for privacy-preserving recommender systems, highlighted the limitations in some of the studies, and the current approaches being used. As services providing recommendations are widespread and extremely useful, we hope this work can be used to guide the future trends so as to advance the quality of solutions being used in privacy-preserving recommender systems.

# References

[AF14] Hassan Artail and Raja Farhat. A privacy-preserving framework for managing mobile ad requests and billing information. *IEEE Transactions on Mobile Computing*, 14(8):1560–1572, 2014.

[Agg06] Charu C Aggarwal. On randomization, public information and the curse of dimensionality. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 136–145. IEEE, 2006.

[AK06] Naveen Farag Awad and Mayuram S Krishnan. The personalization privacy paradox: an empirical evaluation of information transparency and the willingness to be profiled online for personalization. *MIS quarterly*, pages 13–28, 2006.

[BEKR07] Shlomo Berkovsky, Yaniv Eytani, Tsvi Kuflik, and Francesco Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 9–16. ACM, 2007.

[Ber06] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of cryptography conference*, pages 325–341. Springer, 2005.

[BYKB17] Shahriar Badsha, Xun Yi, Ibrahim Khalil, and Elisa Bertino. Privacy preserving user-based recommender system. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1074–1083. IEEE, 2017.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. Multiparty computation from threshold homomorphic encryption. In *International conference on the theory and applications of cryptographic techniques*, pages 280–300. Springer, 2001.

[CGH18] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The guardian*, 17(1):22, 2018.

[CKN$^+$11] Joseph A Calandrino, Ann Kilzer, Arvind Narayanan, Edward W Felten, and Vitaly Shmatikov. "You might also like:" privacy risks of collaborative filtering. In *2011 IEEE Symposium on Security and Privacy*, pages 231–246. IEEE, 2011.

[DEC$^+$16] Yashar Deldjoo, Mehdi Elahi, Paolo Cremonesi, Franca Garzotto, Pietro Piazzolla, and Massimo Quadrana. Content-based video recommendation system based on stylistic visual features. *Journal on Data Semantics*, 5(2):99–113, 2016.

[DGK07] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian conference on information security and privacy*, pages 416–430. Springer, 2007.

[DGK09] Ivan Damgård, Martin Geisler, and Mikkel Kroigard. A correction to 'efficient and secure comparison for on-line auctions'. *International Journal of Applied Cryptography*, 1(4):323–324, 2009.

[DK11] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. *Recommender systems handbook*, pages 107–144, 2011.

[EVTL12] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Reginald L Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE transactions on information forensics and security*, 7(3):1053–1066, 2012.

[FIPR05] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference*, pages 303–324. Springer, 2005.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *International Conference on Cryptology in India*, pages 60–79. Springer, 2012.

[FKV$^+$15] Arik Friedman, Bart P Knijnenburg, Kris Vanhecke, Luc Martens, and Shlomo Berkovsky. Privacy aspects of recommender systems. In *Recommender Systems Handbook*, pages 649–688. Springer, 2015.

[GCF11] Saikat Guha, Bin Cheng, and Paul Francis. Privad: Practical privacy in online advertising. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 276–288. Springer, 1984.

[GSFS18] G. Geetha, M. Safa, C. Fancy, and D. Saranya. A hybrid approach using collaborative filtering and content based filtering for recommender system. *Journal of Physics: Conference Series*, 1000(1):012101, 2018.

[HH19] Syed Mahbub Hafiz and Ryan Henry. A bit more than a bit is more than a bit better. *Proceedings on Privacy Enhancing Technologies*, 4:112–131, 2019.

[HK15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[IFO15] Folasade Olubusola Isinkaye, Yetunde O. Folajimi, and Bolande Adefowoke Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian informatics journal*, 16(3):261–273, 2015.

[JBE$^+$13] Arjan JP Jeckmans, Michael Beye, Zekeriya Erkin, Pieter Hartel, Reginald L Lagendijk, and Qiang Tang. Privacy in recommender systems. In *Social media retrieval*, pages 263–281. Springer, 2013.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[KHG12] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. *Cryptology ePrint Archive*, 2012.

[LSY03] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[LU12] Ting Li and Till Unger. Willing to pay for quality personalization? trade-off between quality and privacy. *European Journal of Information Systems*, 21(6):621–642, 2012.

[NIW$^+$13] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 801–812. ACM, 2013.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

[OY18] Taiwo Blessing Ogunseyi and Cheng Yang. Survey and analysis of cryptographic techniques for privacy protection in recommender systems. In *International Conference on Cloud Computing and Security*, pages 691–706. Springer, 2018.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[PD03] Huseyin Polat and Wenliang Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *Third IEEE International Conference on Data Mining*, pages 625–628. IEEE, 2003.

[RCR15] Vineeth Rakesh, Jaegul Choo, and Chandan K Reddy. Project recommendation using heterogeneous traits in crowdfunding. In *Ninth International AAAI Conference on Web and Social Media*, 2015.

[RV97] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[SCJS15] Bharath K Samanthula, Lei Cen, Wei Jiang, and Luo Si. Privacy-preserving and efficient friend recommendation in online social networks. *Trans. Data Privacy*, 8(2):141–171, 2015.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SL17] Brent Smith and Greg Linden. Two decades of recommender systems at amazon.com. *IEEE internet computing*, 21(3):12–18, 2017.

[SPTH09] Reza Shokri, Pedram Pedarsani, George Theodorakopoulos, and Jean-Pierre Hubaux. Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *Proceedings of the third ACM conference on Recommender systems*, pages 157–164, 2009.

[Uni13] University of Waterloo. CrySP RIPPLE Facility. https://ripple.uwaterloo.ca/, 2013. [Online; accessed 23-Oct-2018].

[VBH21] Adithya Vadapalli, Fattaneh Bayatbabolghani, and Ryan Henry. You may also like... privacy: Recommendation systems meet PIR. *Proceedings on Privacy Enhancing Technologies*, 2021(4):30–53, 2021.

[WBIT12] Udi Weinsberg, Smriti Bhagat, Stratis Ioannidis, and Nina Taft. Blurme: inferring and obfuscating user gender based on ratings. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 195–202. ACM, 2012.

[WZJR18] Cong Wang, Yifeng Zheng, Jinghua Jiang, and Kui Ren. Toward privacy-preserving personalized recommendation services. *Engineering*, 4(1):21–28, 2018.

[Yin20] Senci Ying. Shared MF: A privacy-preserving recommendation system. *arXiv preprint arXiv:2008.07759*, 2020.