Cryptographically-Enhanced Privacy for Recommender Systems

Arjan Jeckmans

CRYPTOGRAPHICALLY-ENHANCED PRIVACY FOR RECOMMENDER SYSTEMS

ARJAN JECKMANS

CTIT Ph.D. Thesis Series No. 13-290
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, The Netherlands

IPA Dissertation Series No. 2014-03
The research in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

This research is supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO) and partly funded by the Ministry of Economic Affairs (project number 10527).

ISBN: 978-90-365-3593-9
ISSN: 1381-3617 (CTIT Ph.D. Thesis Series No. 13-290)
DOI: 10.3990/1.9789036535939

Printed by: Gildeprint Drukkerijen
Back comic by: XKCD (http://xkcd.com/958/)

CRYPTOGRAPHICALLY-ENHANCED PRIVACY FOR RECOMMENDER SYSTEMS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op woensdag 5 februari 2014 om 14.45

door

ADRIANUS JOHANNUS PAULUS JECKMANS

geboren op 22 augustus 1984
te Boxmeer, Nederland

Dit proefschrift is goedgekeurd door:
prof.dr. P.H. Hartel

ABSTRACT

Automated recommender systems are used to help people find interesting content or persons in the vast amount of information available via the internet. There are different types of recommender systems, for example collaborative filtering systems and content-based recommender systems. However, all recommender systems share a common trait: in order to generate personalized recommendations, they require information on the attributes, demands, or preferences of the user. Typically, the more detailed the information related to the user is, the more accurate the recommendations for the user are. Service providers running the recommender systems collect large amounts of personal information to ensure accurate recommendations. This data must be protected to increase the privacy of all participating users.

Privacy is typically enhanced through one (or more) of three methods: (1) decentralization, (2) introduction of uncertainty, and (3) secure computation.

Decentralization aims to remove the central service provider and gives more control to the individual users. However, decentralized systems cannot guarantee the availability of data as users go online and offline as they please. Furthermore, no single entity is responsible for data that does not belong to a specific user (such as item data).

Uncertainty is typically introduced by adding random noise to the data, which provides a mask over the user information. However, this noise negatively impacts the accuracy of the recommender system. When the users introduce their own noise, then the system consists mainly of noise. To preserve accuracy, only the service provider introduces noise, therefore no privacy is achieved against the service provider.

Secure computation protects the data that is used during the computation of recommendations by providing confidentiality, both at rest and during computation. However, it suffers from a large computational overhead, due to the use of cryptography and secure multi-party protocols.

In this thesis we focus on the use of secure computation to enhance the privacy of recommender systems, where we strive to make the computations as efficient as possible. To provide this, we build specialized secure computation protocols based on homomorphic encryption schemes and secure multi-party computation. Each protocol is tailored to the specific problem that is addressed, with a minimum of expensive operations and interactions. These protocols address the following challenges: (1) fostering cooperation between competing service providers, (2) coping with the unavailability of users, and (3) dealing with malicious intent by the users.

Cooperating service providers are able to leverage each others databases to provide better recommendations. However, privacy of users and secrecy of a service provider's database normally prevents competing service providers from collaborating based on sharing their plaintext databases. We provide a secure protocol that allows competing service providers to collaborate and share their respective databases of information, without leaking the database to the competitor.

Most existing secure computation protocols for recommender systems require interaction between the service provider and its users, which makes unavailability of users a serious issue. Secure computation protocols that do not rely on the availability of users are therefore preferred. We contribute a secure protocol that allows users to be unavailable during the computation of a recommendation for a specific user (this specific user is still required to be online). The typical approach to deal with unavailable users is to introduce a second (independent) server, which needs to be (partly) trusted by the users. Our protocol does not rely on an additional server, but instead relies on existing trust relationships (e. g. friendship) between users who wish to share their preferences.

In general, secure computation protocols for recommender systems assume honest behaviour of participating users. However, this assumption is not valid in most cases, as users attempt to exploit the recommender system for their own gain. More robust protocols for recommender systems are preferred. We present a secure framework for recommender systems that can cope with malicious user behaviour. The framework consists of two protocols for users to update ratings and retrieve recommendations. The framework can be instantiated with different types of recommender systems.

SAMENVATTING

Geautomatiseerde aanbevelingssystemen worden gebruikt om mensen te helpen met het vinden van interessante inhoud of personen in de grote hoeveelheid beschikbare informatie op het internet. Er zijn verschillende soorten aanbevelingssystemen, bijvoorbeeld collaboratieve filtering systemen en inhoud gebaseerde aanbevelingssystemen. Echter, alle aanbevelingssystemen delen een gemeenschappelijk kenmerk: om gepersonaliseerde aanbevelingen te genereren, vereisen zij informatie over de attributen, eisen of voorkeuren van de gebruiker. In het algemeen, hoe gedetailleerder de informatie gerelateerd tot de gebruiker is, hoe accurater de aanbevelingen voor de gebruiker zijn. Serviceaanbieders die de aanbevelingssystemen opereren verzamelen grote hoeveelheden persoonlijke informatie om accurate aanbevelingen te waarborgen. Deze informatie dient beschermt te worden om de privacy van alle deelnemende gebruikers te verhogen.

Privacy wordt vaak verhoogd door één (of meer) van deze drie methoden: (1) decentralisatie, (2) introductie van onzekerheid en (3) beveiligde berekeningen.

Decentralisatie heeft als doel om de serviceaanbieder te verwijderen en meer controle te geven aan de individuele gebruikers. Echter, gedecentraliseerde systemen kunnen de beschikbaarheid van informatie niet garanderen, want gebruikers kunnen online en offline gaan wanneer ze willen. Verder is er geen enkele partij verantwoordelijk voor informatie die niet aan een specifieke gebruiker toebehoord (zoals artikel informatie).

Onzekerheid wordt normaal geïntroduceerd door willekeurige ruis aan de informatie toe te voegen, wat dan de gebruikers informatie verbergt. Echter, deze ruis beïnvloedt de accuraatheid van het aanbevelingssysteem op een negatieve manier. Wanneer gebruiker hun eigen ruis toevoegen, dan bestaat het systeem voornamelijk uit ruis. Om de accuraatheid te waarborgen, voegt alleen de serviceaanbieder ruis toe, daarom is er geen privacy tegen over de serviceaanbieder.

Beveiligde berekeningen beschermen de informatie die wordt gebruikt tijdens het uitrekenen van de aanbevelingen door de vertrouwelijkheid van informatie te verstrekken, zowel tijdens de opslag als de berekening. Echter, hebben ze een grote overhead door het gebruik van cryptografie en beveiligde protocollen met meerdere partijen.

In dit proefschrift concentreren wij ons op het gebruik van beveiligde berekeningen om de privacy in aanbevelingssystemen te verhogen, waar we streven de berekeningen zo efficiënt mogelijk te maken. Om dit te realiseren, bouwen we specifieke beveiligde protocollen gebaseerd op homomorfe versleuteling en beveiligde berekeningen met meerdere partijen. Elk protocol is afgestemd op het specifieke pro-

bleem dat wordt aangepakt met een minimum aan dure berekeningen en interacties. Deze protocollen pakken de volgende uitdagingen aan: (1) het bevorderen van de samenwerking tussen rivaliserende serviceaanbieders, (2) omgaan met de onbeschikbaarheid van gebruikers en (3) omgaan met kwaadaardige bedoelingen van de gebruikers.

Samenwerkende serviceaanbieders zijn in staat elkaars database te gebruiken om betere aanbevelingen te doen. Echter, de privacy van gebruikers en de geheimhouding van de database van de serviceaanbieder houden normaal gesproken de samenwerking van serviceaanbieders op basis van database uitwisselingen tegen. Wij leveren een beveiligd protocol dat toestaat dat rivaliserende serviceaanbieders samenwerken en hun respectievelijke databases met informatie te delen, zonder daarbij de database aan de rivaal te onthullen.

De meeste bestaande beveiligde berekening protocollen voor aanbevelingssystemen vereisen interactie tussen de serviceaanbieder en zijn gebruikers. Dit maakt onbeschikbaarheid van de gebruikers een serieus probleem. Beveiligde berekening protocollen die niet rekenen op de beschikbaarheid van gebruikers worden geprefereerd. Wij dragen een beveiligd protocol bij dat het toestaat dat gebruikers niet beschikbaar zijn tijdens de berekening van een aanbeveling voor een specifieke gebruiker (deze specifieke gebruiker moet nog wel beschikbaar zijn). The standaard oplossing om met de onbeschikbaarheid van gebruikers om te gaan is het introduceren van een tweede (onafhankelijke) server. Deze dient (deels) vertrouwd te worden door de gebruikers. Ons protocol hangt niet af van een tweede server, maar hangt in plaats daarvan af van de bestaande vertrouwens relaties (e. g. vriendschappen) tussen gebruikers die hun voorkeuren willen delen.

Over het algemeen nemen beveiligde berekening protocollen aan dat deelnemende gebruikers zich betrouwbaar gedragen. Echter is in de meeste gevallen deze aanname niet geldig, wanneer gebruikers proberen het aanbevelingssysteem voor hun eigen gewin uit te buiten. Robuustere protocollen voor aanbevelingssystemen zijn gewenst. We presenteren een beveiligd framework voor aanbevelingssystemen dat om kan gaan met kwaadaardig gedrag van gebruikers. Het framework bestaat uit twee protocollen voor gebruikers om beoordelingen up te daten en aanbevelingen op te halen. Het framework kan met verschillende soorten aanbevelingssystemen geïnstantieerd worden.

ACKNOWLEDGEMENTS

Almost five years ago, while I was still working on my master thesis and enjoying the student life at the University of Twente, I had no idea what the future would bring to me. The work for my master thesis was done under the supervision of Qiang and Pieter. While it took some time to complete, I was happy with the final result. Getting closer to my masters defence, Pieter and Qiang offered me a chance to continue working with them and start a PhD. Being still unsure about my future, I went for a summer stroll to determine my future. I decided to take on the challenge given to me. Up to this point in my life, I have not regretted that decision and I doubt I ever will.

Pieter, as my supervisor and promotor, you have given me support, guidance, and opportunities, for which I am ever grateful. Qiang, as a daily supervisor and mentor, you taught me to be critical, ask questions, and helped me develop my research. Thank you for our time together. During the course of my PhD, Qiang left our university to pursue his own work. This left me with a time without a mentor and forced me to become more self reliant. After a long search for a suitable replacement, the group finally found Andreas. Andreas, you became my new mentor and offered me help in presenting my ideas and solutions, and asked me questions that I was not asking myself. Thank you.

Of course, during my time at the university, I was not only surrounded by my supervisors, but also by colleagues. First and foremost, there was Christoph. Our four years of PhD overlapped perfectly, starting on the same day and always sharing an office. If there was anything, I could always count on you. I'm really happy we could take this journey together. Of course, there were more people that I could count on to distract me, have a discussion with, and from whom I would gain some valuable information. For that, I would like to thank Jonathan, Elmer, and Stefan. Of course no PhD would be complete without coffee (tea) brakes. Thank you all, Eleftheria, Dina, Michael, Trajce, Luan, Saeed, Marco, Begul, Andre, and more. I would also like to thank our secretaries Bertine and Nienke before her for all their supporting activities making life a lot easier without having to worry too much about bureaucracy.

My thesis would not have been shaped like it has if it was not for my project, the Kindred Spirits project. I would also like to take this opportunity to thank the persons involved in the Kindred Spirits project outside of my university. Thanks to Zeki, Michael, Inald, Mihai, Jeroen, Thijs, Tanya, and all others involved with the project. And a special thanks to STW for making the project possible.

A thanks also to my friends Frank, Rob, and Peter. We always have had a lot of fun together and even after university life was over, having weekends to just relax, have fun, and do stuff is awesome.

Thanks to my old flatmates at Matenweg 75, housing me during my studies and the early parts of my PhD. Plenty of days and nights of hanging out gave me a comfortable place to return to after coming back from the university. Even after I moved, we had some fun.

A special thanks to my family, Kid, Heli, and Remco, for all their support. Without you, I would not have come as far as I did. You have always believed in me.

Last, but not least, thank you Dora. We met during my PhD and my life has been improved for the better since. Let us never be far apart. To me, you bring even more value to this thesis. Tvb!

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

Online applications are an important part of daily life for millions of users. People consume media (Youtube, Flickr, LastFM), do their shopping (Amazon, Ebay), and interact (Facebook, Gmail) online. Because the range and amount of content that is offered to users is often huge, automated recommender systems are employed. By providing personalized suggestions, these systems can help people find interesting media, boost sales through targeted advertisements, or help people meet new friends. Because of their automated nature, recommender systems can meet the demands of large online applications that operate on a global scale.

All recommender systems share a common trait: in order to generate personalized recommendations, they require information on the attributes, demands, or preferences of the user. Typically, the more detailed the information related to the user is, the more accurate the recommendations for the user are. Service providers running the recommender systems collect information where possible to ensure accurate recommendations. The information supplied can either be automatically collected, or specifically provided by the user. Automatically collected information is the result of users interacting with the recommender systems and making choices based on recommendations. For example, video views on Youtube are used to automatically present a selection of recommended similar videos (*recommendations for you*). Based on purchases by other users, items on Amazon are accompanied by package deals (*frequently bought together*) or related items (*customers who bought this item also bought*). Based on your friends and social interactions, Facebook suggests new friends to make. LinkedIn, based on a user's CV and connections, recommends interesting companies, job offers, and news. Vice-versa, LinkedIn also recommends people to recruiters posting new job openings. Users can also explicitly provide information. In this way, users build their own profile specifying their likes and dislikes, or containing general information (such as age and gender) about themselves. For example, Youtube allows users to specify their favorites. Facebook allows listing profile information as well as interests.

However, potential threats to user privacy are often underestimated. Users usually do not take the time to fully understand the privacy policies and its implications, while service providers aim to not bother users with the details of such policies. As such, the user often does not have a good picture of his level of privacy with the service provider.

Furthermore, the more detailed the information related to the user is, the larger the threat to the user's privacy is. In order to enhance their recommender systems, service providers are collecting and consolidating more and more information. For example, in recent privacy policy updates Google stated that they consolidate information from all their services to a single profile. Facebook continues to expand its reach around the internet, giving the ability to share more and *like* almost everything. Information might be abused by the service provider, sold to a third party, or leaked by a hacker. This data must be protected to increase the privacy of all participating users.

Lam et al. [56] note that the information the user shares with the service provider to create useful recommendations, also leads to higher risks re-identification of the user. Indeed, the information published by Netflix as part of their recommender systems prize, though anonymized, allowed for re-identification [68]. Narayanan and Shmatikov linked the anonymized records to publicly available records (such as IMDb) based on rating similarity and time of rating. If two records give a similar rating to a movie around the same time, they are likely to be from the same person. A higher number of similar movie ratings (in rating and in time) increases the confidence of the link between the records.

1.2 USER PRIVACY

The word privacy has many subtly different meanings. We give an overview of the privacy notions that are most relevant for recommender systems. On the internet, privacy revolves mainly around *information privacy*. Kang [51] used the wording of the Information Infrastructure Task Force (IITF), as cited below:

> Information privacy is "an individual's claim to control the terms under which personal information — information identifiable to the individual — is acquired, disclosed or used."

Note that the focus of information privacy is on the control of the individual. Weiss [87] stated that on the Web, privacy is maintained by limiting data collection, hiding users' identities and restricting access to authorized parties only. In practice, information and identity often become closely linked and visible to large groups of people. Profiles may be publicly visible, comments can be seen by all viewers of a content item, and some sites list the last users to visit a particular page. It becomes harder for a user to monitor and control his personal information, as more of it becomes available online. This problem mainly applies to systems where the user logs in to an account, and where tools are available to express a user's preferences, such as recommender systems.

When using recommender systems (and other online applications), users generally share a lot of (personal) information. Whether it is uploading ratings or comments, posting personal information on a profile, or making purchases, information is always shared within a particular *scope* [72]. Privacy involves keeping a piece of information in its intended scope. This scope is defined by *breadth* (the size of the audience), *depth* (extent of usage allowed), and *lifetime* (storage duration). When a piece of information is moved beyond its intended scope in any of these dimensions (be it accidentally or maliciously), a privacy breach occurs. So, a breach may occur when information is disclosed to a party for whom it was not intended, when information is abused for a different purpose than was intended, or when information is stored beyond its intended lifetime.

The concept of information privacy is strongly related to the notion of *confidentiality*, from the field of information security, but not to be used interchangeably. Information privacy focusses on the individual who is the subject of said information, the effects that disclosure have on this person, and his or her control and consent. Confidentiality is concerned with the secrecy of individual pieces of information. In this thesis, the focus will lie on preventing unwanted disclosure and usage of information, but not on the effects on the person. The focus on confidentiality implicitly defines a scope for pieces of information. This gives a solid, but static, expectation of privacy to the user.

1.3 OVERVIEW OF RECOMMENDER SYSTEMS

In this section, we give an overview of the different types of recommender systems and their relation to user privacy. A recommender system provides a set of *items* (e.g. content, solutions, or other users) that is most relevant to a particular user of the system. Typically, recommender systems achieve this by predicting *relevance scores* for all items that the user has not seen yet. Items that receive the highest score get recommended (typically the top-N items, or all items above a threshold t). The prediction is made by considering both the traits of the item and user. Typically, systems look at similarities between items, similarities between users, or relations between particular types of items and particular types of users. The performance of a recommender system is determined by the recommendation accuracy, i.e. the error between given and expected results.

Adomavicius and Tuzhilin [10] give an overview of the state of the art in recommender systems and possible extensions. They list only the three popular types of that time: collaborative filtering, content-based, and hybrid. In their list, the hybrid type is a combination of the two other types. They purposely omitted the other types of recommender systems that were not popular. We make a different distinction with four core recommender system types, taking the first two types of Adomavicius and Tuzhilin and adding two less popular, but important, types. The four core types represent the different approaches to generating recommendations and are also based around different information. Thus the core types have a significantly different impact on the privacy of the user. Possibly, these core types can be augmented with additional information. We list the following core recommender system types:

COLLABORATIVE FILTERING: One of the first collaborative filtering recommender systems is Tapestry, by Goldberg et al. [41]. This system was designed to retrieve email messages from Usenet mailing lists, relevant to a user's particular interests. Goldberg et al. observed that conventional mailing lists are too static, and rarely form a perfect match to a user's demands. Tapestry relies on what the authors termed *collaborative filtering techniques*, which are still widely used today. In collaborative filtering, each user rates content items. These ratings determine similarity between either users (similar users like similar items) or items (users like items similar to highly rated items). Different metrics exist to compute similarity. Recommended for the current user are those items that are rated highest by his most similar peers, or contain those items that are rated most similar to his favourite items. Collaborative filtering relies on the personal rating information of a lot of users. To compute recommendations, the data of every user in the system is used. Collaborative filtering is therefore very privacy invasive. The privacy impact is somewhat mitigated by the fact that the recommendations are based on the aggregate of (potentially) a lot of users. However, auxiliary information [26] and users with eclectic tastes [76] still pose risks to privacy.

CONTENT-BASED: Content-based recommender systems use item similarity to determine recommendations. Unlike the collaborative filtering method, item similarity is computed by item meta-data. Examples of meta-data are, kitchen for restaurants, genre for movies, and artist for music. Recommended are those items that are most similar to the user's favourite items. An example of a content-based recommender system is Newsweeder, by Lang [57]. Since the meta-data on which the item similarity is based is generally public information, no privacy concerns are associated with this information (still service providers might like to keep this secret). However, to compute the recommendations also the ratings of the user are required. This information is privacy sensitive. Because no private information from other users is used, the privacy impact is limited to the service provider and not to other users.

DEMOGRAPHIC: When detailed information about the user's preferences is not available, demographic information can lead to somewhat personalized recommendations. Grundy, by Rich [80], is an example of this. Demographic information may include age, gender, country of residence, education level, etc. The demographic information is matched to a stereotype, and the items attached to this stereotype are recommended. Personalization for the user is limited due to the generalization to a stereotype. It is possible to generalize this approach to categories (instead of demographics). For example, users can be categorized based on their illness in medical recommender systems. Generally, the information about the preferences of a certain demographic is public information. However, the categorization of a user to a certain demographic is based on personal information. Even the demographic that a user belongs to is considered to be personal information. Therefore, this information should be kept private from the service provider. There is no privacy impact on other users.

KNOWLEDGE-BASED: When requiring a recommendation, the user enters his preferences in the recommender system. The system then outputs a (number of) potential recommendations based on (expert) knowledge contained in the system. Possibly, the user can give feedback and the recommendation is refined. After a few iterations, the recommendation is tailored to the user. Entree [23] is an example of such a system, built to help diners find a suitable restaurant. In learning knowledge-based recommender systems, feedback from the user is fed back into the system to add to the knowledge [62]. The knowledge in a knowledge-based recommender system can either be public or private information. In the case of a learning system, the feedback of users is generally considered to be private information. This has implications on the privacy of users, as the knowledge that is build up is a combination of a lot of users and the privacy has to be respected for all users. Furthermore, the preferences that are used to determine the recommendations are personal information. These preferences should be kept private from the service provider.

Collaborative filtering, while being the most popular recommender system type, also has the highest potential impact on privacy. The ratings of a user are exposed to the service provider and to other users. Knowledge-based recommender systems also potentially leak personal information to other users. Choices that are made by users can be fed back into the system and are therefore exposed to other users and the service provider. The current recommendation preferences are also exposed to the service provider. Both content-based and demographic recommender systems do not expose personal information to other users. However, the content-based recommender system still exposes ratings to the service provider and the demographic recommender system exposes personal categorical information to the service provider.

The additional information that can be used to augment these core types can broadly be categorized in augmenting with more personal information and augmenting with additional recommender systems. For example, information about the context of a request can be added to the recommender system to increase its accuracy on a per request basis [11]. Information about the social ties of a user can also be used to improve the recommendations [54]. This additional information is also subject to exposure and thus to privacy concerns by the user.

Recommender systems can also be augmented with other recommender systems. This can be done with the same [82] or different [24] types of recommender systems. The idea is that multiple recommender systems can make decisions on different data or with different training parameters to generate different opinions that can strengthen each other to improve recommender accuracy. These other recommender systems naturally also require input data and enlarge the exposure surface for the user.

1.4 METHODS FOR ENHANCING PRIVACY

In the case of recommender systems, privacy is typically enhanced through one (or more) of three methods: (1) decentralization, (2) introduction of uncertainty, and (3) secure computation.

Decentralization aims to remove the central service provider and gives more control to the individual users. However, decentralized systems cannot guarantee the availability of data as users go online and offline as they please. Furthermore, no single entity is responsible for data that does not belong to a specific user (such as item data). These issues impact the quality of the recommendation service and, since there is no responsible service provider, might not be resolved.

Uncertainty is typically introduced by adding random noise to the data, which provides a mask over the user information. Alternatively, data from multiple users is aggregated into the profile of a single user. However, both approaches negatively impact the accuracy of the recommender system. The specific functions inside the recommender system and the time when uncertainty is introduced determine the amount of uncertainty that is required to guarantee certain levels of privacy. A higher level of privacy requires a larger amount of noise, or more data to aggregate. Furthermore, the sooner the uncertainty is introduced, the more uncertainty is required. When this uncertainty then propagates through the system, it is amplified. Therefore, when the users introduce their own noise when presenting their private information, or aggregate their profile with a lot of other users, the system then consists mainly of uncertainty. To preserve accuracy, typically only the service provider introduces uncertainty, therefore no privacy is achieved against the service provider. Furthermore, when aggregation is used, some user privacy is lost as genuine data is required to create the aggregate.

Secure computation protects the data that is used during the computation of recommendations by providing confidentiality, both at rest and during computation. However, it suffers from a large computational overhead, due to the use of cryptography and secure multi-party protocols.

As decentralization does not offer a central authority on public information, it cannot be used to provide privacy in three out of the four recommender system types. Furthermore, the continues flux of users greatly impacts the availability, reliability, and freshness of data. This leads to a decrease in the service quality of the recommender system. Therefore, decentralization is not a good candidate to preserve the privacy in recommender systems.

Introduction of uncertainty cannot guarantee both accuracy of the recommendation and privacy against the service provider at the same time. Due to the amount of uncertainty that needs to be added to have privacy, this trade-off is inherent to this approach to privacy. A recommender system with low accuracy does not present any utility to the user, and a having no option for privacy against the service provider is not desirable for the user either. Therefore, the introduction of uncertainty is not a good approach for enhancing the privacy of users of recommender systems.

Secure computation does not suffer from service quality and accuracy loss and is able to provide privacy against the service provider while having a central authority on public information. However, the privacy offered by secure computation does come at a cost. The overhead caused by secure computation is far greater than for recommender systems without privacy. As opposed to the previous enhancement methods, this is a cost that can be compensated for up to a certain extent. The development of faster primitives and the increase in computational power of computers, can potentially lower the overhead to reasonable levels. Therefore, in this thesis we focus on the use of secure computation to enhance the privacy of recommender systems, where we strive to make the computations as efficient as possible.

