University of Twente

Master Thesis

Privacy-Preserving Social DNA-Based Recommender

Author:
Inés Carvajal Gallardo

Supervisor:
Dr. Andreas Peter

July 29, 2015

Abstract

Recommender systems are generally used to generate recommendations for items based on user tastes. Recommender systems that provide privacy of user ratings and recommendation requests are called privacy-preserving. In this thesis, we research how to design and implement a privacy-preserving recommender system that uses DNA-similarity as a basis for the recommendation generation. The use of DNA-similarities between users of a recommender system is an interesting problem, because it has many relevant applications. An example of this is online dating platforms that use DNA-matching algorithms to determine compatibility between users. The DNA similarities are computed in a privacy-preserving way between users of the system. This similarity score is combined with familiarity links between a user and his friends in a social network to recommend items based on the friends' preferences, thereby using a social filtering approach to recommend items.

We first design our recommender system in the semi-honest user model, in which users and a central server that assists in computations are assumed to be semi-honest. Users may be offline during recommendation generation, which means that these computations are done on behalf of other users. We then research the malicious user model, in which users are no longer assumed to be semi-honest. To enable our recommender system in this security model, we employ a second server and require that the two servers are semi-honest and non-colluding.

The encyption scheme that is used in the system is a somewhat homomorphic encryption scheme, which enables homomorphic addition and multiplication of ciphertexts up to a certain limit of allowed operations. Proxy re-encryption and additive secret sharing are used to split user data into secret shares, where one share of user data can be re-encrypted to another party that performs computations on behalf of the user.

We design and implement privacy-preserving protocols for computing the similarity scores, the recommender protocol and a rating update protocol with which users can update their ratings in a privacy-preserving way. All protocols have a variant for both security models (with exception of the rating update protocol). We determine two similarity scores that fit the design of our recommender system, called the edit distance and the Smith-Waterman score. Analysis of the performance of both protocols shows that the edit distance protocol is much more efficient and is the preferred choice for similarity computations.

A possible and important application of the developed recommender system could be a social network of members in a self-help group who can get drug recommendations, including non-prescribed drugs, based on other members' experiences, where the accuracy of the recommendation depends on their genetic similarity.

Contents

1 Introduction

Recommender systems are commonly used to gather recommendations for books, movies or music for example. People use these systems to add ratings for items and to get recommendations for items they might be interested in. These recommendations are generated using the ratings supplied by other users that have similar tastes (this is the case in a similarity-based recommender system) or that are connected to the user through a social network.

In a similarity-based recommender system, the recommendations are made based on a similarity measure between users, most commonly similar tastes. For example, in a book-related domain, ratings of other users that have given similar ratings to books as the user who is asking for a recommendation, will have a greater weight in the recommendation. This approach often has the drawback of requiring a lot of input for each recommendation, since all other users in the system are considered.

If the acquantainces that link users in a social network are used for generating recommendations, then the recommender system is said to be familiarity-based. In other words, a familiarity between two users exists if they are acquaintances in a social network and these familiarities form the basis for the recommendations in the recommender system. Users get recommendations based on the ratings of people they know. The reasoning behind this is that users will generally want to have recommendations based on their friends' ratings and this approach has shown to give results that are comparable in accuracy to the approach used in similarity-based recommender systems [GE07, Ler07, SS01, GZC+09]. We will refer to a familiarity-based recommender system as a social recommender system.

Many recommender systems have been proposed that preserve the users' privacy. This is an important topic, because users might not want their friends or complete strangers to know which ratings they gave to certain items. If we think of a recommender system for drugs, users might not even want anyone to know that they use a certain type of drug. Therefore commonly, the generated recommendations (based on possibly sensitive input from other users) and the inquiry after certain recommendations are the information that has to be kept private.

There are, however, application domains for recommender systems where not only the content that is recommended, but also the similarity on which recommendations are based needs to be protected. An example of such an application is a recommender system where DNA similarity is taken as the weight for recommendations. DNA profiles are stored more and more fre-

quently, by governments and private companies alike, but the high sensitivity of this data calls for a way to compute DNA similarities in a privacy-sensitive way. DNA data can contain markers for diseases for example [BKKT08], or for drug allergies [ER04]. It can also be used to test for family relations. It is therefore necessary to keep DNA data that is used in a recommender system private, because it may contain information that individuals would not want their government, health care insurance company or employer to know.

This study focuses on the design of a recommender system that uses DNA similarities between users as a weight for the recommendations, while also using the familiarity links from social networks as a basis for recommendations. By using the the familiarity between users from a social network when generating recommendations, we hope to design an efficient recommender system. Using DNA-similarity as a similarity measure on top of this familiarity will offer new possibilities for the use of recommendations. The familiarity and similarity between users will be combined by generating a recommendation over the friends of a user, while using their DNA-similarities as weights for the relevance of each friend's ratings.

The use of DNA-similarity between users in a recommender system offers new possibilities for the use of the recommender system. A setting in which this social DNA-based recommender system could be used can for example be a social network of members of a self-help group, who give and get drug treatment recommendations for a specific illness. Members of self-help groups may already be giving drug recommendations, for non-prescribed drugs especially. In a privacy-preserving recommender system, the members of the self-help group will now be able to give and get ratings for (non-prescribed) drugs that take genetic factors into account for the expected drug treatment outcome. The drug recommendations in settings like this will be done based on a DNA similarity between users of the system, where the motivation for taking DNA similarity as a weight comes from the fact that similarities in DNA can predict similar responses to drugs. The familiarity between users of the system in this example lies in their visiting the same self-help group. This setting brings us to the following general problem statement.

> Can a privacy-preserving and efficient social DNA-based recommender system be designed for a (medical) setting, where users can privately share drug treatment experiences and get private recommendations based on DNA-similarity to other users in their social network?

The recommender system which we will design in this study will be an extension to a previous recommender system designed by Jeckmans et al. [JPH13], which introduced the use of somewhat homomorphic encryption to build an efficient privacy-preserving recomender system. We will use a somewhat homomorphic encryption scheme (namely that of [BV11]) to get an efficient homomorphic encryption in which a limited number of additions and multiplications is allowed. We will follow the same basic architecture for the recommender system, where a central server does most of the computational work on behalf of the users.

For the security model of the recommender system, we first consider semi-honest users and a semi-honest server. We then extend the design of the system to the malicious user model, where users are allowed to deviate from protocols and collude with other users. A second server will be introduced under an assumption of non-collusion between the two servers, who are both still semi-honest.

All protocols for the recommender system and the privacy-preserving DNA-matching will be implemented and analyzed for efficiency in both the semi-honest security model and the malicious user security model.

Contribution

This research contributes to the field of private recommender systems a new type of recommender system that is DNA-based, which means that the basis for recommendations lies in DNA-similarity of the users. We design privacy-preserving protocols using somewhat homomorphic encryption for the computation of DNA-similarities and for an offline recommender, both in a semi-honest user model as well as in a malicious user model. We design a privacy-preserving protocol for rating updates in the malicious user model. Though most of the protocols are based on existing protocols, they are altered to fit the requirements of our recommender system and the use of somewhat homomorphic encryption in the context of privacy-preserving DNA-matching is new. The results of this research are of importance to real life applications where DNA-matching is used in privacy-sensitive environments. We provide an informal security proof for the rating updates protocol, which is not based on any existing protocols. For the other protocols, security is deduced from the security of the underlying protocols.

The experimental results of the performance of our recommender system are overall very promising.

The similarity computations can be performed in a precomputation phase during system set-up, but at this point will be inefficient for real-time use.

This is not a huge drawback, since the recommender system is designed in such a way that all of these similarity computations can be done during the set-up phase of the system. For the similarity computations, the semi-honest versions of the protocols perform better than the malicious versions. Another contribution is that by using somewhat homomorphic encryption in privacy-preserving DNA-matching protocols and analyzing the performance of the resulting protocols, we determined the applicability and performance of using somewhat homomorphic encryption in this context, which has not been done before.

For the recommender protocol, the malicious version performs notably better than the semi-honest version of the protocol, which makes it the better choice. This has some important advantages; the malicious user model is stronger in security than the semi-honest user model and the protocol in the malicious user model removes a lot of computational load from the users of the system.

In this research, we succeeded in extending the work by Jeckman's et. al [JPH13] to the malicious user model. The design of the recommender protocol in the malicious user model can be viewed independently from the DNA-matching as a contribution as well.

The performance of the rating updates protocol, which was designed in the malicious user model, is very efficient and can be run during real-time use of the system.

2 Related Work

In this section, we will present the current research on recommender systems and privacy-preserving computations on DNA.

We consider the current research on privacy-preserving recommender systems that are either similarity or familiarity-based, since we intend to combine these techniques in the design of a recommender system. A recent study by Jeckmans et al. [JPH13] used a specific type of homomorphic encryption in their recommender system, which we plan to use in our system as well. Therefore, we also discuss the different cryptographic techniques that can be used for preserving privacy in a recommender system, focusing on the technique used by Jeckmans et al. Related work on privacy-preserving recommender systems is presented in section 2.1.

Since we plan to use DNA-similarities in the computation of recommendations, we look into privacy-preserving techniques for DNA-matching and present an overview of relevant research carried out in this field, which can be found in section 2.3.

Then, section 2.4 reviews the current research in the field of pharmacogenmics, which motivates the study of a DNA-based recommender system.

2.1 Recommender Systems

Recommender systems are systems that recommend content to users which they might be interested in. In this section, we consider collaborative filtering recommender systems, which are the recommender systems that generate recommendations based on ratings by other users that have similar tastes as the user requesting the recommendation. Specifically, we only consider recommender systems that are privacy-preserving, which means that the ratings of users are kept private and that the request for a recommendation remains private. For an overview of types of recommender systems, refer to Figure 1. (Note that this is not an exhaustive classification of types of recommender systems, but rather one that is used in this thesis.) The privacy-preserving social DNA-based recommender system that we intend to design falls in both the similarity-based and familiarity-based categories and will therefore combine both types of recommender systems.

2.1.1 Private recommender systems

Private recommender systems have been researched and implemented in a number of ways. Almost all of the private recommender systems that exist use additive public-key homomorphic encryption to enable computation

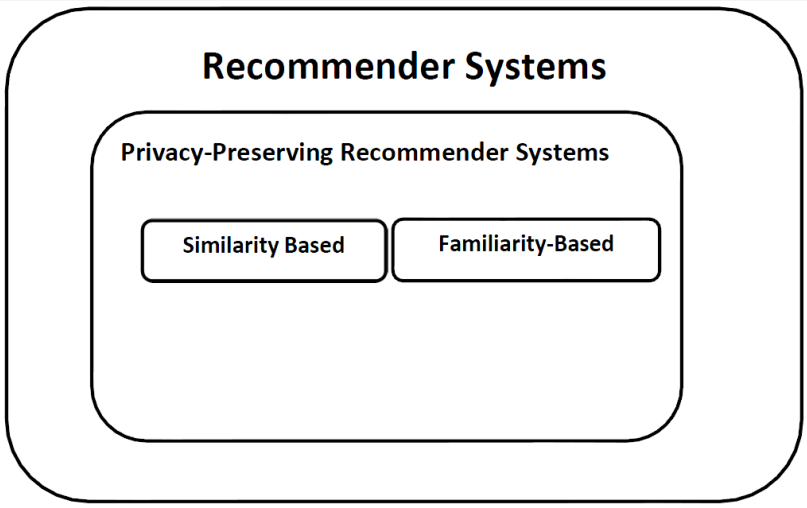

Figure 1: Recommender Systems

of recommendations in a private manner. Additive (public-key) homomorphic encryption has the following useful property: if two ciphertexts are encrypted under the same public key, then addition of the ciphertexts results in a ciphertext which is the encryption of the two added plaintexts. This can be expressed as: $E(x)_{pk} + E(y)_{pk} = E(x + y)_{pk}$, where x, y are plaintexts and pk is the public key.

A study by Canny et al. [Can02] shows an algorithm by which users can aggregate their data without exposing their inputs. Their scheme is a practical implementation of multi-party computation, users compute an aggregate that is publicly available and is composed of all users' data, which allows them to compute personal recommendations locally in a private way. Additive homomorphic encryption is used to apply collaborative filtering, the aggregate is calculated through iterations in which user data has to be added.

Erkin et al. [EVTL12] also used additive homomorphic encryption (using Paillier's scheme [Pai99]) to generate recommendations and introduced a trusted third party and data packing. The trusted third party, the privacy service provider (PSP), performs computations but is not allowed to obtain private data. In Erkin et al.'s approach, a minimum of user interaction is preferred. Their protocol is faster than the protocol proposed by Canny et al., but is still not very efficient. The protocol is also not dynamic (updating

ratings will cause the trusted third party and a service provider to start the whole protocol anew, causing lots of computations to be done again) and the use of a trusted third party would not be preferable if it can be avoided.

Hoens et al. [HBSC13] developed a recommender system in which patients can inquire after or give doctor recommendations, while keeping the recommendations private. This recommender system offers two different ways to keep information private. One of these ways is by data perturbation, the other is by using homomorphic encryption (again, Paillier's scheme is used). The Secure Processing Architecture is proposed by Hoens et al. in which additive homomorphic encryption is used in combination with secure multi-party computation with a certain threshold. The paper also proposes the use of Zero Knowledge Proofs to prove the validity of inputs. The practical implementation of this architecture, however, is too slow to use without a lot of pre-computation, because the time it takes to make recommendations is in the order of hours.

The study by Jeckmans et al. [JPH13] designed two protocols for a privacy-enhanced familiarity-based recommender system that used a different manner of encryption compared to what was used in most of the existing privacy-preserving recommender systems. Their protocols are more efficient than existing protocols such as the one proposed by Canny et al. or Erkin et al., because firstly, it is a familiarity-based recommender system. This means that the costly computations that are needed to determine similarity between users can be left out. Instead, social connections are used to determine which users are considered for the recommendation computation. Secondly, the scheme does not use additive homomorphic encryption, but rather somewhat homomorphic encryption. A strong point of the system they propose is that users can supply the weights that are taken into account for the recommendations. Jeckmans et al. designed two protocols, one in which users need to be online to perform the computations and one in which users may also be offline. This last protocol is the most practical protocol, because it would be unrealistic to expect users of a recommender system to be on-line all the time. The protocols are proven to be secure in the semi-honest attacker model. Since the research done by Jeckmans et al. comes close to what is needed for the proposed social DNA-based recommender system, it will be discussed in more detail here. In section 2.1.2, more will be said on the concept of familiarity-based recommender systems. Section 2.2 will go into more detail on the somewhat homomorphic encryption scheme used in their protocol. Below, the protocol for offline friends will be discussed.

The recommendation formula that Jeckmans et al. designed for their recommender system is the following:

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

Here, $p_{u,b}$ is the recommendation for user u for book b. The summation is taken over all of user u's friends, which are indicated with the symbol f. $q_{f,b}$ indicates whether friend f rated book b. If he did, the value is equal to 1, otherwise it is equal to 0. $r_{f,b}$ is the rating for book b by friend f. The values $w_{u,f}$ and $w_{f,u}$ are the weights that user u gives to friend f and reversed, they define the importance of the friend's input for the user and the importance of the user's input to the friend.

A formula similar to equation 6 could be used in the proposed setting of a medical recommender system that recommends drugs, where not the value of a friend's opinion is used as a weight, but where DNA-similarity is taken into account. This DNA-similarity would then be used instead of the sum $w_{u,f} + w_{f,u}$. The user will then also not have to provide these weights.

The two protocols that are presented, one for on-line friends and one for offline friends, are very similar. The protocol for offline users is an extension of the protocol for on-line users. It has been proven to be secure and is the one of the most efficient privacy-preserving recommender protocols that exist up till now.

A server is used on which encrypted information of users is stored. This encrypted information can be used for the recommendation computations. Proxy re-encryption is applied to re-encrypt information that a user stored for one of his friends, who does not need knowledge of the secret key of the user in order to decrypt the information.

The protocol for offline friends splits the rating vector value R_f (composed of the values $r_{f,b}$ for all books) into two secret shares S_f and T_f with an additive secret sharing scheme. The weight $w_{f,u}$ is also split into two secret shares $x_{f,u}$ and $y_{f,u}$. At the server, one of the rating shares and one of the weight shares are stored. The other shares are stored under encryption at the server, so that the server will not be able to reconstruct the rating vector or the weight value.

In order to compute the rating value in equation 6, both the server and the user compute the sum of the weights, by adding the user's weight value $w_{u,f}$ and the two shares of the friend's weight value $w_{f,u}$ under somewhat homomorphic encryption. To do so, the user sends $[w_{u,f} + y_{f,u}]_u$ to the

server, encrypted under his own key and the server sends $[x_{f,u}]_s$ to the user, encrypted under the server's key. When all values are added, the sum of the weights has been computed. Addition can be done under encryption at both the user and the server.

Then, for each book, the corresponding rating value from the friend's rating vector is taken. The user has a share through proxy re-encryption and the server also has a share of this rating value. They multiply these shares with the combined weight that was previously computed and sum the resulting value for all books, resulting in the values $[z_b]_s = \sum_{f=1}^{F_u}[w_{u,f} + w_{f,u}]_s * t_{f,b}$ and $[a_b]_u = \sum_{f=1}^{F_u}[w_{u,f} + w_{f,u}]_u * s_{f,b}$. The user sends his sum $[z_b]_s$ (that is blinded additively) to the server, along with an unblinding value that is encrypted under the user's key (this has the consequence that the value z_b can only be unblinded under encryption with the user's key).

The user then computes and sends a normalization weight per book: $[d_b]_s = \sum_{f=1}^{F_u}[w_{u,f} + w_{f,u}]_s * q_{f,b}$. These values are blinded by multiplication. An encryption for removing the blinding as well as the blinded normalization values are sent to the server.

The server removes the blinding from the value z_b that was encrypted under the server's key to get $[z_b]_u$. He adds this to his own summed value to get $[n_b]_u$. For the normalization weights that were sent after being multiplicatively blinded, the server decrypts the encryptions and inverts the blinded normalization weights. Then the blinding is removed under the user's key, so that the inverted normalization weights are computed. The value $[n_b]_u$ is then multiplied with the inverted normalization weight for the book b to get the recommendation value $[p_{u,b}]_u$ and this value is sent to the user, who can decrypt it.

2.1.2 Similarity- and familiarity-based recommender systems

Many recommender systems are similarity-based. They use collaborative filtering and perform expensive computations to compute which users are similar to each other. Jeckmans et al. [JPH13] used familiarity instead of similarity as a weight for generating recommendations in their protocol. They point out that previous studies [GE07, Ler07, SS01, GZC+09] have shown that using familiarity between users in a social network instead of similarity gives comparable results if the recommender system is used in a taste-related domain. The benefit of using familiarity instead of similarity, is that is computationally less expensive. Jeckmans et al. implement an efficient protocol for generating recommendations with this approach.

When considering using DNA similarity as a weight for recommenda-

tions, it is not at first obvious that familiarity should also play a role in the recommender system. However, in certain settings this could be useful and improve the efficiency of the recommender system. For example, if the recommender system is used by self-help groups or by patients in a hospital then patients who visit the same doctor or members of the same self-help group form social networks. Users of the recommender system that are linked by familiarity can then be compared to each other using the DNA similarity to determine what treatment or what drug might be recommended for them. This may greatly reduce the overall computational power that is needed.

2.2 Somewhat homomorphic encryption versus additive homomorphic encryption

Several schemes for somewhat homomorphic encryption have been proposed in the past. Van Dijk et al. [vDGHV10] described a SWHE scheme that works over the integers, of which the security is reduced to the security of the hardness of the approximate integer greatest common divisors problem. However, their scheme is very noisy and inefficient for use in practical situations.

Brakerski and Vaikuntanathan described another SWHE scheme [BV11], that was proven to be semantically secure under the assumption of polynomial learning with errors. The scheme uses the ring $\mathbb{Z}_q[x]/\langle f(x)\rangle$ to represent encrypted messages. Outputting ciphertexts is done by multiplying the key with a random parameter and some noise and by adding the resulting value to the message. Decryption is done by performing a modulo operation. Keys can be generated by every party by first choosing a secret key and then adding some randomness to create the public key.

Jeckmans et al. [JPH13] used this SWHE scheme by Brakerski and Vaikuntanathan in their recommender system as opposed to additive homomorphic encryption which is used in most recommender systems (using a scheme such as Paillier's). Their implementation is much more efficient than the recommender systems that make use of additive homomorphic encryption, it runs in the order of minutes instead of hours. The resulting recommender system is secure in the semi-honest attacker model.

The advantage of using somewhat homomorphic encryption over additive homomorphic encryption schemes in a recommender system lies in the fact that it makes the recommender system more efficient. Most public-key additive homomorphic encryption schemes use exponentiations, which are costly computations.

One example of an implementation of a SWHE scheme exists in the HElib

library, which is an open-source C++ library and implements the BGV scheme by Brakerski et al. [BGV11]. Halevi and Shoup [HS14] described some of the algorithms and optimization techniques that are used in this library in their report on HElib.

The BGV scheme is a SWHE scheme based on the scheme by Brakerski and Vaikuntanathan, that implements some changes that increase performance and security. One of the main improvements is the use of modulus switching, a technique where noise in the ciphertext is reduced by transforming a ciphertext into another ciphertext under a different, smaller modulus, without losing any of the information contained in the ciphertext.

2.3 Privacy-sensitive DNA matching

2.3.1 Edit distance and Smith-Waterman distance

The edit distance and Smith-Waterman distance are two DNA-similarity measures that are commonly used to compute the similarity between two strings, that can be arbitrary strings or DNA sequences. The edit distance is defined as the minimum cost of transforming a string x into a string y with the operations deletion, insertion and subtitution. The Smith-Waterman distance is a more fine-grained similarity score. Gaps are used when computing this distance that represent empty spaces within strings, where deletions or insertions have taken place. By comparing segments of various lengths, the Smith-Waterman algorithm outputs a similarity score that takes into account similar regions in two sequences. Both the edit distance and the Smith-Waterman distance have been used in the past as similarity measures in privacy-sensitive DNA matching.