Next to the overhead caused by secure computations, there is another challenge in the design of privacy-enhanced recommender systems. Because the secure computation is not allowed to leak personal information, the computations have to assume that all possible data is present and/or relevant. In a demographic recommender system, the service provider will have to assume that the user to recommend for is part of all demographics, compute recommendations for all of them, and then at the end select the appropriate demographic. This leads to an expected complexity in the order of number of demographics. In a knowledge-based recommender system, this means applying the preferences to all rules (if expressed in rules) in the knowledge base. Leading to an expected complexity in the order of number of rules in the knowledge base. For collaborative filtering this implies that each element in the entire database of user ratings has to be touched at least once. This leads to an expected complexity in the order of the number of users times the number of items. This is opposed to non-private recommender systems, where dismissing useless data early greatly increases the scalability of recommender systems. For example, there is no need to compute recommendations for items that the user has already given a rating to. In collaborative filtering, to increase scalability, a neighbourhood of similar users is selected and the data of all other users is ignored when computing recommendations. This is a second overhead challenge that requires attention when designing secure computation protocols for recommender systems.

1.5 RESEARCH QUESTIONS

Because of the additional overhead caused by secure computation and the need for recommender systems to have huge databases to increase the accuracy of recommendations, efficiency of solutions becomes a main concern. In order to make recommender systems with privacy based on secure computation more practical and encourage deployment, we ask the following main research question:

RESEARCH QUESTION: How to construct efficient privacy-enhanced recommender systems?

Specifically, we focus on three practical scenarios that are motivated by the choice of secure computation and its implications. In these three scenarios, next to addressing the specific problem, we strive for efficiency. These three scenarios are taken from interaction with companies and drawbacks in existing secure computation solutions. We feel that these scenarios represent the more pressing issues and we hope to enable the deployment of privacy-enhanced recommender systems. Of course, more practical scenarios exist, in this thesis we are unable to address all of them. The three scenarios are captured in the following research sub questions:

SUB QUESTION 1: How can competing recommender system service providers collaborate?

Cooperating service providers are able to leverage each others databases to provide better recommendations. However, privacy of users and secrecy of a service provider's database normally prevents competing service providers from collaborating based on sharing their plaintext databases. There is then nothing to stop the competitor from running away with the newly acquired database and immediately stop the collaboration. How can this hurdle of competition be overcome by privacy-enhanced recommender systems, leading to both benefit for the service providers and the users?

SUB QUESTION 2: How to cope with the limited user availability in recommender systems?

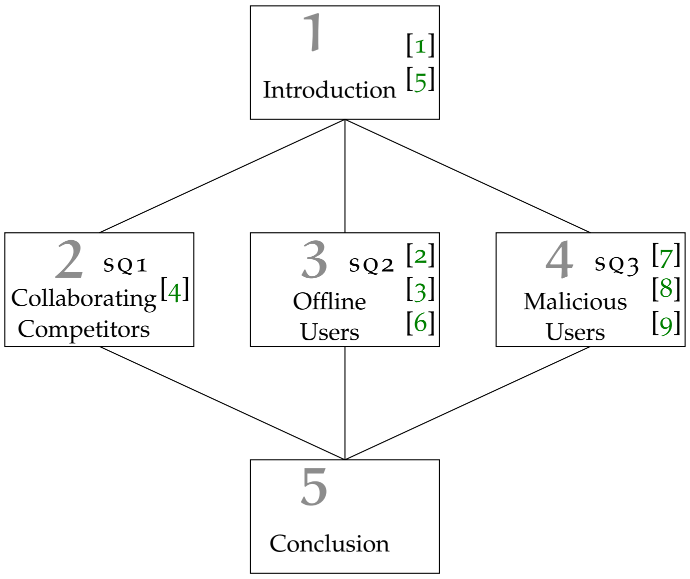


Figure 1: Outline of the thesis

Most existing secure computation protocols for recommender systems require interaction between the service provider and its users, which makes unavailability of users a serious issue (particularly in the case of collaborative filtering). When this issue is not addressed, the efficiency of the recommender system becomes dependent on the availability of the users. If one user goes on holiday for several weeks and all other users have to wait for this user to get back home, efficiency will be low. The typical approach to deal with unavailable users is to introduce a second (independent) server, which needs to be (partly) trusted by the users. We aim for a solution that requires neither availability of users nor a second server.

SUB QUESTION 3: How to deal with malicious intent by the users?

In general, privacy-enhanced recommender systems assume honest behaviour of participating users. However, this assumption is not valid in most cases, as users attempt to exploit the recommender system for their own gain. An author of a book, might try to increase the reputation of his book to increase sales. We aim to increase the robustness of privacy-enhanced recommender systems against malicious intent by users. Furthermore, we aim for a general solution that can be combined with different types of recommender systems.

1.6 CONTRIBUTIONS AND THESIS OUTLINE

Figure 1 shows the outline of this thesis in picture form. In this picture, the title of each chapter is shown, which research sub question is answered, as well as all publications that chapter is based on. Because the three sub questions and scenarios are quite different, each chapter will feature a separate related work section and a separate introduction of used primitives. Furthermore, each chapter has a detailed security and privacy analysis, as well as a detailed performance analysis with actual runtime figures based on a prototype implementation. The protocols and analysis of each chapter have been peer reviewed, except for Chapter 4 which is still in submission to a journal. For the fourth chapter, the protocols have been verified by experts outside the university.

INTRODUCTION: The current chapter, which provides an introduction to recommender systems and privacy, states the research questions, and gives an overview of the thesis.

COLLABORATING COMPETITORS: The second chapter answers sub question 1. We provide a secure protocol that allows competing service providers to collaborate and share their respective databases of information, without leaking the database to the competitor. The recommender system is a collaborative filtering system and it is assumed that the competitors share the same items. As neighbourhood selection does not help to speed up secure computation, because every record needs to be touched anyway, this step is omitted. However, this leads to requiring the computation of absolute values. The impact of this decision as well as different distributions between the competitors are analysed and discussed.

OFFLINE USERS: The third chapter answers sub question 2. We contribute a secure protocol that allows users to be unavailable during the computation of a recommendation for a specific user (this specific user is still required to be online). Typically, protocols rely on an additional server to split the data or trust of the user to ensure privacy. Instead our solution relies on existing trust relationships (e. g. friendship) between users who wish to share their preferences. In this way, the friends act like a second server. However, due to the unavailability of users, transferring the data from the users to their friends cannot be done by simply synchronizing the data when both are available. Proxy re-encryption is used for on demand sharing of data.

MALICIOUS USERS: The fourth chapter answers sub question 3. We present a secure framework for recommender systems that can cope with malicious user behaviour. The framework can be instantiated with different types of recommender systems that are based on ratings. As we only assume the user to be malicious and not the service provider, minimizing the interaction that the user has with the recommender system reduces complexity and increases efficiency. Therefore, the framework consists of two protocols for users to update ratings and retrieve recommendations. In the rating update protocol, no expensive secure comparison protocol is used to check the validity of the user input. To ensure the privacy and the trust of the user in the service provider, the framework assumes two non-colluding servers. In this chapter we also discuss the shilling attack in relation to our framework, an attack which is not covered by the cryptographic definition of a malicious user. This attack allows users to intentionally bias the recommender system, simply by giving valid ratings to the system.

CONCLUSION: The final chapter brings together the answers of the different sub questions to reflect on the main research question of this thesis. Compared to the state of the art, our solutions bring improvements in the assumptions that are made, the private information that is leaked, and the efficiency.

2 COLLABORATING COMPETITORS

2.1 INTRODUCTION

Recommender systems typically use data from the entire customer database of a service provider (company). Companies, which have a lot of customers, are more likely to have enough data to generate good recommendations. However, other companies do not necessarily have enough data to do so [75]. In any case, more customer data can only lead to better recommendations. For companies to gain access to more customer data and to provide more meaningful recommendations for their customers, they can: (1) request the aid of another company which has a large customer database, or (2) collaborate with multiple other companies which contribute their relatively small customer databases to create a large one.

The issue is that companies may not be able to simply share, or give each other full access to, their customer databases. This will result in an undesirable loss of control over their customer database, which is basically their main asset. In addition, sharing customer data may result in loss of customer trust, or privacy regulations may prohibit such data sharing activities. We assume that the customer trusts the company that it chose and we are therefore not concerned about the privacy of the customer towards his chosen company. Companies may be suggested to rely on a third party to generate recommendations. However, this requires companies to share all their data with the third party, and is undesirable as well.

The challenge is to find an efficient privacy-preserving mechanism which allows companies to generate recommendations based on their joint sets of databases, while preserving the privacy of their individual customer database respectively.

2.1.1 *Contribution*

In this chapter, we first detail the used collaborative filtering algorithm without privacy for the two-company setting, where company A requests the aid of company B to get recommendations for its customer. The used formulas are introduced before transferring them to the encrypted domain. Then, we construct a privacy-preserving collaborative filtering algorithm. In our solution, company A uses a homomorphic encryption scheme to hide its customer's data and share the encrypted data with company B, which computes its contributions to the final recommendations in the encrypted domain. From company B, company A only obtains aggregated and anonymized data, which however allow it to generate the top X recommendations for its customer. In the honest-but-curious model (where companies adhere to the protocol, but try to learn additional information), our solution guarantees that: (1) company A has only access to an aggregated and randomized version of the database of company B; (2) company B does not learn any information about the customer data of company A.

To achieve this solution, we propose two secure two-party protocols as building blocks. These protocols, namely the secure absolute value protocol ABS and the secure division protocol DIV, can also be used as building blocks in other protocols outside this work. We then build a prototype of our solution and, based on this prototype, present performance (computation/communication costs and accuracy) results. We show a linear relation between the number of customers and execution time, which is the best that can be achieved (as similarity has to be computed with all other customers). We confirm a larger accuracy gain for company A as the difference in customer population between the companies increases.

2.1.2 *Organization*

In Section 2.2, we formally specify the recommendation scenario. In Section 2.3, we present our solution. In Section 2.4, we analyse the security of our solution. In Section 2.5, we report on the performance of our prototype implementation. In Section 2.6, we review the related work, and Section 2.7 concludes the chapter.

2.2 PROBLEM STATEMENT AND SECURITY MODEL

In this section, we describe the research problem in the two-company setting, and present our security model.

2.2.1 *Problem Statement*

In the two-company setting, company A collaborates with company B in order to get better recommendations for its customers. We assume that company A has n' customers and company B has $n - n'$ customers, so that they have n customers in total. We further assume that both companies share a set of m items. Should this not be the case, excess items can be removed, but no recommendations will be available for them. The customers from both companies have provided some ratings on this set of m items. For the simplicity of description, we assume that there is no common customer between company A and company B. Should a customer be common to both, the companies will not find out as customer information is not shared. However, a customer might get a recommendation based on himself if the ratings are different for the different companies. This is more an annoyance than an actual problem, as the customer receives recommendations for items he is no longer interested in.

Let a rating be an integer from a domain $[v_{min}, v_{max}]$. The ratings of customer y, for $1 \leqslant y \leqslant n$, are denoted as a vector $V_y = (v_{y,1}, v_{y,2}, \cdots, v_{y,m})$ where $v_{y,i}$, for any $1 \leqslant i \leqslant m$, represents customer y's rating for item i. Company A holds the rating vectors V_y $(1 \leqslant y \leqslant n')$, and company B holds the rating vectors V_y $(n'+1 \leqslant y \leqslant n)$. Let the average rating of customer y be denoted by $\bar{v}_y = \frac{\sum_{i=1}^{m} v_{y,i}}{m}$.

The research problem is to design a privacy-preserving collaborative filtering algorithm such that: for customer x, where $1 \leqslant x \leqslant n'$, company A can compute the top X unrated items (by customer x) with the highest predictions, which are computed from its own database and that of company B. A prediction of item i for customer x is denoted by $\mathrm{pred}_{x,i}$.

2.2.2 *Security Model*

We assume that company A and company B are honest-but-curious, which means that they will adhere to the protocol specification but will try to infer information from the protocol execution transcripts. The rationale behind this assumption is that the companies are expected to have signed a service level agreement when engaging in a collaboration. Malicious behaviours will be deterred due to the potential monetary penalties and legal actions. Customer feedback can be used to test the validity of the recommendations. For example, when a number of customers of company A receive useless recommendations, company B might have acted maliciously. As a last resort, company A can create a customer in both companies and compare the recommendations received. We further assume that the customer trusts the company that it chose and is not concerned about his privacy regarding this company. But the customer is concerned regarding his privacy and the other company.

Before describing the privacy requirements, we note an asymmetry between the roles of company A and company B: company A will make use of company B's database to generate recommendations, therefore company A will be able to learn (or, infer) some information about company B's database; on the other side, there is no opportunity for company B to learn anything about company A's database because it will not generate anything. Therefore, we distinguish two cases for privacy protection.

Privacy of Company A

Company A should leak no information about its customer database $(V_y, 1 \leqslant y \leqslant n')$ to company B, namely company B should learn nothing from a protocol execution.

Privacy of Company B

We observe that, if company A learns the predictions $(\mathtt{pred}_{x,i}, 1 \leqslant i \leqslant m)$ for a customer x, then it is able to recommend those with high predictions to the customer. Note that the predictions are generated based on the databases from both company A and company B $(V_y, 1 \leqslant y \leqslant n)$. Based on this observation, we require that, in a protocol execution, company A learns only the information that can be inferred from the predictions $(\mathtt{pred}_{x,i}, 1 \leqslant i \leqslant m)$, but nothing else (e. g. $V_y, n' + 1 \leqslant y \leqslant n$).

Depending on the application scenario, the requirement for the *privacy of company B* can be enhanced. For instance, instead of learning the predictions, we can require that company A only learns the top X items with the highest predictions. Achieving such a strong privacy guarantee may result in an intolerable complexity of the solution. We leave further discussions of such specific scenarios as future work.

2.3 PROPOSED SOLUTION

In this section we first present the collaborative filtering algorithm in the plaintext domain, and then transform the operations into the encrypted domain.

2.3.1 *Recommendation without Encryption*

There are two approaches to design collaborative filtering algorithms. One is the neighbourhood-based approach (e. g. [47]), and the other one is the latent factor based approach (e. g. [55, 77]). In this chapter, we choose a neighbourhood-based collaborative filtering algorithm, as we believe they can be more efficiently represented in the encrypted domain. This is due to the operations involved, which can be computed in an efficient manner, as opposed to the latent factor model building.

Following the framework proposed by Herlocker et al. [47], a neighbourhood-based collaborative filtering algorithm generally operates in three steps:

1. The customer similarity computation step: the similarities between customer x and all other customers are computed based on their ratings.

2. The neighbourhood selection step: the most similar customers to customer x are selected. This step aims to improve recommendation efficiency and accuracy.

3. The prediction generation step: the predictions for customer x are computed. This is done based on the ratings of the neighbourhood selected in the previous step.

In this subsection, we detail the formulas that are used in our solution for each step. There is no privacy protection yet.

Computing Customer Similarity

Herlocker et al. [47] provide a comparison of the most popular similarity metrics for collaborative filtering. They conclude that the Pearson correlation is the best correlation metric to use. In the Pearson correlation the influence of a customers mean rating is taken out, as similar customers might not have a similar rating behaviour. The formula for the Pearson correlation for two customers x and y is given by:

$$sim_{x,y} = \frac{\sum_{i=1}^{m}(v_{x,i} - \bar{v}_x)(v_{y,i} - \bar{v}_y)}{\sqrt{\sum_{i=1}^{m}(v_{x,i} - \bar{v}_x)^2 \cdot \sum_{i=1}^{m}(v_{y,i} - \bar{v}_y)^2}} \tag{1}$$

The result of this formula $sim_{x,y}$ is the similarity between customers x and y. The range of $sim_{x,y}$ is $[-1, 1]$. For our convenience, we rewrite the formula as follows.

$$sim_{x,y} = \sum_{i=1}^{m} c_{x,i} c_{y,i}, \text{where} \tag{2}$$

$$c_{x,i} = \frac{v_{x,i} - \bar{v}_x}{\sqrt{\sum_{j=1}^{m}(v_{x,j} - \bar{v}_x)^2}}, \; c_{y,i} = \frac{v_{y,i} - \bar{v}_y}{\sqrt{\sum_{j=1}^{m}(v_{y,j} - \bar{v}_y)^2}} \tag{3}$$

For $1 \leqslant i \leqslant m$, the range of $c_{x,i}$ (or $c_{y,i}$) is $[-1, 1]$, and only the vector V_x (or V_y) is needed to compute $c_{x,i}$ (or $c_{y,i}$). Define the vector $C_x = (c_{x,1}, c_{x,2}, \cdots, c_{x,m})$. Then the similarity can be computed by taking the inner product of the vectors C_x and C_y, where C_y is defined in the same way as C_x.

Selecting Neighbourhood

Herlocker et al. [47] suggest selecting the top z most similar customers as a neighbourhood, where z is a parameter that depends on the dataset used. This provides a good *coverage* (i. e. having a prediction for many items), while limiting the noise of not so similar customers.

However, instead of selecting a neighbourhood of similar customers, we select the entire customer population as the neighbourhood. We make this choice because it will increase the performance in the encrypted domain. Selecting the neighbourhood is by far the most expensive step in the protocol of Erkin et al. [37]. This choice results in a slightly lower accuracy for items that were already covered in the neighbourhood selection scheme (due to added noise). However, it enables us to use dissimilar customers through negative correlation and increase the coverage to the maximum possible.

Generating Predictions

To generate a recommendation, Herlocker et al. [47] suggest using a prediction algorithm that uses the deviation from mean approach to normalization. The prediction is normalized based on the means of the customers, as again similar customers might not have a similar rating behaviour. We use the following formula, introduced by Resnick et al. [79], to compute predictions:

$$\mathsf{pred}_{x,i} = \bar{v}_x + \frac{\sum_{y=1}^{n}(v_{y,i} - \bar{v}_y)\mathsf{sim}_{x,y}}{\sum_{y=1}^{n}|\mathsf{sim}_{x,y}|} \tag{4}$$

The result of this formula $\mathsf{pred}_{x,i}$ is a predicted rating for item i by customer x. The range of $\mathsf{pred}_{x,i}$ is $[2 \cdot v_{min} - v_{max}, 2 \cdot v_{max} - v_{min}]$. Since we only need the relative order of the predictions to compute the top X recommendations, we use a simplified formula where $\bar{v}_x$ is taken out (it is constant for x). We rewrite the formula for use with two companies, resulting in $\mathsf{pred}'_{x,i} = \frac{E_{x,i}}{D_{x,i}}$, where

$$\begin{aligned} &E_{x,i} = E^A_{x,i} + E^B_{x,i},\ D_{x,i} = D^A_{x,i} + D^B_{x,i},\\ &E^A_{x,i} = \textstyle\sum_{y=1}^{n'}(v_{y,i} - \bar{v}_y)\mathsf{sim}_{x,y},\ D^A_{x,i} = \sum_{y=1}^{n'}|\mathsf{sim}_{x,y}|,\\ &E^B_{x,i} = \textstyle\sum_{y=n'+1}^{n}(v_{y,i} - \bar{v}_y)\mathsf{sim}_{x,y},\ D^B_{x,i} = \sum_{y=n'+1}^{n}|\mathsf{sim}_{x,y}| \end{aligned} \tag{5}$$

Intuitively, company A can compute $E^A_{x,i}$ and $D^A_{x,i}$, and, given C_x, company B can compute $E^B_{x,i}$ and $D^B_{x,i}$. Together, they can compute the order of the predictions $\mathsf{pred}'_{x,i}$. If needed, company A can reconstruct $\mathsf{pred}_{x,i}$ as it knows $\bar{v}_x$.

2.3.2 *Cryptographic Preliminaries*

In this subsection, we first review our main cryptographic primitive, namely the Paillier encryption scheme [71], then show how to encrypt negative values. Note that, we use the symbol $\in_r$ to denote uniform random selection. For example, $x \in_r \mathbb{Z}_N$ denotes taking x as a uniform random element from $\mathbb{Z}_N$.

Paillier Encryption

The $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ algorithms of Paillier encryption scheme [71] are as follows.

$\mathsf{KeyGen}(\ell)$: This algorithm generates a tuple (N, p, q, g, λ), where p and q are two primes with the size determined by the security parameter ℓ. The other values are $N = pq$, $\lambda = lcm(p-1, q-1)$, and $g \in_r \mathbb{Z}^*_{N^2}$. The private key is $SK = \lambda$, and the public key is $PK = (N, g)$.

$\mathsf{Enc}(m, PK)$: The ciphertext for a message $m \in \mathbb{Z}_N$ is $c = g^m r^N \mod N^2$, where $r \in_r \mathbb{Z}_N$. For simplicity, we denote $\mathsf{Enc}(m, PK)$ as $[m]_{PK}$, or $[m]$ when it is clear from the context which public key is used.

$\mathsf{Dec}(c, SK)$: This algorithm computes the message m, from the ciphertext c, as $m = L(c^\lambda \mod N^2)/L(g^\lambda \mod N^2) \mod N$. $L(u)$ is defined as $(u-1)/N$ for $u \in \mathbb{Z}^*_{N^2}$.

The scheme is semantically secure under the decisional composite residuosity assumption [71]. Based on the description, it is straightforward to verify that Paillier scheme possesses the following homomorphic properties.

$$[m_1] \cdot [m_2] = [m_1 + m_2], \ ([m_1])^{m_2} = [m_1 \cdot m_2].$$

Encrypting Negative Integers

To represent negative integers we make use of the cyclic property of the cryptosystem. The top half of the message space will represent negative numbers. When the message space is $m \in \mathbb{Z}_N$, we represent $-m$ by $N - m$, as $N - m \equiv -m \pmod{N}$. We have to be careful of overflows so that a negative number does not suddenly become a positive number or vice versa.

2.3.3 *Cryptographic Sub Protocols*

In this subsection, we describe the sub protocols for secure comparison, secure absolute value, and secure division.

Secure Comparison Sub Protocol

The secure comparison protocol, which is denoted by $\mathsf{COMP}(x, y)$, is run between company A and company B, where company A has x and company B has y. The protocol is used to compare the values of x and y and give an output based on their relation. At the end, company A should learn the result res, which is 1 when $x \geqslant y$ and -1 otherwise. Company B learns nothing from the protocol execution. The secure comparison protocol is used as a building block in the secure absolute value protocol detailed below. Since Yao [89], a lot of solutions have been proposed [40, 53, 52, 85]. In this chapter, we use that of Veugen [85].

Secure Absolute Value Sub Protocol

The secure absolute value protocol computes the absolute value of a value x, and is run between company A and company B. In the protocol company A has a Paillier key pair (PK, SK) and company B has $[x]$ and the public key of company A, PK. Figure 2 shows

	Company A		Company B
	$(\mathsf{PK}, \mathsf{SK})$		$(\mathsf{PK}, [x])$
1.			$b \in_r \{-1, 1\}$, $r_1 \in_r \mathbb{Z}_{2^{200}}$
			$[y] = [x \cdot b + r_1] = [x]^b \cdot [r_1]$
		$\xleftarrow{[y]}$	
2.	decrypt: y		
		$\xrightarrow{y}$ $res = \mathsf{COMP}(y, r_1)$ $\xleftarrow{r_1}$	
		$\xleftarrow{res}$	
	encrypt: $[res]$		
		$\xrightarrow{[res]}$	
3.			$[z] = [res]^b$
			$r_2 \in_r \mathbb{Z}_N$
			$[x + r_2] = [x] \cdot [r_2]$
			$r_3 \in_r \mathbb{Z}_N^*$
			$[z \cdot r_3] = [z]^{r_3}$
		$\xleftarrow{[x+r_2],[z \cdot r_3]}$	
4.	decrypt: $z \cdot r_3$		
	$[s] = [x + r_2]^{z \cdot r_3}$		
		$\xrightarrow{[s]}$	
5.			$[t] = [s]^{\frac{1}{r_3}}$
			$[\lvert x \rvert] = [t] \cdot ([z]^{r_2})^{-1}$
			$([\lvert x \rvert])$

Figure 2: Secure ABS Sub Protocol

our solution to compute the absolute value securely. We require that $-2^{50} \leqslant x \leqslant 2^{50}$, so that x can be hidden statistically without causing an overflow from a positive to a negative number. At the end, company B should learn $[|x|]$ while company A learns nothing. Let the protocol be denoted by $\mathsf{ABS}([x])$.

In more detail, the protocol acts as follows:

1. Company B selects $b \in_r \{-1, 1\}$, $r_1 \in_r \mathbb{Z}_{2^{200}}$. The domain of r_1 is chosen in such a way that it can hide x with statistical security without causing an overflow. Then, company B computes $[y] = [x]^b \cdot [r_1]$ and sends $[y]$ to company A.

2. Company A decrypts y and runs the secure comparison sub protocol with company B who has r_1. Company A obtains res, which is either 1 or -1, encrypts it, and sends $[res]$ to company B.

3. Company B computes $[z] = [res]^b$. The value of z equals 1 when $x \geqslant 0$ and -1 when $x < 0$. Since res contains 1 when $x \cdot b \geqslant 0$, this means that either both x and b are positive, both are negative, or $x = 0$. When both x and b are positive, $z = res \cdot b = 1 \cdot 1 = 1$, $x > 0$ and $|x| = z \cdot x > 0$. When both are negative, z is -1, $x < 0$, and $z \cdot x > 0$. When $x = 0$, $z \cdot x = 0$ independent of z. The relation between z and x similarly holds when res is equal to -1, always leading to $|x| = z \cdot x > 0$. Company B then selects $r_2 \in_r \mathbb{Z}_N$ and $r_3 \in_r \mathbb{Z}_N^*$, and sends $[x + r_2]$ and $[z \cdot r_3]$ to company A.