2.3.2 Techniques for privacy-preserving DNA computation

Privacy-preserving matching of DNA material is a topic that is becoming more interesting while new applications using DNA sequences are on the rise. Bruekers et al. [BKKT08] did a study on ways in which to use cryptography in order to design protocols for matching DNA that would preserve DNA privacy. Their study focused on the most common DNA tests, such as identity, paternity and ancestry tests. The paper deals with the protection of STR (Short Tandem Repeat) profiles that are used for identity tests, which are used commonly by the police for example. The need for privacy enhancing protocols to do identity tests or related tests lies in the fact that these STR profiles do not only hold information about a person's identity, but can also contain information about a pre-disposition to develop

a specific disease, or contain markers for likely drug allergies. Secure multi-party computation is used together with homomorphic encryption (Paillier's scheme for example) to solve this problem and the scheme is proven to be secure in a semi-honest attacker model.

Atallah et al. [AKD03] proposed a protocol for securely computing the edit distance between two sequences. This distance can be used on genome sequences in order to compute similarity and can be computed using dynamic programming.

Jha et al. [JKS08] also did a study on ways to compare strings, particularly DNA sequences, in a privacy-sensitive manner. Their approach, just like Atallah et al.'s, is more general than the protocols developed by Bruekers et al., since they can be used for any piece of genomic data to compute a similarity score, while the protocols that are proposed by Bruekers et al. are focused on paternity testing, ancestry testing or identity testing. Their study presents two distances that can be used to calculate the similarity between two sequences, namely the edit distance which was also studied by Atallah et al. [AKD03] and the Smith-Waterman similarity score, which can also be formulated with a recurrence relation

The cryptographic techniques that are used for these protocols are secure function evaluation, in which two parties can jointly compute a function while they preserve the privacy of their respective inputs, oblivious transfer, oblivious circuit evaluation and secure computation with shares. In oblivious transfer, one out of several inputs is sent to a receiver, without the receiver knowing which of the inputs was received and without the sender knowing which input was received by the receiver. An oblivious transfer of 1-out-of-n values is denoted by OT_1^n. The protocol uses the implementation by Naor-Pinkas [NP01] for this.

Oblivious circuit evaluation is based on Yao's garbled circuits [Yao86, LP09] and secure computation with shares. Inputs of two parties are evaluated on circuits that can be arithmetic or boolean, where inputs of both parties and internal circuit wire values remain secret to the other party. If Alice and Bob use oblivious circuit evaluation, then Alice generates two random keys for every circuit wire, where one key represents a 0 and the other represents a 1. The keys that represents Alice's input for each wire are transferred to Bob, who does not know what values the keys represent. For his own input on each wire, the oblivious transfer protocol OT_1^2 is used n times, if his input consists of n values. For each circuit gate, Alice will produce a garbled truth table: she encrypts the two output wire keys under the encryptions of the input keys for all possible inputs and outputs. These encryptions are randomly permuted. Bob can then decrypt only one output

wire key, since for each input wire he has one input key (Alice's or Bob's input keys). He then learns the mapping of the output wire keys, so that he knows what the output bit on that wire was (which was represented by the key). He learns nothing of the inputs. If a complex circuit is used, only the mapping of keys to values for the wires that represent the output of the entire circuit are revealed.

All of these techniques are used for the protocols for computing both the edit distance and the Smith-Waterman distance. There are several different implementations for evaluating both distance measures that were tested using random strings and real protein sequences, which perform better or worse in certain circumstances. The study shows that scores can be computed in the order of seconds and could therefore be applied in practical situations. For the computations for the edit distance or Smith-Waterman distance, an equality circuit is used (which compares two values to test for equality) and a minimum-of-three circuit is used (which computes the minimum of three values that are shared randomly between two participants).

Jha et al. also showed that the protocol for securely computing the edit distance that was proposed by Atallah et al. [AKD03] was not efficient at all, as it took minutes to solve a problem where the implementation by Jha et al. took only seconds.

Their study clearly does not use homomorphic public-key cryptography, but makes use of garbled circuits in order to keep inputs secret. This is one of the main reasons that the protocol is so efficient.

Another protocol for securely computing a special case of the edit-distance was proposed by Rane and Sun [RS10]. Theirs is an asymmetric protocol for calculating the edit distance in which a lightweight server and a powerful server jointly compute the similarity score. This asymmetric protocol may be a better fitting solution for recommender systems than the protocols proposed by Jha et al, because in the setting of recommender systems it would be preferred to have the server do part of the computations when other users are offline.

The main cryptographic primitives used in their implementation are additive secret sharing and semantically secure additive homomorphic encryption. The cryptographic building blocks that are used in the protocol are a private substitution cost protocol and a privacy-preserving minimum finding protocol for two parties. The private substitution cost protocol is used to compute the substitution costs for substituting one character by another in a privacy-preserving manner. Three different common substitution cost protocols are given by the authors, suchs as a protocol for absolute distance, for polynomial cost and for an indicator function cost. The client in this

protocol computes an encrypted matrix with intermediate edit-distance values, for subsequences of the strings that are compared. The matrix L is used, where $L(i, j)$ is the edit distance between substrings of lengths i and j. The server only learns the total edit distance that is computed through these subproblems of the edit-distance problem. The client keeps a table of insertion costs I and the server keeps a table of deletion costs D, which are both needed for computing the intermediate values in the encrypted matrix. It is possible to use parallelization on multiple servers in a secure manner.

Blanton et al. [BAFM12] did a study on secure and efficient outsourcing of sequence comparisons, which also uses Yao's garbled circuits as the main technique for speeding up computations. They presented a framework in which a client can send two strings to two remote servers who compute the edit distance between these strings, without revealing any information to the servers about the input strings or the outcome of the protocol. They assume that the servers are non-colluding. Their protocol is mostly focused on computing the edit script, which contains information on the operations performed for the optimal edit distance. The protocol is therefore less suited for the medical recommender system that is considered in this literature study.

In a study by Ayday et al. [ARHR13], a new architecture for genetic disease susceptibility tests was proposed based on patient's genomic data for 'privacy-preserving disease susceptibility test' (PDS), in which homomorphic encryption and proxy re-encryption are applied. A storage and processing unit (SPU) stores the genomic data of patients while preserving the genomic privacy and can perform tests on parts of the DNA-data to conduct genetic tests. They use rather specific operations for computing the likelihood of a patient developing a certain disease, tailored to their system architecture, while in our system we aim to keep the DNA-matching very general by computing similarities between patients, but not focusing on disease susceptibility in particular. However, their system is still very similar to our envisioned recommender system regarding the setting and architecture. The security of the patients' genomic data is also somewhat different from the security that we aim to achieve in this research, since the results of the disease susceptibility tests are known to a medical unit who requests the tests and can therefore give information about the patient's genome.

2.4 Drug selection based on DNA

DNA sequences can be used to predict a person's reaction to treatment with a specific drug. The concept of personalized medicine, choosing a certain

drug that will result in the most effective treatment, is very important in the scenario of a medical recommender system where medication can be recommended based on DNA-similarity. This section gives an overview of studies that focus on personalized medicine to get a better understanding of how personalized medicine works and what should be taken into consideration for a social DNA-based recommender system.

Evans and McLoad [EM03] wrote a review article on pharmacogenomics, an area of study that researches differences in responses to drugs between individuals in a population, based on their DNA. The review treats several examples that show how pharmacogenomics can be used to improve drug therapy through molecular diagnostics. The examples show that different reactions to drugs are the result of differences in variations of the genes that encode drug-metabolizing enzymes, drug targets and drug transporters. Single-nucleotide polymorphims (SNPs) are associated with different effects of medication on different individuals and can be used to predict clinical responses. SNPs are DNA sequence variations that occur commonly in a population. They are variations (polymorphisms) of the DNA where only one nucleotide in a sequence of nucleotides differs. Apart from being used for determining the effects of medication, SNPs are also used for other purposes, such as determining kinship.

A more recent review article on the current state of pharmacogenomics was published by Evans and Relling [ER04]. They described an example regarding one of the most common single-gene traits: thiopurine S-methyltransferase, referred to as TPMT. Patients who have non-functional TPMT alleles (they have inherited a certain TPMT polymorphism that does not work as it does for persons with the standard TPMT allele) who are treated with drugs for neoplasias, for example, can develop haematopoietic toxicity, which can be a life-threatening condition. If it is found out that patients have this TPMT deficiency by inspecting their genomic data, they can be treated with lower doses of the drug which will prevent any harmful side-effects. There are known 'candidate genes' that can be used to predict a treatment outcome for a specific drug, these genes have polymorphisms that have been known to affect the drug disposition. The paper gives an overview of some of these genes and states that in some cases drug effects are inherited. At the time that the study was published, pharmacogenomics were not commonly used to individualize patient treatment, partly because drug effects do not only depend on genetic traits but also on drug interaction for example, which is why there had not been many experiments to prove that individualization of medication would improve treatment yet.

3 Research Goals

The goal of this research is to find a solution to the following general problem (which was also stated shortly in section 1):

> Can a private and efficient social DNA-based recommender system be designed for a (medical) setting, in which users of the system can privately share treatment experiences for drugs, where similarity between users' DNA is used to generate recommendations in combination with familiarity links between them. Privacy of treatments and drug ratings and of DNA data needs to be preserved in this recommender system.

The following concrete research questions will be answered in this thesis:

- *How can familiarity and similarity be combined in a recommendation protocol?*
  An equation will be formulated for the recommendation that incorporates both familiarity and similarity. The formula by Jeckmans et al. [JPH13] will be taken as a basis for this formula.

- *How can privacy of patients' DNA material be guaranteed and which method is best for computing DNA similarity?*
  There are several ways to preserve the privacy of DNA material while still being able to compute similarity measures. Ways to compute DNA similarity are for example the edit distance and the Smith-Waterman score. We use both of these measures as similarity scores and compare the efficiency of our implementations of these similarity scores to select the most suitable approach for privacy-preserving computation of DNA-similarity.

- *Can a protocol for offline users be devised, based on the protocol by Jeckmans et al. [JPH13] and what consequences will a protocol with offline users have for privacy requirements when user data has to be stored at a server?*
  As has been stated before, it would be practical to have a protocol for a social DNA-based recommender system for offline users. This means that users have to store (part of) their data at a server under encryption. This has consequences for the DNA-similarity computations and the privacy of these computations. Protocols such as the one by Jha et al. use garbled circuits and have been designed for two

individual parties that keep their own private data, and are therefore less suited to the context of our recommender system. The protocol by Jeckmans et al. will be extended to incorporate the computation of DNA-similarities and this new protocol will be proven to be secure.

- *Can an efficient implementation of a social DNA-based recommender be implemented and which library will be most suitable for this implementation?*
  One goal of this research is to implement the proposed social DNA-based recommender system. We consider a range of libraries that implement somewhat homomorphic encryption, such as the HElib [HS14] library, the jLBC library [1] and the implementation developed by Arjan Jeckmans[2], and compare them to find the one most suitable to our purposes. The implementation of the recommender system will be tested on random DNA-sequences and on randomly generated rating data to measure its performance and the system's performance will be compared to that of other private recommender systems, in particular the previous implementation by Jeckman's et al. [JPH13]. The performance results of the privacy-preserving DNA-matching will be analyzed to check the applicability and performance of using somewhat homomorphic encryption in this context.

- *What security requirements can we set for the recommender system and for the similarity computations? Is it possible to allow for the existence of malicious users?*
  As stated, we will extend the existing protocol by Jeckmans et al. [JPH13] to the setting of this research. This protocol was proven to be secure in the semi-honest attacker model. We will take this security model as a starting point for our system and then try to find a way to extend the system in such a way that malicious users can exist without introducing any risks to privacy or correctness of the results. In our design section, we will formulate the security requirements that the system must fulfill for both security models.

- *Can we develop a protocol for updating ratings in a privacy-preserving way?*
  A privacy-preserving protocol for rating updates would be necessary in any privacy-preserving recommender system, since users may want

[1] `http://gas.dia.unisa.it/projects/jlbc/`
[2] `http://scs.ewi.utwente.nl/other/jeckmanscode/`

to change ratings they entered into the system. We will therefore try to devise a protocol for this in such a way that a user's new rating remains private.

4 Preliminaries

Before going into detail on the design of our recommender system, we present the following (cryptographic) preliminaries that are necessary building blocks to our system.

4.1 Proxy re-encryption

Proxy re-encryption is a technique that can be used to allow a receiver to decrypt on behalf of a sender, without the receiver getting knowledge of the sender's secret key. Blaze et al. [BBS98] describe the concept of atomic proxy cryptography, where through an atomic proxy function a ciphertext under one key is translated to a ciphertext under another key, without the plaintext being exposed at any time during the operation. This is done using a proxy key, with which a message encrypted under one key can be re-encrypted to another key. Proxy re-encryption will be needed to share DNA material and drug ratings privately between users of the recommender system if we want to enable other users to be offline during the computation of recommendations. Data may then be stored at a central server that carries out protocols on behalf of a user's friends.

There are a few requirements to the proxy re-encryption scheme used. The scheme needs to be unidirectional and it needs to be one-hop, meaning that a proxy key from sender to receiver can be constructed from the sender's private key and the receiver's public key (thus not needing interaction with the receiver) and that the proxy re-encryption only works one time from the sender to the receiver. The receiver cannot re-encrypt the message again for a second receiver. This way, when a user gets data from his friends which is proxy re-encrypted, he cannot send it on to yet another friend if the two friends are not connected.

There are several schemes [AFGH06, LV11, CWYD10] that satisfy these requirements and which can be used with the recommender protocol outlined in the rest of this paper. For our recommender system, we will use the second scheme of Ateniese et al. [AFGH06], of which we now present a schematic overview.

The scheme

The scheme that was designed by Ateniese et al. [AFGH06], uses two groups $G1$ and $G2$ of prime order q and a bilinear map $e : G1 \cdot G1 \rightarrow G2$. Here, element G generates $G1$ and $e(G, G) = Z$.

The key-parameters for a user A are: $sk_A = a$, $pk_A = g^a$. A re-encryption key from A to B is computed as: $rk_{AB} = g^{b/a}$

A first-level encryption (which cannot be re-encrypted to another user) is described as follows:

$c_A = (Z^{ak}, m \cdot Z^k)$, where m is the message and k is randomly picked.

A second-level encryption (which can be re-encrypted to another user) is described as:

$c_A = (G^{ak}, m \cdot Z^k)$

To re-encrypt a second-level encryption $c_A = (G^{ak}, m \cdot Z^k)$ to a ciphertext c_B, the scheme uses the following computation:

$c_B = (Z^{bk}, m \cdot Z^k)$, where Z^{bk} is computed as: $e(G^{ak}, G^{b/a}) = Z^{bk}$

To decrypt a first-level encryption $c_A = (u, v)$ with secret key sk_A, compute:

$m = v/(u^{1/a})$

To decrypt a second-level encryption $c_A = (u, v)$ with secret key sk_A, compute:

$m = v/(e(u, G)^{1/a})$

This scheme was proven to be secure under the assumption that in (G_1, G_2), the problem of deciding whether $Q = e(g, g)^{a/b}$ is hard, when given (g, g^a, g^b, Q) for $g \leftarrow G_1, a, b, \leftarrow \mathbb{Z}_q$ and $Q \in G_2$.

Notation

The notation $[[x]]_y$ will be used throughout this paper to denote a encryption of x with the proxy re-encryption scheme with key y, which may belong to a user of the system or which may be a re-encryption key.

4.2 Secret sharing

A secret sharing scheme will be used in the recommender system to split data into two secret shares. Following the research by Jeckmans et al. [JPH13], we use an additive secret sharing [Gol05] scheme to secret share vectors of values. If $v = (v_0, \ldots, v_n)$ is a vector that needs to be secret shared, we choose a vector $s \in R_k$ randomly and compute $r = v - s$. Then, for all $0 \leq i \leq n$ we have $r_i + s_i = v_i$. The vectors r and s are the secret shares to v.

4.3 Somewhat Homomorphic Encryption

Somewhat homomorphic encryption (SWHE) schemes are public-key encryption schemes that allow for a limited number of operations of addition

and multiplication over bits that are encrypted. In these schemes there is a fixed degree of homomorphism. Each ciphertext contains some noise. As long as the noise is small enough, homomorphic operations can be applied to the ciphertext: $E(x) \cdot E(y) = E(x \cdot y)$ and $E(x) + E(y) = E(x + y)$. Noise grows faster with multiplication than with addition. At a certain point, the noise becomes too large and the permitted degree of homomorphism will be reached.

In our recommender system, the somewhat homomorphic encryption scheme by Brakerski and Vaikuntanathan [BV11] is used. The security of this encryption scheme is based on the polynomial learning with errors assumption (PWLE), which is a variant of the ring learning with errors assumption (RWLE) by Lyubashevsky et al. [LPR13]. The RLWE assumption states that if $s \in_r R_q := R_q/R$ (where R is the ring $\mathbb{Z}[x]/\langle f(x)\rangle$, $f(x)$ is a fixed n-degree cyclotomic integer polynomial in $\mathbb{Z}[x]$ and $q \in \mathbb{Z}$ a prime such that $q \equiv 1 \mod 2n$ then given any number of samples $(a_i, b_i = a_i \cdot s + e_i) \in (R_q)^2$ that are polynomial in the security parameter κ, where a_i are uniformly random in R_q and e_i are drawn from the error distribution χ, the b_i are computationally indistinguishable from uniform in R_q.

4.3.1 Notation

The notation $[x]_y$ is used throughout the paper to indicate somewhat homomorphic encryption of x under the public key belonging to y. For homomorphic addition of two messages m_1 and m_2, the notation $[m_1]_y + [m_2]_y = [m_1 + m_2]_y$ is used. For homomorphic multiplication of two messages, the notation $[m_1]_y \cdot [m_2]_y = [m_1 \cdot m_2]_y$ is used. Also, $[m_1]_y + m_2 = [m_1 + m_2]_y$ and $[m_1]_y \cdot m_2 = [m_1 \cdot m_2]_y$.

4.3.2 Parameter selection

In order to use the BV scheme, some public parameters must be specified. The message space of the scheme is $\mathbb{Z}_t[x]/\langle f(x)\rangle$. Here, $t \in \mathbb{Z}_q^*$ is a prime, $f(x)$ is an n-degree cyclotomic integer polynomial in $\mathbb{Z}[x]$ and $q \in \mathbb{Z}$ a prime such that $q \equiv 1(\mod 2n)$. The encrypted messages are in the ciphertext space $\mathbb{Z}[x]/\langle f(x)\rangle$.

M is an upper bound on the allowed number of homomorphic multiplications. An error distribution χ is taken over the ring R_q and is sampled from during key generation and encryption. In order to select a secure parameter set, so that the scheme is correct and secure against attacks, we will follow the approach by Naehrig et al. [NLV11]. They take χ as the discrete

Gaussian $D_{\mathbb{Z}^n,\sigma}$, where σ is the standard deviation of the error distribution and they choose n to be a power of 2.

Naehrig et al. [NLV11] formulate a correctness condition, which for fixed parameters t, σ, n, M and A gives a lower bound on the prime q in order to have a correct encryption scheme. Here, M is the allowed number of multiplications and A is the allowed number of additions. The bound is expressed as: $q \geq 4 \cdot (2t\sigma^2\sqrt(n))^{M+1} \cdot (2n)^{\frac{M}{2}} sqrt(A)$.

The combination of chosen parameters will define the security of the scheme. Specifically, the bound M that we use as a measure for the degree of homomorphism is determined by choice of f, q and χ. Refer to the paper by Naehrig et al. [NLV11] for details on specific parameter selections and their resulting security.

4.3.3 The Encryption Scheme

The SWHE scheme outlined in the paper by Brakerski and Vaikuntanathan [BV11] and in the paper by Naehrig et al. [NLV11] consists of the following routines:

SH.KeyGen A sample s is chosen from the error distribution χ and is set as the secret key $sk := s$. Now, a ring element a_1 is sampled from R_q and e is sampled from χ in order to construct the public key: $pk = (a_0, a_1) := (-(a_1 s + te), a_1)$.

SH.Enc To encrypt a message m from the message space R_t, sample u, f, g from χ. We get $ct = (c_0, c_1) := (a_0 u + tg + m, a_1 u + tf)$, which is a ciphertext belonging to the ring R_q^2. Due to homomorphic operations, the ciphertext can grow to become an element from R_q^d, where $d \leq M$. This is the standard form for ciphertexts: $ct = (c_0, \ldots, c_d)$.

SH.Dec For decryption of $ct = (c_0, \ldots, c_\delta)$ with secret key s, we use the following formula: $m = \sum_{i=0}^{\delta} c_i s^i \in R_q(\mod t)$, which is the original plaintext message m.

SH.Add We can add two ciphertexts $c = (c_0, \ldots, c_\delta)$ and $c' = (c'_0, \ldots, c'_\gamma)$ as $c + c' = (c_0 + c'_0, \ldots, c_{\max(\delta,\gamma)} + c'_{\max(\delta,\gamma)})$. This operation does not increase the number of elements in the ciphertext. The ciphertext containing the least elements is padded with zeroes.

SH.Mul We can multiply two ciphertexts $c = (c_0, \ldots, c_\delta)$ and $c' = (c'_0, \ldots, c'_\gamma)$ as follows: using a symbolic value v, the expression

$(\sum_{i=0}^{\delta} c_i v^i) \cdot (\sum_{i=0}^{\gamma} c'_i v^i) \equiv \sum_{i=0}^{\delta+\gamma} \hat{c}_i v^i$ with terms in R_q can be computed, such that $(\hat{c_0}, \ldots \hat{c_{\delta+\gamma}})$ is the resulting ciphertext (the product of the two original ciphertexts).

4.4 DNA similarity measures

The edit distance and Smith-Waterman distance are two similarity measures that are commonly used to compute the similarity between two strings, that can be arbitrary strings or DNA sequences.