4. Company A decrypts $[z \cdot r_3]$, and sends $[x + r_2]^{z \cdot r_3}$ to company B.

5. Company B computes $[|x|] = [(x + r_2) \cdot z \cdot r_3]^{\frac{1}{r_3}} \cdot ([z]^{r_2})^{-1}$.

Secure Division Sub Protocol

The secure division protocol, shown in Figure 3, computes the division of two variables x and y. The protocol is run between company A and company B, where company A has a Paillier key pair (PK, SK) and company B has $[x], [y]$, and the public key of company A PK. Our solution for secure division computes the division based on the multiplicative inverse and a list lookup. We assume $y \neq 0$. At the end, company A should learn $\frac{x'}{y'}$ while company B learns nothing, where $\frac{x'}{y'} = \frac{x}{y}$ and $GCD(x', y') = 1$. Note that the equation $\frac{x'}{y'} = \frac{x}{y}$ holds in the integer domain instead of $\mathbb{Z}_N$.

In more detail, the protocol acts as follows:

1. Company B selects $r_1, r_2 \in_r \mathbb{Z}_N^*$ and sends $[y \cdot r_1]$ and $[x \cdot r_2]$ to company A.

2. Company A decrypts $[y \cdot r_1]$ and inverts it to obtain $y^{-1} \cdot r_1^{-1}$. It then computes $[x \cdot y^{-1} \cdot r_1^{-1} \cdot r_2] = [x \cdot r_2]^{y^{-1} \cdot r_1^{-1}}$ and sends $[x \cdot y^{-1} \cdot r_1^{-1} \cdot r_2]$ to company B.

3. Company B computes $[x \cdot y^{-1}] = [x \cdot y^{-1} \cdot r_1^{-1} \cdot r_2]^{\frac{r_1}{r_2}}$, and sends $[x \cdot y^{-1}]$ to company A.

4. Company A decrypts to retrieve $x \cdot y^{-1}$, which is in the domain of $\mathbb{Z}_N^*$. Suppose $-T \leqslant x, y \leqslant T$ and $T << N$, then company A can build a list of pairs $(x \cdot y^{-1}, \frac{x'}{y'})$, where $\frac{x'}{y'} = \frac{x}{y}$ and $GCD(x', y') = 1$. Company A then looks up the list and obtains $\frac{x'}{y'}$.

2.3.4 *Recommendation with Encryption*

As specified in Section 2.2.1, company A's customer database size is n' and company B's database size is $n - n'$. For customer x, the ratings

	Company A		Company B
	(PK, SK)		$(PK, [x], [y])$
1.			$r_1, r_2 \in_r \mathbb{Z}_N^*$
			$[y \cdot r_1] = [y]^{r_1}$
			$[x \cdot r_2] = [x]^{r_2}$
		$\xleftarrow{[y \cdot r_1], [x \cdot r_2]}$	
2.	decrypt: $y \cdot r_1$		
	$y^{-1} \cdot r_1^{-1} = (y \cdot r_1)^{-1}$		
	$[x \cdot y^{-1} \cdot r_1^{-1} \cdot r_2] = [x \cdot r_2]^{y^{-1} \cdot r_1^{-1}}$		
		$\xrightarrow{[x \cdot y^{-1} \cdot r_1^{-1} \cdot r_2]}$	
3.			$[x \cdot y^{-1}] = [x \cdot y^{-1} \cdot r_1^{-1} \cdot r_2]^{\frac{r_1}{r_2}}$
		$\xleftarrow{[x \cdot y^{-1}]}$	
4.	decrypt: $x \cdot y^{-1}$		
	compute: x'/y'		
	(x'/y')		

Figure 3: Secure DIV Sub Protocol

are $v_{x,i}$ $(1 \leqslant i \leqslant m)$, where $v_{x,i} = 0$ means that the customer has not rated i. We assume that company A creates a Paillier key pair by running KeyGen.

Scaling, Rounding, and Inner Product

The Paillier cryptosystem deals with encryption/decryption of integers, however, in the recommender system we work with non-integer values. Therefore, in the rest of this chapter, we assume that the values of $c_{x,i}$ and $c_{y,i}$, for all x, y, i, have been scaled by 100 and rounded to integers. The scaling value of 100 gives enough precision to compute recommendations correctly, while limiting additional overhead. In addition, when computation is done with respect to Equation (5), we assume company A and company B have already scaled the values $v_{y,i} - \bar{v}_y$ by 100 and rounded the results, for all y, i.

For our recommender algorithm, the basic operation required is an inner product between two vectors, say C_x and C_y, with different data owners. Given an encrypted vector $[C_x]$ (meaning each element of the vector is encrypted $[c_{x,i}]$) and an unencrypted vector C_y, anyone can compute the encrypted similarity $[sim_{x,y}]$ following Equation (2):

$$[sim_{x,y}] = [\sum_{i=1}^{m} c_{x,i} c_{y,i}] = \prod_{i=1}^{m} [c_{x,i} c_{y,i}] = \prod_{i=1}^{m} [c_{x,i}]^{c_{y,i}} \tag{6}$$

Privacy-Preserving Recommendation Generation

If customer x requires recommendations, company A and company B engage in the protocol shown in Figure 4.

In more detail, the protocol is detailed as follows.

1. Company A computes the Pearson correlations, namely $sim_{x,y}$ $(1 \leqslant y \leqslant n', y \neq x)$, between customer x and all other customers in its own database. For $1 \leqslant i \leqslant m$, company A computes $[c_{x,i}]$, $[E^A_{x,i}]$ and $[D^A_{x,i}]$ according to Equations (3) and (5), and sends them to company B. Note that the encryption is done with PK.

2. Company B computes $[sim_{x,y}]$ following Equation (6) for $n'+1 \leqslant y \leqslant n$. Company B then runs the ABS sub protocol with company A to obtain $[|sim_{x,y}|]$ for $n'+1 \leqslant y \leqslant n$. Company B uses $[sim_{x,y}]$ and $[|sim_{x,y}|]$ $(n'+1 \leqslant y \leqslant n)$ to compute $[E^B_{x,i}]$ and $[D^B_{x,i}] = \prod_{y=n'+1}^{n} [|sim_{x,y}|]$ for $1 \leqslant i \leqslant m$ following Equation (5). Company B computes $[E_{x,i}] = [E^A_{x,i}] \cdot [E^B_{x,i}]$ and $[D_{x,i}] = [D^A_{x,i}] \cdot [D^B_{x,i}]$ for $1 \leqslant i \leqslant m$.

3. Company A and company B run the DIV protocol for company A to retrieve $pred'_{x,i}$ for $1 \leqslant i \leqslant m$. Company A then chooses the top X predicted items among the unrated ones, and sends them to customer x.

2.4 SECURITY ANALYSIS

We analyse the privacy properties of the protocols. The sub protocols in Section 2.3.3 are secure. Given that the COMP sub protocol is secure, we analyse the ABS sub protocol. Company A learns nothing about x because of the randomization resulted from b, r_1, r_2, r_3, r_4 and company B learns nothing about x because everything is encrypted under company A's public key. In particular, we let $r_1 \in_r \mathbb{Z}_{2^{200}}$, so that r_1 can statistically hide x from company A. Intuitively, in the DIV sub protocol, company A learns nothing about x, y due to the randomization resulted from r_1, r_2 and company B learns nothing about x, y because everything is encrypted under company A's public key.

Based on the security of sub protocols in Section 2.3.3, the recommendation algorithm in Section 2.3.4 is secure with respect to the security model in Section 2.2.2. Given the security of the sub protocols, the algorithm is secure for company A because everything sent to company B is encrypted under the public key PK. Similarly, the algorithm is secure for company B based on the security of the ABS and DIV sub protocols. Here, we have a minor note on using DIV sub protocol in the recommendation algorithm in Section 2.3.4. If $D_{x,i} = 0$ for any $1 \leqslant i \leqslant m$, then the DIV protocol will not work. Note the fact that $D_{x,i} = 0$ means that the similarities $|sim_{x,y}| = 0$ for all $1 \leqslant y \leqslant n$ that rated i, which can be assumed to be negligible due to the randomness

	Company A		Company B
	$(x, V_y, 1 \leqslant y \leqslant n')$		$(V_y, n'+1 \leqslant y \leqslant n)$
	(PK, SK)		(PK)
1.	compute: $sim_{x,y}, 1 \leqslant y \leqslant n'$		
	$\forall i : 1 \leqslant i \leqslant m;$		
	compute: $c_{x,i}$		
	compute: $E^A_{x,i}$		
	compute: $D^A_{x,i}$		
	encrypt: $[c_{x,i}]$		
	encrypt: $[E^A_{x,i}]$		
	encrypt: $[D^A_{x,i}]$		
		$\xrightarrow{[c_{x,i}],[E^A_{x,i}],[D^A_{x,i}],1\leqslant i\leqslant m}$	
2.			$\forall y : n'+1 \leqslant y \leqslant n;$
			compute: $c_{y,i}$
			$[sim_{x,y}] = \prod_{i=1}^{m} [c_{x,i}]^{c_{y,i}}$
		$\forall y : n'+1 \leqslant y \leqslant n;$	
		$\rightarrow [\lvert sim_{x,y}\rvert] = \mathsf{ABS}([sim_{x,y}])$	$\xleftarrow{[sim_{x,y}]}$ $\xrightarrow{[\lvert sim_{x,y}\rvert]}$
			$\forall i : 1 \leqslant i \leqslant m;$
			$[E^B_{x,i}] = \prod_{y=n'+1}^{n} [sim_{x,y}]^{v_{y,i}-\bar{v}_y}$
			$[D^B_{x,i}] = \prod_{y=n'+1}^{n} [\lvert sim_{x,y}\rvert]$
			$[E_{x,i}] = [E^A_{x,i}] \cdot [E^B_{x,i}]$
			$[D_{x,i}] = [D^A_{x,i}] \cdot [D^B_{x,i}]$
3.		$\forall i : 1 \leqslant i \leqslant m;$	
	$\rightarrow$ $\xleftarrow{pred'_{x,i}}$	$pred'_{x,i} = \mathsf{DIV}([E_{x,i}], [D_{x,i}])$	$\xleftarrow{[E_{x,i}],[D_{x,i}]}$
	$(pred'_{x,i}, 1 \leqslant i \leqslant m)$		

Figure 4: Collaborative Filtering in Two-Company Setting

in customers' ratings and the size of customer population. Our implementation in Section 2.5 partially validates this assumption. Should this assumption be untrue, company A would be unable to generate a prediction for item i. But, this would also happen in the unsecured version of the protocol.

2.5 PERFORMANCE ANALYSIS

We have created a prototype implementation in C++. The prototype uses the GNU Multiple-Precision (GMP) library and consist of roughly 750 lines of code. To test this prototype, we use the MovieLens 1M dataset (taken from http://grouplens.org/), which contains 1 million ratings for 3900 movies by 6040 users. The ratings are on an integer scale from 1 to 5. We split the rating dataset in two parts by randomly selecting users as either a customer of company A or company B. We set the bit-length of the Paillier modulus N to 1024. All tests are carried out on an Intel Xeon at 3 GHz, with 2 GB of RAM.

2.5.1 *Computation Cost*

Referring to the proposed protocol, the computational complexity is related to both the number of customers of company B and number of items. Theoretically, the computational complexities is $O(m(n-n'))$ for both companies. To obtain concrete numbers for running time, we investigate two cases, in which the total number of items is fixed.

Case 1

In this case, we want to investigate the running time with respect to the total customer population. We take a fixed population distribution as example, where company A has 20% of the total population and company B has 80% of the total population.

We compute the running time values for ten different total populations, namely $604 \times i$ for $1 \leqslant i \leqslant 10$. The running time figures are shown in Figure 5, where the x-axis denotes the total customer population and the y-axis denotes the running time. The solid line indicates the total running time for company A and company B, while the dashed line indicates only the running time for company A. As expected, the graph shows a linear relation between the number of customers and the running time of the algorithm. The running time for both company A and B individually increases linearly with the customer population.

When the total population is 6040 (the full dataset), the running time for the two companies is 354 seconds. Which is not efficient enough for practice, we will discuss how to improve the efficiency later, however the customer is not involved and thus the recommendations can be precomputed. Thus for the customer, getting a recommendation

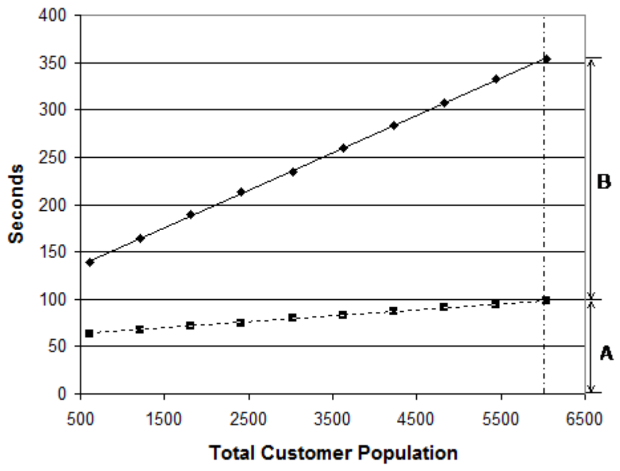


Figure 5: Running Time w.r.t. Total Population

only takes the amount of time it takes company A to lookup the pre-computed recommendations.

Case 2

In this case, we want to investigate the running time with respect to the population distribution between company A and company B. The total population is 6040.

We compute the running time values for eight different population proportions for company A, namely 1%, 2%, 5%, 10%, 20%, 30%, 40%, 50%. The running time figures are shown in Figure 6, where the x-axis denotes the population distribution and the y-axis denotes the running time. Again, the solid line indicates the total running time and the dashed line indicates the running time for company A. The dashed vertical line shows the intersection with Figure 5. In particular, when company A only has 1% of the population the running time is 414 seconds, when company A is given 50% of the population the running time is 270 seconds. As stated in Case 1, the running time for company A and B increases linearly with the customer population of company B. Note that when company A has less customers, company B also has more customers due to the way the dataset is split. As expected, we see a linear relationship between the running time and the distribution of the customers.

2.5.2 *Communication Cost*

The communication cost between company A and company B is proportional to the number of items and the number of customers held by company B. We give an estimate of the communication cost for the

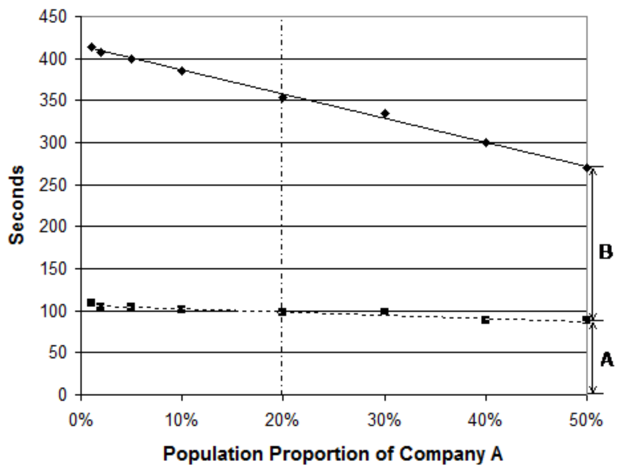


Figure 6: Running Time w.r.t. Population Distribution

scenario where the customer population is split between company A and B at a 20%
80% ratio. Given the composite number N of 1024 bits, an encryption, which is essentially a number modulo N^2, has a maximum size of 2048 bits. When adding everything up this results in a transmission of 15.2 MB of data.

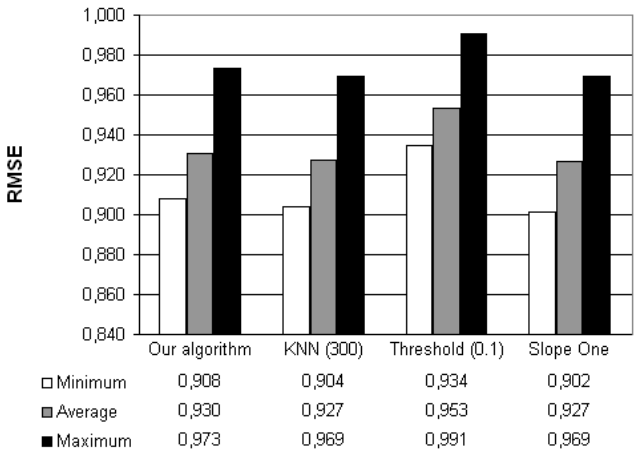
2.5.3 *Recommendation Accuracy*

To evaluate the overall accuracy property of the protocol on the whole customer population (namely, 6040 customers), we randomly select 10% of the 6040 customers, denoted as $\mathcal{S}$, and remove 5 ratings for each of them. The removed ratings are used to compare against their predictions to determine the accuracy. We use root mean squared error (RMSE) as the accuracy measure:

$$\mathrm{RMSE} = \sqrt{\frac{1}{t} \sum_{x \in \mathcal{S}, 1 \leqslant i \leqslant m} (\mathrm{pred}_{x,i} - v_{x,i})^2}, \tag{7}$$

where t is the total number of predictions. Note that in our case, the prediction formula $\mathrm{pred}'_{x,i}$ is only used to predict the ordering of items, we use the formula for $\mathrm{pred}_{x,i}$, defined in Equation (4), to normalize back to predictions of actual ratings. This formula will give the same ordering results and meaningful accuracy figures. The computation shows an average RMSE of 0.930. This is compared against k-nearest neighbours, threshold neighbourhood, and slope one prediction in Figure 7. The results show that our algorithm is comparable to these established recommendation methods.

To evaluate the recommendation accuracy gained by company A, we consider eight cases, where the population proportion of company



	Our algorithm	KNN (300)	Threshold (0.1)	Slope One
□ Minimum	0,908	0,904	0,934	0,902
▨ Average	0,930	0,927	0,953	0,927
■ Maximum	0,973	0,969	0,991	0,969

Figure 7: Accuracy Comparison with Established Algorithms

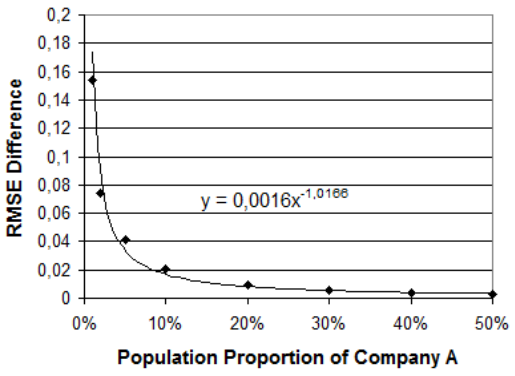


Figure 8: Accuracy Change w.r.t. Population Distribution

A in the whole population are 1%, 2%, 5%, 10%, 20%, 30%, 40%, 50% respectively. When the proportions are 1%, 2%, 5%, 10%, we let company A's customers be from the set $\mathcal{S}$. When the proportions are 20%, 30%, 40%, 50%, we let company A's customers consist of all customers from the set $\mathcal{S}$ plus customers from the rest of the whole population. For every case, we compute the RMSE value for company A, where the computation is only based on company A's customer data, and compute a difference by subtracting the RMSE value based on the data of the whole population. The RMSE differences of all eight cases are shown in Figure 8, where the x-axis denotes the population distribution and the y-axis denotes the RMSE difference. From the figure, the accuracy difference decreases when company A's population proportion increases. This implies that the accuracy gain becomes less when company A has more customers.

2.5.4 *Improving the Efficiency*

We foresee two ways to reduce the computation costs of the proposed algorithm. Instead of using the data from all customers, company B can use those from a subset of its customers. For example, those customers that have rated a lot of items. By using a subset of the customers, company B has to compute fewer similarities between customers and has fewer entries to compute the prediction with. When company B uses a dense subset of the customers, the combined dataset will also have a higher density. In this case, we will expect the recommendation accuracy, compared to the accuracy computed from the full set of B's customers, will show only a small difference. As to the computational efficiency, both companies will be more efficient because $[sim_{x,y}]$ and $[|sim_{x,y}|]$, for any y not in the subset, will not be computed. In summary, the performance will depend on how to choose the subset by company B, and we leave it as a future work to perform further investigation.

In the proposed algorithm, if company A discloses the items that customer x has rated, then the computation becomes less complex. This is reasonable because it does not make sense to make a prediction for an already rated item. Note that company A does not disclose the ratings for the rated items. In more details, for any item i which has been rated by the customer, the values $[E_{x,i}]$, $[D_{x,i}]$, their components, and their division $\mathtt{pred}_{x,i}$ do not have to be computed. Furthermore, for company B, computing $[sim_{x,y}]$ can be done faster. In summary, company A can sacrifice a bit of the privacy of its customer for a better computational performance.

2.6 RELATED WORK

In the literature, the most relevant work to ours is that of Basu et al. [16, 17] and that of Polat and Du [75]. Basu et al. proposed a privacy-preserving version of the slope one predictor. They pre-compute the deviation and cardinality matrices under encryption and make the cardinality matrix public. Then the prediction for a single item can be computed under encryption and all parties collaborate to decrypt the result. Making the cardinality matrix public in the case of two parties will leak information. Furthermore, their timing information is based on a single prediction for a single customer and item. When predicting the top X recommendations, this timing information has to be increased proportional to the number of items in the database (predictions can be computed in parallel). The setting for Polat and Du [75] is slightly different: a customer, who is not a member of either company, wants company A and B to compute recommendations for him/her. This customer plays an active role in the protocol and privacy is based on randomizing values, rather than encryption. Another difference in their work is that a rating is either 0 or 1, which makes protocol design easier than our case.

Other privacy-preserving recommender algorithms focus on the privacy of individual customers. Aïmeur et al. [13] provided a framework where customer data are separately stored over two parties, where an agent has access to ratings and the company has access to the items so that they together can generate recommendations for customers. Polat and Du [74] proposed a singular value decomposition predictor based on random perturbation of data. They go on to show the impact on privacy and accuracy, and their inherent trade-off due to perturbation. Berkovsky et al. [18] proposed to combine random perturbation with a peer-to-peer structure to create a form of dynamic random perturbation. For each request, the customer can decided what data to reveal and the amount of protection that is put on the data. Different perturbation strategies are compared based on accuracy and perceived privacy. The requirement for a peer-to-peer structure makes this approach less suitable for our scenario, where only two parties are involved. McSherry and Mironov [65] proposed collaborative filtering algorithms in the differential privacy framework. Similar to other perturbation and anonymization based approaches, this approach still has a trade-off between privacy and accuracy. In our approach, we preserve privacy without decreasing accuracy.

Canny [28, 27] uses homomorphic encryption to privately compute intermediate values of the collaborative filtering process. These intermediate values are made public and used in singular value decomposition and factor analysis, which leads to recommendations. However, because the intermediate values are made public, this leaks a lot of information about the customers when all data is held by only two parties (as is our case). Erkin et al. [37] proposed a collaborative filtering algorithm based on homomorphic cryptosystems. This algorithm requires every customer to take part in the protocol execution in order to compute recommendations for a single customer, and this makes the solution unscalable in practice.

There are two approaches to achieve privacy-preserving data mining, which is similar in nature to recommender systems. One is perturbation and anonymization based following the work of Agrawal and Srikant [12], and the other is cryptography-based following the work of Lindell and Pinkas [60]. In dealing with a large data set, the perturbation and anonymization based approach is generally efficient and flexible, however this approach usually does not provide rigorous security guarantees. For example, Narayanan and Shmatikov [69] have demonstrated serious de-anonymization attacks against the Netflix Prize dataset. Recently, McSherry and Mironov [65] applied the concept of differential privacy to recommender systems. With differential privacy they achieved rigorous security. However, this approach in general may reduce the computation accuracy as it will modify the original data. In contrast, the cryptography-based approach can provide rigorous security guarantees and will not affect computation accuracy, but it is often too complex to be practical. Our work demonstrates that, for (at least some) recommender algorithms, the cryptographic approach can be feasible, which means both efficiency and rigorous privacy protection can be achieved at the same time.

2.7 CONCLUSION

We have proposed a privacy-preserving collaborative filtering algorithm for companies to compute recommendations based on a joint set of customer databases. Based on the experimental results from a prototype implementation, we have shown that an individual company can generate more accurate recommendations. The performance analysis shows that it takes about six minutes to computes recommendations for a customer using a PC. Notice that company A can pre-compute the encryptions and company B can perform most of the computations in a parallel manner. Therefore, the performance can be significantly improved in practical deployment, and the solution is in fact feasible. Furthermore, since the customer is not involved in the protocol, the companies can pre-compute a recommendation for the customer and present it immediately when requested.

The solution is limited in the fact that no privacy is offered for the customer against its chosen company, only against the other company. The solutions only considers a horizontal partitioning of the dataset, the scenario where the dataset is vertically (or even arbitrarily) partitioned is also interesting. This case seems to have more privacy concerns because we somehow need to link the records in different customer databases.

3 OFFLINE USERS

3.1 INTRODUCTION

Collaborative filtering recommender systems rely on a large database of information from a lot of different users. With such a database the systems then recommend content based on similarity (agreement in rating behaviour) between users. However, studies [43, 46, 58, 84] have shown that for taste related domains, such as movies and books, familiarity (social closeness between users) gives comparable accuracy to using similarity. Familiarity captures how well users know each other (and thus their preferences). Using familiarity instead of similarity removes the information need from unknown users, thus increasing privacy between users, as some trust already exists between friends. Users in recommender systems are not always available to share information and contribute to computations. As a benefit, this trust between friends can also be leveraged to facilitate information sharing when users are unavailable. Further, since no information from unknown users is needed, a recommender system based on familiarity also works on a smaller dataset, leading to a higher efficiency. In this chapter we focus on the generation of recommendations using only familiarity. We leave as future work, a recommender system that combines both similarity and familiarity.