4.4.1 Edit Distance

The edit distance is defined as the minimum cost of transforming a string x into a string y with the operations deletion, insertion and subtitution. If x and y are viewed as vectors of respectively n and m characters long, then $x = x_1...x_n$ and $y = y_1...y_m$. The edit-distance between the strings is then $L(n, m)$ for the input strings x and y, where n and m denote the number of characters that are considered for the edit-distance. So in general, $L(i, j)$ will be the edit distance between the substrings $x_1...x_i$ and $y_1...y_j$. Using this notation, the edit-distance can be formulated recursively as follows, for $1 \leq i \leq n, 1 \leq j \leq m$:

$$L(i,j) = min \left\{ \begin{array}{l} L(i-1,j)+1 \\ L(i,j-1)+1 \\ L(i-1,j-1)+S(x_i,y_j) \end{array} \right\} \tag{2}$$

Here, $S(x_i, y_j)$ denotes the cost for substituting the i-th element in string x by the j-th element in string y. The value of $L(0, 0) = 0$. The edit distance is at least 0, in the case that two strings are identical, and is at most $max(n, m)$, in the case that the two strings have no matching characters at the same position. To use the edit distance as a similarity weight between 0 and 1, we use the following formula:

$$s_{x,y} = 1 - \frac{L(n,m)}{\max(n,m)} \tag{3}$$

4.4.2 Smith-Waterman similarity

The Smith-Waterman distance is a more fine-grained similarity score. This measure is commonly used for sequence alignment to determine similar regions between two strings of nucleotides. Nucleotide segments of all possible lengths are compared to find the optimal alignment of the two sequences and

to determine the similarity score. It is defined by Jha et al. [JKS08] as follows, for $1 \leq i \leq n, 1 \leq j \leq m$:

$$H(i,j) = max \left\{ \begin{array}{l} 0, \\ max_{1 \leq o \leq i}(H(i-o,j) - g(o), \\ max_{1 \leq l \leq j} H(i,j-l) - g(l), \\ H(i-1,j-1) + c(x_i, y_j) \end{array} \right\} \tag{4}$$

Here, as before, x_i and y_j denote the i-th and j-th characters in the strings x and y. The Smith-Waterman distance between the strings x and y is $H(m,n)$. Gaps (usually denoted by the $-$ character) are used for the alignment and represent empty spaces within strings, where deletions or insertions have taken place. The function g is the gap-function which scores gaps and is defined as: $g(k) = x + y(k-1)$. This is the affine form of a gap scoring function. The term k is the size of the gap and the terms x and y are positive integer constants. The value of x determines the initial cost of the gap, the value of y determines the gap-cost per character. In equation 4 we see that the gap-score is always subtracted, this is because a gap due to an insertion or deletion decreases the similarity of the two strings. The function c is the cost-function that determines the cost of substituting one character by another and is defined as $c(a,b) = e$ if $a = b$ and $-f$ if $a \neq b$. The maximal value that the Smith-Waterman score can take is $e \cdot min(n,m) - |n-m|$, which only happens if the two strings are identical. The minimal value is 0, when there is no similarity between the sequences. To use the Smith-Waterman score as a similarity weight between 0 and 1, we use the following formula:

$$s_{x,y} = \frac{H(n,m)}{e \cdot \min(n,m) - |n-m|} \tag{5}$$

4.4.3 Technique for privacy-preserving computation

Both the edit-distance and the Smith-Waterman distance are considered as a similarity weight in our envisioned recommender system, see Section 5. While the edit distance would be enough to test for similarity between SNPs that are related to a set of drugs, having the Smith-Waterman score as a similarity weight has some advantages as well. A large amount of research still has to be done in the field of pharmacogenomics. The interaction between drug and multiple SNPs is not always known and some SNPs may not have been found yet [ER04]. It would therefore be too limiting to only consider the correspondence of a certain set of SNPs between individuals

to predict drug treatment effects. Also, the recommender system that is presented in this research generally takes DNA similarity as a weight for recommendations. The recommendations need not be drug treatment recommendations for patients, this is merely the setting that we keep in mind as we develop the recommender system.

The methods that we came across for privacy-preserving computation of the edit distance or Smith-Waterman score can be roughly divided into two categories. The first category is composed of all methods that use additive homomorphic encryption to apply operations on encrypted data, the second category is composed of all methods that use garbled circuits for secure function evaluation. The protocols by Jha. et al [JKS08] are among the most efficient techniques for computing these similarity scores in a privacy-preserving way, because of their efficient use of garbled circuits. However, their protocols are designed for two parties who both have to provide plaintext input to the circuits. In the setting of our research, where a central server represents users and is involved in the computation of similarity weights and in the generation of recommendations, this presents a problem. The server should only have access to encrypted data. However, the protocols presented by Jha. et al require the parties to enter unencrypted data.

The protocols by Rane and Sun [RS10] are intended to be used by a client and server, who jointly compute the edit distance or a related similarity score (such as the longest common subsequence). Their method uses additive homomorphic encryption and is suited for a setting in which a server holds part of the data over which a similarity needs to be computed.

Other earlier techniques for computing the edit distance using additive homomorphic encryption also exist, such as the technique by Atallah et al. [AKD03], but are too slow for use in practical situations. The technique by Rane and Sun puts most of the computational load on the server, so that a client can do most computations in constant time.

We choose the technique presented by Rane and Sun [RS10] as the basis for our similarity computations and will alter their protocols slightly to fit the setting of our recommender system. The main difference is that we use somewhat homomorphic encryption for the encryption in our recommender system, which not only allows for addition but also for multiplication of ciphertexts. We expect that this will speed up the protocols that Rane and Sun presented.

4.5 Encrypted Division

The homomorphic encryption scheme by Brakerski and Vaikuntanathan [BV11], which we use in this research, works on integers from the message space $R_t = \mathbb{Z}_t[x]/\langle f(x)\rangle$. In the protocols of our recommender system, sometimes an encrypted division will be needed. Because we work with integers, this means that while we are still using encryption, we cannot actually divide the integers. During this time, the integers will have to be inverted modulo a certain prime number. Following the construction by Jeckmans et al. [JPH13], we will use a look-up table to store the actual fractions of the integers, so that these can be replaced later. For two integers with $x, y \in \mathbb{Z}_t[x]$ with $gcd(x, y) = 1$, the lookup table stores the value $\frac{x}{y}$ at index positions $x \cdot y^{-1}(\mod t)$. If $gcd(x, y) \neq 1$, then we can divide out the greatest common divisor and use the fraction $\frac{x'}{y'}$ for index position $x' \cdot y'^{-1}(\mod t)$ instead (where $x' = \frac{x}{gcd(x,y)}, y' = \frac{y}{gcd(x,y)}$), since it is the same as the fraction $\frac{x}{y}$.

5 Design

In this section we will explain the recommender system design. Throughout the section, the application of drug recommendations will be considered. Similarity computations are performed to establish DNA similarity between patients and ratings for drugs are shared throughout a social network. Users of the recommender system can inquire after recommendations for a certain drug which are made using their friends' ratings. Here, the DNA similarity between user and friend is taken as a weight for the rating. Other application areas of a privacy-preserving DNA-based social recommender system also exist, some of which are discussed in section 6.5. For simplicity's sake, however, during the discussion of the protocols used in the recommender system, we will use the drug recommendation application as a setting.

We discuss the system architecture and system requirements in subsections 5.1 and 5.3.2. We then look at the recommendation formula used for the recommender system in section 5.2. Section 5.3 looks at the security requirements of the recommender system.

5.1 System Components

The recommender system has the following actors:

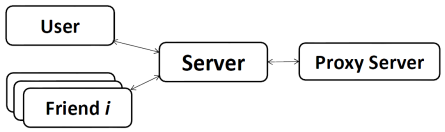

Figure 2: System Components

User The user of the system, who inquires after a drug recommendation.

Server The server, which could be located at a hospital for instance. The server assists in recommendation computations and performs the similarity computations between users. It also stores encrypted DNA-data for the users and stores encrypted similarity scores between users. The server serves as a proxy for proxy re-encryption operations.

Friends The friends of the user, who for instance visit the same self-help

group or are connected to each other by some other means. The friends provide ratings for drug treatments.

Proxy Server An additional server, which can be used to assist in privacy-preserving computations.

Recommendations should be generated while friends are offline. Therefore, the recommender protocol should be an offline protocol, in which the server performs computations on behalf of the friends.

5.2 Recommendation Formula

The recommender system generates recommendations for a drug by evaluating the ratings that users of the system provide. Based on these user-supplied ratings, a recommendation for a specific user is made that expresses the likelihood that the user will get good results from using the drug. Equation 6 specifies the recommendation formula used in our recommender system, which is based on the formula by Jeckmans et al. [JPH13].

$$p_{u,d} = \frac{\sum_{f=1}^{F_u} q_{f,d} \cdot r_{f,d} \cdot s_{u,f}}{\sum_{f=1}^{F_u} q_{f,d} \cdot s_{u,f}} \tag{6}$$

The drug prediction $p_{u,d}$ is generated for a user u and a drug d. The total number of users in the system is U, the total number of drugs known to the system is D. The drug prediction formula takes into account all friends of the user, with F_u being total number of friends of user u. Each friend f supplies a rating $r_{f,d}$ for drug d. The value $q_{f,d}$ is used as an indicator value, it has a value of 1 if friend f rated drug d and a value of 0 otherwise. The DNA similarity $s_{u,f}$ between user u and friend f serves as a weight and is multiplied with the rating. Note that $s_{u,f} = s_{f,u}$. The similarity score $s_{u,f}$ lies between 0 and 1. The higher the similarity score, the higher the friend's rating is valued.

The sum of ratings multiplied with the indicator values and similarities is divided by the normalization sum, which is the sum of all ratings multiplied with the corresponding indicator values.

The similarity $s_{u,f}$ will never be known to the user or the friend or the server, it is computed once by the user and the server or by the friend and the server (using the user's and friends' DNA material) and is then stored under encryption at the server. The user cannot change his DNA data at some point to initiate new similarity computations and can therefore not influence the similarity value to try and find out some of the other values in

the formula. The indicator $q_{f,d}$ and the rating $r_{f,d}$ remain hidden from the user as well as the server.

The DNA-similarity used in the formula can be either the edit distance or the Smith-Waterman score, both of which were discussed in section 4.4. Privacy-preserving protocols for determining these DNA similarity measures are presented later on. However, this recommender system could also be used with a different similarity score.

5.3 Security

5.3.1 Security Model

Initially, we design our recommender system for the semi-honest model. Here, the server is semi-honest. It will try to find out as much as possible about the users, but will not deviate from the protocols as set out in section 6. Since the server would be located at a hospital or a pharmacy for instance, this is a reasonable assumption. The user and his friends are also considered to be semi-honest. They will try to learn as much as possible about the other parties.

5.3.2 Privacy Requirements

The system should fulfill the following set of privacy requirements:

- The user and friend cannot find out the similarity score $s_{u,f}$. The user cannot find out any of the ratings $r_{f,d}$ of his friends. The user will try to find these values, since he is semi-honest, but it should be computationally hard for him to find them.

  To fulfill this requirement, similarity scores are computed by a user and the server. Only the server sees the similarity value encrypted under the user's key. The server therefore does not know the value and neither does the user, since he does not get to see it. Other values that users may try to find, such as ratings or indicator values, cannot be recovered through the recommendation formula, since the similarity weight stays secret and cannot be changed to influence the recommendation result.

- The server will try to learn the values $s_{u,f}$, $r_{f,d}$, $q_{f,d}$, $p_{f,d}$, but should not be able to succeed in this.

The similarity value is stored under encryption at the server. The rating and indicator values are never shared with the server, nor is the recommendation.

- The server does not learn the contents of the users' DNA data.

  All user data that is stored at the server is either stored under encryption with the user's key or after secret sharing has been applied to the data.

- Users do not learn the contents of other users' DNA data.

  Users keep their own DNA data and only store their data at the server under encryption with their public key. Other users can therefore not find out the contents of their DNA data.

- Users cannot learn whether a friend submitted a rating for a certain condition or for a specific drug. The values $q_{f,d}$ should therefore also be kept hidden from the user. Likewise, the friend does not find out the same information about the user.

- The inquiry after a prediction should be considered sensitive data.

5.4 Joining and Leaving the System

During the set-up of the recommender system, similarities between users that are friends will be computed. Users provide secret shares to some of their data as input for the similarity computations and as input that may later be re-encrypted to their friends through proxy re-encryption, when it is used in a recommendation protocol.

When a user joins the system, we assume that a social network is already in place. The user therefore already has friends who use the system, whom he is immediately connected to.

Upon registration, a user has to provide his/her DNA data to the server. The DNA data will be split into secret shares using the secret sharing mentioned in section 4.2. One share is stored in plaintext and one share is stored encrypted under encryption with the user's key, along with a proxy re-encryption key for each of the user's friends. Details on the specific representation of DNA data at the user and at the server are given in section 5.5.

The user also has a rating vector, R_u, and an indicator vector, Q_u, which may need to be used by his friends in recommendation generations at some point and which contain all rating values $r_{u,d}$ and indicator values $q_{u,d}$. These pieces of data are each split into two secret shares using the

same secret sharing scheme, where one share of R_u and Q_u respectively is stored in plaintext at the server and the other share is stored at the server under encryption with the user's key. The re-encryption keys needed to translate the encryption of the shares to encryptions under the friends' keys have already been stored at the server in the previous step.

The reason for storing shares of the user's data at the server, where the server is only able to see one of the shares, is the concern of privacy. The server can access one share for each piece of data and he can re-encrypt the other share for one of the user's friends using a proxy re-encryption scheme (examples of appropriate schemes were discussed in section 4.1). The protocols for the recommender system are set up in such a way that a user and the server can use the shares of a friend's data to generate recommendations and similarity weights while preserving the friend's privacy.

Once the user has provided the necessary data to the server, they initiate the similarity computations. The results of the computations are similarity scores between the user and each friend, encrypted under the user's key. Using proxy re-encryption, this similarity score is then also stored under encryption with the friend's key.

Upon leaving the system, the server will delete all data that was stored on behalf of the user, namely his secret shared DNA data, rating vector and indicator vector and also all of his similarity scores. The server is assumed to be honest-but-curious, we therefore expect the server to cooperate in this process.

In short, the following user data is stored at the server:

- The user's DNA data (refer to section 5.5 for more details).
- Encryption keys from the user to each of his friends.
- Two shares to the rating vector R_u, one share encrypted under the user's public key.
- Two shares to the indicator vector Q_u, one share encrypted under the user's public key.
- Similarity values between the user and each of his friends, under encryption with the user's public key.

5.5 DNA data representation

The DNA data over which a similarity score is computed remains generic throughout this research. The DNA data used can be a partial genome

sequence, the whole genome, or can also only hold the status of the SNPs that are of interest for a certain drug. The implementation of selecting the right parts of DNA remains as future work and depends on the specific setting in which the recommender system is used.

DNA-data is represented as a vector of integer-values. Each nucleotide-base is represented by a number. We use the alphabet $A = \{1, 2, 3, 4\}$ to represent the bases $\{A, C, G, T\}$. DNA-data belonging to user u is represented as G_u, a vector of arbitrary length n.

This DNA-data is stored at the server as follows: for each position i in the DNA-sequence G_u, the character $G_{u,i}$ is represented by an indicator vector $I_{u,i}$. This indicator vector has length 4 and contains zeroes in all places, except at the $G_{u,i}$'th position, where the value is 1. For example, if part of a DNA sequence is (T, A, C, G, G), then this sequence is represented in our alphabet as $(4, 1, 2, 3, 3)$. The vector of indicator vectors that represent this DNA sequence will be represented as: $((0, 0, 0, 1), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 1, 0))$.

For each character in G_u, we construct such an indicator vector. The usefulness of these indicator vectors will become apparent when we discuss the edit distance protocol. The collection of all indicator vectors is denoted by $(I_{u,i})$. The vector $(I_{u,i})$ is split into two parts using a secret sharing scheme (refer to section 4.2), so that $(I_{u,i}) = (X_{u,i}) + (Y_{u,i})$. Vector $(X_{u,i})$ is stored under encryption with the user's key at the server along with a proxy re-encryption key for each of his friends, so that it can be re-encrypted for another user of the system. Vector $(Y_{u,i})$ is stored in plaintext at the server.

What is stored at the server regarding a user's DNA data is therefore the following: a plaintext secret share $(Y_{u,i})$ and a secret share $(X_{u,i})$ which is stored under encryption with the user's key. Together, these shares form the collection of indicator vectors for the user's DNA sequence.

5.6 Summary of DNA notations

G_u The genome sequence of user u of arbitrary length n, consisting of integers from the alphabet $A = \{1, 2, 3, 4\}$.

$G_{u,i}$ The i'th character from G_u.

$I_{u,i}$ The indicator vector for $G_{u,i}$.

$(I_{u,i})$ The collection of all indicator vectors $G_{u,i}$ for $0 \leq i \leq n$.

$(X_{u,i})$ A share of $[I_{u,i}]$ for another user of the system.

($Y_{u,i}$) A share of $[I_{u,i}]$ for the server.

6 Construction

We will initially consider the semi-honest security model, which was discussed in section 5.3, when designing the protocols for our recommender system (similarity protocols can be found in section 6.2.1 and the offline recommender protocol can be found in section 6.2.7). Then, we will make an extension to the malicious user model in section 6.3. The existing protocols for the security model of our recommender system will be altered to extend them to the malicious user model.

6.1 Transfer of Data through Proxy Re-Encryption

For many of the protocols described below, a user and the server jointly compute some value using (partial) data from the user's friends. Because of the system's privacy requirements, the friend's data is stored neither at the server nor at the user in full plaintext. Instead, as we saw in section 5.4, the friend's data is always stored in two parts at the server, after it has been split using an appropriate secret sharing scheme. One part is stored in plaintext, the other part is stored under encryption with the friend's key. The server also has a proxy re-encryption key from the friend to the user. Whenever a protocol is initiated in which the user needs some share to a friend's data, the server will re-encrypt this data for the user and send it to him using proxy re-encryption. This will happen during similarity computations, where a secret share of the collection of indicator vectors ($I_{u,i}$) is re-encrypted for the user, but also during the recommender protocol, where secret shares to the friend's rating vector and indicator vector are re-encrypted and then sent to the user.

6.2 Protocols in the Semi-Honest Model

6.2.1 Similarity Protocols

The similarity computations between users are performed during the set-up phase of the system. As a similarity measure, both the edit distance and the Smith-Waterman score are considered. However, in theory these similarity measures could also be replaced by some other similarity measure to use in combination with the recommender system. Different privacy-preserving protocols would then be needed to perform the similarity computations.

6.2.2 Edit Distance Protocol

We first look at a privacy-preserving protocol to compute the edit distance of two strings, which in our recommender system will be two DNA-sequences. The protocol presented here is based on the privacy-preserving Levenshtein distance protocol by Rane and Sun [RS10]. The protocol that they devised uses additive homomorphic encryption. In our protocol, we will use a somewhat homomorphic encryption scheme. We alter the Levenshtein distance protocol in some places to fit it into our system architecture.

The protocol for computing the edit distance uses two subroutines; the substitution cost protocol, which determines the cost of substituting one character by another in a sequence, and the minimum-finding protocol, which finds the minimum element of a vector in a privacy-preserving manner. Both of these subroutines are carried out by the server and a user, where most work is done on the server side.

To find the edit distance between two strings x and y of lengths n and m, we use the dynamic formulation of the edit distance of section 4.4. We present the protocol for strings of unequal length. However, in our application of DNA-sequence similarity, strings of equal length will always be used, since we do a comparison on two sequences of DNA and having strings of unequal length as input to the protocol may leak information.

An encrypted edit distance matrix M is used, where $M(i, j)$ is the encryption under the user's public key of the edit distance $L(i, j)$ between substrings $x[0 \dots i]$ and $y[0 \dots j]$. $M(n, m)$ is the encryption of the total edit distance. The server has matrix M. The user does not get to see any of the sub-distances nor the total edit distance, because then the privacy of the similarity which is used in the recommendation formula would be violated.

Before the protocol can be carried out, the user has to obtain his secret share $(X_{f,i})$ (containing partial indicator vectors for string y, which belongs to friend f) from the server through proxy re-encryption. The user already has his own string x, his public key PK_U and secret key SK_U. The server has the user's public key PK_U and the other share $(Y_{f,i})$. Note that $(X_{f,i}) + (Y_{f,i}) = (I_{f,i})$.

The steps of the protocol are as follows:

1. The server initiates values $M(0, 0) = [0]_U$, $M(0, j) = [j]_U$ for $0 \leq j \leq m$ and $M(i, 0) = [i]_U$ for $0 \leq i \leq n$.

2. For $1 \leq i \leq n, 1 \leq j \leq m$:

   The user and server initiate the substitution cost protocol for the characters x_i and y_j, at the end of which the server has $[S(i, j)]_U$.

The server computes the values $[L(i-1,j)+1]_U, [L(i,j-1)+1]_U$ and $[L(i-1,j-1)+S(i,j)]_U$ using the additive homomorphic properties of the encryption scheme used.

The user and server initiate the minimum-finding protocol to find $M(i,j) = [min(L(i-1,j)+1, L(i,j-1)+1, L(i-1,j-1)+S(i,j))]_U$. The server gets this minimum value, the user learns nothing during this subroutine.

3. At the end of the last round, the server has computed $M(n,m) = [L(n,m)]_U$. This is the edit distance between strings x and y, encrypted under the user's key. $M(n,m)$ is used to compute the similarity weight $s_{u,f}$ following Equation 3 and this weight is stored at the server as $[s_{u,f}]_U$ for future reference. The user does not obtain the edit distance.

6.2.3 Substitution Cost Protocol

The substitution cost protocol is an altered version of the indicator function substitution cost protocol by Rane and Sun [RS10]. The protocol is used to evaluate the cost of substituting a character a (one of the x_i) by a character b (one of the y_j) in a string. For the edit distance, the cost is 1 if the characters are not the same and the cost is 0 if the characters are the same. We denote the indicator variable which takes a value of 1 if a and b are the same and takes a value of 0 if a and b are not the same by $1_{(a=b)}$. The substitution cost can then be expressed as $S(a,b) = (1 - 1_{(a=b)})$.

The differences with [RS10] are that instead of taking the characters a and b as input to the protocol, one of these characters is represented as an indicator vector which is split according to a secret sharing scheme and shared between the two participating parties of the protocol.

In this protocol, the server and a user jointly compute the substitution cost. The protocol takes as input a character b from the alphabet A, which is known to the user and two secret shares to an indicator vector $E^{(a)}$, where the character a is also from the alphabet A. The indicator vector has previously been split into two shares $X^{(a)}$ and $Y^{(a)}$ with one of the secret sharing schemes mentioned in section 4.2. The user receives a share $X^{(a)}$ through proxy re-encryption beforehand. The share $Y^{(a)}$ is already stored at the server.