As a pre-requisite for a familiarity-based recommender system, a familiarity network needs to be known to the recommendation provider. Since this familiarity information is already present in online social networks, we can leverage these networks to provide recommendations. Our aim is to build a recommendation system on top of existing social networks (utilizing the familiarity relationship that is present), while preventing the social network from learning the users' taste preferences (not giving the social network any information that it does not have already).

While the general tastes (and possibly some specific tastes) of friends are known, the exact details of a friend's complete taste are usually not known. Revealing a specific taste to friends can be embarrassing [64] as it does not conform to the group norm, or to the societal norm as a whole. For example, if all friends of a person dislike 'The Hunger Games', but that person loves the book, if the friends find out this could be embarrassing. As such, the privacy of the user with regards to their taste needs to be protected from both friends (specific taste) and the online social network (general and specific taste).

Our contribution is the following: First, we look at the privacy of the commonly used recommendation formula based on the weighted average of ratings [43], where the strength of the relationship determines the weighting factor. Using this formula, the ratings of close friends have a higher impact on the prediction, than the ratings of distant friends. We observe that weighted average based on user supplied weights does not provide enough privacy. Based on this, we propose an adjusted formula that offers more privacy. Second, utilizing this adjusted formula, we construct a secure protocol that computes the recommendation for a user, when all his friends are online. For privacy reasons, friends do not give their data to the online social network, therefore all friends have to be actively involved in the computation. However, users are not guaranteed to be online in a social network. Third, as users can be offline, we also construct a secure protocol where the users friends are offline, and the user works together with the social network server to compute the recommendations. Not having to wait for all friends to have been online to do their part in the protocol increases the efficiency of the solution.

To ensure the privacy of the users, we make use of secure multiparty computation and a somewhat homomorphic encryption scheme. The motivation for a somewhat homomorphic encryption scheme (we use [21]) is: 1) it allows us to do a (bounded) number of additions and at least one multiplication on encrypted data, and 2) the message space is pre-determined by public parameters and is the same across key pairs. The latter property allows for blinding values under one key and unblinding under another. In constructing our solution, next to privacy, we focus on the efficiency of the solution. Both protocols are secure, assuming honest-but-curious participants.

In this chapter, we will use books as our running example for recommendations. The chapter is structured as follows: Section 3.2 details the state of the art and related work. Section 3.3 gives the problem specification and details the adjusted recommendation formula. Section 3.4 outlines the cryptographic primitives that are used. Section 3.5 details the solution with online friends and the solution with offline friends. Section 3.6 analyses the solutions in terms of security and privacy. Section 3.7 analyses the complexity of the solutions and gives performance figures based on our prototype. And Section 3.8 gives concluding remarks with regard to the solutions.

3.2 RELATED WORK

In this section, we show related work in privacy-preserving recommender systems that protect privacy through the use of cryptography and multi-party computation. In 2002, Canny [28] proposed using additive homomorphic encryption to privately compute intermediate values of the collaborative filtering process. These intermediate values are made public and used in singular value decomposition and factor analysis, which leads to recommendations. However, the presented approach suffers from a heavy computational and communication overhead. Moreover, due to the nature of the used recommender system (singular value decomposition), users cannot input their familiarity information.

Hoens et al. [48] designed a privacy-preserving recommender system for social networks that computes the weighted average rating for items. It gathers input from friends and friends of friends and onwards by first defining a group of users involved in the computation. Then a threshold homomorphic cryptosystem is set up. This cryptosystem, together with multi-party computation, is used to compute the weighted average. The weights are defined by the user for his friends, and by the friends for the friends of friends, and so on. Privacy is achieved through both cryptographic protocols as well as anonymity through multiple participants. The downsides of this solution are the requirement that users are online, the setup of a big group in advance, and the heavy computational load in the order of hours for a recommendation of a single book for a single user. Hoens et al. [49] designed a private recommender system for doctors, where patient ratings are aggregated. In this scenario, there is not a predefined group of patients and no weights are given to individual ratings or patients. Hoens et al. offer two solutions, one based on anonymized ratings, and one based on cryptography and multi-party computation. Again, the timing (of the solution based on cryptography) for computing a single recommendation is in the order of hours.

Basu et al. [17] proposed a privacy-preserving version of the slope one predictor, using a threshold additive homomorphic cryptosystem. In their scenario, different parties hold different parts of the data. In a social network setting, this means that each friend holds his own data. The parties pre-compute the deviation and cardinality matrices under encryption and make the cardinality matrix public. Then the prediction for a single item can be computed under encryption and all parties collaborate to decrypt the result. Their timing information, in the order of seconds, is based on a prediction for a single user and single book. This is after pre-computation of the matrices, which is in the order of hours. There is no support for offline users, nor for familiarity due to the way predictions are computed.

Erkin et al. [38] proposed a collaborative filtering algorithm based on additive homomorphic cryptosystems. This algorithm requires a second semi-trusted server to allow for users to be offline. However, in practical scenarios such a server is usually not available. The protocol of Erkin et al. does not give weights to the ratings. The runtime to compute the recommendations for all books for a single user is in the order of minutes for a dataset of 1000 books and several thousand (variable) users.

3.3 PROBLEM SPECIFICATION

We consider the following problem scenario: A user has just finished reading a book and has to decide which book to read next. He turns to his online social network of choice, but none of his friends are online. With nobody to ask, he consults the book recommender system of the online social network (which bases the recommendation on the information from his friends). Knowing that his taste information remains protected, the user also inserts a rating for the book he just finished ('Breaking Dawn'). The following subsections go into more detail about the entities and their relationship, the suggested method of using the taste information, and what constitutes a breach of privacy.

3.3.1 *Architecture*

The system consists of three entities:

1. the user, for whom a prediction has to be generated,
2. the online social network, also denoted as the server, acting as a gateway to access the user's friends and assisting in the prediction computation, and
3. the friends of the user, giving their opinions as input for the book predictions.

Because of the nature of online social networks, not all friends will be online when the request for book recommendations is made. For privacy reasons the server does not have access to the taste information of friends and the user is unlikely to want to wait until all friends have come online. Therefore, the online social network acts as a gateway for the information of the user's friends (while not learning information about the friends' preferences) and an assistant in the computation (to prevent the user from learning the friends' preferences). As such, we distinguish two scenarios; book recommendation when the user's friends are all online, and book recommendation when the user's friends are all offline. It is also possible that some friends of the user are online, while some are offline. For simplicity we take this third scenario to be equal to book recommendation when the user's friends are all offline.

3.3.2 *Recommendation Formula*

Before predictions can be made, the familiarity between users has to be captured. Towards this end, the user can score his friends on their familiarity (social closeness) and the expected overlap in reading habits. Each user will have to give one score to each of his friends. This is something that is currently unavailable in online social networks (only the fact that they are friends is available). It is possible to infer an approximation of the strength of a relationship based on profiles and interaction (e. g. [88]). However, we believe that the user is in the best position to determine his own scores.

Scoring a friend essentially gives that friend a weight that determines how heavy his opinion counts towards a specific book recommendation. Based on the friends' ratings for books and the weight for each friend, the recommender system predicts a score for each book. This helps the user to select the next book to read.

A book prediction is denoted by $p_{u,b}$, for user $u, 1 \leqslant u \leqslant U$, of book $b, 1 \leqslant b \leqslant B$, where U is the total number of users and B is the total number of books. The recommendation formula is as follows:

$$p_{u,b} = \frac{\sum_{f=1}^{F_u} q_{f,b} \cdot r_{f,b} \cdot w_{u,f}}{\sum_{f=1}^{F_u} q_{f,b} \cdot w_{u,f}}, \tag{8}$$

where F_u is the number of friends of a user u, $q_{f,b}$ is 1 if friend f rated book b and 0 otherwise, $r_{f,b}$ the rating of friend f for book b, and $w_{u,f}$ the weight given by the user u to friend f. The indication, $q_{f,b}$, if a book b has been rated a friend f is either 0 or 1, $q_{f,b} \in \{0, 1\}$. The range of the prediction, $p_{u,b}$, is equal to the range of the ratings given to a book, $r_{f,b}$. For example, this range can be between 0 and 5 for a 0 to 5 star rating system. The weight given to a friend, $w_{u,f}$, can be in the range between 0 and 1 excluding 0, as 0 would indicate no friendship. This formula has been used in previous research in similarity-based [47], familiarity-based [43] and trust-based recommendation systems [86].

However, when looking at the inherent privacy this formula can give us, we notice two things:

1. Due to the fact that the user u learns the predictions $p_{u,b}$ and determines the weights $w_{u,f}$, with two prediction requests the user can learn which books are rated by one friend, i. e. learn $q_{f,b}$. This is accomplished by changing the weight $w_{u,f}$ for that specific friend. For example, suppose that the user has three friends who have rated two books. The first friend rated the first book with a 5, the second friend rated both books with a 4, and the third friend rated the second book with a 3. When the user request a prediction with all weights set to 1, he will receive a prediction of 4.5 for the first book and 3.5 for the second book. Next, the user requests a prediction with the weights of the first and second friend set to 1, and the weight of the third friend set to 0.5. He will receive a prediction of 4.5 for the first book and 3.67 for the second book, thus he learns that the third friend rated the second book. Given enough runs, the user can learn $q_{f,b}$ for all his friends.

2. Because the user knows $p_{u,b}$, $w_{u,f}$, and $q_{f,b}$, the only unknown values are that of $r_{f,b}$. Given enough predictions with different

weights, the user gets more equalities and can also compute $r_{f,b}$. Then the user knows $p_{u,b}$, $w_{u,f}$, $q_{f,b}$, and $r_{f,b}$ and has completely breached the privacy of his friends.

Consequently, when using this formula, we cannot achieve privacy at all. Intuitively, the user has full control, and the friends have no input beyond their fixed ratings. This asymmetry in the formula leads to an asymmetrical relationship between the user and his friends. As stated by Carley and Krackhardt [29], friendship is not necessarily symmetric, but tends in the direction of symmetry. In general, long strong friendships are symmetric, and newly forged friendships are not symmetric. As such, we aim to bring symmetry to the recommendation formula and balance out the power in the relationship between the user and his friends.

Since the weight from the user to his friends is asymmetrical, we propose to make the weight, and thus the formula, symmetrical. This is accomplished by taking the average of the weight from the user to his friend and the weight from the friend to the user. This results in the following formula:

$$p_{u,b} = \frac{\sum_{f=1}^{F_u} q_{f,b} \cdot r_{f,b} \cdot (\frac{w_{u,f}+w_{f,u}}{2})}{\sum_{f=1}^{F_u} q_{f,b} \cdot (\frac{w_{u,f}+w_{f,u}}{2})}, \tag{9}$$

where $w_{f,u}$ is the weight given by friend f to user u, with range between 0 and 1 excluding 0. Since the weight from the friend to the user is required to compute recommendations for that friend, no new information has to be added compared to the previous formula. Note that this also requires a bi-directional relationship between the friends and that both friends have specified a weight for one another. When looking back to the two points made before in light of this adjusted formula, we can say:

1. Since the user can still change the weights that are given to his friends $w_{u,f}$, the user can influence the averaged weight, $\frac{w_{u,f}+w_{f,u}}{2}$. Based on the changed weights and change in predictions, the user can still determine $q_{f,b}$ as before.

2. When the user knows $p_{u,b}$, $w_{u,f}$, and $q_{f,b}$, the values for $r_{f,b}$ and $w_{f,u}$ remain unknown. The fact that both the upper and lower part of the prediction formula remain unknown increases the difficulty of breaching privacy.

To prevent the user from learning $q_{f,b}$, the user's influence on the weight can be removed. However, then this recommender system would lose the user's control and reduce the value of the predictions. Instead, we refer to profile aggregation methods [83], methods that add random ratings [34, 63], or methods that add randomness to the output [65]. These solutions can be applied *independent* of our solution and will not be addressed in this chapter.

Note that the impact on accuracy of this adjusted formula (from asymmetrical to symmetrical) has not been determined. Collaborative filtering algorithms typically rely on the Pearson correlation for similarity [47], which is also a symmetrical correlation. Furthermore, friendship tends to be in the direction of symmetry [29], therefore we assume the impact to be minimal. As this chapter focusses on privacy and efficiency, and a suitable dataset (with fine-grained familiarity information) to test accuracy could not be found, we leave it as future work to validate this assumption.

Further note that, in this chapter, we assume that the user supplies the weights. Should the weights be inferred by the server and remain unknown to the users, the first recommendation formula (asymmetric) can still be used. Our protocols can be adjusted accordingly without major changes.

3.3.3 *Security Model*

Both the user and his friends are considered to be honest-but-curious; they will follow the protocol but try to learn the taste of their friends. More specifically, the user u will try to learn $r_{f,b}$ and $w_{f,u}$, while the friends of u will try to learn $w_{u,f}$.

We also assume that the social network server is honest-but-curious; the server will follow the protocol, while trying to learn the tastes of users. The server will try to learn $q_{f,b}$, $r_{f,b}$, $w_{u,f}$, $w_{f,u}$, and $p_{u,b}$. We assume that the users do not collude with the server, as they do not want to impact the privacy of their friends.

3.4 CRYPTOGRAPHIC PRIMITIVES

To build our solutions, we make use of the cryptographic primitives described in this section. The primitives of additive secret sharing and proxy re-encryption are only used in the solution with offline friends.

3.4.1 *Somewhat Homomorphic Encryption*

To protect information during the protocol, we use the somewhat homomorphic encryption scheme of Brakerski and Vaikuntanathan [21]. Specifically, we use the fact that this somewhat homomorphic encryption scheme allows both addition and multiplication of the encrypted messages (though a limited, but configurable amount), and the fact that the message space is the same across multiple key pairs (given the same public parameters).

In the setup phase of the encryption system, the public parameters are chosen. Among others, these are: the message space (which equals $\mathbb{Z}_t$ for some prime number t), the encrypted messages (which are represented in the ring $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ of polynomials over $\mathbb{Z}_q$ for some prime number q, where the polynomial $f(x)$ is cyclotomic and of degree n), and the degree D of allowed homomorphism (which indicates the amount of multiplications that can occur under encryption). The choice of the ring R_q in relation to the prime t and degree of homomorphism D defines the security of the encryption system.

Each party can, based on these public parameters, create a key pair consisting of the secret key SK and the public key PK. The secret key is randomly chosen and the public key is based on the secret key and some randomness. The public key of user u is denoted by PK_u. Given an encryption of m under the public key PK_u, denoted by $[m]_u$, the following homomorphic properties hold (until the error overflows, typically when the degree D has been reached): $[m_1]_u + m_2 = [m_1 + m_2]_u$, $[m_1]_u + [m_2]_u = [m_1 + m_2]_u$, $[m_1]_u \cdot m_2 = [m_1 \cdot m_2]_u$, $[m_1]_u \cdot [m_2]_u = [m_1 \cdot m_2]_u$.

This scheme is semantically secure under the polynomial learning with errors assumption. For more details, we refer to the work of Brakerski and Vaikuntanathan [21].

3.4.2 *Encrypted Division*

Because the homomorphic encryption system can only encrypt integers, and thus only operate on integers, division of encrypted values is not straightforward. For example $[5]/[2] \neq [2.5]$ as $[2.5]$ cannot be represented as such. Given that the message space $\mathbb{Z}_t$ is known and the range of the predictions $p_{u,b}$ is also known and significantly smaller, a lookup table can be constructed (and precomputed) to quickly translate the integers after division into the actual fractions they represent. The lookup table looks like this: given two integers x and y, with $\gcd(x, y) = 1$ and x/y as a possible result for $p_{u,b}$, the index is $x \cdot y^{-1} \mod t$ and the resulting value x/y. For integers x' and y' with $\gcd(x', y') \neq 1$, the division result is the same as for $x = x'/\gcd(x', y')$ and $y = y'/\gcd(x', y')$. We denote the set of possible integers for x, X, the set of possible integers for y, Y, and the range of possible predictions $p_{u,b}$, P. The lookup table then has size $|\{x/y \mid \gcd(x, y) = 1, x/y \in P, x \in X, y \in Y\}|$. The size of the lookup table is upper bounded by the size of the message space $\mathbb{Z}_t$. As such, division can happen under encryption and after decryption a table lookup retrieves the actual result.

3.4.3 *Additive Secret Sharing*

An alternate method to protect information from multiple parties, while still providing operations on that information, is additive secret sharing [42]. Unlike encryption, where only the party with the key can decrypt it, anybody with enough shares can extract the information. Distribution of the shares prevents extraction of the information, but still allows us to run a protocol to use the information. When a party has a value x that it wants to protect, it creates a random value $r \in_R \mathbb{Z}_k$, where k is the modulus based on a security parameter. The party then creates $s = x - r$. It can give r to a second party, and s to a third. Together the second and third party can reconstruct x by $x = r + s$.

It is also possible to secret share a vector of values, X, of length n. The secret sharing algorithm is then applied to each element of X individually, resulting in the two vectors R and S, both of length n. When combined the vectors R and S sum up to the vector X, $x_i = r_i + s_i$, where $1 \leqslant i \leqslant n$.

3.4.4 *Proxy Re-encryption*

To share information between two friends in the social network, we use proxy re-encryption [19]. Proxy re-encryption allows us to send a (secret) message from one user to his friends through the social network. In proxy re-encryption, based on the keys of two users a re-encryption key can be derived. This re-encryption key is then given to the proxy (the social network server). When given a message encrypted under the key of one user, using the re-encryption key the proxy can translate the message, to a message encrypted under the key of the second user. This way an offline user can store his information on the social network encrypted under his own key. When a friend requires access to that information, the server can translate the information to be encrypted under the key of the friend (provided a re-encryption key has been setup). The friend can then decrypt and use the information left by the offline user. Alternatively, a user could encrypt his information for each of his friends separately. However, this increases the workload of the user by a factor of the number of friends.

We require that the re-encryption scheme is unidirectional.In a unidirectional scheme the users do not have to share their private keys to create a re-encryption key. To create a re-encryption key from the user to a friend, only the user's private key and the friend's public key are needed. We further require that the re-encryption scheme is one-hop only, so that only friends of the user can read his information. Some examples of schemes that satisfy these requirements are: Ateniese et al. [15], Libert and Vergnaud [59], and Chow et al. [32]. The proxy re-encryption scheme can be chosen independent of our protocol and is only used to give the friends' information to the user beforehand.

3.5 PROPOSED SOLUTIONS

In this section we provide the details of the protocols to compute the book recommendations. A protocol is given when all friends are online, and a protocol is given when all friends are offline. For convenience, we make some small cosmetic alterations to the prediction formula 9.

We set the value of $r_{f,b}$ *to 0 when* $q_{f,b} = 0$*, thus* $r_{f,b}$ *becomes equal to* $q_{f,b} \cdot r_{f,b}$*. We also divide* $w_{u,f}$ *and* $w_{f,u}$ *by 2 before running the protocols (without renaming), remove the need to divide by 2 during the protocol. We further assume that* $r_{f,b}$*,* $w_{u,f}$*, and* $w_{f,u}$ *are integer values and if needed scaled to preserve precision.*

3.5.1 *Solution with Online Friends*

Figure 9 shows the recommendation protocol for user u with online friends. We assume that, before the protocol is run, the user u has set up his keys for the somewhat homomorphic encryption scheme, $\{PK_u, SK_u\}$, and distributed the public key. The protocol works as follows:

1. Each friend f of the user u computes their weight $w_{u,f} + w_{f,u}$. To do this, the user u encrypts $w_{u,f}$ for each friend under his own key, and sends $[w_{u,f}]_u$ to the corresponding friend f. The friends compute $[w_{u,f} + w_{f,u}]_u = [w_{u,f}]_u + w_{f,u}$.

2. Given the encrypted weight, each friend computes the impact of his ratings, $(w_{u,f} + w_{f,u}) \cdot r_{f,b}$, for each book. Recall that $r_{f,b} = 0$, when the book is unrated. The friends compute $[n_{f,b}]_u = [w_{u,f} + w_{f,u}]_u \cdot r_{f,b}$, and send $[n_{f,b}]_u$ to the server. The server sums the values received by the friends into $[n_b]_u = \sum_{f=1}^{F_u} [n_{f,b}]_u$ for each book.

3. In similar fashion, the normalization factor d_b is computed. The friends compute $[d_{f,b}]_u = [w_{u,f} + w_{f,u}]_u \cdot q_{f,b}$, and send $[d_{f,b}]_u$ to the server. The server sums the values received by the friends into $[d_b]_u = \sum_{f=1}^{F_u} [d_{f,b}]_u$ for each book.

4. To compute the predictions $p_{u,b}$, a division has to be performed. Towards this end, the server selects random values ξ_b from the multiplicative domain of the message space $\mathbb{Z}_t^*$ and blinds d_b multiplicatively for each book, $[d_b \cdot \xi_b]_u = [d_b]_u \cdot \xi_b$. The resulting values $[d_b \cdot \xi_b]_u$ are sent to the user u. The user u decrypts to $d_b \cdot \xi_b$ and computes the inverse, $d_b^{-1} \cdot \xi_b^{-1}$, for each book. These inverses are encrypted again under the users key, $[d_b^{-1} \cdot \xi_b^{-1}]_u$, and sent to the server. The server removes the blinding by multiplying with the random values ξ_b again, $[d_b^{-1}]_u = [d_b^{-1} \cdot \xi_b^{-1}]_u \cdot \xi_b$. The server then divides n_b by d_b for each book to determine the predictions, $[p_{u,b}]_u = [n_b]_u \cdot [d_b^{-1}]_u$. The encrypted predictions are then sent to the user u.

5. The user u decrypts the received predictions and uses the precomputed division lookup table to determine the actual predictions.

	User u	Server	Friends F_u
	(PK_u, SK_u)	(PK_u)	(PK_u)
	$(w_{u,f}, 1 \leqslant f \leqslant F_u)$		$(R_f, Q_f, w_{f,u})$
	$\forall f : 1 \leqslant f \leqslant F_u;$		
1.	encrypt: $[w_{u,f}]_u$		
		$\xrightarrow{[w_{u,f}]_u}$	
			$[w_{u,f} + w_{f,u}]_u = [w_{u,f}]_u + w_{f,u}$
		$\forall b : 1 \leqslant b \leqslant B;$	
2.			$[n_{f,b}]_u = [w_{u,f} + w_{f,u}]_u \cdot r_{f,b}$
			$\xleftarrow{[n_{f,b}]_u}$
		$[n_b]_u = \sum_{f=1}^{F_u} [n_{f,b}]_u$	
3.			$[d_{f,b}]_u = [w_{u,f} + w_{f,u}]_u \cdot q_{f,b}$
			$\xleftarrow{[d_{f,b}]_u}$
		$[d_b]_u = \sum_{f=1}^{F_u} [d_{f,b}]_u$	
4.		$\xi_b \in_r \mathbb{Z}_t^*$	
		$[d_b \cdot \xi_b]_u = [d_b]_u \cdot \xi_b$	
		$\xleftarrow{[d_b \cdot \xi_b]_u}$	
	decrypt: $d_b \cdot \xi_b$		
	$d_b^{-1} \cdot \xi_b^{-1} = (d_b \cdot \xi_b)^{-1}$		
	$[d_b^{-1} \cdot \xi_b^{-1}]_u$		
		$\xrightarrow{[d_b^{-1} \cdot \xi_b^{-1}]_u}$	
		$[d_b^{-1}]_u = [d_b^{-1} \cdot \xi_b^{-1}]_u \cdot \xi_b$	
		$[p_{u,b}]_u = [n_b]_u \cdot [d_b^{-1}]_u$	
		$\xleftarrow{[p_{u,b}]_u}$	
5.	decrypt: $p_{u,b}$		
	$(p_{u,b}, 1 \leqslant b \leqslant B)$		

Figure 9: Book Recommendation Protocol with Online Friends

3.5.2 *Solution with Offline Friends*

To cope with offline friends, we split the data of friends between the server and the user using secret sharing (thus reducing trust requirements). As the data is split neither the server nor the user has access to the friends data. Proxy re-encryption is used to give part of the data from the friend to the user. The re-encryption key from the friend to the user is used as a manifestation of their familiarity relationship (uni-directional from the friend to the user).

Usage of Secret Sharing and Proxy Re-encryption

Each friend f of the user secret shares the rating vector R_f and weight $w_{f,u}$. The rating vector R_f is split into the vectors S_f and T_f following the secret sharing method. Similarly, the weight $w_{f,u}$ is split into $x_{f,u}$ and $y_{f,u}$. As the secrets will be reconstructed under encryption, we set the modulus k of the secret sharing scheme equal to the message space t of the homomorphic encryption system. The friend stores S_f and $x_{f,u}$ on the server. The vectors T_f and Q_f as well as the value $y_{f,u}$ will be distributed to the user u using proxy re-encryption. Therefore, these values are stored under encryption at the server and the re-encryption key to the user u is computed and also stored on the server.

Protocol

Figure 10 shows the recommendation protocol for user u with offline friends. We assume that, before the protocol is run, the required secrets $T_f, Q_f, y_{f,u}, 1 \leqslant f \leqslant F_u$ have been distributed and that both the user u and the server have set up their keys for the somewhat homomorphic encryption scheme, $\{PK_u, SK_u\}$ and $\{PK_s, SK_s\}$ respectively, and exchanged public keys. The protocol works as follows:

1. Both user u and the server compute the weight, $w_{u,f} + w_{f,u}$, for each friend under one another's public key. The weight is computed by $w_{u,f} + w_{f,u} = w_{u,f} + y_{f,u} + x_{f,u}$, where u holds $w_{u,f}$ and $y_{f,u}$, and the server holds $x_{f,u}$. The user u computes $[w_{u,f} + y_{f,u}]_u$ and sends this to the server, while the server computes and sends $[x_{f,u}]_s$. This allows the user to compute $[w_{u,f} + w_{f,u}]_s$ and the server to compute $[w_{u,f} + w_{f,u}]_u$.