The vector $E^{(a)}$ (which is one of the indicator vectors from $I_{f,j}$ for some user f) has values 0 at each position, except at the b'th position (note that b in this protocol represents a character from an alphabet consisting

User (PK_S, PK_U, SK_U) $X^{(a)}, b$		Server (PK_U, PK_S, SK_S) $Y^{(a)}$
1.	$\longrightarrow$ $[b]_U$	
		$\forall 0 \leq i \leq \lvert A \rvert :$ $[\beta_i \cdot (i-b) + (1 -$ $Y_i^{(a)}) + r]_U$
	$\longleftarrow \forall 0 \leq i \leq \lvert A \rvert :$ $[\beta_i \cdot (i-b) + (1 - Y_i^{(a)}) + r]_U$	
2. $(1 - Y_b^{(a)}) + r$		
	$\longrightarrow$	
$[(1 - Y_b^{(a)}) + r]_U$	$[(1 - Y_b^{(a)}) + r]_U$ $[X_b^{(a)}]_U$	
3.		$[(1 - Y_b^{(a)}) + r - X_b^{(a)}]_U$ $= [(1 - E_b^{(a)}) + r]_U$ $= [(1 - 1_{a=b}) + r]_U$ $[(1-1_{a=b})+r]_U + [-r]_U$ $= [(1 - 1_{a=b})]_U$

Figure 3: Substitution Cost Protocol

of integers), which takes the value of 1. To protect the friend's privacy, the server does not know a nor the indicator vector $E^{(a)}$.

An overview of the protocol can be found in Figure 3. The steps of the protocol are the following:

1. The user encrypts b under his own public key and sends it to the server. The server then for all i in the alfabet computes the value $[\beta_i \cdot (i-b) + (1 - Y_i^{(a)}) + r]_U$ using homomorphic addition. This value

is computed as: $(i-[b]_U)\cdot\beta_i+(1-Y_i^{(a)}+r)$. Here, β_i and r are random integers from some integer field that is large enough to provide security. These encryptions are sent to the user in the right order.

2. The user picks the b'th term of the received encryptions and decrypts it to obtain $(1-Y_b^{(a)})+r$. He re-encrypts this value as $[(1-Y_b^{(a)})+r]_U$ and sends it back to the server. Because of the re-encryption, the server does not know which term was picked. The user also sends his encrypted share $[X_b^{(a)}]_U$.

3. The server uses additive homomorphic properties to get $[(1-Y_b^{(a)})+r-X_b^{(a)}]_U$ and removes the noise-term r, also using homomorphic properties. He ends up with $[(1-1_{a=b})]_U = [S(a,b)]_U$, which is the encrypted substitution cost for substituting character a by character b.

The substitution cost protocol outputs an encrypted substitution cost (either an encryption of 1 or 0) to the server. The substitution cost is encrypted under the user's key. The ciphertext contains the noise of two additions, namely the addition of the server's indicator vector share and the unblinding of factor r. During the protocol, a few homomorphic additions are applied to ciphertexts occurring in the protocol (both at the user and the server) and some scalar multiplications (at the server).

6.2.4 Minimum-Finding Protocol

The minimum-finding protocol is used to find the minimum value out of n values and was developed by Rane and Sun [RS10]. The input for this protocol is a vector z with n entries. The goal is for the server to find $[min_{1\leq i\leq n} z_i]_U$, the minimum encrypted under the user's key. For the edit-distance protocol, a vector with 3 entries will always be entered into the minimum-finding protocol. An overview of the protocol can be found in Figure 4.

1. The server generates a permutation π on the set $(1,\ldots,n)$. He computes the permutation of the component-wise encrypted vector z, which he wants to find the minimum of, to obtain $[v]_U = \pi([z]_U)$. The server now picks an integer $g > 0$. Using this value, he creates an order-preserving map $G \in \mathbb{Z}^{n\times n}$ (refer to Figure 6.2.4). The server then computes $w = [Gv]_U$, using homomorphic properties. Because of the order-preservingness of G,

User: (PK_S, PK_U, SK_U)		Server: $(PK_U, PK_S, SK_S); [z]_U$
1.		$[v]_U = \pi([z]_U)$ $[w]_U = [Gv]_U$ $[b]_U = [w-a]_U$
	$\longleftarrow$ $[w-a]_U$	
2. $b = w - a$		$a_\delta =$ $(a_1 - a_2, \ldots, a_{n-1} - a_n)$
	$\longleftarrow$	
$b_\delta =$ $(b_1 - b_2, \ldots, b_{n-1} - b_n)$ $\forall 1 \leq i < j \leq n$: $a_i - a_j - \eta_{i,j} \lessgtr b_j - b_i$	$a_\delta - \eta$	
	$\longrightarrow$ $[\alpha]_U = [\arg\min_{1 \leq i \leq n} v_i]_U$	
3.		$\forall 1 \leq i \leq n$: $[\beta_i \cdot (i - \alpha) + v_i + r]_U$
	$\longleftarrow \forall 1 \leq i \leq n$: $[\beta_i \cdot (i - \alpha) + v_i + r]_U$	
$v_\alpha + r$ $[v_\alpha + r]_U$	$\longrightarrow$ $[v_\alpha + r]_U$	
4.		$[v_\alpha]_U = [v_\alpha + r]_U + [-r]_U$ $= [min_{1 \leq i \leq n} z_i]_U$

Figure 4: Minimum-Finding Protocol

the following property holds: $v_i < v_j \iff w_i < w_j$. The server picks

$$\begin{pmatrix} g+g_1 & g_2 & \cdots & g_{n+1} & g_n \\ g_1 & g+g_2 & \cdots & g_{n+1} & g_n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_1 & g_2 & \cdots & g+g_{n+1} & g_n \\ g_1 & g_2 & \cdots & g_{n+1} & g+g_n \end{pmatrix} \tag{7}$$

Figure 5: Matrix G

a vector $a \in \mathbb{Z}^n$ randomly and sends $[w-a]_U$ to the user.

2. The user decrypts $[w-a]_U$ element-wise and obtains a vector $b = w - a$. The user generates a vector b_δ of size $\binom{n}{2}$ such that $b_\delta = (b_1 - b_2, \ldots b_1 - b_n, b_2 - b_3, \ldots, b_2 - b_n, \ldots, b_{n-1} - b_n)$. The server generates a similar vector a_δ of size $\binom{n}{2}$. The server then chooses an integer value $\eta' \in [-g, g]$ randomly and uses this to generate the vector $\eta = (\eta_{i,j})_{1 \le i < j \le n} \in \mathbb{Z}^{\binom{n}{2}}$. The sum of all $\eta_{i,j}$ is equal to η'. The server sends $a_\delta - \eta$ to the user, who for all $1 \le i < j \le n$ compares $a_i - a_j - \eta_{i,j}$ to $b_j - b_i$.

   Note that: $w_i \le w_j$ iff $a_i - a_j \le b_j - b_i$. By determining $a_i - a_j - \eta_{i,j} \le b_j - b_i$, the server checks whether $w_i - w_j - \eta_{i,j} \le 0$. The value $\eta_{i,j}$ was picked from $[-g, g]$, so if $a_i - a_j - \eta_{i,j} < b_j - b_i$ then $v_i < v_j$ and if $a_i - a_j - \eta_{i,j} > b_j - b_i$ then $v_i > v_j$. With these comparisons, the user is able to determine $\alpha = \arg\min_{1 \le i \le n} v_i$. The user sends $[\alpha]_U$ to the server.

3. The server picks a random integer r and random integers $\beta_i \forall 1 \le i \le n$. He uses homomorphic properties to compute $[\beta_i(i-\alpha) + v_i + r]_U$ and sends these values in the right order to the user. For index $i = \alpha$, the user decrypts $[\beta_i(i-\alpha) + v_i + r]_U$ to $v_\alpha + r$. He re-encrypts this value and sends it to the server in the form $[v_\alpha + r]_U$.

4. The server now removes the blinding factor r using homomorphic properties and ends up with $[v_\alpha]_U = [min_{1 \le i \le n} v_i]_U = [min_{1 \le i \le n} z_i]_U$.

The encrypted minimum which is outputted to the server at the end of the protocol, is a ciphertext to which two additions have been applied, namely the blinding and unblinding of v_α. During the protocol, element-wise multiplication is performed on the server once and several additions take place. However, the output of the protocol only contains the noise caused by two additions.

6.2.5 Smith-Waterman Distance

For the Smith-Waterman similarity score, we use the same subroutines as for the edit distance, with very few alterations. To find the similarity score, we use the recursive relation as was specified in section 4.4.

An encrypted edit distance matrix M is used, where $M(i,j)$ is the encryption under the user's public key of the Smith-Waterman score $H(i,j)$ between substrings $x[0\dots i]$ and $y[0\dots j]$. $M(n,m)$ is the encryption of the total score. The server has matrix M. The user does not get to see any of the sub-distances nor the total edit distance, because then the privacy of the similarity which is used in the recommendation formula would be violated.

Instead of using minimum-finding, it is apparent that we will have to do maximum-finding to find the Smith-Waterman score. We can reuse the minimum-finding protocol by using the following relation:
$-\min(-z_1,\dots,-z_n) = \max(z_1,\dots,z_n)$. We provide the negative vector as input to the minimum-finding protocol and negate the output to obtain the maximum value. The substitution cost protocol can also be reused, however, we do not use an indicator cost function anymore. We use the cost function c. In order to use this cost function, we alter the substitution cost protocol so that (if we use an indicator vector for the character y) instead of computing $1 - E_i^a$ we compute $(E_i^a - 1)f + E_i^a \cdot e$. If the two characters are the same, e.g. as input characters we have a and b and for a we have the indicator vector E^a, then this will evaluate to $(1-1)f + 1 \cdot e = e$ and if the two characters are different then this will evaluate to $(0-1)f + 0 \cdot e = -f$.

The steps of the substitution cost protocol are now the following for computing the cost $c(a,b)$ for characters a, b:

1. The user encrypts b under his own public key and sends it to the server. The server then for all i in the alfabet computes the value $[\beta_i \cdot (i-b) + (Y_i^{(a)} - 1)f + Y_i^{(a)} \cdot e + r]_U$ using homomorphic multiplication and addition. Here, β_i and r are random integers from some integer field that is large enough to provide security. These encryptions are sent to the user in the right order.

2. The user picks the b'th term of the received encryptions and decrypts it to obtain $(Y_b^{(a)} - 1)f + Y_b^{(a)} \cdot e + r$. He re-encrypts this value as $[(Y_b^{(a)} - 1)f + Y_b^{(a)} \cdot e + r]_U$ and sends it back to the server. Because of the re-encryption, the server does not know which term was picked. The user also sends his encrypted share $[X_b^{(a)} \cdot (e+f)]_U$.

3. The server uses additive homomorphic properties to get $[(Y_b^{(a)} - 1 +$

$X_b^{(a)})f + (Y_b^{(a)} + X_b^{(a)}) \cdot e + r]_U$ and removes the noise-term r, also using homomorphic properties. He ends up with $[(1_{a=b} - 1)f + 1_{a=b} \cdot e]_U = [c(a, b)]_U$, which is the encrypted substitution cost for substituting character a by character b.

Using this altered substitution cost protocol and the maximum-finding protocol, we devise the following protocol for the Smith-Waterman score:

1. The server initiates values $M(0,0) = [0]_U$, $M(0,j) = [0]_U$ for $0 \leq j \leq m$ and $M(i,0) = [0]_U$ for $0 \leq i \leq n$.

2. For $1 \leq i \leq n, 1 \leq j \leq m$:

3. The user and server initiate the altered substitution cost protocol for the characters x_i and y_j, at the end of which the server has $[c(x_i, y_j)]_U$.

4. The server computes the values $[H(i-o,j)-g(o)]_U \forall 1 \leq o \leq i, [H(i,j-l)-g(l)]_U \forall 1 \leq l \leq j$ and initiates the maximum-finding protocol with the user to find $[\max_{1 \leq o \leq i}(H(i-o,j)-g(o)]_U)$ and $[\max_{1 \leq l \leq j} H(i,j-l)-g(l)]_U$. The server also computes the value $[H(i-1,j-1) + c(x_i, y_j)]_U$, using homomorphic encryption properties.

5. The user and server initiate the maximum-finding protocol to find: $[\max(0, \max_{1 \leq o \leq i}(H(i-o,j)-g(o)]_U), \max_{1 \leq l \leq j} H(i,j-l)-g(l), H(i-1,j-1)+c(x_i,y_j))]_U$. The server gets this maximum value and stores it as $[H(i,j)]_U$, the user learns nothing during this subroutine.

6. At the end of the last round, the server has computed $M(n,m) = [H(n,m)]_U$. This is the edit distance between strings x and y, encrypted under the user's key. $M(n,m)$ is used to compute the similarity weight $s_{u,f}$ according to Equation 5, which is stored at the server as $[s_{u,f}]_U$ for future reference. The user does not obtain the Smith-Water score nor the similarity measure.

6.2.6 Analysis and Complexity of the Similarity Protocols

For the similarity computation (edit distance or Smith-Waterman) and the subprotocols, security derives from the original protocols on which they were based [RS10, JKS08]. Here, we will not give a formal proof of security for each of these protocols, because they are straightforward. With the composition theorem for the semi-honest model [Gol05] we can conclude that the composition of all proven secure subprotocols is secure as well.

For the substitution cost protocol, the complexity for user as well as server is $O(1)$ for both computation and communication. For the minimum-finding protocol (or the altered maximum-finding protocol), the same applies, since in our setting we do minimum-finding on three values.

The complexity of these subroutines translate directly into the complexity analysis for the edit distance and Smith-Waterman protocols. The complexity for the protocols can be found in Figures 6 and 7.

	User		Server	
step	comp	comm	comp	comm
1.	-	-	$O(\max(n, m))$	-
2.	$O(n \cdot m)$	$O(n \cdot m)$	$O(n \cdot m)$	$O(n \cdot m)$

Figure 6: Complexity of the Edit Distance Protocol for a user and a server (semi-honest model), n and m are the respective lengths of sequences that are compared.

	User		Server	
step	comp	comm	comp	comm
1.	-	-	$O(\max(n, m))$	-
3.	$O(m \cdot n)$	$O(n \cdot m)$	$O(m \cdot n)$	$O(n \cdot m)$
4.	$O(m^2 \cdot n^2)$	$O(m^2 \cdot n^2)$	$O(m^2 \cdot n^2)$	$O(m^2 \cdot n^2)$
5.	$O(m \cdot n)$	$O(m \cdot n)$	$O(m \cdot n)$	$O(m \cdot n)$

Figure 7: Complexity of the Smith-Waterman Protocol for a user and a server (semi-honest model), n and m are the respective lengths of sequences that are compared.

We see that the complexity of the Smith-Waterman Protocol is greater than that of the edit distance protocol. This is due to the fact that for finding the Smith-Waterman distance, subsequences must also be inspected.

The only homomorphic operations that are used in the protocols are homomorphic additions. These protocols can therefore also be carried out using an additive homomorphic encryption scheme.

6.2.7 Offline Recommender Protocol

Here we present the recommender protocol for offline friends, which is based on the solution with offline friends by Jeckmans et al. [JPH13]. An overview of the protocol can be found in Figure 8. Before the run of the protocol,

for all friends of the user, similarities between the user and the friend $s_{u,f}$ have been computed by the user and the server and stored under encryption of both the user and the friend at the server, using one of the protocols presented in section 6.2.1.

As was mentioned before in section 5.4, the friend's rating vector $r_{f,d}$ and his indicator vector $q_{f,d}$ are split into two shares following the secret sharing method that was discussed before. The rating vector R_f for a friend f is split as follows: $R_f = T_f + S_f$. The indicator vector is split in the same way: $Q_f = U_f + V_f$. This secret sharing scheme has already been applied during the set-up of the recommender system. The shares S_f and V_f have been stored at the server in plaintext and the other shares T_f and U_f have been stored under encryption with the friend's key at the server. A proxy re-encryption key from the friend to the user has been stored at the server as well. For use in this protocol, we set $r_{f,d}$ to zero when $q_{f,d}$ is zero, in order to simplify the operations.

User: $(PK_U, SK_U, PK_S, PRE_{SKU}, PRE_{PKU})$		Server: (PK_S, SK_S, PK_U) $\forall 1 \leq f \leq U^f$: $S_f, V_f, [[T_f]]_{PRE_{PKf}}$ $[[U_f]]_{PRE_{PKf}}, RK_{fU}$
1.	$\forall 1 \leq f \leq U^f$	
	$\longleftarrow$	$[[T_f]]_{PRE_{PKU}}, [[U_f]]_{PRE_{PKU}}$
	$[[T_f]]_{PRE_PK_U}, [[U_f]]_{PRE_PK_U}$	
T_f, U_f		
2.	$\forall 1 \leq f \leq U^f$	
	$\longleftarrow$	$[s_{u,f} + b]_U$
	$[s_{u,f} + b]_U, [-b]_S$	
$[s_{u,f}]_S$		
3.	$\forall 1 \leq d \leq D$	
$[z_d]_S = \sum_{f=1}^{F_u} [s_{u,f}]_S \cdot t_{f,d}$		$[a_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot s_{f,d}$
$[z_d + \sigma_{1,d}]_S = [z_d]_S + \sigma_{1,d}$	$\longrightarrow$	
	$[z_d + \sigma_{1,d}]_S, [-\sigma_{1,d}]_U$	
4.	$\forall 1 \leq d \leq D$	
$[g_d]_S =$ $\sum_{f=1}^{F_u} [s_{u,f}]_S \cdot u_{f,d}$		$[h_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot v_{f,d}$
$[g_d + \sigma_{2,d}]_S = [g_d]_S + \sigma_{2,d}$	$\longrightarrow$	
	$[g_d + \sigma_{2,d}]_S, [-\sigma_{2,d}]_U$	
5.	$\forall 1 \leq d \leq D$	
		$z_d + \sigma_{1,d}, g_d + \sigma_{2,d}$
		$[z_d]_U = [-\sigma_{1,d}]_U + (z_d + \sigma_{1,d})$
		$[n_d]_U = [z_d]_U + [a_d]_U$
		$[g_d]_U = [-\sigma_{2,d}]_U + (g_d + \sigma_{2,d})$
		$[e_d]_U = [g_d]_U + [h_d]_U$
6.	$\forall 1 \leq d \leq D$	
	$\longleftarrow$	$[e_d \cdot \sigma_{3,d}]_U = [e_d]_U \cdot \sigma_{3,d}$
	$[e_d \cdot \sigma_{3,d}]_U, [\sigma_{3,d}^{-1}]_S$	
$[e_d \cdot \sigma_{4,d}]_S =$ $e_d \cdot \sigma_{3,d} \cdot [\sigma_{3,d}^{-1}]_S \cdot \sigma_{4,d}$	$\longrightarrow$	
	$[e_d \cdot \sigma_{4,d}]_S, [\sigma_{4,d}]_U$	
		$e_d^{-1} \cdot \sigma_{4,d}^{-1}$
		$[e_d^{-1}]_U = e_d^{-1} \cdot \sigma_{4,d}^{-1} \cdot [\sigma_{4,d}]_U$
	$\longleftarrow$	$[p_{u,d}]_U = [e_d^{-1}]_U \cdot [n_d]_U$
	$[p_{u,d}]_U$	

Figure 8: Recommender Protocol for offline friends

1. Before the start of the actual offline recommender protocol, the main server will act as a proxy by re-encrypting the shares T_f and U_f for the user, translating the encryption under key PRE_{PKf}, belonging to the friend, to encryption under the user's key PRE_{PKU}, using the re-encryption key RK_{fU}.

2. During the first step of the protocol, the server has the similarity measure for each friend f that is considered in the protocol. He blinds the value $[s_{u,f}]_U$ additively with a blinding value b_f and sends the encryption $[s_{u,f} + b_f]_U$ to the user, along with the unblinding value $[-b_f]_S$, encrypted under the server's key. The user decrypts this to $s_{u,f} + b_f$ and uses the unblinding value to get $[s_{u,f}]_S = [s_{u,f} + b_f]_S + [-b_f]_S$.

3. For each drug for which a recommendation needs to be generated, the server and user compute their shares of the sum of ratings multiplied with the similarity between user and friend. The user computes the value $[z_d]_S = \sum_{f=1}^{F_u} [s_{u,f}]_S \cdot t_{f,d}$, using the similarity that he received during step 1 and his share of the friend's rating vector, and the server computes $[a_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot s_{f,d}$. Together, the values $z_d + a_d$ evaluate to $n_d = \sum_{f=1}^{F_u} s_{u,f} \cdot r_{f,d}$. The user blinds his value $[z_d]_S$ with a random integer $\sigma_{1,d} \in \mathbb{Z}_t$ and sends this value to the server accompanied by the unblinding value $[-\sigma_{1,d}]_U$, which can only be removed under encryption with the user's key.

4. Likewise, the server and user compute their shares of the normalization sum. The user computes $[g_d]_S = \sum_{f=1}^{F_u} [s_{u,f}]_S \cdot u_{f,d}$ and blinds this value with $\sigma_{2,d} \in \mathbb{Z}_t$ to obtain $[g_d + \sigma_{2,d}]_S$ and sends this to the server along with the unblinding value $[-\sigma_{2,d}]_U$, which can also only be removed under encryption with the user's key. The server simultaneously computes the value $[h_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot v_{f,d}$.

5. The server now removes the blinding values $\sigma_{1,d}$ and $\sigma_{2,d}$ using homomorphic properties of the encryption scheme. He computes $[z_d]_U = [-\sigma_{1,d}]_U + (z_d + \sigma_{1,d})$ and is now able to construct the sum-value $[n_d]_U = [z_d]_U + [a_d]_U$. The server also computes $[g_d]_U = [-\sigma_{2,d}]_U + (g_d + \sigma_{2,d})$ which allows him to compute $[e_d]_U = [g_d]_U + [h_d]_U$. The normalization-sum $[e_d]_U$ evaluates to $\sum_{f=1}^{F_u} [s_{u,f}]_U \cdot q_{f,d}$ and needs to be inverted before the drug prediction can be made. The only problem that might occur here is when e_d is equal to 0. This would occur when none of the user's

friends have rated drug d. However, the likelihood of this situation is negligibly small.

6. The server blinds $[e_d]_U$ multiplicatively using $\sigma_{3,d} \in \mathbb{Z}_t^*$, obtaining $[e_d \cdot \sigma_{3,d}]_U$. He sends this to the user, accompanied by the inverted blinding value $[\sigma_{3,d}^{-1}]_S$. The blinding can only be removed under the server's key. The user uses the unblinding value to obtain $[e_d]_S$ and adds his own blinding $\sigma_{4,d} \in \mathbb{Z}_t^*$ multiplicatively. He sends it to the server along with the unblinding value $[\sigma_{4,d}]_U$, which can only be removed under encryption with the user's key. The server decrypts the message to $e_d \cdot \sigma_{4,d}$. He inverts this value to $e_d^{-1} \cdot \sigma_{4,d}^{-1}$ and removes the blinding under encryption with the user's key. Now the server has $[e_d^{-1}]_U$. He computes the recommendation $[p_{u,d}]_U = [n_d]_U \cdot [e_d^{-1}]_U$ and sends it to the user. The user can decrypt this and uses a look-up table for divisions to get recommendation value.

6.2.8 Analysis and Complexity of the Recommender Protocol

The differences with Jeckman et al.'s [JPH13] protocol, is that some operations have been added to preserve the privacy of the indicator vector Q_f and that the weights from user to friend and from friend to user do not need to be added, but that another weight is used (the similarity score), which needs to be sent from the server to the user while maintaining privacy. We accomplish this by blinding the similarity score under the server's public key.

The recommender protocol uses a few more homomorphic additions than the protocol by Jeckmans et al., but the number of homomorphic multiplications stays the same (only one homomorphic multiplication is performed).

An analysis of the protocol's complexity can be found in figure 15.

	User		Server	
step	comp	comm	comp	comm
1.	$O(F_u)$	-	$O(F_u)$	$O(F_u)$
2.	$O(F_u)$	-	$O(F_u)$	$O(F_u)$
3.	$O(F_u \cdot D)$	$O(D)$	$O(F_u \cdot D)$	-
4.	$O(F_u \cdot D)$	$O(D)$	$O(F_u \cdot D)$	-
5.	-	-	$O(D)$	-
6.	$O(D)$	$O(D)$	$O(D)$	$O(D)$

Figure 9: Complexity of the Offline Recommender Protocol for a user and a server (semi-honest model), F_u is the number of friends of user u, D is the number of drugs that is being rated.

6.3 The Malicious User Model

The recommender system that we have discussed so far was designed for a semi-honest setting, in which users and the server are all semi-honest. However, this is a relatively weak security model. It would be preferable to allow malicious users in our security model. That is why we now extend the recommender system to a new security model where malicious users are allowed, but where the server is still considered to be semi-honest. The server will still try to find out as much as possible about the system's users, but will not deviate from protocol. To accomodate for the existence of malicious users, a second server is needed, which we shall call the proxy server and which was briefly mentioned before in section 5.1. The proxy server is also required to be semi-honest and the two servers are required to not collude. Section 6.3.3 discusses the reasonableness of the assumption of non-collusion between the two servers. All computations are from now on now carried out by the server and the proxy server on behalf of the user.

For the sake of simplicity, throughout the remainder of this section we shall refer to the main server and the assisting proxy server as server A and server B, respectively.

6.3.1 Additional Privacy Requirements

Additional privacy requirements in the malicious user model are the following:

- Users cannot submit false ratings that lie outside the boundaries of permitted rating values.

- The assisting server is also considered to be semi-honest, but cannot find out any of the values $s_{u,f}$, $r_{f,d}$, $q_{f,d}$, $p_{f,d}$ without computationally hard work. The assisting server can also not find out users' DNA data.

6.3.2 Data Storage

All user data is stored at the same server as before, which is the main server. The proxy server only serves as an assisting server for the similarity computations and the recommendation generation and can therefore be a lightweight server.

To enable privacy-preserving computations on behalf of the user, the user needs to store some additional data at server A: he needs to encrypt the collection of indicator vectors that represent his genome sequence as well as his genome sequence under the assisting server's public key and store this at the main server. Also, a proxy re-encryption key for server B has to be generated by the user, which is then stored at the main server (server A).

It may seem unsecure to store confidential user data under the assisting server's public key at the main server, since in real life the two servers could collude. However, since the user already stored two secret shares to his genome data at the main server, storing the non-splitted data under encryption with the assisting server's key practically makes no difference and does not decrease security in any way when compared to the previous situation.

In summary, what is now stored for each user at server A is:

- The user's DNA data: the plaintext share ($X_{u,i}$), the encrypted share ($Y_{u,i}$) which can be proxy re-encrypted, the non-splitted collection $[I_{u,i}]_B$ and the encrypted genome sequence $[G_u]_B$.
- Encryption keys from the user to each of his friends and to the proxy server.
- Two shares to the rating vector R_u, one share encrypted under the user's public key.
- Two shares to the indicator vector Q_u, one share encrypted under the user's public key.
- Similarity values between the user and each of his friends, under encryption with server B's public key.

Server B also needs to generate re-encryption keys to all users in the system, which are stored at server A.

6.3.3 Non-Collusion Assumption

For the malicious user model, we assume that the two servers are still semi-honest: they do not deviate from protocol and do not collude. The non-collusion assumption for two servers has been used in real-life applications and in theoretical applications before.

The research by Peter et al. [PTK13] uses the concept of two non-colluding servers for outsourcing privacy-preserving computations on encrypted user data. Their design is very similar to ours. Choi et al. [CEJ+07] also use this concept in their design for online secure computation on encrypted input. Veugen et al. [VdHCM15] use two non-colluding servers in their provably secure recommender system to outsource computations to. In their recommender system, a server that acts maliciously cannot do so without a user detecting this malicious behaviour. They provide security for their recommender system in the malicious model, where an adversary can take control of one of the servers. The pre-processing phase for their recommender system is not included in this security model, however.

Catrina and Kerschbaum [CK08] wrote about e-commerce applications that use secure multi-party computation and mentioned the combination of cryptographic and non-cryptographic tools and architectures to make efficient SMC systems possible, for instance by combining cryptographic techniques and the use of service providers. They note that the advantages of using a service provider, such as the main server and proxy server in our recommender design for malicious users, are that service providers can be set-up with more resources than normal clients in an SMC application and that network requirements are lower, since fewer network traffic is necessary. The same may be true in our recommender system, the main server especially can be set with more resources and the amount of network traffic between the main and the proxy server is kept to a minimum throughout all protocols that are presented in the remainder of this section. Other real-world applications using non-colluding servers are [BLW08, BCD+09].

The requirement that the two servers do not collude is a reasonable one, since our main server will be provided by a hospital, for instance, and since the proxy server could be a security provider (refer to section 6.3.12 for a discussion of the proxy server's role in the system). These servers will not likely collude or cheat to recover users' data, since this would be damaging to their reputation.

6.3.4 Similarity Protocols in the Malicious User Model

We will now discuss the alterations made to the already existing similarity protocols, to extend them to the malicious user model.

6.3.5 Edit Distance

We alter the edit distance protocol discussed previously in section 6.2.2, which is based on the privacy-preserving Levenshtein distance protocol by Rane and Sun [RS10], to extend it to the malicious user model.

As before, an encrypted edit distance matrix M is used, where $M(i, j)$ is the encryption under server B's public key of the edit distance $L(i, j)$ between substrings $x[0 \dots i]$ and $y[0 \dots j]$. $M(n, m)$ is the encryption of the total edit distance. Server A has matrix M. The user u, to whom string x belongs and friend f, to whom string y belongs, do not get to see any of the sub-distances nor the total edit distance, since they are not involved in the protocol. Server B also does not get to see these values; server A never shares the encrypted values with him, since that would cause the privacy of the protocol to be violated.

Before the protocol can be carried out, server B has to obtain his secret share $(X_{f,j})$ (containing partial indicator vectors for string y, which belongs to friend f) from server A through means of proxy re-encryption. He already has his own public key PK_B and secret key SK_B. Server A has Server B's public key PK_B, the user's public key PK_U and the public key PK_F of friend f. He also has the other share $(Y_{f,j})$. Note that $(X_{f,j}) + (Y_{f,j}) = (I_{f,j})$. Lastly, server A has the homomorphically encrypted indicator vector collection $[(I_{u,i})]_B$ and the homomorphically encrypted genome sequence $[G_u]_B$.

The steps of the protocol are as follows:

1. Server A initiates values $M(0, 0) = [0]_B$, $M(0, j) = [j]_B$ for $0 \leq j \leq m$ and $M(i, 0) = [i]_B$ for $0 \leq i \leq n$.

2. For $1 \leq i \leq n, 1 \leq j \leq m$:

   Server A and server B initiate the substitution cost protocol for the characters x_i and y_j, at the end of which the server has $[S(i, j)]_B$. Inputs to this protocol are: $X_{f,j}, Y_{f,j}, [I_{u,i}]_B, [G_{u,i}]_B$.

   Server A computes the values $[L(i-1, j)+1]_B, [L(i, j-1)+1]_B$ and $[L(i-1, j-1) + S(i, j)]_B$ using the additive homomorphic properties of the encryption scheme used.

Server A and server B initiate the minimum-finding protocol to find $M(i,j) = [min(L(i-1,j)+1, L(i,j-1)+1, L(i-1,j-1)+S(i,j))]_B$. Server A gets this minimum value, server B learns nothing during this subroutine.

3. At the end of the last round, server A has computed $M(n,m) = [L(n,m)]_B$. This is the edit distance between strings x and y, encrypted under server B's key. $M(n,m)$ is used to compute the similarity weight $s_{u,f}$ following Equation 3 and this weight is stored at server A as $[s_{u,f}]_B$ for future reference. The user, friend and Server B never obtain this (encrypted) edit distance.

6.3.6 Substitution Cost

The substitution cost protocol from section 6.2.3 is altered to extend it to the malicious user model and is still based on the indicator function substitution cost protocol by Rane and Sun [RS10].

Server A and Server B now jointly compute the substitution cost. The protocol takes as input an encrypted indicator vector $E^{(b)} := [I_{u,i}]_B$ that represents a character $b = G_{u,i}$ from the alphabet A, the encrypted b: $[b]_B$, and two secret shares to an indicator vector $I_{f,j}$, which represents a character a that is also from the alphabet A. Server A has $E^{(b)}$ and $[b]_B$. $I_{f,j}$ has previously been split into two shares $X^{(a)} := X_{f,j}$ and $Y^{(a)} := Y_{f,j}$ with one of the secret sharing schemes mentioned in section 4.2. Server B receives his share $X^{(a)}$ through proxy re-encryption before the start of this protocol; the share $Y^{(a)}$ is already stored at server A.

An overview of the protocol can be found in Figure 10. The steps of the protocol are the following:

Server B (PK_A, PK_B, SK_B) $X^{(a)}$		Server A (PK_B, PK_A, SK_A) $Y^{(a)}, [E^b]_B, [b]_B$
1.	$\longleftarrow$ $\pi([E^b])_B$	$\pi([E^b])_B$
	$\longleftarrow \forall 1 \leq i \leq \lvert A \rvert :$ send t_i in ascending order of $\pi^{-1}(i)$	$\forall 1 \leq i \leq \lvert A \rvert :$ $t_i = [\beta_i \cdot (i-b) + (1 - Y_i^{(a)}) + r]_B$
2. pick $\pi(b)$-th term $(1 - Y_b^{(a)}) + r$ $[(1 - Y_b^{(a)}) + r]_B$	$\longrightarrow$ $[(1 - Y_b^{(a)}) + r]_B$	
3.	$\longrightarrow$ $[X^{(a)}]_B$	
	$\longleftarrow$ $\pi([X^{(a)} + r_2]_B)$	$\pi([X^{(a)} + r_2]_B)$
$[X_b^{(a)} + r_2]_B$	$\longrightarrow$ $[X_b^{(a)} + r_2]_B$	
		$[X_b^{(a)}]_B$
4.		$[(1 - Y_b^{(a)}) + r - X_b^{(a)}]_B$ $= [(1 - E_b^{(a)}) + r]_B$ $= [(1 - 1_{a=b}) + r]_B$ $[(1-1_{a=b})+r]_B + [-r]_B$ $= [(1 - 1_{a=b})]_B$

Figure 10: Substitution Cost Protocol in the malicious user model

1. Server A picks a random permutation π of the alphabet A and permutes $[I_{u,i}]_B$ and $[E^{(b)}]_B$. He sends the obtained permutation $[\pi(E^{(b)})]_B$ to server B. Server A then for all i in the alphabet computes the value $[\beta_i \cdot (i-b) + (1 - Y_i^{(a)}) + r]_B$ using homomorphic addition. This value is computed as: $(i - [b]_B) \cdot \beta_i + (1 - Y_i^{(a)} + r)$. Here, β_i and r are random integers from some integer field that is large enough to provide security. These encryptions are sent to the user according to the ascending order of $\pi^{-1}(i)$.

2. Server B gets $[\pi(E^{(b)}]_B$ and decrypts this to $\pi(E^{(b)})$. He infers $\pi(b)$ from the index where a 1 appears in the resulting indicator vector. Server B picks the $\pi(b)$'th term of the received encryptions, which comes down to him picking the b'th term, and decrypts it to obtain $(1 - Y_b^{(a)}) + r$. He re-encrypts this value as $[(1 - Y_b^{(a)}) + r]_B$ and sends it back to the server. Because of the re-encryption, the server does not know which term was picked.

3. Server B sends his share $X^{(a)}$ after encrypting it under his public key to Server A. Server A additively blinds the vector element-wise with a blinding factor r_2. He then permutes the vector with π and sends the result back to Server B. Server B now picks the $\pi(b)$'th index of the vector that he receives, which is $[X_b^{(a)} + r_2]_B$. He sends this to Server A and Server A removes the blinding factor.

4. Server A uses additive homomorphic properties to get $[(1 - Y_b^{(a)}) + r - X_b^{(a)}]_B$ and removes the noise-term r, also using homomorphic properties. He ends up with $[(1 - 1_{a=b})]_B = [S(a,b)]_B$, which is the encrypted substitution cost for substituting character a by character b.

The substitution cost protocol outputs an encrypted substitution cost (either an encryption of 1 or 0) to server A. The substitution cost is encrypted under server B's public key.

6.3.7 Minimum Finding

For minimum finding, we now look to a protocol by Erkin et al. [EFG+09], which is a minimum finding protocol for two encrypted values. We alter this protocol so that it is performed by two servers using SWHE. The original protocol offers the option to compare two encrypted values and to get the

encrypted minimum. We will have to perform the routine twice to get the encrypted minimum of three values.

We opt for this altered version of Erkin's protocol, instead of extending the protocol in section 6.2.4, because it is easier to extend to the malicious user model and because the extension we present takes advantage of the SWHE scheme used. In Erkin et al.'s original protocol, a multiplication is performed by using an interactive protocol. By using SWHE, we expect to improve on efficiency, since we can perform a homomorphic multiplication. In the original protocol an encrypted value needs to be blinded, sent to the other party, who then decrypts it and performs an exponentiation and re-encrypts the result, after which the first party removes the blinding from the received encryption. The efficiency gain would lie in the fact that no communication is required and that we expect a homomorphic multiplication to be faster than a blinding, exponentiation and unblinding.

Refer to Figure 11 for an overview of the protocol. Below, we describe the steps taken in the protocol for our setting, where the protocol differs slightly from the one by Erkin et al. [EFG+09]. At the beginning of the protocol, the main server has three encrypted values of which he would like to get the encrypted minimum. We call these values a, b and c.

Server B: (PK_B, SK_B, PK_A)		Server A: (PK_A, SK_A, PK_B) $[a]_B, [b]_B, [c]_B$
1.		
		$[z]_B = [2^l + a - b]_B$
	$\longleftarrow$	$[d]_B = [z + r]_B$
$z + r \mod 2^l$	$\longrightarrow$	
	$[z + r \mod 2^l]_B$	
		$[z']_B = [d \mod 2^l - r \mod 2^l]$
2.	$\forall 0 \leq i \leq l-1$ $\longrightarrow [d'_i]_B$	
		$s \in_r \{-1, 1\}$
		$[w'_i]_B = [d'_i + r'_i]_B$
	$\longleftarrow [w'_i]_B$	
$w_i = w'i \mod 2$		
	$\longrightarrow [w_i]_B$	
		$[c_i]_B = [d'_i - r'_i + s + 3\sum_{j=i+1}^{l-1} w_j]_B$
		$[e_i]_B = [c_i]_B \cdot [q_i]_B$
	$\longleftarrow [e_i]_B$	
check for zero		
	$\longrightarrow [\lambda']_B$	
		$[\lambda]_B = [\lambda']_B \cdot s$
3.		
		$[z \mod 2^l] = [z' + \lambda 2^l]_B$
		$[z_l]_B = [2^{-l} \cdot (z - (z \mod 2^l))]_B$
		$[m]_B = [z_l]_B \cdot [a - b]_B + [b]_B$

Figure 11: Mininimum Finding in the malicious user model

1. Server A has $[a]_B$ and $[b]_B$. The variable l is the bit-length of the inputs. Server A determines $[z]_B$, whose most significant bit is 0 iff

$a < b$. We want to obtain the value $z \mod 2^l$, to determine the most significant bit of z. Server A blinds z additively with a random blinding factor r which is $\kappa + l + 1$ bits long (κ being the security parameter of our SWHE scheme) and sends it to server B. Server B now performs a modulo operation on the blinded and decrypted value and returns $[z+r \mod 2^l]_B = [d \mod 2^l]_B$ to server A. Server A subtracts $r \mod 2^l$ from this value using homomorphic addition and gets $[z' \mod 2^l]_B$. This subtraction does not occur modulo 2^l of course, so a reduction modulo 2^l has to be performed to get $[z \mod 2^l]_B$. (Only if $d \mod 2^l > r \mod 2^l$, then $z' \mod 2^l \equiv z \mod 2^l$).

2. In the second step, Erkin et al. [EFG+09] switch to another homomorphic encryption scheme with a smaller plaintext space that allows for efficient multiplicative masking. We do switch, however, since we use the BV scheme [BV11], in which multiplicative masking comes down to a multiplication rather than an exponentiation.

   Server B encrypts all bits in $d' \equiv d \mod 2^l$ and sends these to server A. Server A computes the sums $[c_i]_B = [d_i - r_i + s + 3\sum_{j=i+1}^{l-1} w_j]_B$, using the bits of $r' \equiv r \mod 2^l$. Here, $w_j = d'_j \oplus r'_j$. To compute this xor-value under homomorphic encryption, we use the formula $a \oplus b = a + bmod2$, where a and b are bits. Server A first adds all d and r' bits under homomorphic encryption and sends them in a random order to server B. Server B decrypts these values and performs a reduction modulo 2. Server B sends the results back to server A in the same order as he received them, after which server A stores the values in the right order (unrandomized). Now, if $r' > d'$, then one of the c_i will be equal to zero. If $d' > r'$, then all c_i are non-zero. Server A sends the encrypted c_i to server B, after multiplicatively blinding them with random values $q_i \in \mathbb{Z}_t^*$, who decrypts all values and checks for zeroes. Server B sends back an encrypted bit λ', which is 1 if a zero was encountered during the check and is 0 otherwise. Server B now does not know which of the two initial values r' and d' was greater, because this depends on the value of the random s that server A chose. Server A computes $[\lambda]_B \equiv [\lambda' \cdot s]_B$ to get the encrypted bit indicating whether $r' > d'$. As Erkin et al. [EFG+09], we compare the values $2d$ and $2r + 1$ in this step instead of d and r. This does not change any results, but has the advantage of avoiding the case where d and r are equal.

3. Server A now has the encrypted bit indicating whether $r \mod 2^l > d$

mod 2^l. He uses this bit λ to compute: $[z \mod 2^l] = [z' + \lambda 2^l]_B$. Now Server A can compute the most significant bit of z, through computing $[z_l]_B = [2^{-l} \cdot (z - (z \mod 2^l))]_B$. The encrypted minimum of the values $[a]_B$ and $[b]_B$ is computed by Server A as: $[m]_B = [z_l]_B \cdot [a - b]_B + [b]_B$, using a homomorphic multiplication.

4. Server A and B repeat steps 1 to 3 on the inputs $[m]_B$ and $[c]_B$, to get the encrypted minimum of values $[a]_B, [b]_B$ and $[c]_B$.

6.3.8 Smith-Waterman

For the extension of the Smith-Waterman protocol to the malicious user model, we use the same subroutines as for the edit distance in the malicious user model, with very few alterations.

An encrypted edit distance matrix M is used, where $M(i, j)$ is the encryption under Server B's public key of the Smith-Waterman score $H(i, j)$ between substrings $x[0 \dots i]$ and $y[0 \dots j]$. $M(n, m)$ is the encryption of the total score. Server A has matrix M.

We use the minimum-finding protocol for the malicious user model to do maximum finding in the same manner as before using: $-\min(-z_1, \dots, -z_n) = \max(z_1, \dots, z_n)$. The substitution cost protocol for the malicious user model can also be reused by making the same alterations as in section 6.2.5, the substitution cost protocol carried out by the two servers will then compute the cost $c(a, b)$ for characters a, b.

The altered protocol for the Smith-Waterman score is as follows:

1. Server A initiates values $M(0, 0) = [0]_B$, $M(0, j) = [0]_B$ for $0 \leq j \leq m$ and $M(i, 0) = [0]_B$ for $0 \leq i \leq n$.

2. For $1 \leq i \leq n, 1 \leq j \leq m$:

3. Server A and server B initiate the altered substitution cost protocol for the characters x_i and y_j, at the end of which Server A has $[c(x_i, y_j)]_B$.

4. Server A computes the values $[H(i - o, j) - g(o)]_B \forall 1 \leq o \leq i, [H(i, j - l) - g(l)]_B \forall 1 \leq l \leq j$ and initiates the maximum-finding protocol with server B to find $[\max_{1 \leq o \leq i}(H(i - o, j) - g(o)]_B)$ and $[\max_{1 \leq l \leq j} H(i, j - l) - g(l)]_B$. Server A also computes the value $[H(i-1, j-1) + c(x_i, y_j)]_B$, using homomorphic encryption properties.

5. Server A and server B initiate the maximum-finding protocol to find: $[\max(0, \max_{1 \leq o \leq i}(H(i - o, j) - g(o)]_B), \max_{1 \leq l \leq j} H(i, j - l) - g(l), H(i -$

$1, j-1) + c(x_i, y_j))]_B$. Server A gets this maximum value and stores it as $[H(i,j)]_B$, server B learns nothing during this subroutine.

6. At the end of the last round, server A has computed $M(n,m) = [H(n,m)]_B$. This is the Smith-Waterman similarity of strings x and y, encrypted under server B's key. $M(n,m)$ is used to compute the similarity weight $s_{u,f}$ according to Equation 5, which is stored at server A as $[s_{u,f}]_B$ for future reference. The user, friend and server B do not obtain this (encrypted) Smith-Waterman score or similarity measure.