2. Given the encrypted weights, both the user u and the server can compute the impact of the secret shared ratings $r_{f,b} = t_{f,b} + s_{f,b}$ for each book. The user u computes $[z_b]_s = \sum_{f=1}^{F_u} [w_{u,f} + w_{f,u}]_s \cdot t_{f,b}$ and the server computes $[a_b]_u = \sum_{f=1}^{F_u} [w_{u,f} + w_{f,u}]_u \cdot s_{f,b}$. Together, this sums up (ignoring encryption for a moment) to $z_b + a_b = \sum_{f=1}^{F_u} (w_{u,f} + w_{f,u}) \cdot (t_{f,b} + s_{f,b}) = \sum_{f=1}^{F_u} (w_{u,f} + w_{f,u}) \cdot r_{f,b} = n_b$. The user u selects random values $\xi_{1,b}$ from the domain of message space $\mathbb{Z}_t$ and uses them to blind $[z_b]_s$. The resulting encryptions, $[z_b + \xi_{1,b}]_s$, and the encryptions to remove

	User u		Server
	(PK_u, SK_u, PK_s)		(PK_u, PK_s, SK_s)
	$(T_f, Q_f, w_{u,f}, y_{f,u}, 1 \leqslant f \leqslant F_u)$		$(S_f, x_{f,u}, 1 \leqslant f \leqslant F_u)$
		$\forall f : 1 \leqslant f \leqslant F_u;$	
1.	encrypt: $[w_{u,f} + y_{f,u}]_u$		encrypt: $[x_{f,u}]_s$
		$\xrightarrow{[w_{u,f}+y_{f,u}]_u}$	
		$\xleftarrow{[x_{f,u}]_s}$	
	$[w_{u,f} + w_{f,u}]_s = [x_{f,u}]_s + (w_{u,f} + y_{f,u})$		
			$[w_{u,f} + w_{f,u}]_u = [w_{u,f} + y_{f,u}]_u + x_{f,u}$
		$\forall b : 1 \leqslant b \leqslant B;$	
2.	$[z_b]_s = \sum_{f=1}^{F_u} [w_{u,f} + w_{f,u}]_s \cdot t_{f,b}$		
	$\xi_{1,b} \in_r \mathbb{Z}_t$		$[a_b]_u = \sum_{f=1}^{F_u} [w_{u,f} + w_{f,u}]_u \cdot s_{f,b}$
	$[z_b + \xi_{1,b}]_s = [z_b]_s + \xi_{1,b}$		
	encrypt: $[-\xi_{1,b}]_u$		
		$\xrightarrow{[z_b+\xi_{1,b}]_s, [-\xi_{1,b}]_u}$	
3.	$[d_b]_s = \sum_{f=1}^{F_u} [w_{u,f} + w_{f,u}]_s \cdot q_{f,b}$		decrypt: $z_b + \xi_{1,b}$
	$\xi_{2,b} \in_r \mathbb{Z}_t^*$		$[z_b]_u = [-\xi_{1,b}]_u + (z_b + \xi_{1,b})$
	$[d_b \cdot \xi_{2,b}]_s = [d_b]_s \cdot \xi_{2,b}$		$[n_b]_u = [z_b]_u + [a_b]_u$
	encrypt: $[\xi_{2,b}]_u$		
		$\xrightarrow{[d_b \cdot \xi_{2,b}]_s, [\xi_{2,b}]_u}$	
4.			decrypt: $d_b \cdot \xi_{2,b}$
			$d_b^{-1} \cdot \xi_{2,b}^{-1} = (d_b \cdot \xi_{2,b})^{-1}$
			$[d_b^{-1}]_u = [\xi_{2,b}]_u \cdot (d_b^{-1} \cdot \xi_{2,b}^{-1})$
			$[p_{u,b}]_u = [n_b]_u \cdot [d_b^{-1}]_u$
		$\xleftarrow{[p_{u,b}]_u}$	
5.	decrypt: $p_{u,b}$		
	$(p_{u,b}, 1 \leqslant b \leqslant B)$		

Figure 10: Book Recommendation Protocol with Offline Friends

the blinding, $[-\xi_{1,b}]_u$, are sent to the server. Note that the server can only remove the blinding using encryptions under the user's public key.

3. The user u computes the combined weight to normalize the prediction using $[d_b]_s = \sum_{f=1}^{F_u} [w_{u,f} + w_{f,u}]_s \cdot q_{f,b}$ for each book. These encrypted values are blinded multiplicatively with random values $\xi_{2,b}$, taken from the multiplicative domain of the message space $\mathbb{Z}_t^*$. The resulting encryptions, $[d_b \cdot \xi_{2,b}]_s$, and encryptions to remove the blinding after inversion, $[\xi_{2,b}]_u$, are sent to the server. Meanwhile, the server removes the blinding values $\xi_{1,b}$ and reconstructs $[n_b]_u = [z_b]_u + [a_b]_u$.

4. The server decrypts the received encryptions, $d_b \cdot \xi_{2,b}$, and inverts them, resulting in $d_b^{-1} \cdot \xi_{2,b}^{-1}$. Under the public key of u, the blinding values $\xi_{2,b}$ are removed, resulting in the encryptions $[d_b^{-1}]_u$. The server divides n_b by d_b under the public key of u, $[p_{u,b}]_u = [n_b]_u \cdot [d_b^{-1}]_u$, for each book. The resulting encrypted predictions $[p_{u,b}]_u$ are sent to the user u.

5. The user u decrypts the received predictions and uses the precomputed division lookup table to determine the actual predictions.

3.6 SECURITY ANALYSIS

In this section, we look at the security and privacy that the two protocols offer in relation to the security model. Recall from the security model that all parties are honest-but-curious. The user u will try to learn $r_{f,b}$ and $w_{f,u}$. Friends will try to learn $w_{u,f}$. The server will try to learn $q_{f,b}$, $r_{f,b}$, $w_{u,f}$, $w_{f,u}$, and $p_{u,b}$. Given that the parties are honest-but-curious, each party should not be able to distinguish between a protocol execution and a simulation of the protocol based only on the party's input and output. However, only the user u has an output in the protocol. As such, for the server and friends, each message they receive should be indistinguishable from random messages. For the user, messages may depend on the output $p_{u,b}$.

3.6.1 *Online Friends*

In this protocol, the user's friends only see encrypted values, encrypted under the key of the user u. Given that the homomorphic encryption scheme is semantically secure [21], the encrypted values are indistinguishable from encryptions of random messages. As the friends also get no output from the protocol, the protocol can easily be simulated and the friends learn nothing from the protocol.

The server also only sees encrypted values. As the homomorphic encryption scheme is semantically secure, the encrypted values are indistinguishable from encryptions of random messages. The server receives no output from the protocol, and the protocol can easily be simulated. Thus the server learns nothing from running the protocol.

After the user encrypts and sends $w_{u,f}$, the user only receives $d_b \cdot \xi_b$ and $p_{u,b}$ for all books. As $p_{u,b}$ is the output of the prediction formula 9, the user should always learn this and does not constitute a breach of privacy. The other value, d_b, is randomized multiplicatively over the full multiplicative domain by ξ_b, and is thus indistinguishable from a value chosen at random from the domain. Because this can also be easily simulated, the privacy of d_b is preserved. The only exception to this is when $d_b = 0$, in this case $d_b \cdot \xi_b$ is also equal to 0. This only happens when none of the users friends have given a rating for b, i. e. $q_{f,b} = 0$ for $1 \leqslant f \leqslant F_u$. This situation is deemed acceptable as $q_{f,b}$ is not required to be private. By setting $d_b^{-1} \cdot \xi_b^{-1}$ to 0, the protocol can continue without the server learning anything, resulting in the prediction $p_{u,b} = 0$.

3.6.2 *Offline Friends*

In the protocol with offline friends, the privacy of the user towards his friends is not in danger, as they are not involved in the protocol. In the other direction, each friend shares some information with both the user and the server. The user receives through the proxy re-encryption T_f, Q_f, and $y_{f,u}$, and the server receives S_f and $x_{f,u}$. Except for Q_f, all these values are additive secret shares and hence indistinguishable from random values [42]. This means that these values can be used as inputs to the protocol. Given that the proxy re-encryption scheme is secure, and Q_f is not required to be private from the user u, the privacy of each friend is not breached.

During the protocol, next to encrypted values, the user only receives $p_{u,b}$. As the homomorphic encryption scheme is semantically secure, the encrypted values are indistinguishable from encryptions of random messages. These messages can thus be simulated. Furthermore, the user receives $p_{u,b}$, as intended, as output of the prediction function. Thus from the user's perspective the protocol can be completely simulated.

Next to encrypted values, which are indistinguishable from encryptions of random values, the server only receives $z_b + \xi_{1,b}$ and $d_b \cdot \xi_{2,b}$. The value of z_b is protected by additive blinding, using $\xi_{1,b}$, and thus indistinguishable from a random value and possible to simulate. For d_b, as in the protocol with online friends, multiplicative blinding, using $\xi_{2,b}$, is used. Thus d_b is indistinguishable from a random value and can be simulated. Only in the case that $d_b = 0$, will the server learn something about $q_{f,b}$, which is a violation of the privacy of the user's friends. This can be avoided by setting $[d_b \cdot \xi_{2,b}]_s$ to $[\xi_{2,b}]_s$ and $[\xi_{2,b}]_u$ to $[0]_u$ when $d_b = 0$. This is only the case when $q_{f,b} = 0$, for $1 \leqslant f \leqslant F_u$, which the user knows. The server will receive $\xi_{2,b}$ instead

Table 1: Complexity of the protocol with online friends, F_u is the number of friends and B the number of books

	User u		Server		Friend	
step	comp	comm	comp	comm	comp	comm
1.	$O(F_u)$	$O(F_u)$			$O(1)$	$O(1)$
2.			$O(BF_u)$	$O(BF_u)$	$O(B)$	$O(B)$
3.			$O(BF_u)$	$O(BF_u)$	$O(B)$	$O(B)$
4.	$O(B)$	$O(B)$	$O(B)$	$O(B)$		
5.	$O(B)$					

of 0, which is a random value, and be unable to decrypt $[0]_u$ as it is protected by the user's key. The resulting prediction $p_{u,b}$ will then still be 0.

3.7 PERFORMANCE ANALYSIS

In this section we look at the complexity (computational and communicational) of the two protocols. Then, we look at the performance (runtime) of the protocols with different sized datasets based on our prototype implementation.

3.7.1 *Theoretical Complexity*

Table 1 shows the complexity of the computational (comp) and communicational (comm) costs of each step in the protocol with online friends. The costs are given in big-O notation and for each party. The first step shows a complexity related to the number of friends for the user u, and constant for each friend. The second and third step, where the friends contribution is calculated, shows a complexity in the order of number of books for each friend, and in the order of both the number of books and friends for the server. These steps have the largest complexity. The fourth step shows a complexity in the order of number of books for both the user and the server. The final step shows a complexity on the order of the number of books for the user. All steps together it seems that the server has the most work to do.

Table 2 shows the complexity of the protocol with offline friends. The notation is the same as the previous table. The first step shows a complexity in the order of number of friends for both the user u and the server. The second step shows a complexity related to both the number of books and number of friends for both the user and the server. This step has the greatest complexity in the protocol. The third step shows a complexity in the order of number of books and number of friends for the user, and a complexity in the order of number of

Table 2: Complexity of the protocol with offline friends, F_u is the number of friends and B the number of books

	User u		Server		Friend	
step	comp	comm	comp	comm	comp	comm
1.	$O(F_u)$	$O(F_u)$	$O(F_u)$	$O(F_u)$		
2.	$O(BF_u)$	$O(B)$	$O(BF_u)$	$O(B)$		
3.	$O(BF_u)$	$O(B)$	$O(B)$	$O(B)$		
4.		$O(B)$	$O(B)$	$O(B)$		
5.	$O(B)$					

books for the server. The fourth step shows a complexity in the order of the number of books. The final step shows a complexity in the order of number of books for the user.

The complexity of the homomorphic operations on the ciphertexts depends mainly on the degree of the used polynomials n. However, n also has an impact on the ring R_q and thus on the security of the encryption scheme. As such, there exist a trade-off between the complexity (and efficiency) of the individual homomorphic operations and the security offered to the user. In the performance section, we shall come back to this trade-off.

3.7.2 *Prototype Results*

To analyze the performance of the two protocols, an implementation of the somewhat homomorphic encryption scheme has been made in C++ based on the FLINT library. Based on this implementation a prototype program of the protocols has been constructed. The prototype consists of roughly 1050 lines of code. The prototype is single threaded and computes the different steps for each party sequentially on the same machine. As such, network latency is not taken into account. All tests are carried out on an Intel Xeon at 3 GHz, with 2 GB of RAM. As input data, a synthetic dataset has been constructed, as there are no publicly available datasets that have explicit fine-grained familiarity values. Some datasets have friendship links, but only as a binary value. The synthetic dataset consists of either 50, 100, or 200 friends (based on the number of average friends in Facebook) that have each rated 25 books. The total number of books is either 500, 1000, or 2000 (comparable to other datasets used in privacy-preserving recommender systems). Note that it is not possible for 50 friends to rate 2000 books, with only 25 ratings per friend (denoted with n/a). This gives us performance information for different numbers to observe how the solutions scale. A rating is a score between 1 and 100, and the weights between users, after division by 2, is between 1 and 50. Since the dataset is synthetic,

Table 3: Runtime of the prototype with attacker runtime logarithm of 255

online	books			offline	books		
friends	500	1000	2000	friends	500	1000	2000
50	113 s	236 s	n/a	50	132 s	282 s	n/a
100	149 s	309 s	706 s	100	182 s	387 s	1021 s
200	222 s	456 s	988 s	200	282 s	588 s	1477 s

no information can be derived about the accuracy of the recommender system.

We set the parameters of the somewhat homomorphic encryption scheme to the following, based on the suggestions of Naehrig et al. [67]. The message space t is set to 5000011, to allow for protocol runs with a maximum of 500 friends (500 friends times a maximum rating of 100 times a maximum weight of 100 is 5 million). Based on the message space and the potential security of the encryption scheme, we take 4096 for the polynomial degree n. This results in a polynomial coefficient size q of 84 bits and a logarithm of the attacker runtime of 255 for the decoding attack [61]. Successfully running the decoding attack breaks the security of the encryption scheme, therefore Naehrig et al. [67] suggest an attacker runtime for the decoding attack of at least 128, giving an equivalent of 128 bits security, or an attack complexity of 2^{128}. Table 3 shows the runtime performance of the prototype implementation with these parameters.

As can be seen from the table, the prototype for the protocol with online friends requires just under 2 minutes for the smallest dataset and over 16 minutes for the largest dataset. As expected, the prototype for the protocol with offline friends is slower. This prototype takes a little over 2 minutes for the smallest dataset and over 24 minutes for the largest dataset. This protocol has the benefit that friends need not be online, but requires more time to protect the information of those friends. When looking at the running times for the different datasets, we see a linear trend with respect to the number of books and a sub linear trend with respect to the number of friends. When looking at the protocol complexity, this is to be expected. Most operations have to be done per book and not per friend, but computing the impact of each friend on each book is linear in both (and the slowest step in the protocols).

We can lower the security of the somewhat homomorphic encryption scheme in order to gain a speed increase of the protocols. This lowered security implies that it takes less time to break the semantic security of the encryption scheme and recover encrypted messages. Should encrypted messages be recovered, privacy is lost. Towards this end, we take for n 2048, resulting in a q of 83 bits and a logarithm

Table 4: Runtime of the prototype with attacker runtime logarithm of 75

online	books		
friends	500	1000	2000
50	50 s	102 s	n/a
100	68 s	137 s	287 s
200	104 s	209 s	441 s

offline	books		
friends	500	1000	2000
50	59 s	120 s	n/a
100	85 s	170 s	442 s
200	134 s	267 s	617 s

of the attacker runtime of 75. Table 4 shows the runtime performance with these parameters offering lowered security, but more speed.

From the table we can see that these parameters result in runtimes that are more than 2 times faster than the more secure parameters. As expected, the running time relations between the different datasets remains the same. The desired level of security has a large impact on the running time of the protocols, but it does not change the basic properties of the protocols.

3.8 conclusion

In this chapter, we proposed an efficient privacy-enhanced familiarity-based recommender system. We proposed an adjusted recommendation formula that provides more privacy than weighted average with user supplied weights. Furthermore, two different protocols have been given, one where all friends of the user are online, and another where friends are offline. In both cases, a bi-directional friendship is assumed. The privacy of these protocols has been analysed, and two edge cases have been found and fixed. The protocols achieve privacy in the honest-but-curious model. A drawback of our solution is the reliance on friendship (and the accompanied trust) to distribute data. This means that our solution cannot be applied to similarity-based recommender systems.

We have made an implementation of the somewhat homomorphic encryption scheme of Brakerski and Vaikuntanathan [21]. Based on this implementation, a prototype of the two protocols has been built and the efficiency of them has been analysed. The prototype is limited to a single machine and single thread, and does not show the impact of latency. The prototype shows a runtime in the order of minutes with a linear trend with regards to scaling of the input set. This is a significant improvement over the work of Hoens et al. [48], the previous privacy-enhanced recommender systems with user supplied weights, which also assumed honest-but-curious participants and ran in the order of hours. Furthermore, not all users need to be online at some or all stages of the protocol, which is required by most related work. When we compare our work to the work of Erkin et al. [38], which assumes honest-but-curious participants and allows for offline users, we can see the difference in slowdown of the protocol when going from online to offline. The slowdown caused by our protocol is less than 1.5 times, while the slowdown of Erkin et al. is more than 6 times.

4 MALICIOUS USERS

4.1 INTRODUCTION

Considerable attention has been given to privacy in recommender systems, however (unlike this chapter) mainly dealing with honest-but-curious users (as the previous chapters). In such a setting, the user is assumed to adhere to the protocol specifications while still trying to breach the privacy of other users. In practice however, users do not always follow the rules. This gap between research and practice holds back the deployment of privacy-preserving recommender systems. Users may not give proper ratings, and in privacy-preserving recommender systems that assume honest-but-curious users, improper ratings will go unnoticed. Users may even try to exploit the recommender system for their own gain. In the most common attack [44], called the shilling attack, a user tries to introduce intentional bias into the recommender system. This is done by crafting a special attack profile within the constraints of the recommender system. For example, to increase the number of recommendations for an item made or sold by the user and by decreasing the number of recommendations for an item from a competitor.

We provide a framework, based on secure computation, for a privacy-preserving recommender system that can cope with malicious users, i. e. users who do not follow the rules. In this framework, recommendations are based around ratings given by users. We assume that the recommender system itself consists of two non-colluding honest-but-curious servers. The second server can act as both a privacy provider and an efficiency provider to the first server and users. The advantages of a second server are: 1) The users computational load is significantly reduced. Even to the point where users are not required to be online during the recommendation process [38]. 2) The user does not need to fully trust a single server, but instead only semi-trust two servers, increasing privacy for the user [13]. For similar reasons, two servers have also been proposed in private database queries by Boneh et al. [20] and in private data aggregation by Applebaum et al. [14]. This second server could be run by a user representative company, or a privacy watchdog, e. g. Electronic Frontier Foundation. A collaboration with such a server could boost the reputation of the service provider, where as any form of malicious behaviour from the servers may harm their reputation. Assuming honest-but-curious users is less realistic as there are no penalties for malicious behaviour. When the malicious user gets caught, he simply creates a new account and continues as before. Using homomorphic encryption and secure two-party computation, the two servers will be able to compute the recommendations amongst themselves in an oblivious manner. As the recommendations are computed between the two servers, this framework gives flexibility in the choice of the specific recommender system.

For the interaction with the users, we offer two secure protocols, one for updating the ratings of a user and one for retrieving a recommendation. The protocol for updating a rating has two major challenges: 1) The protocol requires sanitization of the (malicious) user's input. 2) The rating should be updated without either of the servers learning the rating, or the item associated with that rating. To sanitize the user's input, our protocol does not rely on expensive secure comparisons, but instead utilizes the fact that the rating space is typically small. To preserve the privacy of the user, our protocol uses double blinding and shifting to hide both the rating and the associated item. The protocol for retrieving a recommendation has the challenge of not allowing the (malicious) user control over what can be retrieved (i. e. decrypted). During retrieval, our protocol has no information flow from the user to the servers, thus removing user control. After detailing the framework, we analyse the security and privacy of this solution with malicious users and two non-colluding honest-but-curious servers. We also analyse the performance (theoretical and practical) of the framework.

Furthermore, we discuss the shilling attack in relation to our privacy-preserving recommender system framework. The shilling attack is not covered by the cryptographic definition of malicious users, as the shilling attack subverts the recommender system within the constraints of the system. All ratings given by the user (who is performing the shilling attack) are valid, given the rating constraints. Therefore, from the cryptographic sense, the ratings are valid and not malicious. We offer an approach to hinder the rapid building of attack profiles. However, similar to non-private recommender systems, the shilling attack remains a concern.

4.1.1 *Organization*

Section 4.2 discusses related work. Section 4.3 outlines the problem, details the architecture and security model, and gives an overview of the envisioned framework. Section 4.4 introduces the cryptographic primitive, supporting functions, and notation. Section 4.5 details our solution for the privacy-preserving protocols. Section 4.6 analyses the security and privacy of our solution. Section 4.7 analyses the performance of our solution and prototype implementation. Section 4.8 discusses the shilling attack in relation to our privacy-preserving framework. Section 4.9 concludes.

4.2 RELATED WORK

Canny [28] proposed using additively homomorphic encryption to privately compute a model for the collaborative filtering process. This model is made public and used in singular value decomposition and factor analysis, and can be used to make recommendations. Canny takes malicious users into account by limiting the amount of influence one user can have on the model each step. Furthermore, redundancy is in place to prevent a malicious coalition from reporting incorrect values and significantly altering the model. The main drawbacks are the heavy computational and communication overhead, and the continuing involvement of users (or user representatives). As redundancy is used to combat malicious users, most of the work has to be carried out multiple times, increasing the overhead of privacy. Furthermore, it is required that most users are online for several hours during the execution of the protocol. Cheng and Hurley [30] investigate the impact of the shilling attack on the work of Canny, specifically with the model made public. They conclude that releasing the model increases the knowledge for the attacker and makes it possible to more efficiently introduce bias.

Alternatively, it has been proposed to add noise to the information used in, or output from, recommender systems [18, 65, 74]. The added noise adds uncertainty about user ratings, thus increasing privacy. Because ratings are not encrypted, it can be easily checked if they are in the valid range. A downside of this approach is a loss in accuracy for the recommendations. Furthermore, in case of a central trusted party collecting true ratings, there is no privacy against this party [65]. In the case of an untrusted central party [74] or a distributed setting [18], the result is an even greater loss in accuracy. Gunes et al. [45] investigate the impact of the shilling attack when applying the privacy protection technique of Polat and Du [74], and conclude that the impact of malicious user profiles remain almost the same compared to a non private recommender system.

The usage of two servers has been proposed before. Aïmeur et al. [13] proposed a second server (or trusted component within the server) to promote the deployment of privacy-preserving recommender systems. Their approach relies on two non-colluding honest-but-curious servers, a merchant and an agent. It is unclear if malicious users are taken into account, but a collusion between a user and an agent results in the loss of some privacy for the merchant. Furthermore, no protection or robustness against the shilling attack is given. Erkin et al. [38] proposed to use a second server to increase efficiency and unburden the users of the recommender system. In their work, all entities are honest-but-curious and are not allowed to collude. The shilling attack and its implications are not mentioned.

Several solutions exist where the user data is distributed over multiple servers [17, 4, 75]. Each server has full access to the data that it is responsible for. As a result the user cannot give incorrect ratings, but has no privacy against the server that holds his data. The user does have privacy against the other servers. All these solutions rely on honest-but-curious servers. The shilling attack and its implications are not mentioned in these solutions.

As opposed to these solutions, our solution preserves the privacy of the users against both servers and other users. In our framework, the users are not required to be online and can be malicious and collude with other users or one of the servers. Furthermore, we provide an efficient solution that is flexible and does not suffer any accuracy loss. We also discuss the shilling attack in relation to our framework.

4.3 PROBLEM SPECIFICATION

As previously stated, we aim to cope with malicious users while still preserving privacy. To reduce the complexity of the system we assume two honest-but-curious non-colluding servers. We further assume that the recommender systems that will be instantiated in the recommender system framework are based around user ratings.

4.3.1 *Architecture*

We assume that there is a large number of users and two servers. The users give ratings and require recommendations, while the servers compute the recommendations based on the user ratings. We represent the number of items, for which a recommendation can be given, by n. The vector $I = (1, 2 \ldots, n)$, denotes the item indexes. Each user has a vector of ratings $R = (r_1, r_2 \ldots, r_n)$, where each rating $r_i, i \in I$, is an integer ($r_i \in \mathbb{Z}$) either between the minimum ($r_{min} > 0$) and maximum ($r_{max} > r_{min}$) value of a rating (depending on the rating system used), or 0 if the item is not rated. We denote the number of possible ratings by ν (i. e. $\nu = r_{max} - r_{min} + 1$).

Both servers are together responsible for providing user privacy in the recommender system. One of the two servers, which we shall denote the *key server*, is responsible for limiting access to the ratings of the users, while allowing the recommender system functionality. This is achieved through an additively homomorphic encryption scheme [39]. The key server holds a key pair for this encryption scheme, denoted by PK for the public key and SK for the secret key, but is not allowed to have access to any user data.

The other server, which we shall denote the *data server*, is responsible for storing an encrypted version of the user ratings. The user ratings are encrypted using the public key held by the key server. Each rating in the vector of ratings R is encrypted element wise, $[R]_{PK} = ([r_1]_{PK}, [r_2]_{PK} \ldots, [r_n]_{PK})$, where $[x]_{PK}$ denotes the encryption of x under public key PK. The data server is not allowed to have access to the secret key SK, but does know the public key PK.

4.3.2 *Security Model*

We assume that the users of the recommender system are potentially malicious, i. e. they don't have to conform to the protocol specifications and can behave arbitrarily. We also assume that they can collude with each other, and we do not put a bound on the number of malicious users.

For the two servers, we assume that both of them are honest-but-curious, i. e. they follow the protocol but try to learn any information they can from the data being processed. Furthermore, we assume that the two servers do not collude with one another. These are realistic assumptions, as the companies that will run these servers have a reputation to lose (thus causing severe damage). This is opposed to users, that have very little to lose when behaving maliciously.

We also exclude collusion between one server and users. As the servers are assumed to behave in an honest-but-curious manner, colluding with a malicious entity leads to a confusing state about the capabilities of the colluding parties. The colluding parties cannot be both malicious and honest-but-curious at the same time.

4.3.3 *Framework Overview*

The recommender system framework is based on three different actions:

1. Users should be able to update their ratings. The new or updated rating of the user should be changed also on the data server, without loss of the user's privacy. However, the user is considered malicious, so his input has to be validated. To achieve this action, we propose a privacy-preserving rating update protocol (Section 4.5.1).

2. Users can request a recommendation. To compute this, a protocol is run between the data and key server. The user is not involved in this action, which ends with an encrypted recommendation for the user at the data server. This action is not the focus of this chapter, as it is already solved, e. g. Erkin et al. [38]. Because we only impose some limitations (see Section 4.5.2) about the input (encrypted ratings) and output (encrypted recommendation) of the recommender system, the framework can be instantiated with different types of the recommender systems. In Section 4.6, we instantiate our framework with the work of Erkin et al.

3. Users retrieve their recommendation. After the computation of the recommendation, the encrypted recommendation has to be

sent from the data server back to the user. To achieve this action, we propose a privacy-preserving recommendation retrieval protocol (Section 4.5.3).

4.4 PRELIMINARIES

In our framework, we make use of the additively homomorphic encryption scheme by Bresson et al. [22], which can be seen as the additive version of El Gamal [36]. This scheme offers a double decryption mechanism, that successfully decrypts either by a master key or a local key. The message space is $\mathbb{Z}_N$, where N is the product of two large safe primes. These two primes form the master secret key. The local key pair (SK, PK) is computed as follows: the generator g is chosen as the square of a random element α from $\mathbb{Z}^*_{N^2}$, $g = \alpha^2 \mod N^2$. The secret key a is chosen at random between 1 and the order of the generator g (inclusive). The element h is computed as $g^a = h \mod N^2$. The public key consists of N, g, and h.

To encrypt a message m under public key PK, first a uniform random element r is chosen from $\mathbb{Z}_{N^2}$. Encryption is then as follows: $A = g^r \mod N^2, B = h^r \cdot (1 + m \cdot N) \mod N^2$. The ciphertext $[m]_{PK}$ consists of the tuple (A, B). To decrypt a message using the local secret key $SK = a$, compute the following: $m = \frac{B/(A^a)-1 \mod N^2}{N}$. There is a second decryption mechanism that uses the master secret key. Since we do not need it in our work, we omit the details.

The homomorphic properties are as follows: $[m_1]_{PK} \cdot [m_2]_{PK} = [m_1 + m_2]_{PK}, ([m_1]_{PK})^{m_2} = [m_1 \cdot m_2]_{PK}$. By generating the parameters and throwing away the master key, an encryption scheme where multiple (local) keys have the same message space can be derived. This latter property will be used in our rating update protocol. The scheme of Bresson et al. is semantically secure under the decisional Diffie-Hellman assumption. The efficiency of this scheme is comparable to that of Paillier [71]. However, as a ciphertext is represented by two elements instead of only one, the speed of operations is essentially halved.

We note that Peter et al. [73] suggested to use the scheme of Bresson et al. [22] to realize general-purpose outsourcing of computations in the two-server setting. In contrast to Peter et al. we do not use the double decryption mechanism of Bresson et al. but only need the fact that the plaintext space is the same across different public keys.

Recall from the previous chapters that we use the symbol $\in_r$ to denote uniform random selection. We use the following supporting functions throughout our framework:

PERM(VECTOR): A permutation function that randomly permutes the elements in the given vector. The output of this function is a vector with the same size as the input vector.

CS(VECTOR, MAGNITUDE): A circular shift function operating on a vector and having a certain magnitude. The elements of the vector are shifted to the end of the vector by the magnitude amount of spaces. The number of elements in the vector remains the same. In a circular shift, the element of the vector that is shifted out at the end, is shifted in again at the beginning of the vector, thus all elements remain in the vector. The output of this function is a vector with the same size as the input vector.

REP(VECTOR, INDEX, NEW ELEMENT): A replacement function that replaces the element of the vector at the given index with the given new element. The first element of the vector is indexed with 1. The output of this function is a vector with the same size as the input vector.

4.5 OUR FRAMEWORK

Recall that our framework consists of three different actions. Users should be able to update their ratings, request a recommendation, and retrieve their recommendation. In this section, we discuss each action. We also give one protocol for users to update their ratings and one for users to retrieve their recommendations.

4.5.1 *Privacy-Preserving Rating Update*

A trivial solution for updating user ratings would be for the user to send a complete encryption of his new data after every update. To check if each rating, $r_i, i \in I$, is within the correct range, between r_{min} and r_{max}, would require running $2n$ encrypted comparison protocols between the two servers. For each of the n items a comparison is needed to check if the rating is above or equal to r_{min} and below or equal to r_{max}. If any comparison does not satisfy the desired outcome, the user ratings are not valid. However, this is undesirable as it requires the user to store his own data, and the servers to check the ratings of each item for validity with expensive comparison protocols.

Instead, the privacy-preserving rating update protocol has as user input a single new rating r'_i and an item index i. The output of this protocol is the updated encrypted ratings vector $[R']_{PK}$, or the original encrypted ratings vector $[R]_{PK}$ in case of an incorrectly provided rating. The rating update protocol only updates one rating at a time. Since to fully preserve privacy, each element of the ratings vector needs to be touched, the complexity cannot be lower than linear in the number of items. We add the requirement that the data server has his own key pair $PK^\star, SK^\star$, with the same plaintext space $\mathbb{Z}_N$ as the key pair of the key server (recall that this is possible due to the use of the scheme of Bresson et al. [22]). The protocol, shown in Figure 11 and Figure 12, consists of six steps:

	User	Data Server
	(PK)	$(PK, SK^\star, PK^\star)$
	(r'_i, i)	$([R]_{PK})$
1.	encrypt: $[r'_i]_{PK}$	
	encrypt: $[i]_{PK}$	
	$\xrightarrow{[r'_i]_{PK}, [i]_{PK}}$	
		$([r'_i]_{PK}, [i]_{PK})$

Figure 11: Privacy-Preserving Rating Update Protocol (Step 1)

1. The first step is shown in Figure 11. The user prepares his input. The new rating and the item index are both encrypted using the public key of the key server, $[r'_i]_{PK}$ and $[i]_{PK}$. These encryptions are then sent to the data server.

2. From the second step onwards, see Figure 12. The data server prepares to check if $[r'_i]_{PK}$ is in the range between r_{min} and r_{max} (in collaboration with the key server). Since the number of possible ratings ν in a recommender system is usually quite small $(r_{max} - r_{min} + 1)$, the data server subtracts each possibility f $(r_{min} \leqslant f \leqslant r_{max})$ from the rating received from the user. If the rating given by the user is valid, one of these ciphertexts $[e_f]_{PK}$ will be a ciphertext of 0. The data server selects random values $d_f \in_r \mathbb{Z}^*_N$ and multiplicatively blinds these ciphertexts $[e_f \cdot d_f]_{PK} = [e_f]^{d_f}_{PK}, r_{min} \leqslant f \leqslant r_{max}$. The resulting vector $[E \cdot D]_{PK}$ is randomly permuted using $\mathsf{PERM}([E \cdot D]_{PK})$, leading to the vector $[E']_{PK}$. This vector $[E']_{PK}$ is sent to the key server.

3. The key server decrypts the vector E'. If the user submitted a valid rating, one of the elements in E' will be 0. In this case, the rating was valid and the key server sends the bit $\nu = 1$ to the data server and the protocol continues with step 4. If the vector E' does not contain an element 0, the rating given by the user was invalid. In this case, the key server sends the bit $\nu = 0$ to the data server and the protocol is aborted.

4. If the protocol is not aborted $(\nu = 1)$, the data server selects a random value $a \in_r \mathbb{Z}_n$. With this random value a as a magnitude the data server performs a circular shift of the encrypted ratings vector, i. e. $\mathsf{CS}([R]_{PK}, a) = [S]_{PK}$. The index received from the user is updated to match this shift, $[i]_{PK} \cdot [a]_{PK} = [i + a]_{PK}$. This allows the key server to insert the new rating at the correct location, without learning the actual item the rating is for. Then the data server selects n random values to blind all old ratings,

Data Server	Key Server
$(PK, SK^\star, PK^\star)$	$(SK, PK, PK^\star)$
$([R]_{PK}, [r'_i]_{PK}, [i]_{PK})$	

2. $\forall f : r_{min} \leqslant f \leqslant r_{max};$
   $[e_f]_{PK} = [r'_i]_{PK} \cdot [-f]_{PK}$
   $d_f \in_r \mathbb{Z}_N^*$
   $[e_f \cdot d_f]_{PK} = [e_f]_{PK}^{d_f}$
   $[E']_{PK} = \mathsf{PERM}([E \cdot D]_{PK})$

$$\xrightarrow{[E']_{PK}}$$

3. decrypt: E'
   if $\exists e'_f \in E' : e'_f = 0, r_{min} \leqslant f \leqslant r_{max},$
   then: $v = 1,$ else: $v = 0$

$$\xleftarrow{v}$$

4. if $v = 0,$
   then: $[R']_{PK} = [R]_{PK}$ and abort,
   else: $a \in_r \mathbb{Z}_n$
   $[S]_{PK} = \mathsf{CS}([R]_{PK}, a)$
   $[i + a]_{PK} = [i]_{PK} \cdot [a]_{PK}$
   $\forall j \in I;$
   $b_j, b' \in_r \mathbb{Z}_N$
   $[s_j + b_j]_{PK} = [s_j]_{PK} \cdot [b_j]_{PK}$
   $[r'_i + b']_{PK} = [r'_i]_{PK} \cdot [b']_{PK}$
   encrypt: $[b_j]_{PK^\star}, [b']_{PK^\star}$

$$\xrightarrow{[S+B]_{PK},[r'_i+b']_{PK},[i+a]_{PK},[B]_{PK^\star},[b']_{PK^\star}}$$

5. decrypt: $i + a$
   $k = (i + a - 1 \mod n) + 1$
   $[S' + B']_{PK} = \mathsf{REP}([S + B]_{PK}, k, [r'_i + b']_{PK})$
   $[B']_{PK^\star} = \mathsf{REP}([B]_{PK^\star}, k, [b']_{PK^\star})$
   $\forall j \in I;$
   $c_j \in_r \mathbb{Z}_N$
   $[s'_j + b'_j + c_j]_{PK} = [s'_j + b'_j]_{PK} \cdot [c_j]_{PK}$
   $[b'_j + c_j]_{PK^\star} = [b'_j]_{PK^\star} \cdot [c_j]_{PK^\star}$

$$\xleftarrow{[S'+B'+C]_{PK},[B'+C]_{PK^\star}}$$

6. decrypt: $B' + C$
   $\forall j \in I; [s'_j]_{PK} = [s'_j + b'_j + c_j]_{PK} \cdot [-b'_j - c_j]_{PK}$
   $[R']_{PK} = \mathsf{CS}([S']_{PK}, -a)$

$([R']_{PK})$

Figure 12: Privacy-Preserving Rating Update Protocol (Step 2-6)

$\forall j \in I; b_j \in_r \mathbb{Z}_N$, resulting in the random vector B, and 1 random value to blind the new rating, $b' \in_r \mathbb{Z}_N$. All the ratings of the user (including the new rating) are then blinded, $\forall j \in I; [s_j]_{PK} \cdot [b_j]_{PK} = [s_j + b_j]_{PK}$ (leading to the vector $[S + B]_{PK}$), and $[r'_i]_{PK} \cdot [b']_{PK} = [r'_i + b']_{PK}$. These blinding values are also encrypted under the data server's public key, $[B]_{PK^\star}$ (element wise) and $[b']_{PK^\star}$. The data server sends the following encryptions to the key server: $[S + B]_{PK}, [r'_i + b']_{PK}, [i + a]_{PK}, [B]_{PK^\star}$, and $[b']_{PK^\star}$.

5. The key server decrypts the received index and aligns it to the proper range, $k = (i + a - 1 \mod n) + 1$. This ensures that k is an element of I. Using this index location k, the key server replaces the old rating in the received ratings vector with the new rating, $\mathsf{REP}([S + B]_{PK}, k, [r'_i + b']_{PK})$, in the shifted ratings vector received from the data server. The key server also replaces the blinding value in the blinding vector to match the blind with the new rating, $\mathsf{REP}([B]_{PK^\star}, k, [b']_{PK^\star})$. This results in the vectors $[S' + B']_{PK}$ and $[B']_{PK^\star}$. The key server selects n random values to blind the ratings, $\forall j \in I; c_j \in_r \mathbb{Z}_N$. The blinding values are added to the ratings and the blinding values from the data server, resulting in $[S' + B' + C]_{PK}$ and $[B' + C]_{PK^\star}$. The key server sends the following encryptions to the data server: $[S' + B' + C]_{PK}$ and $[B' + C]_{PK^\star}$.

6. The data server decrypts the blinding values, $B' + C$, and removes the blinding values from the ratings, resulting in $[S']_{PK}$. Then the data server performs a circular shift to re-align the ratings to their original position, $\mathsf{CS}([S']_{PK}, -a) = [R']_{PK}$. The output is the vector of the new encrypted ratings original encrypted ratings of the user $[R']_{PK}$, this vector replaces the old vector $[R]_{PK}$.

4.5.2 *Privacy-Preserving Recommendation*

As mentioned before, the framework can be instantiated with different types of recommender systems. However, these recommender systems have to be based around user ratings, as this is the input to our framework. Since the data server has all user ratings encrypted under an additively homomorphic encryption scheme, $[R]_{PK}$, this server can process additions, subtraction, and scaling. Together with the key server, both can interactively compute other operations such as multiplication, division, and comparison. These operations form the basis of many different recommender systems, and can all be computed privately. In our analysis in Section 4.7, we instantiate the framework with the memory-based collaborative filtering recommender system of Erkin et al. [38] as an example. However, it would also be possible to

	User	Data Server	Key Server
	(PK)	(PK)	(SK, PK)
		$([x]_{PK})$	
1.		$y \in_r \mathbb{Z}_N$	
		encrypt: $[y]_{PK}$	
		$[x+y]_{PK} = [x]_{PK} \cdot [y]_{PK}$	
	$\xleftarrow{y}$		$\xrightarrow{[x+y]_{PK}}$
2.			decrypt: $x+y$
		$\xleftarrow{x+y}$	
3.	$x = x+y-y$		
	(x)		

Figure 13: Privacy-Preserving Recommendation Retrieval Protocol

do for example model-based collaborative filtering [27], content-based recommendation [57], or hybrid recommendation systems [24].

We denote the output of the recommendations by $[x]_{PK}$, in the case of a single value, or $[X]_{PK}$, in case of a vector. Since the data server has all the rating information, and no input from the users is required, all recommendations can be pre-computed by the servers. There are some limitations to having the rating data stored under an additively homomorphic encryption scheme. As homomorphic operations and secure two-party protocols are less efficient than their non privacy-preserving counterparts, the servers are bounded (depending on the recommender system used) in the number of recommendations they can compute in any given period.

4.5.3 *Privacy-Preserving Recommendation Retrieval*

The privacy-preserving recommendation retrieval has as input an encrypted result value $[x]_{PK}$ held by the data server and as output the same value x, but unencrypted, held by the user. This protocol can be run for multiple values in parallel, in case the input is the encrypted vector $[X]_{PK}$, which results in the output vector X for the user. The protocol, shown in Figure 13, consists of the following three steps:

1. The data server selects a random value used for blinding, $y \in_r \mathbb{Z}_N$. This blinding is applied to the result value, $[x]_{PK} \cdot [y]_{PK} = [x+y]_{PK}$. The data server sends y to the user and $[x+y]_{PK}$ to the key server.

2. The key server decrypts $[x+y]_{PK}$ to $x+y$. The key server sends $x+y$ to the user.

3. The user removes the blinding on the result value and retrieves the result x, $x + y - y = x \mod N$.

4.6 SECURITY ANALYSIS

In this section we analyse the security and privacy of the update and retrieval protocols. We look at the security and privacy of the protocols from the perspective of the different parties in the system, a malicious user, the data server, and the key server. We give a simulation-based proof sketch of how privacy is preserved. For the malicious user, we show how a malformed input impacts the system. Also, we discuss the impact of colluding users.

4.6.1 *Malicious User*

In both protocols, the interaction between the user and the servers is limited. The view of a malicious user in these protocols can be simulated. In the case of the rating update protocol, the user receives no information at all and thus cannot learn anything new. The simulation ends immediately after the user encrypts his input. In the case of the recommendation retrieval protocol, the user only receives two values, $x + y$ and y, which are both indistinguishable from a random value taken from $\mathbb{Z}_N$. The only correlation they have is that the formula $x + y - y = x \mod N$ holds. As x is the output of the protocol, this can also be simulated as any y taken from from $\mathbb{Z}_N$ will satisfy this relationship. During the interactions of the user with the servers, the user only receives the recommendations provided by the recommender system. This recommendation may leak some information, but this is considered as intended information leakage as it is the result of the recommender system formulas.

A malicious user might attempt to provide bad ratings to the servers. Particularly, ratings that are not between r_{min} and r_{max}, ratings for an item that does not exist (i. e. $i \notin I$), or invalid ciphertexts. If the rating is not in the proper range, none of the elements $r'_i - f, r_{min} \leqslant f \leqslant r_{max}$, will result in a 0. Therefore, even after blinding, the vector E' does not contain a 0 either and the key server will send the bit 0 to the data server and abort the protocol. Thus, the servers detect when a rating is invalid and consequently abort the update protocol. If the user provides an item index i that does not point to an item, this index is forced to a proper index of a random item when the key server computes $k = i + a \mod n$. According to Bresson et al. [22], when decrypting using the local key (which we use) an invalid ciphertext (a ciphertext for which $B/(A^a) = 1 \mod N$ does not hold) will be detected. Therefore, when a malicious user sends an invalid ciphertext, this will be detected by the key server during the decryption of the vector E'. As a result of this the key server sends the bit 0 to the data server and the protocol is aborted.

4.6.2 *Data Server*

In the rating update protocol, everything the data server sees is either encrypted or blinded. The new rating $[r_i']_{PK}$ and index $[i]_{PK}$ received from the user are encrypted with a semantically secure encryption scheme, and thus can be simulated. The bit received from the key server in step 3 reveals if a rating is in the proper range or not, as intended, but nothing more. The updated ratings $[S' + B' + C]_{PK}$ that are received from the key server are encrypted and blinded, and the blinds of the data server $[B' + C]_{PK^\star}$ are also blinded. This additional blinding C prevents correlation between the data that is sent to the key server and the data that is received. This blinding makes the values uniform random values and these values can therefore be simulated. The blinding of the encrypted values adds fresh randomness to the ciphertext and together with the semantic security, these ciphertexts can be simulated and cannot be correlated. The retrieval protocol can be simulated as the data server does not receive anything during the protocol execution.

4.6.3 *Key Server*

In the rating update protocol, the key server first receives the vector E' from the data server. Each element in this vector is either multiplicatively blinded with a uniform random value, and thus indistinguishable from a random value, or 0, which leaks that the rating r_i' is valid. Because of the permutation operation PERM performed by the data server, the position of the 0 does not reveal any information about the rating r_i' to the key server. Thus, the key server learns that the user's rating is in the valid range or not, but nothing else. In step 5, the key server receives blinded, $S + B, r_i' + b', i + a$, and encrypted values, $[B]_{PK^\star}, [b']_{PK^\star}$, from the data server. The blinded values are indistinguishable from random values due to the blinding with uniform random values, and can thus be simulated. The encryptions are encrypted under a semantically secure scheme, and can also be simulated. In the retrieval protocol the key server receives an encryption of a blinded value, $x + y$. This blinded value is indistinguishable from a random value and can thus be simulated. Therefore, from the key server's point of view, the retrieval protocol can be simulated.

4.6.4 *Collusion*

When users collude with each other, they share their respective inputs and outputs. Collusion can occur to either disrupt the functionality of the recommender system, or to learn private information about other (non-colluding) users. Colluding users can coordinate their inputs to subvert the recommender system. However, since the user input is sanitized by the servers, users can only coordinate on a shilling attack. We discuss this attack in Section 4.8.

Colluding users can try to learn the private information of other users. This occurs by looking at the recommender system output in relation to the input. The recommendation protocol is the only part of the system where user data from multiple users is potentially combined to form an output, a recommendation. If the details of the recommender system are known, the colluding users can cancel out each others influence and effectively lower the total number of users in the system. It then depends on the specific recommender system how this impacts the privacy of other users. For example, a content-based recommender would not suffer as only the information of the user seeking a recommendation is used. For collaborative filtering on the other hand, in the extreme scenario that there is only one other user next to the malicious users, all privacy for the honest user is likely to be lost (as this would be similar to a collaborative filtering system with only two users).

This is not limited to our private recommender system framework, as non-private recommender systems face the same issue. A collusion of users can pool their inputs and outputs and try to infer knowledge about the other users in the system. In collaborative filtering, this is especially dangerous for eccentric users [76]. This issue remains both in non-private recommender systems and in our framework.

4.7 PERFORMANCE ANALYSIS

We look at the theoretical complexity of the framework first. Then we look at the practical implications of this framework using a prototype. We also instantiate our framework prototype with the recommender system of Erkin et al. [38] and give performance numbers.

4.7.1 *Theoretical Complexity*

We look at the theoretical complexity of the two protocols. Specifically, we look at the expensive cryptographic operations. Recall that we use the additively homomorphic encryption scheme of Bresson et al. [22]. Table 5 shows the meaning of the abbreviations used in the complexity tables. Further, we denote the data server by $\mathcal{S}_{data}$, the key server by $\mathcal{S}_{key}$, and the user by $\mathcal{U}$.

Table 6 shows the computational and communicational complexity of the privacy-preserving rating update protocol. Recall that n is the number of items in the recommender system and ν is the number of possible ratings. In total, to update one rating, $5 + 5n + \nu$ encryption operations, $1 + n + \nu$ decryption operations, $2 + 4n + \nu$ homomorphic

Table 5: Legend for performance analysis

Computation	
enc	encryption
dec	decryption
add	homomorphic addition
sc	homomorphic scaling

Communication	
ciph	ciphertext
plai	plaintext value from $\mathbb{Z}_N$
bit	a 0 or a 1

additions, and ν homomorphic scalings are made. Along with the elements transmitted for the 2 encrypted comparisons, $5 + 4n$ ciphertexts are communicated. The majority of the computation happens on the servers and the majority of the communication is between the servers. The amount of work per rating update is linear in the number of items n in the recommender system.

When pre-computing the random values, i. e. generating and encrypting the random values, the data server can save ν encryption operations in step 2 and $2 + 2n$ encryption operations in step 4. By pre-computing, the key server can save $2n$ encryption operations in step 5. This reduces the total cost to $3 + n$ encryption operations, $1 + n + \nu$ decryption operations, $2 + 4n + \nu$ homomorphic additions, and ν homomorphic scalings.

Because the framework can be instantiated with different types of recommender systems, we cannot give complexity results for this step. For privacy reasons, all encrypted information available to the two servers that can potentially be used by the recommender system has to be touched at some point. The amount of data gives an early indication to the complexity of the recommender system. We note that all computations take place on the servers and all communication is between the servers. In the next section, we instantiate our framework with a recommender system and give timing information.