6.3.9 Analysis and Complexity of the Similarity Protocols

For the similarity computation (edit distance or Smith-Waterman) and the subprotocols, security again derives from the original protocols on which they were based [RS10] [JKS08] [EFG+09]. Using the composition theorem [Gol05] again, the edit distance and Smith-Waterman protocol are secure in the malicious user model.

For the new substitution cost protocol, the complexity for user as well as server stays $O(1)$ for both computation and communication. For the new minimum-finding protocol (or the altered maximum-finding protocol), the complexity for communication and computation is now $O(l)$, where l is the bit-length of the inputs.

The complexity of these subroutines translate directly into the complexity analysis for the edit distance and Smith-Waterman protocols in the malicious extension. The complexity for the protocols can be found in Figures 12 and 13.

	Server B		Server A	
step	comp	comm	comp	comm
1.	-	-	$O(\max(n,m))$	-
2.	$O(n \cdot m \cdot l)$	$O(n \cdot m \cdot l)$	$O(n \cdot m \cdot l)$	$O(n \cdot m \cdot l)$

Figure 12: Complexity of the Edit Distance Protocol for two servers (malicious user model)

	Server B		Server A	
step	comp	comm	comp	comm
1.	-	-	$O(\max(n, m))$	-
3.	$O(m \cdot n)$	$O(n \cdot m)$	$O(m \cdot n)$	$O(n \cdot m)$
4.	$O(m^2 \cdot n^2 \cdot l)$	$O(m^2 \cdot n^2 \cdot l)$	$O(m^2 \cdot n^2 \cdot l)$	$O(m^2 \cdot n^2 \cdot l)$
5.	$O(m \cdot n \cdot l)$	$O(m \cdot n \cdot l)$	$O(m \cdot n \cdot l)$	$O(m \cdot n \cdot l)$

Figure 13: Complexity of the Smith-Waterman Protocol for two servers (malicious user model)

6.3.10 Offline Recommender Protocol in the Malicious User Model

Here we present an extension to the protocol discussed in section 6.2.7, so that the solution for offline friends by Jeckmans et al. [JPH13] now works in a setting with malicious users. An overview of the protocol can be found in Figure 14. Before the run of the protocol, for all friends of the user, similarities between the user and the friend $s_{u,f}$ have been computed by server A and server B and stored under encryption of server B at server A, using one of the protocols presented in section 6.3.4.

The same input is used as before in the semi-honest user case. Instead of re-encrypting the shares T_f, U_f for the user, server A now re-encrypts these shares for server B. Server A has the similarities between the user and his friends, encrypted under server B's key.

Server B: $(PK_B, SK_B, PK_A, PK_U, PRE_PK_B, PRE_S K_B)$		Server A: (PK_A, SK_A, PK_B, PK_U) $\forall 1 \leq f \leq F^u$: $S_f, V_f, [[U_f]]_{PRE_{PKf}}$ $[[T_f]]_{PRE_{PKf}}, RK_{fB}, [s_{u,f}]_B$
1.	$\forall 1 \leq f \leq U^f$	
	$\longleftarrow$	$[[T_f]]_{PRE_{PKB}}, [[U_f]]_{PRE_{PKB}}$
	$[[T_f]]_{PRE_{PKB}}, [[U_f]]_{PRE_{PKB}}$	
T_f, U_f		
2.	$\forall 1 \leq f \leq U^f$	
	$\longleftarrow$	$[s_{u,f} + b_f]_B$
	$[s_{u,f} +$ $b_f]_B, [-b_f]_A, [-b_f]_U, [s_{u,f}]_U$	
$[s_{u,f}]_A$		
$[s_{u,f}]_U$	$\longrightarrow$	
	$[s_{u,f}]_U$	
3.	$\forall 1 \leq d \leq D$	
$[z_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot t_{f,d}$		$[a_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot s_{f,d}$
	$\longrightarrow$	
	$[z_d]_U$	
4.	$\forall 1 \leq d \leq D$	
$[g_d]_A = \sum_{f=1}^{F_u} [s_{u,f}]_A \cdot u_{f,d}$		$[h_d]_B = \sum_{f=1}^{F_u} [s_{u,f}]_B \cdot v_{f,d}$
$[g_d + \sigma_{1,d}]_A = [g_d]_A + \sigma_{1,d}$	$\longrightarrow$	
	$[g_d + \sigma_{1,d}]_A, [-\sigma_{1,d}]_B$	
5.	$\forall 1 \leq d \leq D$	
		$[n_d]_U = [z_d]_U + [a_d]_U$
		$[g_d]_B = [-\sigma_{1,d}]_B + (g_d + \sigma_{1,d})$
		$[e_d]_B = [g_d]_B + [h_d]_B$
6.	$\forall 1 \leq d \leq D$	
		$[e_d]_B$
	$\longleftarrow$	$[e_d \cdot \sigma_{2,d}]_B = [e_d]_B \cdot \sigma_{2,d}$
	$[e_d \cdot \sigma_{2,d}]_B, [\sigma_{2,d}^{-1}]_A$	
$[e_d \cdot \sigma_{3,d}]_A =$	$\longrightarrow$	
$e_d \cdot \sigma_{2,d} \cdot [\sigma_{2,d}^{-1}]_A \cdot \sigma_{3,d}$		
	$[e_d \cdot \sigma_{3,d}]_A, [\sigma_{3,d}]_U$	
		$e_d^{-1} \cdot \sigma_{3,d}^{-1}$
		$[e_d^{-1}]_U = e_d^{-1} \cdot \sigma_{3,d}^{-1} \cdot [\sigma_{3,d}]_U$
	$\longleftarrow$	$[p_{u,d}]_U = [e_d^{-1}]_U \cdot [n_d]_U$
	$[p_{u,d}]_U$	

Figure 14: Recommender Protocol for offline friends in the malicious user model

1. Before the start of the actual offline recommender protocol, the main server, server A, will act as a proxy by re-encrypting the shares T_f and U_f for server B. He translates the encryptions under key PRE_PK_F, belonging to the friend, to encryptions under server B's key PRE_{PKB}, using the re-encryption key RK_{fB}.

2. Server A blinds the encrypted similarity $[s_{u,f}]_B$ with blinding factor b_f. He sends the blinded similarity along with the unblinding factors $[-b_f]_A, [-b_f]_U$ to server B. Server B can now compute the similarity under encryption with server A's key as well as the user's key to get $[s_{u,f}]_A$ and $[s_{u,f}]_U$. Server B sends $[s_{u,f}]_U$ to server A, who needs this value in the next step of the protocol.

3. For each drug for which a recommendation needs to be generated, server A and server B compute their shares of the sum of ratings multiplied with the similarity between user and friend. Server B computes the value $[z_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot t_{f,d}$, using the similarity that he received during step 1 and his share of the friend's rating vector. Server B computes $[a_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot s_{f,d}$. Together, the values $z_d + a_d$ evaluate to $n_d = \sum_{f=1}^{F_u} s_{u,f} \cdot r_{f,d}$. Server B sends $[z_d]_U$ to Server A.

4. In a similar manner, the servers compute their shares of the normalization sum. Server B computes $[g_d]_A = \sum_{f=1}^{F_u} [s_{u,f}]_A \cdot u_{f,d}$ and sends it to Server A after blinding it additively with blinding factor $\sigma 1, d$, along with the unblinding factor $[-\sigma 1, d]_B$. Server A simultaneously computes the value $[h_d]_U = \sum_{f=1}^{F_u} [s_{u,f}]_U \cdot v_{f,d}$.

5. Server A now computes the sum-value $[n_d]_U = [z_d]_U + [a_d]_U$ and the normalization-value $[e_d]_B = [g_d]_B + [h_d]_B$. To compute the normalization-value, server A first decrypts $[g_d + \sigma 1, d]_A$ and adds the unblinding factor, to get $[g_d]_B$. The normalization-sum $[e_d]_B$ evaluates to $\sum_{f=1}^{F_u} [s_{u,f}]_B \cdot q_{f,d}$ and needs to be inverted before the drug prediction can be made. The only problem that might occur here is when e_d is equal to 0. This would occur when none of the user's friends have rated drug d. However, the likelihood of this situation is negligibly small.

6. Server A uses multiplicative blinding to blind $[e_d]_B$ and obtains $[e_d \cdot \sigma 2, d]_B$. He sends this, along with unblinding factor $[\sigma 2, d^{-}1]_A$ to server B. The blinding can only be removed under server A's key. Server B uses the unblinding value to obtain $[e_d]_A$ and adds his own

multiplicative blinding value $\sigma_{3,d} \in \mathbb{Z}_t^*$. He sends the result to server A along with the unblinding value $[\sigma_{3,d}]_U$, which can only be removed under encryption of the user's key. Server A decrypts the message to $e_d \cdot \sigma_{3,d}$. He inverts this value to $e_d^{-1} \cdot \sigma_{3,d}^{-1}$ and removes the blinding under encryption with the user's key. Now the server has $[e_d^{-1}]_U$. He computes the recommendation $[p_{u,d}]_U = [n_d]_U \cdot [e_d^{-1}]_U$ and sends it to the user. The user can decrypt this and uses a look-up table for divisions (refer to section 4.5) to get the recommendation value.

6.3.11 Analysis and Complexity of the Recommender Protocol

For the complexity of this protocol, refer to Figure 15. With regards to the SWHE encryption scheme, two multiplications are necessary to carry out this protocol.

	Server B		Server A	
step	comp	comm	comp	comm
1.	$O(F_u)$	-	$O(F_u)$	$O(F_u)$
2.	$O(F_u)$	$O(F_u)$	$O(F_u)$	$O(F_u)$
3.	$O(F_u \cdot D)$	$O(D)$	$O(F_u \cdot D)$	-
4.	$O(F_u \cdot D)$	$O(D)$	$O(F_u \cdot D)$	-
5.	-	-	$O(D)$	-
6.	$O(D)$	$O(D)$	$O(D)$	$O(D)$

Figure 15: Complexity of the Offline Recommender Protocol for two servers (malicious user model)

6.3.12 Role of the Proxy Server

The second server that was introduced for the malicious model, referred to as server B in the discussion of the protocols, has the task of assisting the main server in computations.

For the edit-distance and Smith-Waterman score computations, this proxy server is merely involved in the execution of the malicious minimum-finding and substitution-cost protocols. In the malicious substitution-cost protocol, the proxy server's task is to pick one of the terms that the main server sends to him and send it back after re-encrypting it. All other computations are done by the main server. The operations performed by the proxy server are outsourced by the main server with the purpose to preserve privacy, the proxy server does not need to perform intensive computations.

In the malicious minimum-finding protocol, the proxy server assists by performing modulo reductions (for which the values need to be decrypted), checking for zero-values in a collection of values sent to him by the main server and outputting corresponding encrypted bits. In this case, the operations performed by the proxy server are merely to preserve privacy as well.

In both cases, the main workload lies at the main server, the proxy server is only involved in those steps of the protocol that can't be performed by the main server alone without losing the privacy-preservingness of the protocol.

The recommender protocol in the malicious user model also uses the existence of the second server to outsource operations that cannot be performed by the main server alone without losing security. The proxy server computes sums based on his secret shares and multiplicatively blinds a value so that the main server can decrypt and then invert it. All other computations are done by the main server.

We can conclude that the role of the proxy server is therefore that of a security provider. This underlines the previously made statement that the second server can be a lightweight server that is only needed for security. The proxy server could be supplied by a company specialising in providing security services and in which some degree of trust can be placed. Having a second server whose only goal is to provide security in combination with a main server that is governed by a party (such as a hospital) that needs to maintain an image of being trustworthy further strengthens our grounds for the non-collusion assumption between the two servers in the malicious user model.

6.4 Rating Updates

As users change their ratings for drug treatments, their rating vectors need to be updated accordingly. A privacy-preserving protocol is needed for this. The main server, who stores the ratings, should not find out what rating is being updated nor what the new rating value is. Since malicious users are now allowed in our recommender system, users may try to inject false ratings that lie outside the range of permitted rating-values. The range of permitted rating values can be defined as: $(r_{\min}, \ldots, r_{\max})$. Therefore, the server should be able to do some checks on the new rating value, to make sure that it is a valid rating.

The following protocol (refer to Figure 16) offers a solution to the posed problems. It presents a privacy-preserving way to update users' rating vectors (which are stored at the main server in two secret shares) and indicator

vectors. We discuss in detail the way to update a rating vector. Translation to the case of updating an indicator vector is very simple, the only difference is the values that an element in the indicator vector can take (either 1 or 0). The same protocol can therefore be used for the indicator vector.

The main server, Server A, has the user's current rating vector shares S_u and $[[T_u]]_{pru}$ (where pru is the public key of user u in the proxy re-encryption scheme, likewise prb will occur during the protocol as the public key of server B and $prub$ will occur as the re-encryption key from user u to server B). Together, these shares form the rating vector $R_u = S_u + T_u$. We set the length of these vectors to be $k \in \mathbb{N}$. The requested update is for the new rating value $[r_x]$, where the index x lies between 1 and k. Before the start of this protocol, the user has put in the update request to the main server, supplying the new rating value $[r_x]_B$ and the index at which the update needs to take place, $[x]_B$. The value $[r_x]_B$ has to be split through additive secret sharing during the protocol and both secret shares to the user's rating value have to be updated accordingly at the right index.

Server B $(PK_A, PK_B, SK_B, prb, prskb, pru)$		Server A $(PK_B, PK_A, SK_A, prb, pru, prub)$ $[[T_u]]_{pru}, S_u, [r_x]_B, [x]_B$
1.	$\forall 1 \leq i \leq r_{\max} - r_{\min}$	
	$\longleftarrow$	$a_i = [(r_x - (r_{min} + i - 1)) * \delta_1]_B$
	a_i	
$(r_x - (r_{\min} + i - 1) * \delta_1 \stackrel{?}{=} 0$	$\longrightarrow$	
	true/false	
2.		
		s_x, a
		$[t_x]_B := [r_x - s_x]_B, [x + a]_B$
	$\longleftarrow$	
	$[x + a]_B, [t_x]_B, [s_x]_A$	
$x + a$, t_x		
$y := (x + a - 1)(\mod k) + 1$		
3.		
$\sigma_1 := \{\sigma_{j,1}\} \forall 1 \leq j \leq k$		$\sigma_2 := \{\sigma_{j,2}\} \forall 1 \leq j \leq k$
		$[[T_u]]_{prb}$
	$\longleftarrow$	$[[T'_u]]_{prb} = c([T_u]_{prb}, a)$
	$[[T'_u]]_{prb}$	
$T''_u := T_u + \sigma_1$		
$T''_{u,y} = t_x + \sigma_{1,y}$	$\longrightarrow$	
	T''_u	
		$T'''_u = c(T''_u + \sigma_2, -a)$
	$\longleftarrow$	$[S'_u]_A = c([S_u]_A, a)$
	$[S'_u]_A, T'''_u$	
$[[]T_u]]_{pru} = [[T'''_u]]_{pru}, [S''_u] :=$		
$[S'_u - \sigma_1]_A$		
$[S''_{u,y}]_A = [s_x - \sigma_{1,y}]_A$	$\longrightarrow$	
	$[T_u]_{pu}, [S''_u]_A$	
4.		
		$S_u = c(S''_u - \sigma_2, -a)$

Figure 16: Rating Update Protocol (malicious user model)

1. To check whether user u has put forth a valid update request, server A subtracts all valid rating values in the range $r_{\min}, \ldots, r_{\max}$ from r_x and blinds the resulting values multiplicatively with a blinding value $\delta_1 \in \mathbb{Z}_t^*$. He sends these values $[(r_x - (r_{min} + i - 1)) * \delta_1]$ for $1 \leq i \leq r_{\max} - r_{\min}$ to server B. Server B decrypts these values and checks whether any of them evaluate to 0. If none of the values evaluate to 0, then the user has supplied a false rating which does not lie in the valid range of rating values. In this case, server B sends "false" to server A, who terminates the protocol. Otherwise, server B sends "true" to server A and the protocol continues in step 2.

2. Server A generates a random value $s_x \in F_t$ and a random integer a between 1 and k. He uses the properties of SWHE to get $[t_x]_B$, which is the updated share for server B and to blind the index x at which the update needs to take place, resulting in the value $[x + a]$. a also determines the circular rotation that will be applied in step 3. Server A sends $[x + a]_B$ and $[t_x]_B$ to server B. He encrypts the random value s_x to $[s_x]_A$, which will be used later by Server B to update server A's rating share. Server B receives and decrypts. Lastly, server B computes the new index value y, at which the rating update needs to take place for both secret shares to the user's rating vector. We subtract 1 before the modulo operation and add it after, because our indexes range from 1 to k. If the circular shift by a is such that $x + a$ evaluates to 0, we want to have an index of k.

3. Server A and B simultaneously generate vectors of random values σ_1 and $\sigma_2 \in F_t$. Server A applies the circular shift by a to the proxy re-encrypted share T_u, which he re-encrypts to server B using the re-encryption key $prub$ and sends it to server B, who decrypts his share using his secret key $prskb$ and adds the random vector σ_1, to blind the values. Server B now replaces the y-th position of the share with t_x and subtracts the randomness $\sigma_{1,y}$ here as well and sends the resulting vector T''_u to server A in plaintext. Server A undoes the circular shift and adds in his own randomness σ_2. He sends the result, T'''_u back to server B, who encrypts the updated share under proxy re-encryption with the user's key pru. For the share S_u, server A uses homomorphic encryption under his own key to encrypt the share and then applies the circular shift to the result. He sends $[S'_u]_A$ to server B. Server B replaces the y-th position of the encrypted share with s_x and subtracts his randomness σ_1 from all terms. The result is sent back to server A.

4. Server A subtracts his own randomness σ_2 and then removes the circular rotation from S''_u, resulting in the updated share S_u. After this operation, server A has the updated and proxy-encrypted secret share $[T_u]_p u$, where at the x'th position the value of t_x has been added and has the updated plain share S_u, where at the x'th position the value of s_x has been added.

Privacy

We now look at the privacy of the presented protocol for rating updates. Recall that the two servers in the protocol are semi-honest and non-colluding. The user who supplies the new rating r_x at position x can be malicious. The user may therefore try to supply a 'false' rating, that lies outside the boundaries of permitted values r_{min} and r_{max} and the two servers will try to learn the value of r_x or the position x at which the update takes place. To prove security of the protocol, it should be clear that the servers are not be able to distinguish between a real run of the protocol and a simulation of one, according to the 'real-vs.-ideal' framework [Gol05]. Any messages that the servers receive during a protocol run should therefore be indistinguishable from random.

Privacy of the user towards any of the other users in the system is never in danger, since they are not involved in the protocol. They receive no output from the protocol.

The value r_x, which is the new rating, is sent under somewhat homomorphic encryption to server A. Since the encryption is semantically secure, server A cannot distinguish this message from a random value. The same holds for the message x containing the index at which the update has to be performed. For server B, who can decrypt these messages, the values are first transformed before he receives them. The actual value of the new rating is split at the beginning of the protocol into two shares: server A subtracts a random share-value s_x under somewhat homomorphic encryption from the encrypted update value r_x, so that server B does not get the actual new rating value, but only an additive share t_x which contains randomness chosen by server A. Because of this, the message is indistinguishable from random in the eyes of server B and can therefore be simulated from his point of view. Server A encrypts the random s_x under encryption with his own key, so that server B can add this to server A's share at the right time, but once again, because of the semantic security of the encryption scheme, this message can be simulated from server B's point of view.

Regarding the secret shares T_u and S_u, which are additive secret shares to

the user's rating vector R_u for which an update is requested, we have indistinguishability of these shares from random vectors throughout the protocol, since they are additive secret shares and T_u is only known to server B and S_u is only known to server A. The servers do not collude and therefore are not able to recover R_u.

The changes made to T_u and S_u during the protocol run, are done in such a way that at no point any of the two servers can distinguish the secret shares from vectors of random elements. Server A applies a circular shift with a random value a to the secret share T_u, before re-encrypting it to server B using the proxy re-encryption key from the user to server B. Since a circular shift with a random value is applied, server B cannot decipher at which index x he will perform the update to the secret share. To make sure that the update is performed at the right place, the index at which the update has to be performed is also translated according to the circular shift. The index message is indistinguishable from random by server A, because it is encrypted when server A receives it. After server A translates it according to the circular shift, server B receives it and can decrypt it, but he can at this point also not distinguish it from a random message, because of the added random factor that determines the circular shift.

Server B updates his rating share T_u with t_x and then adds in random additive blinding values to all other positions in T_u, after which he sends it to server A in plaintext. Because of the blinding with random values, server A cannot distinguish the share from a vector of random values. Server A adds in his own randomness before removing the circular rotation, so that server B cannot decipher the location at which the update was made.

The two random vectors that are added to the share T_u, need to be subtracted from the share S_u, otherwise the protocol does not work correctly. This is done during the protocol, partly under somewhat homomorphic encryption, so that server B can subtract his randomness from S_u without needing server A's plaintext secret share to do so. Because this subtraction takes place under somewhat homomorphic encryption, server B cannot distinguish the old or the new value from an encryption of a random message.

We can conclude that during the full run of the protocol, the (encrypted) values T_u and S_u are indistinguishable from vectors of random messages and can therefore be simulated from the points of view of both server A and server B.

With regards to the user who requests an update, there is a chance that the user inserts a false rating value, where r_x lies outside the allowed range of values. The protocol takes care of this by checking the value. Server A sees the encrypted r_x and subtracts all allowed values from it, under

somewhat homomorphic encryption. He cannot distinguish any of these messages from random, due to the semantic security of the used encryption scheme. These values are then blinded multiplicatively and sent to server B. Server B decrypts the messages and checks them zeroes (if no zero occurs, the rating is false). Because of the multiplicative blinding, server B cannot distinguish any of the received messages from random and from his point of view, this part of the protocol can therefore be simulated.

We conclude that the protocol is privacy-preserving. All messages that are received during the protocol are either indistinguishable from random to the point of view of the receiver because they are encrypted with a semantically secure encryption scheme, because they are secret shares or because they are blinded either additively or multiplicatively with random values. The user receives no output from the protocol and is caught if he submits a false rating. The two servers cannot distinguish a real run of the protocol from a simulation.