Table 7 shows the computational and communicational complexity of the privacy-preserving recommendation retrieval protocol per recommendation that is transferred to the user. In case of an output vector X for the user, multiply the costs by the size of the vector. The total computational cost is 1 encryption operation (which can be pre-computed), 1 decryption operation, and 1 homomorphic addition. The

Table 6: Per step complexity of the privacy-preserving rating update protocol

Step		Computation	Communication	
1	$\mathcal{U}$	2 enc	$\mathcal{U} \rightarrow \mathcal{S}_{data}$	2 ciph
2	$\mathcal{S}_{data}$	ν enc, ν add, ν sc	$\mathcal{S}_{data} \rightarrow \mathcal{S}_{key}$	ν ciph
3	$\mathcal{S}_{key}$	ν dec	$\mathcal{S}_{key} \rightarrow \mathcal{S}_{data}$	1 bit
4	$\mathcal{S}_{data}$	$3 + 2n$ enc, $2 + n$ add	$\mathcal{S}_{data} \rightarrow \mathcal{S}_{key}$	$3 + 2n$ ciph
5	$\mathcal{S}_{key}$	1 dec, $2n$ enc, $2n$ add	$\mathcal{S}_{key} \rightarrow \mathcal{S}_{data}$	$2n$ ciph
6	$\mathcal{S}_{data}$	n dec, n enc, n add	n/a	

Table 7: Per step complexity of the privacy-preserving recommendation retrieval protocol

Step		Computation	Communication	
1	$\mathcal{S}_{data}$	1 enc, 1 add	$\mathcal{S}_{data} \rightarrow \mathcal{U}$	1 plai
			$\mathcal{S}_{data} \rightarrow \mathcal{S}_{key}$	1 ciph
2	$\mathcal{S}_{key}$	1 dec	$\mathcal{S}_{key} \rightarrow \mathcal{U}$	1 plai
3	$\mathcal{U}$		n/a	

total communication cost is 1 ciphertext and 2 plaintext elements from $\mathbb{Z}_N$.

The bulk of the work is done by the two servers, which can be expected to have powerful computing at their disposal. The impact on the user is small, only two cryptographic operations per user rating are required, and no cryptographic operations to get recommendations. Even though the complexity is linear in the number of items (required for privacy), our framework has good efficiency with regards to the computational and communicational overhead as the constants are small.

4.7.2 *Prototype Results*

We have created a prototype implementation in C++ based on the GNU Multiple-Precision (GMP) library. The prototype only uses a single thread and computes the different steps for each party sequentially on the same machine. As such, network latency is not taken into account. The prototype consists of over 800 lines of code. All tests are carried out on an Intel Xeon at 3 GHz, with 2 GB of RAM. To test the performance of the protocols we take different sizes (1000, 4000, 10000, 50000) for the item sets. As both the update and retrieval protocols are run per user, there is only a single user in this data. The data is randomized for each run and the average of 10 runs is taken. We set the

Table 8: Per step prototype timing results of the privacy-preserving rating update protocol

Items	1000	4000	10000	50000
Step 1 ($\mathcal{U}$)	0.01 s	0.01 s	0.01 s	0.01 s
Step 2 ($\mathcal{S}_{data}$)	0.11 s	0.11 s	0.11 s	0.11 s
Step 3 ($\mathcal{S}_{key}$)	0.05 s	0.05 s	0.05 s	0.05 s
Subtotal	0.17 s	0.17 s	0.17 s	0.17 s
Step 4 ($\mathcal{S}_{data}$)	13.7 s	55.0 s	137 s	688 s
Step 5 ($\mathcal{S}_{key}$)	13.8 s	55.0 s	137 s	687 s
Step 6 ($\mathcal{S}_{data}$)	13.6 s	54.6 s	137 s	684 s
Total	41.3 s	165 s	412 s	2059 s

bit-length of the modulus N to 1024. The number of rating options v for the user is set to 10, with $r_{min} = 1$, and $r_{max} = 10$.

Table 8 shows the timing results of the rating update protocol without using pre-computation of random values. As expected, the first 3 steps do not show any difference depending on the number of items. These steps only depend on the number of possible ratings v. In the first step, the user only needs to spend 0.01 seconds, which should not impact the user at all. After step 3, the servers have determined if the rating supplied by the user is valid, and decide if to continue or not. Up to this point the user and server only need 0.17 seconds, which is close to real-time. If the decision about the rating is feed back to the user, the user can then continue with his activities.

In step 4 to 6 the majority of the computational work takes place. These steps are between the two servers only. They are dependent on the number of items n and scale more or less linearly with this number, averaging 0.0136 seconds per item. All these steps are dominated by the cost for encryption and decryption, as homomorphic additions are cheap in comparison. The per item cost of steps 4 to 6 of this protocol is just over 0.04 seconds, which seems reasonable. However, the linear scaling property makes this protocol less attractive for a huge number of items.

Table 9 shows the timing results of the recommendation retrieval protocol without pre-computation. In the first step, the cost is dominated by the encryption operations (which can be pre-computed). In the second step, the cost is dominated by decryption. Both steps take a comparable amount of time, the second step is cheaper as no homomorphic additions need to be performed. In the last step, where the user combines the information received from the server, no cryptographic operations take place. This step is therefore significantly faster than the other steps and provides minimal overhead for the user. In

Table 9: Per step prototype timing results of the privacy-preserving recommendation retrieval protocol

Items	1000	4000	10000	50000
Step 1 ($\mathcal{S}_{\mathtt{data}}$)	7.0 s	27.4 s	68.9 s	344 s
Step 2 ($\mathcal{S}_{\mathtt{key}}$)	6.8 s	27.3 s	68.2 s	340 s
Step 3 ($\mathcal{U}$)	0.001 s	0.001 s	0.002 s	0.009 s
Total	13.8 s	54.7 s	137 s	684 s

total the average time per item is 0.0137 seconds. Similar to the rating update protocol, the cost is reasonable, but scaling to a huge number of items is less attractive.

We have instantiated the work of Erkin et al. [38] into our prototype[1]. The instantiation consists of roughly 1600 lines of code. To pack the ratings into fewer ciphertexts takes 455 seconds (just over 7.5 minutes) for 1000 users and 1000 items. To then generate recommendations for a single user based on these packed ratings takes 550 seconds (just over 9 minutes). In comparison the protocols for updating ratings and retrieving recommendations takes significantly less time. The timing results for our framework are comparable to the timing results presented by Erkin et al. and to other solutions based on honest-but-curious users.

4.8 THE SHILLING ATTACK AND DEFENCES

Our privacy-preserving framework covers malicious users in the cryptographic sense, where users cannot deviate from the functionality of the system. In most recommender systems, users, by design, influence the recommendations for other users. As such, our framework does not prevent users from influencing the recommendations of other users. However, users can also be malicious within the constraints of the recommender system and intentionally introduce bias for their own gains. This is the main attack on non-private recommender systems [44], and is not covered by the cryptographic definition of a malicious user. As we consider users to be potentially malicious, we discuss the shilling attack in relation to our privacy-preserving recommender system framework in this section.

In the shilling attack, a user tries to introduce bias into the recommender system to either increase (push) or decrease (nuke) the popularity of an item. To accomplish this, the user crafts a special attack profile [66] containing ratings specifically chosen to push or nuke a single item and filler ratings to increase the spread and impact of the attack profile. For example, to push (nuke) an item in a collaborative

1 We thank Erkin et al for giving access to their code.

filtering system, give that item the highest (lowest) rating. The filler ratings then consist of average ratings, or ratings similar to specific other users. This will lead to the pushed (nuked) item getting a better (worse) prediction for recommendation. Crafting multiple attack profiles that target the same item also increases the spread and impact.

4.8.1 *Current Defences*

Current proposed defences against the shilling attack [44] consist of monitoring certain statistics of the recommender system, or having an inherent robustness in the recommender system. Several statistics can be monitored to identify attack profiles, such as weighted rating deviation from mean agreement [25] or degree of similarity to neighbouring profiles (in case multiple attack profiles are created with the same method) [31]. Once an attack profile has been discovered, additional measures can be taken.

Recommender systems can be made more robust by, for example, trying to identify and exclude attack profiles [70]. This has a close relation to computing statistics, however some misclassification is allowed as long as most profiles are classified correctly. Classification can occur based on a number of alternatives, e. g. mean rating, or standard deviation. Alternatively, the impact of a newly added rating can be computed on a fixed set of items [78]. If the impact of a newly added rating is too large, the rating is rejected.

Both approaches require extensive computational work, which is inefficient when processing on encrypted data, and are privacy invasive to the user, as computing these metrics leaks information about the ratings of a user. The current defences against the shilling attack are still possible in our privacy-preserving recommender system framework, however we advise against them as they are inefficient and privacy invasive.

An alternative defence that has been proposed is to offer a captcha to the user upon creation of a profile (and possibly other steps) to hinder automated profile building [66]. This idea has been dismissed as being too invasive for the legitimate users of the recommender system [66].

4.8.2 *Privacy-Preserving Defence*

We think that the idea of hindering automatic profile building has merit, especially in the setting of a privacy-preserving recommender system, and go into more details. As the servers know when the user interacts with the system, the servers can then first offer a small challenge to the user. However, we think that captchas are not suited.

Instead, to hinder the rapid building of attack profiles, we suggest using a proof of work system [35]. A proof of work system (or client puzzle scheme) allows a prover to prove to a verifier that some task has been carried out. The difficulty of this task can be set by the verifier. The verifier, only having to check the proof, only has to do a fraction of the work. Such a system has been suggested as a way to defend against denial of service attacks [50]. In such a scenario the connecting client (the prover) has to invest resources before the connection is accepted. The difficulty of the proof of work is increased as the server has to handle more connection requests. This increases the cost for the client to connect, thus also increasing the cost of mounting a denial of service attack.

Similarly, we can ask a user to do a proof of work for every rating that is submitted to the servers. For every rating that the user wants to add or change, he has to invoke the privacy-preserving rating update protocol of our framework. Furthermore, only one rating can be updated per run of the protocol. This prevents the user from circumventing the proof of work by updating multiple ratings for the cost of only one proof of work. Prior to the execution of this protocol, the data server can ask the user to submit a proof of work of a specific difficulty. When a user is quickly building a profile (potentially attacking the recommender system), the difficulty of the required proof can be increased. This requires the user to invest more resources into building the profile and simultaneously slows down the number of ratings that user can insert into the system. By decreasing the difficulty over time, users who only rate a few items each time, but interact with the system over a long period, do not have to give proofs with high difficulty. Thus for the server to compute the difficulty of the proof of work, the server only needs to know when and how many ratings are submitted by the user. This information is known to the server as it coincides with the invocation of the rating update protocol.

4.8.3 *Proof of Work Scheme*

The proof of work scheme we are looking for should have the following properties:

DETERMINABLE DIFFICULTY The determinable difficulty property implies that the verifier (the server) can precisely determine the required resources from the prover (the user) in solving a puzzle. This has two implications: (1) puzzles generated by the verifier are independent from each other, such that solving one puzzle does not help to solve another, and (2) generated puzzles are unpredictable by the verifier, each puzzle looks fresh, so that solutions cannot be pre-computed.

PARALLEL COMPUTATION RESISTANCE A prover cannot accelerate the computation of the proof by parallel computation (e. g. by using a botnet). This property requires that each step in the proof computation cannot be computed before the previous step has been computed, each step is performed sequentially.

Furthermore, the verifier should spend significantly fewer resources then the prover in verifying the proof. The difficulty of the proof of work should be able to be chosen with fine granularity. Ideally, no state information for each proof of work is kept.

We suggest the scheme of Rivest et al. [81], as it achieves determinable difficulty and parallel computation resistance. However, it does not achieve keeping no state information for each proof of work. The description of the scheme is as follows:

$\mathsf{Setup}(\ell, D)$: Based on a security parameter ℓ and a maximum puzzle difficulty D, this algorithm selects two random large primes p, q and a cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}^*_{pq}$. The public parameter is pq and the master key MK is (p, q, H).

$\mathsf{Generate}(MK, d, req)$: This algorithm takes as input the master key, a proof difficulty d, and possibly some additional request information req. A random value r is chosen, $r \in_r \mathbb{Z}_{pq}$, and a generator g computed, $g = H(r||req)$. The proof of work is output as (g, d) and the state information $inf = (r, d, req)$ is stored.

$\mathsf{Solve}(g, d, pq)$: Based on the proof of work (g, d) and the public parameter pq, the proof h is computed as $h = g^{2^d} \mod pq$.

$\mathsf{Verify}(MK, inf, h)$: This algorithm takes as input the master key MK, the state information inf, and the proof. The proof succeeds if $h \equiv (H(r||req))^{2^d \mod \phi(pq)} \pmod p q$. Note that due to the knowledge of p and q, the verifier can compute the proof faster than the prover.

When different proof of works have the same proof difficulty d, they can be verified in a batch, rather than one by one. Suppose there are n proof of works, denoted by $(g_i, d), 1 \leqslant i \leqslant n$, and corresponding solutions $h_i, 1 \leqslant i \leqslant n$. Then batch them as follows, (1) select random values $x_i \in_r \mathbb{Z}^*_X$, where X is an integer and smaller than pq, and (2) check if the following equation holds:

$$(\prod_{i=1}^{n} (g_i)^{x_i})^{2^d \mod \phi(pq)} \equiv \prod_{i=1}^{n} (h_i)^{x_i} \pmod{pq}$$

The chance than an incorrect solution exists is upper-bounded by $1/X$. Note that both the performance gain and the security depend on the choice of X. A small X provides better performance and lower security guarantees, and a large X provides higher security guarantees but worse performance.

4.8.4 *Proof Difficulty*

We give a suggestion about how the difficulty of a proof should increase as more ratings are added and how it should decrease over time. We then analyse the impact of this suggestion.

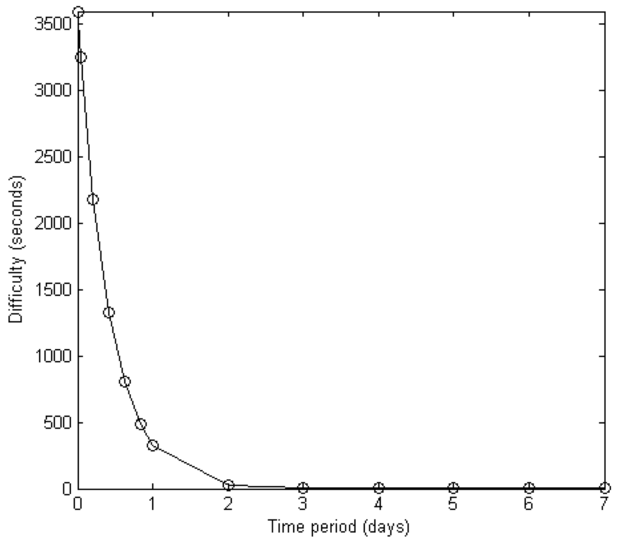


Figure 14: Proof Difficulty δ w.r.t. Time Period τ

We want to allow normal users to be able to insert ratings in the system without being hindered by the proof of work system. However, attackers should be hindered severely in rapidly building profiles. Therefore, users who almost constantly insert ratings should have a severe proof of work difficulty, while users who have long periods of time in between ratings should not have to do any proof. As a starting point we take the time period in which the last 10 ratings (of the user) were inserted and exponentially increase the proof difficulty when this time period is smaller.

We take the time period τ of the last 10 ratings as the time difference (in seconds) between the current time, at which the user wants to insert a rating, and the time when the user inserted his tenth previous rating (the beginning of the period). If the user did not yet rate 10 items, take the origin as the beginning of the time period. The difficulty δ (also in seconds) of a proof of work is then computed as follows:

$$\delta = (60 \cdot 60) \cdot e^{-\tau/(60 \cdot 60 \cdot 10)}$$

This formula results in a maximum proof of work difficulty of 1 hour. The first 10 ratings do not require any proof of work.

An overview of the proof difficulty given a certain time period is shown in Figure 14. Given this formula, a user that rates 10 items in a period of 4 days, does not have to do any proof of work. Should the user rate 10 items over period of 3 days, the difficulty is 3 seconds. A period of 2 days results in a difficulty of only 30 seconds. When the period becomes even shorter, the difficulty for the proof of work increases significantly, a period of 1 day results in a difficulty of 5 minutes and 27 seconds. Should a user decide to constantly insert ratings, i. e. as fast as possible taking the proof of works into account, the difficulty will average out around 35 minutes. This will allow the user to insert around 41 ratings per day. We believe this numbers do not greatly impact regular users, but severely hampers the rating insertion speed of attackers.

Alternative methods of computing the difficulty are available. The most appropriate method is likely to be dependant on the specific scenario and the trade-off between robustness against attackers and burden for honest users. Therefore, we allow the usage of different difficulty computation methods.

While there are no technical limitations to using proof of work systems in non-private recommender systems, it is not advisable to do so. Proof of work systems can only hinder the rapid building of profiles. Unlike traditional approaches that detect attack profiles and remove their influence entirely. Therefore, proof of work systems only delay the building of a profile that will later be removed anyway. The extra work does not seem worth it. In the privacy-preserving setting however, where efficiency and privacy are an issue, proof of work systems provide a useful tool to delay attackers.

4.9 CONCLUSION

We have provided a framework for a privacy-preserving recommender system. This framework can cope with malicious users and is therefore a good candidate for practical deployment. The framework consists of two non-colluding honest-but-curious servers. Having two servers reduces the systems complexity, increases efficiency, and reduces interaction with the users to a minimum. Furthermore, the framework can be instantiated with different types of recommender systems, as long as they are based around user ratings.

The framework is based around two privacy-preserving protocols. First, a rating update protocol, where a user gives a new rating to the servers and the servers privately update the database. Second, a recommendation retrieval protocol, where the user can retrieve the results of the recommender system in a privacy-preserving manner. We have given an analysis of the security and privacy of these protocols and an analysis of the performance of the framework. The performance of our framework is comparable to solutions with honest-but-curious users, and more importantly the overhead for the user is small.

We have discussed the most prevalent attack in recommender systems, the shilling attack, in relation to our privacy-preserving recommender system framework. We concluded that in our setting the current counter measures are privacy invasive and inefficient. Therefore, we offered a defence mechanism that is compatible with our privacy-preserving framework and that reduces the speed at which attack profiles, required for the shilling attack, can be built. However, like with non-private recommender systems, the shilling attack remains a concern.

5 CONCLUSION

5.1 MAIN RESEARCH QUESTION

The data collection of recommendation service providers potentially exposes the data of many users. This may lead to privacy breaches for the users and thus a need for the protection of their privacy. We chose to focus on secure computation as a means to provide privacy. We dismissed the introduction of uncertainty as a privacy mechanism as it reduces the accuracy of recommender systems and provides no privacy against the service provider. Decentralization was dismissed as it cannot deal with the constant flux of users. The biggest drawback of secure computation is its efficiency compared to a non private system, hence our focus on efficiency. This led to our main research question:

RQ: How to construct efficient privacy-enhanced recommender systems?

As efficiency is not the only hurdle for making the secure computation of recommender systems practical, we focussed on three specific practical scenarios to address the (in our view) more pressing issues for the deployment of privacy-enhanced recommender systems. In the rest of this concluding chapter we restate the research sub questions and corresponding scenarios, we discuss the achievements and limitations of our protocols, and we give directions for future work.

5.2 DISCUSSION OF RESEARCH QUESTIONS

In this section, we look at the research sub questions and corresponding protocols. We relate these protocols to the main research question and we look at both the theoretical and the practical efficiency of the work in this thesis.

5.2.1 *Collaborating Competitors*

In the first scenario, competing service providers collaborate to provide more accurate recommendations for their users, without giving up confidential data about their users. The benefit for the users is better recommendations. For the service provider the benefit is more satisfied users, leading to an improved business. This scenario relates to the following research sub question:

SQ1: How can competing recommender system service providers collaborate?

We refined the scenario of collaborating competitors to collaboration based on collaborative filtering. Collaborative filtering makes recommendations based on a dataset of user ratings for items. We assumed that the items held by the two service providers are the same, but that the users subscribed to the service providers are not. This is a horizontal partitioning of the combined dataset. We further assumed that the user ratings are available without protection at the service provider with which the user subscribed. This resulted in a secure protocol for computing collaborative filtering between the two service providers. The protocol does not select a neighbourhood (as is common in collaborative filtering) to increase efficiency. The protocol utilizes two specialized sub-protocols for securely computing an absolute value and a division. From the point of view of the user, the entire protocol can be pre-computed and recommendations can be received instantly.

Our protocol is limited by the assumption on the dataset. Only horizontal partitioning of the data is supported. Furthermore, the users have no privacy protection from the service provider they have subscribed with. Compared to related work, our protocol does not require any user involvement, making it cheaper for the user. Our protocol also does not leak any intermediate information during the protocol, such as a model of the data. Vertical and even arbitrary partitioning of the dataset remains as a potential improvement for our protocol.

5.2.2 *Limited User Availability*

The second scenario addresses the constant flux of user availability and the common requirement of user involvement in secure computations. In this scenario, users are not required to wait for all other users to get their recommendations. The corresponding sub question is as follows:

SQ2: How to cope with the limited user availability in recommender systems?

In the previous scenario, user privacy was sacrificed to avoid issues with user availability. In this scenario, we did not want to sacrifice privacy. Instead, our protocol builds upon the trust that is inherent to the friendship between two users. It further builds upon the fact that in certain domains, familiarity-based collaborative filtering performs comparable to similarity-based collaborative filtering. We offer a protocol in this scenario that requires users to be available and from there build a protocol that does not require user availability. The trust of a user in the friendship is expressed through proxy re-encryption and secret sharing, allowing the service provider to act as an intermediary.

The reliance on the trust of friendship is also a limiting factor. Our protocol assumes a bi-directional friendship between users. It does not generalize to a similarity-based collaborative filtering system due to the (potential) lack of friendship and trust between similar users. Most protocols in related work do not offer any option for limited availability of users, other than waiting for them. The protocols that do cope with limited user availability are often more inefficient than their non-coping counterparts. This is opposed to our protocols that have comparable efficiency in both cases.

5.2.3 *Malicious Intent by Users*

In the final scenario, the malicious intent of users is taken into account. As not all users might behave honestly, the recommender system in this scenario deals with the malicious behaviour of users. This is captured in the following sub question:

SQ3: How to deal with malicious intent by the users?

We offer a framework that makes use of two non-colluding servers. The main protocols of the framework allow for the updating of ratings and the retrieval of recommendations. The part that does the actual computation of recommendations can be instantiated by different types of rating-based recommender systems. Because the user involvement is limited, the burden for the user in participating in the recommender system is small. Next to offering this framework, we also discuss the shilling attack. This is an attack that can be executed by users with malicious intent, but is not covered by the cryptographic definition of a malicious user. We provide a defence mechanism that hinders users in rapidly building shilling attack profiles.

Because our framework is based around ratings, the framework cannot be instantiated with recommender systems that are not based on ratings (i. e. demographic and knowledge-based recommender systems). Furthermore, while the framework hinders the construction of shilling attack profiles, like non-private recommender systems, a full solution for the shilling attack remains to be found. A framework for recommender systems that can deal with malicious intent of users does not exist in related work. Even for specific recommender system based on secure computation, only a few can deal with malicious users.

5.2.4 *Efficient Privacy-Enhanced Recommender Systems*

The protocols for the three scenarios are all based on secure computations. We look at their common traits in relation to the main research question. The main research question is again given:

RQ: How to construct efficient privacy-enhanced recommender systems?

THEORETICAL EFFICIENCY The theoretical efficiency of our work (i. e. the complexity) is based on the requirement imposed by secure computation that all data that can potentially be relevant has to be used in the computation. This means that in the case of collaborative filtering, the complexity of protocols is in the order of users times items. Our protocols improve the theoretical efficiency in the following ways.

The protocol of Chapter 2 processes the information of one service provider before the secure computation, lowering the overall complexity. The protocols of Chapter 3 limit the number of users used in the secure computation from all possible users, to only the friends of the user. The difference between the number of users and number of friends is large and therefore the complexity is greatly lowered. The rating update and retrieval protocols of the framework of Chapter 4 only concern a single user. The complexity of these protocols is only in the number of items.

The theoretical efficiency indicates a lower bound of the privacy-enhanced recommender systems. Solutions using generic constructions for secure computation do not aim to lower the theoretical efficiency and thus will be less efficient than tailored solutions. While the complexity provides a good first estimate, the efficiency of a construction is not solely dependent on the complexity of a protocol. The speed of a protocol is also greatly impacted by its design and implementation.

PRACTICAL EFFICIENCY Through specialized protocols, designed and optimized for specific scenarios, and fast primitives we managed to get the runtime of privacy-enhanced recommender systems down from hours to minutes, and in some cases even to seconds. These runtime figures are measured by the prototype implementations that we wrote in C++.

Table 10 shows an overview of the implementation of the primitives used in the thesis. The number of lines of code and the used libraries are given. The libraries are GMP[1] (GNU Multiple Precision Arithmetic Library) and FLINT[2] (Fast Library for Number Theory). For a description of the primitives, check the chapter in which they are used, or consult the references. All primitive implementations have a common structure, are written by us, and have undergone extensive functionality tests. The code for the Brakerski and Vaikuntanathan [21] primitive was co-developed by C.T. Bösch. The runtime of the prototypes is impacted by the choice of cryptographic primitives, its implementation, and its parameters. For the parameters, there is a balance between speed and security. Bad implementations and wrong parameters can even lead to invalid calculations, thus destroying the functionality of the recommender system.

Table 11 shows an overview of the code for the prototypes of the protocols designed in this thesis. The number of lines of code, the executable size, and the used primitive are given for the prototypes of each chapter and for the instantiation of our framework from Chapter 4

1 http://gmplib.org/
2 http://www.flintlib.org/

Table 10: Overview of the implementation of cryptographic primitives

Primitive	Lines of code	Libraries
Paillier [71]	~380	GMP
Bresson et al. [22]	~480	GMP
Brakerski and Vaikuntanathan [21]	~700	FLINT

Table 11: Overview of the implementation of the prototypes of our protocols

Protocol	Lines of code	Executable size	Primitive
Chapter 2	~750	171.3 kB	[71]
Chapter 3	~1050	38.1 kB & 42.1 kB	[21]
Chapter 4, main	~800	25.8 kB & 22.1 kB	[22]
Chapter 4, [38]	~1600	43.7 kB	[22]

with the work of Erkin et al. [38]. For the instantiation, our code is a modified version of the code written by Erkin et al. For the other protocols, all code is written by us. The primitive and prototype sources will be made available publicly online[3]. Our protocols do not reduce the accuracy of the recommendations and achieve a level of privacy which is equal to confidentiality. As the protocols do not leak the data or intermediate information, the user can be sure that his data will not be abused for other purposes.