6.5 Other Application Areas

We have now put forth the design of a privacy-preserving DNA-based social recommender system in the setting of personalized medicine, where drug treatment recommendations are generated based on DNA similarity between patients. Other application areas for the recommender system that we described and implemented also exist.

An example of an application area in which the recommender system might be interesting, is an online dating service where individuals are matched based on their genetic compatibility. A company that provides such a service is GenePartner[3]. Individuals wishing to use their service are asked to present a sample of their DNA, which is stored by the company. Having privacy-preserving DNA matching and privacy-enhanced recommender protocols in a social network of people who use the dating service (where matches could be made based on genetic compatibility and on ratings by others) would be useful to the users of such a service.

6.6 Limitations in the Malicious User Setting

To accommodate for the existence of malicious users, we introduced a second server that is semi-honest. Users are excluded from all protocol operations (except for setting them in motion by requesting a recommendation) and are only required to provide input during the registration phase. Users then

[3] `http://www.genepartner.com/`

provide DNA-material to the main server (the secret shares to the indicator collection: $(X_{u,i})$ and $[(Y_{u,i})]_U$, the encrypted indicator collection $[I_{u,i}]_U$ and the encrypted genome sequence $[G_u]_U$). Users also provide to the main server a re-encryption key for the proxy server and their rating vector with the accompanying indicator vector, both of which are split into secret shares.

Even though all user data which is stored by the main server is stored either encrypted or as secret shares, where one share is encrypted, quite a lot of user data is stored at the server and so quite a lot of trust is placed in the server. When both servers follow protocol, neither server will ever know any of the user's confidential information, since neither server has access to the unencrypted information or to both secret shares that form a piece of data. However, if the servers were to collude, all user data would be discovered: the genome sequence, the rating vector and the accompanying indicator vector.

This is no different from the security setting in which users were required to be semi-honest: then, if the user and the server were to collude, they could also retrieve the user's friends' dna-material and rating vectors.

However, this means that the security of the malicious user model (also) builds upon a non-collusion assumption, which is a limit to the security. The security setting where two servers are required to be semi-honest and non-colluding is stronger than the security setting in which users are also semi-honest, but it would be even better to have a recommender system in which every entity could be malicious, but could still not recover any information about any of the other entities. To achieve this security requirement, we would need to have protocols in which all operations on the users' data could be carried out without having to share the data between two servers. This is a current limitation of the recommender system and it remains future work to study whether this security requirement can be achieved efficiently.

7 Experimental Results

To analyze the performance of our protocols, we implemented prototypes in C++. We used several libraries as building blocks for our system, especially for the somewhat homomorphic encryption and for the use of elliptic curves in the proxy re-encryption scheme.

The implementation of the system consists of prototypes for: the edit distance protocol, the substitution cost protocol, the minimum finding protocol, the smith-waterman distance protocol and the offline recommender protocol. All of these protocols have been implemented for both the semi-honest user model as well as the malicious user model, except for the minimum-finding protocol, of which only the malicious version was implemented. The reason that the minimum-finding protocol in the semi-honest model was not implemented is that there were some implementationwise technical requirements that made an actual implementation impractical. When using a group of integers to represent the messages, there will be overflows of integers that wrap around the group. However, the protocol that our version of minimum-finding in the semi-honest model was based on, uses comparisons that will give incorrect results when wraparound happens, which will happen often. A very large message space would be needed to solve this problem, but this would lead to a very impractical implementation.

We also implemented the rating update protocol and the proxy re-encryption scheme by Ateniese et al. [AFGH06], which is used in the implementation of the recommender protocols.

All prototypes have been implemented so that for each party in the protocol, steps are taken sequentially on the same machine. All protocols have been implemented in a single thread and have been tested on a virtual machine with a 2x Intel Xeon CPU at 2.33 GHz with 8GB of RAM.

7.1 Random data for tests

For testing the similarity computation protocols, we generated random DNA-sequences represented with the integers $1, 2, 3, 4$ of different lengths. For testing the recommender protocols, we altered the existing datasets that were used by Jeckman's et al. [JPH13] slightly to fit our input requirements, in order to run our data on the same ratings that were used for the evaluation of the performance of their recommender system. In order to test the rating updates protocol, we generated artificial user rating vectors where the ratings lie between 1 and 5. This rating range was chosen out of convenience, but does have some impact on the efficiency of the protocol, since

the two servers compare an updated rating value to all possible rating values at the beginning of the protocol to see whether the supplied and encrypted rating value is legitimate (and does not lie outside the permitted rating value boundaries). The impact on efficiency will not be very great however, because this process is linear in the amount of permitted rating values. This range of allowed rating values is also a good fit for most recommender systems, it would be a good representation of a rating system where users give stars to items for example.

7.2 Libraries

SWHE-encryption

For the practical implementation of our recommender system, we had the option to choose from several libraries that implement an SWHE scheme. At an early stage of the research, we considered the HElib [HS14], which implements the BGV-scheme [BGV11]. However, later on we chose to use the BV-scheme [BV11] as a basis for the SWHE, since it provided some advantages regarding our system design (the main one being that we only needed a relatively small message space for our system). Since we chose the BV-scheme, there were two possible options for a cryptographic library. The first was an open-source library written in Java: the jLBC[4]. The other option was the BV-implementation written by Arjan Jeckmans[5], which is based on the GMP[6] and FLINT[7] libraries. After trying out both implementations, it was apparent that the code written by Jeckmans was faster than the jLBC library, which is why his code was chosen to use as the BV-implementation for our recommender system.

Elliptic Curve Cryptography

For our implementation of the proxy re-encryption scheme, we used the Pairing-Based-Cryptography Library (PBC) [8], which is a free C-library built on GMP that allows us to perform cryptographic operations on elliptic curves.

[4] `http://gas.dia.unisa.it/projects/jlbc/`
[5] `http://scs.ewi.utwente.nl/other/jeckmanscode/`
[6] `https://gmplib.org/`
[7] `http://flintlib.org/`
[8] `http://crypto.stanford.edu/pbc/`

7.3 Choice of Parameters

BV-scheme

For the choosing the parameters for the BV-scheme, we follow the guidelines by Naehrig et al. [NLV11]. Theorem 3.3 from their paper gives a lower limit for the prime q that is used in the scheme as the ciphertext modulus, through the following equation:

$$q \geq 4 \cdot (2t\sigma^2\sqrt{n})^{D+1} \cdot (2n)^{D/2} \cdot \sqrt{A} \tag{8}$$

If the parameters used comply with this equation, then, according to the theorem, the scheme will work correctly and securely under the Ring-LWE assumption. Here, M is the allowed number of multiplications, A the allowed number of additions, n is advised to be a power of 2, and t is the plaintext space modulus. Naehrig et al. also give a relation between the parameters with which the security of the resulting BV-scheme can be computed. We use their proposed parameter $\epsilon = 2^{-32}$ as the success probability of the attacker in the distinguishing attack when computing the optimal runtime of the attacker.

For the edit distance computations that we carry out, if we for example fix $D = 3$, $A = 1000$, $t = 1021$, $n = 4096$, this results in a minimal q of 119 bits with a resulting security of the attacker logarithm runtime in terms of basic CPU operations of 2^{144}. Refer to Table 6 to view the selected parameters for the similarity computations and their resulting security.

We choose a rather small message space for the edit distance computations, since the computations are all performed on relatively small integers that do not exceed the length of the compared DNA-sequences by much. The parameter t is therefore dependent on the lengths of DNA-sequences with which we experiment. Refer to section 7.5 for our motivation for the selected values for t.

For the number of allowed multiplications, we set $M = 3$, since this is the number of multiplications needed for our implemented system to work correctly.

t	D	A	n	$log_2(q)$	security (in bits)
1021	3	1000	4096	119	144
1021	3	1000	8192	122	386

Table 6: BV-scheme parameters for similarity computations - edit distance

We now have two sets of parameters with different security levels of the attacker logarithm runtime and different message spaces with which we run experiments. Both of these parameter sets fulfill the requirements set by Naehrig et al. [NLV11], who state that the security level must be at least 128 in order to have a correctly working secure scheme.

After some initial runs of the Smith-Waterman protocols on the parameters in table 6, we sometimes get incorrect results. This is due to noise overflow in the ciphertexts that are computed during the protocol runs, since there are more subroutines of the protocols used for every computation of a matrix value. Therefore, for the Smith-Waterman computations, we specify parameter sets with slightly larger keys to solve this problem, which somewhat reduces the security parameter. The parameter sets that we use are the following, found in table 7.

t	D	A	n	$log_2(q)$	security (in bits)
1021	3	1000	4096	122	138
1021	3	1000	8192	126	370

Table 7: BV-scheme parameters for similarity computations - Smith-Waterman distance

For the recommender system protocols and the rating update protocol, we again determine different keyparameters. Since the number of needed multiplications is lower for these protocols, we can fix $M = 2$. However, the message space must be much larger for these protocols, to accommodate for a large number of users in the system. Table 8 shows the different parameter sets which will be used for the experiments of these protocols.

t	D	A	n	$log_2(q)$	security (in bits)
5000011	2	1000	4096	126	130
5000011	2	1000	8192	129	358

Table 8: BV-scheme parameters for recommender computations

In all three cases, we have two key parameter sets where the first has a normal security level, of respectively 144 or 138 and 130 bits of the attacker logarithm runtime, and where second keyset has a very high security level of respectively 386 or 370 and 358 bits of the attacker logarithm runtime. We will compare the runtimes of the protocols on these two key parameter

sets, to see what effect the security level has on the efficiency of the system.

Proxy re-encryption

For the proxy re-encryption scheme, a symmetric pairing is required. We use parameters for the elliptic curve provided by the PBC, namely type A pairings. This type of pairing provides a base field size of 512 bits [9]. The parameters for the elliptic curve are: prime q (the order of the field used) is 512 bits and $r = 730750818665451621361119245571504901405976559617$, which is the order of the subgroups used for the pairing and is a Solinas prime. The elliptic curve that is used is $Y^2 = X^3 + X$. Using these parameters, we achieve a security of 256 bits for the proxy re-encryption, which is a good level of security according to the NIST recommendations for key-lengths[10]. The runtimes (in milliseconds) on random data for the first- and second-level decryption and encryption operations, for the re-encryption and for key-generation (two public/private keypairs and a re-encryption key) of our implementation are given in table 9.

Operation n	Runtime *(ms)*
Encryption (1st)	10.6935
Decryption (1st)	0.238691
Encryption (2nd)	2.52118
Decryption (2nd)	1.4941
Re-encryption	1.38101
Keygen	7.65707

Table 9: Proxy re-encryption performance

The scheme is very efficient, since all operations take only *milliseconds*. We use the implementation of this scheme for these parameters in the implementation of other protocols where proxy re-encryption is used.

7.4 Timings of the Protocols

7.4.1 Similarity Computations

The edit distance protocols and Smith-Waterman protocols were tested on inputs of different sizes, using the given key parameter sets. In the tables

[9] `http://crypto.stanford.edu/pbc/times.html`
[10] `http://www.keylength.com/en/4/`

below, the runtimes for these protocols are shown. Runtimes are specified in *seconds*.

Input size n	SH KS1 (s)	Mal KS1 (s)	Mal:SH ratio	SH KS2 (s)	Mal KS2 (s)	Mal:SH ratio
10	382.014	859.754	2.25	464.716	1135.83	2.44
20	1638.76	2101.08	1.28	3830.38	4915.43	1.28
50	11603	14998.4	1.28	–	–	-

Table 10: Execution times of Edit Distance Protocol

The execution speed for the edit distance protocol is in the order of minutes for input lengths of 10, in the order of thousands of seconds for input lengths of 20 and ten thousands of seconds for input lengths of 50. For the keyset one, in both the semi-honest and the malicious protocol, this comes down to approximately half an hour for length 20. For the larger keyset, it takes more than an hour. In the case of an input length of 50, the computation takes hours in the semi-honest case. This inefficiency is largely due to the key parameters that are needed to enable three multiplications during the protocol and due to the fact that the edit distance protocol is a recursive protocol, so the growth of the execution time is exponential in the input size. The difference between runtime of the semi-honest and the malicious version of the edit distance protocol differs for different input sizes, but we see a factor of 1.28 for lengths 20 and 50. For a small input size of 10, the difference is greater, which may be explained by the overall runtime being smaller which would lead to greater variation in timings of specific protocol runs. It is clear that the malicious version of the protocol is slower than the semi-honest version, which leads to especially inefficient timings on large inputs.

The results for the edit distance protocol on input length 50 for keyset 2 are missing from this table. Running the protocol on these specific parameters gave incorrect results, due to noise overflow and would have to be run again on larger parameters such as were set for the Smith-Waterman protocol. However, this would not give an accurate comparison to the other timings in the table, which is why these timings were left out.

Even though the performance of the semi-honest and malicious edit distance protocols are not very efficient, these computations can be done on a server that has more resources than the machine on which these protocols were tested for performance. More importantly, the computations can be done in an off-lne phase during the recommender system set-up, which is also how we designed our recommender system, to allow for some inefficiency in the similarity computations.

Input size n	SH KS1 (s)	Mal KS1(s)	Mal:SH ratio	SH KS2 (s)	Mal KS2 (s)	Mal:SH ratio
10	2027.72	2324.37	1.15	5381.38	5882.59	1.09

Table 11: Execution times of Smith-Waterman Distance Protocol

The execution times for the Smith-Waterman protocol are much slower than those for the edit distance. This is to be expected, as the complexity for the Smith-Waterman protocol is a factor of $n*m$ greater than that of the edit distance protocol for some of the protocol steps, due to the recursive minimum-finding step on all previous column or row values in the Smith-Waterman score matrix. Where the inefficiency of the edit distance protocol could be tolerated as the protocol is ran during an offline set-up phase, the inefficiency of the Smith-Waterman protocol on sequences as small as length 10 would be a deterrent from using this implementation in a real-life application of the system and would be a good argument for using the edit distance as a similarity score instead of the Smith-Waterman distance. The difference between the malicious and the semi-honest versions of the protocol is not very great this time, the malicious protocol is only a factor of 1.15 slower than the semi-honest version on keyset 1 and a factor of 1.09 slower on keyset 2. We did not run the Smith-Waterman protocols on larger input sizes, since the results on sequences of length 10 already proved very inefficient, on keyset 1 the execution time was more than half an hour and on keyset 2 the execution time was more than an hour. Running the protocols on larger input lengths gave wrong results, which is again due to noise overflow in the ciphertexts. An option to remedy this would be to change the keyparameter sets even more to allow for more homomorphic operations. We decided against this, since changing the key parameters to enable the protocol to run on bigger inputs would only decrease the performance more.

7.4.2 Recommender Protocol

We run experiments on the previously mentioned key parameter sets for different input sizes for the number of friends F and the number of drugs D. All results are runtimes for generating D recommendations. An initial run of both the semi-honest and the malicious user protocol for the recommender gives the runtimes shown in table 12. The used input sizes were $F = 10$ and $D = 10$. Runtimes are given in *seconds*.

t	M	A	n	$log_2(q)$	Runtime SH (s)	Runtime Mal (s)
5000011	2	1000	4096	126	6.08213	4.96252
5000011	2	1000	8192	129	14.3631	13.5413

Table 12: Execution times for different keysets on server without proxy re-encryption, $F = 10$,$D = 10$, where F is the number of friends, D is the number of drugs

For the different keysets, we now run the recommender protocols on various larger input sizes. We use the same input sizes as in the research by Jeckman's et al. [JPH13], namely: $F = 50, 100, 200$ and $D = 500, 1000, 2000$. The results of these runs can be found in table 13.

F	D	SH KS1 (s)	Mal KS1 (s)	SH:MAL ratio	SH KS2 (s)	Mal KS2 (s)	SH:MAL ratio
50	500	335.132	318.170	1.05	738.722	705.927	1.05
100	500	522.174	508.343	1.03	1134.356	1108.950	1.02
200	500	904.907	887.699	1.02	1899.185	1878.125	1.01
50	1000	641.521	615.945	1.04	3306.858	1823.016	1.81
100	1000	995.472	975.940	1.02	5011.672	2798.317	1.79
200	1000	1712.765	1687.519	1.01	6539.349	4432.331	1.48
100	2000	3015.882	2354.550	1.28	11081.609	7764.834	1.43
200	2000	4746.540	3801.570	1.25	13552.790	10818.180	1.25

Table 13: Execution times in *seconds* of recommender protocols, including proxy re-encryption. Columns 1 and 2 show the number of friends and drugs. Columns 3 and 4 show the execution times of the semi-honest and malicious protocols on keyset 1, columns 6 and 7 show the execution times of the protocols on keyset 2. Columns 5 and 8 show the ratio between semi-honest and malicious protocols on keysets 1 and 2 respectively.

The execution times are given in *seconds*. For all input sizes for keyset 1, the protocols run in the order of minutes on average. For a very large input size of 200 friends and 2000 drugs, for example, the malicious protocol will take approximately one and a half hour to compute the recommendations for all of these drugs. In comparison, on an input size of 50 friends and 500 drugs, it only takes approximately six minutes for the malicious protocol to compute the recommendations. What is interesting is that the semi-honest protocol is slower than the malicious protocol in all cases. For small input sizes, the difference in runtime is small, but this difference becomes greater on larger inputs. This can be explained by the fact that, though both protocols have the same complexity, except for in the first step of the protocol where there is more communication overhead in the malicious protocol, in

steps three and five, the semi-honest protocol requires more computations over the number of drugs D than the malicious protocol. When the drug parameter rises, this difference becomes apparent by an increase in runtime in the semi-honest protocol.

The following figures 17181920 show the runtimes in *minutes* for the semi-honest and malicious protocols on both keysets.

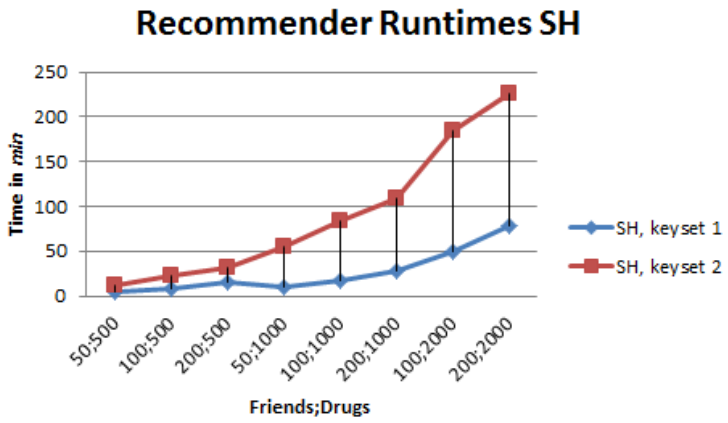

Figure 17: Semi-Honest Recommender Runtimes on keysets 1 and 2

Figure 17 shows the runtimes of the recommender protocol in the semi-honest user model for keysets 1 and 2. There is an exponential growth in runtime as the number of friends F and the number of drugs D increase. The difference in runtime between keysets 1 and 2 increases much as the input parameters grow, which is due to the larger keyparameters for keyset 2. For keyset 1, the protocol runs in under twenty minutes for most input sizes (5 minutes in the best case for 50 friends and 500), but for large input sizes of 1000 drugs and 100 or 200 friends, the protocol will take an hour or longer to run, which would be too inefficient for a real-time application of the recommender system. The timings for keyset 2 get inefficient faster, for input sizes of 200 friends and 500 drugs and onwards, the protocol will take half an hour to run in the best case and more than one hour in the worst case.

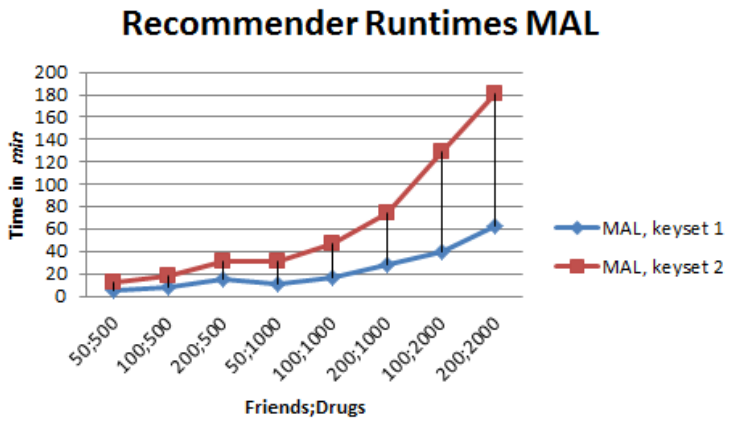

Figure 18: Malicious Recommender Runtimes on keysets 1 and 2

The trend in growth of runtimes for the recommender protocol in the malicious user model on both keysets is roughly the same as for the semi-honest user model. The timings are better overall for the malicious version of the protocol, for most input sizes up to 100 friends and 1000 drugs, the protocol will run in less than 15 minutes on keyset 1 and will take more than forty minutes to complete for large input sizes of 1000 drugs and 100 or 200 friends.

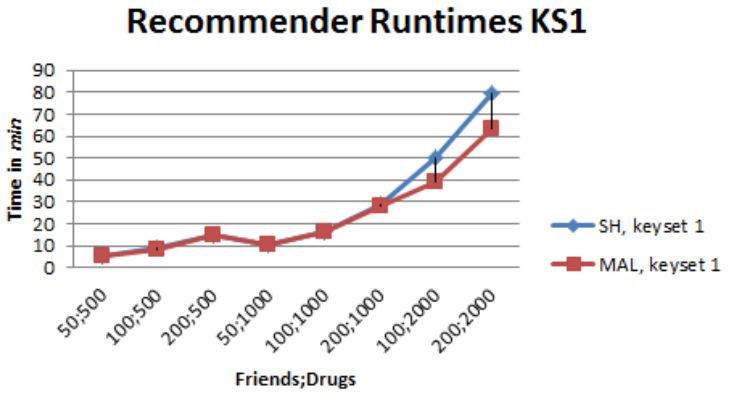

Figure 19: Semi-Honest and Malicious Runtimes on keyset 1

Figure 19 shows the difference in execution time between the semi-honest and the malicious protocol on the first key parameter set. There is very little difference between the execution times here, only for large input sizes of 2000 friends and 100 or 200 drugs does it become apparent that the semi-honest protocol takes longer to execute.