While these prototypes bring the runtime of solutions to manageable times, they are only prototypes. Actual solutions should be made more robust and are likely to run under less favourable conditions. However, significant improvements can be made by introducing threading to the prototypes as many operations can be parallelized. Furthermore, cloud computing can increase the amount of resources available to the entities running the solutions.

5.3 COMPARISON WITH STATE OF THE ART

To investigate the impact of our research, we compare our protocols to the state of the art. The state of the art is given by the work of Basu et al. [17], Canny [28], and Erkin et al. [38]. These research papers present protocols based on secure computation and appear in the related work section of all our chapters. Basu et al. [17] provide a secure protocol for collaborating competitors to compute a model-based collaborative filtering system based on the slope one predictor. Multiple service providers can based on their own data collaboratively compute the model,

3 http://scs.ewi.utwente.nl/other/jeckmanscode/

and based on this model compute predictions. The used cryptographic primitive is a threshold version of Paillier [33]. Canny [28] proposes a model-based collaborative filtering system based on singular value decomposition. Canny assumes a peer-to-peer setting with a central server just for storage of values (with integrity) and as a global source of random values. To handle malicious peers threshold El-Gamal encryption [36] and zero-knowledge-proofs [42] are used. Erkin et al. [38] propose a memory-based collaborative filtering system that copes with the unavailability of users by adding a second non-colluding server. One server holds the encrypted data and the other server holds the key. Data packing (multiple values in one encryption) is used to increase efficiency. The used cryptographic primitive is Paillier [71].

Table 12 shows how our protocols relate to the state of the art. All the protocols are compared based on the assumptions that are made on the user and the service provider (server), the privacy that is offered against other users and the service provider, and the efficiency of initial computations to be done before recommendation (pre-computation) and of the computation of the recommendation itself. These protocols are comparable as the assumptions made are similar, privacy loss is always defined in terms of data leakage, and the setup to benchmark timing results is comparable. The exception is the timing information given by Canny [28], which has been updated to current standards. A brief explanation of uncommon terms used in the table follows. No assumptions on the user means that the service provider has plaintext access to the information of the user. When the protocol is made for competing service providers, the privacy against the other service providers is given instead of other users (other servers:). Assuming that a user is unavailable implies that users can be offline during the computation of a recommendation for another user. The opposite, available, implies that users should be online to contribute their data during key phases of the protocol. The model of a model-based collaborative filtering approach is denoted by CF model. WORM storage stands for write-once-read-many storage, where data can be written to the storage once and not be changed any more, but without restrictions on the number of read operations. A server that acts as a trusted source of global random data is denoted by random source.

As can be seen from the table, our protocols improve upon the state of the art. The protocol of Chapter 2 has better privacy guarantees and improved efficiency (most notably in the pre-computation) compared to the work of Basu et al. [17], while having the same assumptions on the user and service provider. The offline protocol of Chapter 3, compared to the work of Erkin et al. [38], makes fewer assumptions on the server, while having similar privacy protection and efficiency. The framework of Chapter 4 makes fewer assumptions on the user, compared to the work of Canny [28] and Erkin et al. [38], and is the first to provide privacy of the user against the service provider, while only assuming a malicious and unavailable user. However, the work

Table 12: Overview of the protocols in relation to the state of the art

	Assumptions on		Privacy against		Efficiency of	
Scheme	User	Server	Users	Server	Pre-computation	Prediction
Chapter 2	none	honest-but-curious multiple servers	other servers: no leakage	full leakage	none	minutes
Basu et al. [17]	none	honest-but-curious multiple servers	other servers: CF model	full leakage	hours	minutes
Chapter 3	honest-but-curious unavailable	honest-but-curious	friends: rated items	friends of user	seconds - minutes	minutes
Erkin et al. [38]	honest-but-curious unavailable	honest-but-curious two servers	number similar users	no leakage	seconds	minutes
Chapter 4	malicious unavailable	honest-but-curious two servers	no leakage	when user rates	seconds - minutes	minutes
Canny [28]	malicious available	WORM storage random source	CF model	CF model	hours - days	milliseconds
Erkin et al. [38]	honest-but-curious unavailable	honest-but-curious two servers	number similar users	no leakage	seconds	minutes

of Canny makes fewer assumptions about the server. The privacy protection of our framework is better than that of Canny and Erkin et al. The protocol of Canny requires heavy pre-computation to create the model (and to keep it up to date), this is somewhat balanced out by the fact that creating a recommendation is very fast as there are no secure computations involved at this stage. Our framework and the protocol of Erkin et al. both require less time for pre-computation, but more for the actual recommendation. In the case of our framework it is even dependent on the specific recommender system instantiation, but it is unlikely to be faster than the protocol of Canny due to the usage of cryptography during recommendation.

At the beginning of this thesis we set out to encourage deployment of privacy-enhanced recommender systems. This table can help service providers choose an appropriate recommender system based on their needs. All protocols in the table allow for privacy-enhanced collaborative filtering. Furthermore, the protocols in this thesis should be implementable without many issues, as they are designed to avoid complex operations. For specific scenarios that are not detailed in this thesis, the design methods for our protocols point others to problem areas and optimization opportunities. Furthermore, sub-protocols can be reused and protocol specifics can be tweaked as needed.

When such privacy-enhanced recommender systems are deployed, this will benefit users everywhere, as recent revelations reveal that mass surveillance is getting more and more common. In areas where privacy is a must (e. g. healthcare), our protocols bring new functionalities, leading to improved services for users.

5.4 FUTURE WORK DIRECTIONS

While the efficiency of privacy-enhanced recommender systems has been advanced significantly in this thesis, there is still room for improvement. Not only might protocols be made even faster, attention should also be paid to lowering the overall complexity, and thus improving scalability. This efficiency line of research is ongoing and might never be completely solved.

In this thesis, we explicitly chose three scenarios to work on. But other scenarios exist, such as privately updating the knowledge base in knowledge-based recommender systems, or privately selecting the proper demographic and associated recommendations in a demographic recommender system. Tackling other scenarios that require privacy-enhanced recommender systems is left as future work. It is also interesting to investigate slightly different scenarios, to the ones presented in this thesis. We use trust in friendship to cope with limited user availability, but other avenues of trust exist (e. g. co-workers, communities) and can lead to a similar scenario. While we deal with malicious users, a scenario where also the malicious intent by service providers is dealt with remains.

In the area of privacy, we chose to focus on confidentiality. However, information privacy is also related to control. The control for the user to chose the terms for his data usage. Would it be possible to build such control into the secure computation protocols for recommender systems? As an example, consider that the user has expressed that his ratings be deleted after one year. How can the service provider keep track of this one year, when he is not allowed to know which item the user is rating at a given point?

5.5 FINAL WORDS

In this thesis the state of the art in secure computation for recommender systems has been improved, with practical scenarios being tackled. It brings us closer to the deployment of recommender systems that respect the privacy of its users. In this digital age, privacy becomes increasingly important and is not something that should just be given up. Luckily, the potential for secure computation to be used for practical applications is ever increasing.

BIBLIOGRAPHY

AUTHOR REFERENCES

[1] Michael Beye, Arjan Jeckmans, Zekeriya Erkin, Qiang Tang, Pieter Hartel, and Inald Lagendijk. *Privacy in Online Social Networks*, chapter 4, pages 87–113. Springer, 2012.

[2] Arjan Jeckmans, Qiang Tang, and Pieter Hartel. Poster: privacy-preserving profile similarity computation in online social networks. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 793–796, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2093476.2093495.

[3] Arjan Jeckmans, Qiang Tang, and Pieter Hartel. Privacy-preserving profile matching using the social graph. In *International Conference on Computational Aspects of Social Networks*, pages 42–47. IEEE, 2011. ISBN 978-1-4577-1132-9. doi: 10.1109/CASON.2011.6085916.

[4] Arjan Jeckmans, Qiang Tang, and Pieter Hartel. Privacy-preserving collaborative filtering based on horizontally partitioned dataset. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 439–446. IEEE, may 2012. doi: 10.1109/CTS.2012.6261088.

[5] Arjan Jeckmans, Michael Beye, Zekeriya Erkin, Pieter Hartel, Inald Lagendijk, and Qiang Tang. *Privacy in Recommender Systems*, chapter 12, pages 263–281. Springer, 2013.

[6] Arjan Jeckmans, Andreas Peter, and Pieter Hartel. Efficient privacy-enhanced familiarity-based recommender system. In *Proceedings of the 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 400–417. Springer, 2013.

[7] Arjan Jeckmans, Andreas Peter, and Pieter Hartel. Towards practical private recommender systems: Security against malicious users. In *In submission*, 2013.

[8] Qiang Tang and Arjan Jeckmans. Efficient client puzzle schemes to mitigate dos attacks. In *Computational Intelligence and Security, International Conference on*, pages 293–297, Los Alamitos, CA, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4297-3. doi: 10.1109/CIS.2010.70.

[9] Qiang Tang and Arjan Jeckmans. Towards a security model for computational puzzle schemes. *International Journal of Computer Mathematics*, 88(11):2246–2257, 2011. doi: 10.1080/00207160.2010.543951.

OTHER REFERENCES

[10] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, June 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.99.

[11] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 217–253. Springer US, 2011. ISBN 978-0-387-85820-3.

[12] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450, 2000.

[13] Esma Aïmeur, Gilles Brassard, José Fernandez, and Flavien Mani Onana. Alambic: a privacy-preserving recommender system for electronic commerce. *International Journal of Information Security*, 7:307–334, 2008. ISSN 1615-5262.

[14] Benny Applebaum, Haakon Ringberg, Michael J. Freedman, Matthew Caesar, and Jennifer Rexford. Collaborative, privacy-preserving data aggregation at scale. In *Proceedings of the 10th international conference on Privacy enhancing technologies*, PETS'10, pages 56–74, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-14526-4, 978-3-642-14526-1. doi: 10.1145/1881151.1881155.

[15] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security*, 9(1):1–30, February 2006. ISSN 1094-9224. doi: 10.1145/1127345.1127346.

[16] Anirban Basu, Hiroaki Kikuchi, and Jaideep Vaidya. Privacy-preserving weighted slope one predictor for item-based collaborative filtering. In *Proceedings of the international workshop on Trust and Privacy in Distributed Information Processing*, 2011.

[17] Anirban Basu, Jaideep Vaidya, and Hiroaki Kikuchi. Efficient privacy-preserving collaborative filtering based on the weighted slope one predictor. *Journal of Internet Services and Information Security (JISIS)*, 1(4):26–46, November 2011.

[18] 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, 2007.

[19] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer Berlin / Heidelberg, 1998.

[20] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and DavidJ. Wu. Private database queries using somewhat homomorphic encryption. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 102–118. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38979-5. doi: 10.1007/978-3-642-38980-1_7.

[21] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 505–524, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22791-2.

[22] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *Advances in Cryptology - ASIACRYPT 2003*, pages 37–54, 2003. doi: 10.1007/978-3-540-40061-5_3.

[23] Robin Burke. Knowledge-based recommender systems. In *Encyclopedia of Library and Information Systems*, volume 69, pages 180–200, 2000.

[24] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, November 2002. ISSN 0924-1868. doi: 10.1023/A:1021240730564.

[25] Robin Burke, Bamshad Mobasher, Chad Williams, and Runa Bhaumik. Classification features for attack detection in collaborative recommender systems. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 542–547, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5. doi: 10.1145/1150402.1150465.

[26] Joseph Calandrino, Ann Kilzer, Arvind Narayanan, Edward Felten, and Vitaly Shmatikov. "you might also like:" privacy risks of collaborative filtering. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 231–246, 2011. doi: 10.1109/SP.2011.40.

[27] John Canny. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th annual international conference on Research and development in information retrieval*, pages 238–245, 2002.

[28] John Canny. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy*, pages 45–57, 2002.

[29] Kathleen M. Carley and David Krackhardt. Cognitive inconsistencies and non-symmetric friendship. *Social Networks*, 18(1):1–27, 1996. ISSN 0378-8733. doi: 10.1016/0378-8733(95)00252-9.

[30] Zunping Cheng and Neil Hurley. Trading robustness for privacy in decentralized recommender systems. In *Proceedings of the Twenty-First Conference on Innovative Applications of Artificial Intelligence*, pages 79–84, 2009.

[31] Paul-Alexandru Chirita, Wolfgang Nejdl, and Cristian Zamfir. Preventing shilling attacks in online recommender systems. In *In WIDM '05: Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management*, pages 67–74. ACM Press, 2005.

[32] Sherman Chow, Jian Weng, Yanjiang Yang, and Robert Deng. Efficient unidirectional proxy re-encryption. In *Proceedings of the Third international conference on Cryptology in Africa*, AFRICACRYPT'10, pages 316–332, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-12677-4, 978-3-642-12677-2. doi: 10.1007/978-3-642-12678-9_19.

[33] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-41658-6. doi: 10.1007/3-540-44586-2_9.

[34] Nima Dokoohaki, Cihan Kaleli, Huseyin Polat, and Mihhail Matskin. Achieving optimal privacy in trust-aware social recommender systems. In *Proceedings of the Second international conference on Social informatics*, SocInfo'10, pages 62–79, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16566-4, 978-3-642-16566-5. doi: 10.1145/1929326.1929331.

[35] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag. ISBN 3-540-57340-2. doi: 10.1145/646757.705669.

[36] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE*

Transactions on, 31(4):469–472, Jul 1985. ISSN 0018-9448. doi: 10.1109/TIT.1985.1057074.

[37] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Inald Lagendijk. Efficiently computing private recommendations. In *International Conference on Acoustic, Speech and Signal Processing-ICASSP*, pages 5864–5867, 2011.

[38] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Inald Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE Transactions on Information Forensics and Security*, 7(3):1053–1066, 2012.

[39] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007:15:1–15:15, January 2007. ISSN 1687-4161. doi: 10.1155/2007/13801.

[40] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography 2007*, pages 330–342. Springer, 2007.

[41] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35:61–70, December 1992. ISSN 0001-0782. doi: 10.1145/138859.138867.

[42] Oded Goldreich. Foundations of cryptography: a primer. *Foundations and Trends in Theoretical Computer Science*, 1:1–116, April 2005. ISSN 1551-305X. doi: 10.1561/0400000001.

[43] Georg Groh and Christian Ehmig. Recommendations in taste related domains: Collaborative filtering vs. social filtering. In *In Proc ACM Group '07*, pages 127–136, 2007.

[44] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. Shilling attacks against recommender systems: a comprehensive survey. *Artificial Intelligence Review*, pages 1–33, 2012. ISSN 0269-2821. doi: 10.1007/s10462-012-9364-9.

[45] Ihsan Gunes, Alper Bilge, Cihan Kaleli, and Huseyin Polat. Shilling attacks against privacy-preserving collaborative filtering. *Journal of Advanced Management Science*, 1(1):54–60, March 2013.

[46] Ido Guy, Naama Zwerdling, David Carmel, Inbal Ronen, Erel Uziel, Sivan Yogev, and Shila Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, pages 53–60, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-435-5. doi: 10.1145/1639714.1639725.

[47] Jonathan Herlocker, Joseph Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999.

[48] Thomas Ryan Hoens, Marina Blanton, and Nitesh Chawla. A private and reliable recommendation system for social networks. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 816–825, August 2010. doi: 10.1109/SocialCom.2010.124.

[49] Thomas Ryan Hoens, Marina Blanton, and Nitesh Chawla. Reliable medical recommendation systems with patient privacy. In *Proceedings of the 1st ACM International Health Informatics Symposium*, IHI '10, pages 173–182, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0030-8. doi: 10.1145/1882992.1883018.

[50] Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999*. The Internet Society, 1999.

[51] Jerry Kang. Information privacy in cyberspace transactions. *Stanford Law Review*, 50(4):1193–1294, 1998. ISSN 00389765.

[52] Florian Kerschbaum, Debmalya Biswas, and Sebastiaan de Hoogh. Performance comparison of secure comparison protocols. In *Proceedings of the 20th International Workshop on Database and Expert Systems Application*, pages 133–136, 2009.

[53] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security (CANS)*, pages 1–20, 2009. doi: 10.1007/978-3-642-10433-6_1.

[54] Ioannis Konstas, Vassilios Stathopoulos, and Joemon M. Jose. On social networks and collaborative recommendation. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 195–202, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-483-6. doi: 10.1145/1571941.1571977.

[55] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009. doi: 10.1109/MC.2009.263.

[56] Shyong Lam, Dan Frankowski, and John Riedl. Do you trust your recommendations? an exploration of security and privacy

issues in recommender systems. In Günter Müller, editor, *Emerging Trends in Information and Communication Security*, volume 3995 of *Lecture Notes in Computer Science*, pages 14–29. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-34640-1.

[57] Ken Lang. Newsweeder: Learning to filter netnews. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann, 1995.

[58] Kristina Lerman. Social networks and social information filtering on digg. *Computing Research Repository (CoRR)*, abs/cs/0612046: 1–8, 2006.

[59] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *Proceedings of the Practice and theory in public key cryptography, 11th international conference on Public key cryptography*, PKC'08, pages 360–379, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-78439-X, 978-3-540-78439-5. doi: 10.1145/1793774.1793804.

[60] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology, CRYPTO – 00*, pages 36–54, 2000.

[61] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Proceedings of the 11th international conference on Topics in cryptology: CT-RSA 2011*, CT-RSA'11, pages 319–339, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-19073-5. doi: 10.1145/1964621.1964651.

[62] Fabiana Lorenzi and Francesco Ricci. Case-based recommender systems: A unifying view. In *Intelligent Techniques for Web Personalization*, volume 3169 of *Lecture Notes in Computer Science*, pages 89–113. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-29846-5.

[63] Ashwin Machanavajjhala, Aleksandra Korolova, and Atish Das Sarma. Personalized social recommendations: accurate or private. *Proceedings of the VLDB Endowment*, 4(7):440–450, April 2011. ISSN 2150-8097.

[64] Judith Masthoff and Albert Gatt. In pursuit of satisfaction and the prevention of embarrassment: affective state in group recommender systems. *User Modeling and User-Adapted Interaction*, 16: 281–319, 2006. ISSN 0924-1868. doi: 10.1007/s11257-006-9008-3.

[65] Frank McSherry and Ilya Mironov. Differentially private recommender systems: building privacy into the netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636, 2009.

[66] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology*, 7(4), October 2007. doi: 10.1145/1278366.1278372.

[67] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1004-8. doi: 10.1145/2046660.2046682.

[68] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR: Computing Research Repository*, pages 1–24, 2006.

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

[70] Michael O'Mahony, Neil Hurley, and Guénolé Silvestre. Collaborative filtering - safe and sound? In Ning Zhong, Zbigniew W. Raś, Shusaku Tsumoto, and Einoshin Suzuki, editors, *Foundations of Intelligent Systems*, volume 2871 of *Lecture Notes in Computer Science*, pages 506–510. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20256-1. doi: 10.1007/978-3-540-39592-8_72.

[71] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology, EUROCRYPT – 99*, pages 223–238, 1999.

[72] Leysia Palen and Paul Dourish. Unpacking "privacy" for a networked world. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 129–136, New York, NY, USA, 2003. ACM. ISBN 1-58113-630-7. doi: 10.1145/642611.642635.

[73] Andreas Peter, Erik Tews, and Stefan Katzenbeisser. Efficiently outsourcing multiparty computation under multiple keys. *IEEE Transactions on Information Forensics and Security*, 8(12):2046–2058, 2013.

[74] Huseyin Polat and Wenliang Du. Svd-based collaborative filtering with privacy. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 791–795, 2005.

[75] Huseyin Polat and Wenliang Du. Privacy-preserving top-n recommendation on distributed data. *Journal of the American Society for Information Science and Technology*, 59:1093–1108, 2008.

[76] Naren Ramakrishnan, Benjamin Keller, Batul Mirza, Ananth Grama, and George Karypis. Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54–62, November 2001. ISSN 1089-7801. doi: 10.1109/4236.968832.

[77] Jasson Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719, 2005.

[78] Paul Resnick and Rahul Sami. The influence limiter: provably manipulation-resistant recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 25–32, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-730–8. doi: 10.1145/1297231.1297236.

[79] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, 1994.

[80] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3 (4):329–354, 1979. ISSN 0364-0213. doi: 10.1016/S0364-0213(79) 80012-9.

[81] Ronald Rivest, Adi Shamir, and David Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, Cambridge, MA, USA, 1996.

[82] Alon Schclar, Alexander Tsikinovsky, Lior Rokach, Amnon Meisels, and Liat Antwarg. Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, pages 261–264, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-435-5. doi: 10.1145/1639714.1639763.

[83] 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*, RecSys '09, pages 157–164, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-435-5. doi: 10.1145/1639714.1639741.

[84] Rashmi Sinha and Kirsten Swearingen. Comparing recommendations made by online systems and friends. In *In Proceedings of the DELOS-NSF Workshop on Personalization and Recommender Systems in Digital Libraries*, 2001.

[85] Thijs Veugen. Improving the dgk comparison protocol. In *Information Forensics and Security (WIFS), 2012 IEEE International Workshop on*, pages 49–54, 2012. doi: 10.1109/WIFS.2012.6412624.

[86] Patricia Victor, Martine Cock, and Chris Cornelis. Trust and recommendations. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 645–675. Springer US, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_20.

[87] Stefan Weiss. The need for a paradigm shift in addressing privacy risks in social networking applications. In *The Future of Identity in the Information Society*, volume 262, pages 161–171. IFIP International Federation for Information Processing, 2008. doi: 10.1007/978-0-387-79026-8_12.

[88] Rongjing Xiang, Jennifer Neville, and Monica Rogati. Modeling relationship strength in online social networks. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 981–990, New York, NY, USA, 2010. ACM. doi: 10.1145/1772690.1772790.

[89] Andrew Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

IPA DISSERTATION SERIES

Stream Processing Systems. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. VAN DER HORST. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. GRAY. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. CALAMÉ. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. MUMFORD. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. DE GRAAF. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. BRIJDER. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. KOPROWSKI. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. KHADIM. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. MARKOVSKI. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. KASTENBERG. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. BUHAN. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. MARIN-PERIANU. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. VERHOEF. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. DE MOL. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. LORMANS. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. VAN OSCH. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. SOZER. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. VAN WEERDENBURG. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. HANSEN. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. MESBAH. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. RODRIGUEZ YAKUSHEV. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. OLMOS JOFFRÉ. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. VAN DEN BERG. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. KHATIB. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. CORNELISSEN. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. BOLZONI. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. JONKER. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. CZENKO. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. CHEN. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. KALISZYK. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'CONNOR. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. PLOEGER. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. HAN. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. LI. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. KWISTHOUT. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. COCX. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. BAARS. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. DEKKER. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. LAROS. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. BOOGERD. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. NEUHÄUSSER. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. ENDRULLIS. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. STAIJEN. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. WANG. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. BERENDSEN. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. NUGROHO. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. SILVA. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. DE BRUIN. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. COSTA. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. JAGHOORI. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. BAKHSHI. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. ARNOLDUS. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. ZAMBON. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. ASTEFANOAEI. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. PROENÇA. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. MORALI. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. VAN DER BIJL. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. KRAUSE. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. ANDRÉS. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. ATIF. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. VAN TILBURG. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. PROTIC. *Configuration management for models: Generic methods for model comparison and model *co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. GEORGIEVSKA. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. MALAKUTI. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. RAFFELSIEPER. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. TSIROGIANNIS. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. MOON. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. MIDDELKOOP. *Capturing and Exploiting Abstract Views of States in OO Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. VAN AMSTEL. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. TAMALET. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. BASTEN. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

M. IZADI. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. KATS. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. KEMPER. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. WANG. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. KHOSRAVI. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. MIDDELKOOP. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

Z. HEMEL. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. DIMKOV. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. SEDGHI. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. HEIDARIAN DEHKORDI. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. VERBEEK. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. NADALES AGUT. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

H. RAHMANI. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. VERMOLEN. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. ENGELEN. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. STAPPERS. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. HEIJSTEK. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. KOP. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. OSAIWERAN. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. KUIJPER. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. BEOHAR. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. IGNA. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. ZAMBON. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. LIJNSE. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. DE KONING GANS. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. GREILER. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. MAMANE. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. VAN DEN HEUVEL. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. BUSINGE. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. VAN DER BURG. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. KEIREN. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. GERRITS. *Pushing and Pulling: Computing push plans for* *disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. TIMMER. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. ROELOFFZEN. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. LENSINK. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. TANKINK. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. DE GOUW. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. VAN DEN BOS. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. HADZIOSMANOVIC. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. JECKMANS. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

THESIS PROPOSITIONS

1. Secure two-party computation computes functions with limited overhead, until you have to compare two values. Then the overhead quickly grows beyond a practical limit. (Chapter 4)

2. Privacy is about giving a user control over his data, privacy policies are about giving that control to the service provider. (General)

3. Additive blinding of values often results in simpler solutions compared to multiplicative blinding. (Chapter 3)

4. Service providers do not let company secret information leave their network, so similarly users should not give their private information to service providers or other unfamiliar entities. (General)

5. Prototyping not only gives performance estimates to the reader, it also increases the understanding of the researcher. (Chapter 5)

6. Cars get you from A to B. Good cars get you from A to B fast. Great cars just get you into trouble. (Redline, 2007)