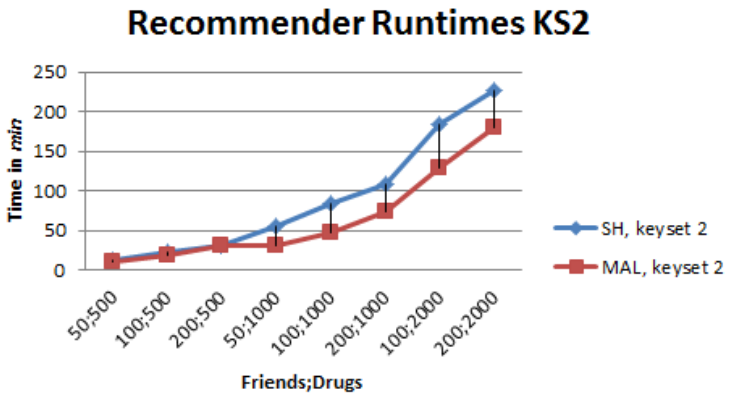

Figure 20: Semi-Honest and Malicious Runtimes on keyset 2

On the second key parameter set, the difference is much clearer. Figure 20 shows the different runtimes for the semi-honest and the malicious protocols for keyset 2. The timings for small input sizes are very similar, but for inputs with more than 500 drugs, the semi-honest protocol performs clearly worse than the malicious protocol.

The factor with which the semi-honest version of the protocol differs from the malicious version are shown in table 13. An interesting observation here is that the difference in execution time is almost negligible for small parameter sets on both keysets, where the semi-honest protocol is 1 to 5 per cent slower than the malicious protocol. For an input size of 2000 drugs however, however, we see a large jump in these execution times on keyset 1. Here, the semi-honest protocol is 28 and 25 per cent slower than the malicious protocol for 100 and 200 friends respectively. On keyset 2, these differences are even greater. For 1000 drugs, the semi-honest protocol is approximately 80 per cent slower than the malicious protocol for 50 and 100 friends, but is only 48 per cent slower for 200 friends. The difference in timings therefore becomes greater as the input sizes get larger and as the

key parameters get larger. The sudden jump in ratios can be explained by the fact that the input parameters for F and D that are chosen are not chosen linearly. For 2000 drugs on keyset 2, the semi-honest protocol is 43 per cent slower for 100 friends and 25 per cent slower for 200 friends. This decrease in timing difference for keyset 2 on parameters of 1000 drugs and 200 friends or larger is an unexpected result. It is possible that the results for 1000 drugs and 50 or 100 friends are outliers.

We do see a general decreasing trend in the ratios of the execution times for a constant input size for the number of drugs and an increasing input size for the number of friends. This holds for both keyparameter sets and all possible input sizes for D. The reason for the execution times of the semi-honest and malicious protocol to lie closer together when a larger number of friends is taken for the recommendation generation, is that the difference in computational work between the two versions of the protocol is mostly dependent on the input size of D. When the number of friends F grows larger, the influence of D becomes slightly smaller, thus resulting in a smaller difference between the execution times between the semi-honest and malicious recommender protocol.

In general, we can conclude that the malicious protocol is more efficient than the semi-honest protocol and that the difference in performance generally rises as the combined input size for the number of drugs and friends used in the computation or the key parameter size grows larger.

7.4.3 Rating Updates

The rating updates protocol was performed on random rating vectors, where R denotes the number of ratings in each vector. We chose a constant number of users for testing the efficiency of the protocol and generated data for 50 users, since this factor does not influence the speed of the protocol (the update takes place for one user only). We ran the protocol on both keysets for $R = 10, 100, 200, 500, 1000, 2000$. The results given in table 14 show the time needed to update one rating in a rating vector of size R for one user. Runtimes are given in *seconds*.

R	Runtime for KS1 (s)	Runtime for KS2 (s)	$\frac{KS2}{KS1}$
10	0.751304	1.77817	2.36678
100	6.10326	14.376	2.35546
200	12.0838	28.7247	2.37713
500	29.9851	716.828	2.39061
1000	60.3183	144.544	2.39635
2000	120.756	288.278	2.38728

Table 14: Execution times of rating updates protocol, including PRE.

The ratio of the runtime on keyset 2 divided by the runtime of the protocol on keyset 1 is approximately 2.4 and is the same for different sizes of the rating vector. The increase of security therefore slows down the protocol by more than a factor of two.

For the increase in the number of ratings that is considered for the update, we see a semi-exponential growth in the runtime of the protocol, for both keysets. This can be seen in figure 21.

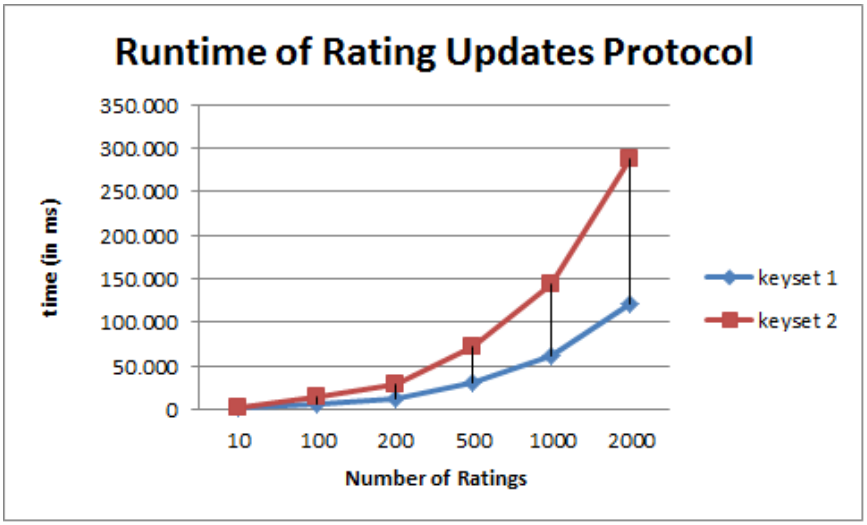

Figure 21: Rating Updates Runtimes

7.5 Example use case for Similarity Computations

An example of a situation where DNA similarities can be computed to determine whether an individual is a carrier for a certain disease is found in a research by Gordillo and Vega [GV08], who studied the autosomal recessive disorder known as the Roberts Syndrome. This syndrome is caused by mutations on the ESCO2 gene. Molecular genetic testing can identify the risk of inheritance of the mutations that cause Robert's syndrome in a family. A number of 26 SNPs need to be examined in order to find all possible mutations. This study motivates our choice for input DNA lengths of 20 and 50; by using privacy-preserving DNA matching on relatively small input lengths, a mutation such as one of the mutations on the ESCO2 gene that causes Robert's Syndrome could be tracked down.

8 Discussion and Conclusion

In this research, we designed a recommender system that incorporates privacy-preserving DNA-matching into a social and privacy-preserving recommender system. We studied techniques for privacy-preserving DNA-matching and studied various types of privacy-preserving recommender systems that exist. Then, we designed and implemented the edit-distance and Smith-Waterman algorithms for computing DNA-similarities between users of our envisioned recommender system in a privacy-preserving way, based on previous work by Rane and Sun [RS10] and Erkin et al. [EFG+09]. These similarities were then incorporated into a previously designed social recommender system by Jeckman's et al. [JPH13] in the semi-honest user model, which we extended to add extra security guarantees for the users of the system (specifically regarding the privacy of the indicator vector with which ratings that a user adds can be hidden). For the encryption in all of the designed protocols for our recommender system, we used the somewhat homomorphic encryption of the BV-scheme [BV11], which is an efficient form of encryption that allows for a limited number of homomorphic additions and multiplications of ciphertexts.

Our designed recommender system was initially developed for the semi-honest user model, as was the original recommender system that we based our design upon, but we later extended it to the malicious user-model in which two non-colluding servers are employed. This is a stronger security model and is more realistic than the semi-honest user model for a real-life application of the recommender.

To accomodate for changes in ratings made by users of the system, we then designed and implemented a secure rating update protocol, which is carried out by two non-colluding servers in the malicious user model.

In this section, we will recall our research goals and will discuss how we achieved these goals. We will also discuss the design and implementation (in terms of efficiency and security) of our recommender system and compare the efficiency of the implementation to that of the recommender system by Jeckman's et al. [JPH13], which it was based upon. We then draw a comparison between the efficiency of our DNA-matching and the disease susceptibility results by Ayday et al. [ARHR13] and compare the efficiency of our implemented proxy re-encryption scheme to the benchmark of Ateniese et al. [AFGH06].

We conclude this section with a discussion of the limitations of our research and recommendations for future work.

Research Goals

Our main research goal was to design and implement a privacy-preserving DNA-based social recommender system, that uses privacy-preserving DNA-matching to compute similarities between users of the recommender system and combines this with familiarity between the users to generate recommendations in a privacy-preserving manner.

To do so, we slightly altered the recommendation formula by Jeckman's et al. [JPH13] to include the DNA-similarity score as a weight for the rating prediction.

Then, we looked at different techniques for privacy-preserving DNA-matching to see which would fit into our recommender system best. We chose the technique of a substitution cost protocol by Rane and Sun [RS10] and the minimum finding technique by Erkin et al. [EFG+09] to adapt to our system architecture and chosen cryptographic scheme, namely the somewhat homomorphic encryption scheme by Brakerski and Vaikantunathan [BV11]. These subprotocols were adapted to both the semi-honest and the malicious user model and were used in the computation of the edit distance and the Smith-Waterman score. The main differences between the original subprotocols and our adaptations were the use of somewhat homomorphic cryptography and the use of secret shares and indicator vectors as input to the protocols, which made some adjustments to the computations necessary. We chose to enable both of these similarity scores to make the recommender system more dynamic, so that a choice for a specific similarity score could be made later on. The edit distance will in most use cases suffice for DNA-matching, since most of the time a set of SNPs needs to be compared, but the Smith-Waterman gives an additional score that takes into account multiple alignments of two DNA-sequences, which might prove useful in some cases.

One of our concrete research questions was whether we could devise an offline recommender protocol and what the privacy requirements would be for storing user data on the server. We designed the offline recommender based on the recommender for offline friends by Jeckman's et al. [JPH13] and altered it to fit our system architecture. The privacy requirements that followed from having a recommender system that can compute recommendations when users are offline, were that users need to store their data in secret shares at the server under proxy re-encryption, so that for each piece of data one of these shares could be re-encrypted for another user or another server with whom the main server initiates the protocol. In our system design, this meant that not only the rating data needed to be divided into shares and

stored on the server, but that the DNA data also needed to be stored at the server in secret shares, since we could count upon both users being online at the same time to perform the similarity computations. The requirement that users and friends can be offline also meant that DNA-similarity computations need to be performed by the server and a user (or the user and a proxy in the case of the malicious model) for all of the user's friends once the user registers, in a system-set up phase.

Summary of Results

DNA-matching

The implementations of the edit distance in the semi-honest and the malicious user model were not very fast, but could be used in an offline phase where DNA-similarities are computed between users beforehand. This would take much computational power of a main server, especially if multiple similarities are computed at the same time. A drawback of the inefficiency is that, depending on the length of DNA over which similarities need to be computed, the user will have to wait for minutes or even an hour to have access to the recommender system. The Smith-Waterman similarity score proved too inefficient to use, even in a precomputation phase when the user registers with the recommender system.

The performance of the similarity protocols that we implemented indicate that somewhat homomorphic encryption may not be the best encryption scheme to use for the implementation of privacy-preserving DNA-matching. The edit distance protocol performed reasonably well, but its efficiency is not very good. The Smith-Waterman distance protocol was very inefficient. Using somewhat homomorphic encryption for protocols that require many multiplications will therefore not give good performance. However, new somewhat homomorphic encryption schemes that may be developed in the future may change this and boost the performance of the privacy-preserving DNA-matching protocols.

Off-line Recommender

Our offline recommender protocol gave good results. Though it is slow on very large inputs, the performance of the protocols is good enough for use in a real-life application of the recommender system. What is most notable is that the protocol performs better in the malicious user model, which is the preferred security model.

Rating Updates

The rating updates protocol was implemented and ran very efficiently. It can be used in the malicious user model when two non-colluding servers are employed.

Comparison of Results

We will now compare the timings of our DNA-similarity computations to the timings given in the paper by Ayday et al. [ARHR13]. Recall that their research focused on a new architecture for privacy-preserving disease susceptibility tests, where SNPs that contain markers for certain diseases are compared per patient to compute a disease susceptibility. Though their genetic tests and system architecture are different from ours, since we consider DNA-similarity between users of our recommender system (either the edit distance or the Smith-Waterman similarity score), the main idea behind it is to compare SNPs in a privacy-preserving way, which can be translated to the setting of our similarity computations by selecting the right subsequence of a DNA-string. Of the reviewed literature in section 2.3 that have implementations and performance results, their manner of privacy-preserving genetic tests comes closest to our implemented techniques for similarity computations.

The performance of Ayday et al.'s [ARHR13] was tested on DNA-sequences of length 10 (10 SNPs) on an Intel Core i7-2620M CPU with 2.70 GHz processor. For a key-size of 2048 bits, the homomorphic operations of the susceptibility test took 25 seconds, while for a key-size of 4096 bits, the operations took 100 seconds. The security for their 4096 bits key corresponds roughly to our keyset 1 for the BV-scheme. Our timings on this keyset for sequences of length 10 are 382 seconds in the semi-honest edit distance protocol and 465 seconds in the malicious edit distance protocol. We see that our genetic test is therefore around 4 times slower than the results by Ayday et al. [ARHR13].

In order to make a realistic comparison on the efficiency of the recommender system, we ran the code for the offline protocol by Jeckman's et al. [JPH13] on our own machine, using keyset 1 which was defined in section 7.3, for large input parameters of 100 friends and 2000 books. The protocol executed in slightly over 26 minutes on our machine, whereas the performances listed in their paper noted a runtime of approximately 17 minutes for the same input. The reason for this difference lies mainly in the usage of different parameters for the BV-scheme, which is necessary for the

larger amount of multiplications that must be supported in order for our protocols to run correctly, and may also be due to the different set-up of machines on which the code was tested. The larger key parameters that we use are necessary because of the fact that in our recommender system, we also hide the indicator vector from the user or from the second server. This means that the main server has two sums to compute, of which one needs to be inverted. To invert this without loss of privacy, the value has to be blinded multiplicatively two times, whereas in the original protocol, the indicator vector is not hidden from the user (or second server in our case) and the main server therefore does not need to compute a sum before inversion. In that case, the user, or second server, can already do the multiplicative blinding without the main server needing to add his own multiplicative blinding to hide the sum from the server.

Where the protocol by Jeckmans et al. [JPH13] took 26 minutes on our machine on keyset 1, our own implementation in the semi-honest user model took slightly over 51 minutes to execute. Our implemented recommender system has worse performance than the original recommender for offline friends by Jeckmans et al. [JPH13], this is also apparent from the other timings given in their paper and the timings given in table 13. This is to be expected, however, since the recommender protocol that we designed and implemented requires more computations on ciphertexts during the protocol run to generate the recommendations.

One of the main differences between the results in this research and those of Jeckmans et al. [JPH13] is that we also implemented the recommender for the malicious user model, where two non-colluding semi-honest servers are employed. We therefore implemented a recommender in a stronger security model, which is an addition to the current research on recommender systems.

Veugen et al. [VdHCM15] presented a framework for computing recommendations by using two non-colluding servers. Their architecture is very similar to ours, which is why a comparison to their results is also in order. Their security model however, differs from ours in the sense that one of the servers is allowed to be malicious, where in our case both servers are required to be semi-honest. Their recommendations are computed using homomorphic encryption in a collaborative filtering approach, which is similar to the technique that we use (only we use somewhat homomorphic encryption). Another difference in their recommendation computation is that they use a pre-processing phase for generating some system parameters, which takes a load of the work to be performed by the two servers later on. The timing for one recommendation when $10,000$ users are considered is 0.34 seconds, but this is excluding four hours of pre-computation.

The results by Veugen et al. [VdHCM15] are very efficient, especially since they are able to move a lot of the computations to a pre-computation phase. Their recommender is therefore more efficient than ours. However, this statement should be moderated by adding that their execution time is only for the computation of one recommendation, while our performance results compute the ratings for all drugs D. On the other hand, their implementation was tested on a greater number of users.

For proxy re-encryption, we implemented the second option by Ateniese et al. [AFGH06]. Our timings for this implementation were given in the previous section. The paper by Ateniese et al. describes their own timings for the third option, which is similar to the second option but is dependent on a different security assumption. They tested their protocol on an AMD Athlon 2100+ 1.8 GHz with 1 Gbyte RAM client with an IBM 7200 RPM, 40 Gbyte, Ultra ATA/100 hard drive and an Intel Pentium 4 2.8 GHz with 1 Gbyte RAM server with a Seagate Barracuda 7200 RPM, 160 Gbyte, Ultra ATA/100 hard drive. We compare our timings to their server timings for a 512 bit parameter size. The respective timings are given in table 15.

Operation	Runtime *(ms)*	Runtime Ateniese implementation *(ms)*
Encryption (1st)	10.6935	9.3
Decryption (1st)	0.238691	26.5
Encryption (2nd)	2.52118	-
Decryption (2nd)	1.4941	4.1
Re-encryption	1.38101	26.7
Keygen	7.65707	-

Table 15: Efficiency of the implementation

The differences in timings are very slight. Our implementation is slightly faster in most cases, which may be due to the fact that the second option from the paper requires less computation than the third option.

Limitations and Future Work

There were a few limitations to our recommender system design that could be improved upon.

Firstly, the similarity computations run in the order of hours on relatively large inputs and will take even longer for very large inputs, such as DNA-sequences of length 1000 and more. We did not test on these inputs, since for length 50 the protocol for the edit distance already took more than

an hour to execute. The similarity computations can be computed by two servers during an offline phase, but it would still be a great improvement to the recommender system if these computations could be more efficient. To achieve this, different subroutines for minimum finding and substitution cost might need to be developed that require less homomorphic multiplications, so that the key parameters chosen can be smaller. This will speed up the somewhat homomorphic encryption scheme.

Secondly, a problem that we ran into during the implementation of our protocols was that the semi-honest minimum finding protocol could not be implemented due to an integer wrap around in the message space that we used for our encryption system. As a result, the malicious minimum finding protocol was used instead, but this may have had a negative impact on the efficiency of the semi-honest edit distance and Smith-Waterman distance protocol timings.

A minor limitation of our current system design is that both servers are required to be semi-honest and non-colluding. Although this is a realistic scenario, when for example the main server is governed by a health care company and the second server is a security provider, it would be even better if the recommender system could be designed in such a way that one or two malicious servers would not impact the privacy of the system. This would lead to a stronger security model.

As a recommendation for future research, it would be interesting to study whether verifiability of recommendation results can be introduced to the protocol in the malicious or the semi-honest user model. In our current design, the user who requests a recommendation has no way of knowing that he receives a correct result that is computed over all of his friends' ratings. If the current protocols were to be extended, the fact could be used in the malicious user model that the two servers are non-colluding and could therefore be required to check each other's results.

References

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.

[AKD03] Mikhail J. Atallah, Florian Kerschbaum, and Wenliang Du. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society, WPES 2003, Washington, DC, USA, October 30, 2003*, pages 39–44, 2003.

[ARHR13] Erman Ayday, Jean Louis Raisaro, Jean-Pierre Hubaux, and Jacques Rougemont. Protecting and evaluating genomic privacy in medical tests and personalized medicine. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 95–106, 2013.

[BAFM12] Marina Blanton, Mikhail J. Atallah, Keith B. Frikken, and Qutaibah M. Malluhi. Secure and efficient outsourcing of sequence comparisons. In *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, pages 505–522, 2012.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 127–144, 1998.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, pages 325–343, 2009.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.

[BKKT08] Fons Bruekers, Stefan Katzenbeisser, Klaus Kursawe, and Pim Tuyls. Privacy-preserving matching of DNA profiles. *IACR Cryptology ePrint Archive*, 2008:203, 2008.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, pages 192–206, 2008.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 505–524, 2011.

[Can02] John F. Canny. Collaborative filtering with privacy. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*, pages 45–57, 2002.

[CEJ+07] Seung Geol Choi, Ariel Elbaz, Ari Juels, Tal Malkin, and Moti Yung. Two-party computing with encrypted data. In *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, pages 298–314, 2007.

[CK08] Octavian Catrina and Florian Kerschbaum. Fostering the uptake of secure multiparty computation in e-commerce. In *Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, 2008, Technical University of Catalonia, Barcelona , Spain*, pages 693–700, 2008.

[CWYD10] Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient unidirectional proxy re-encryption. In *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch,*

South Africa, May 3-6, 2010. Proceedings, pages 316–332, 2010.

[EFG+09] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies, 9th International Symposium, PETS 2009, Seattle, WA, USA, August 5-7, 2009. Proceedings*, pages 235–253, 2009.

[EM03] William E. Evans and Howard L. McLeod. Pharmacogenomics – drug disposition, drug targets, and side effects. *N Engl J Med*, 348(6):538–549, 2003.

[ER04] William E. Evans and Mary V. Relling. Moving towards individualized medicine with pharmacogenomics. *Nature*, 429:464–468, 2004.

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

[GE07] Georg Groh and Christian Ehmig. Recommendations in taste related domains: collaborative filtering vs. social filtering. In *Proceedings of the 2007 International ACM SIGGROUP Conference on Supporting Group Work, GROUP 2007, Sanibel Island, Florida, USA, November 4-7, 2007*, pages 127–136, 2007.

[Gol05] Oded Goldreich. Foundations of cryptography - A primer. *Foundations and Trends in Theoretical Computer Science*, 1(1), 2005.

[GV08] Miriam Gordillo and Hugo Vega. The molecular mechanism underlying roberts syndrome involves loss of esco2 acetyltransferase activity. *Human Molecular Genetics*, 17(14):21722180, 2008.

[GZC+09] 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 2009 ACM Conference on Recommender*

Systems, RecSys 2009, New York, NY, USA, October 23-25, 2009, pages 53–60, 2009.

[HBSC13] T. Ryan Hoens, Marina Blanton, Aaron Steele, and Nitesh V. Chawla. Reliable medical recommendation systems with patient privacy. *ACM TIST*, 4(4):67, 2013.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. Cryptology ePrint Archive, Report 2014/106, 2014. `http://eprint.iacr.org/`.

[JKS08] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 216–230, 2008.

[JPH13] Arjan Jeckmans, Andreas Peter, and Pieter H. Hartel. Efficient privacy-enhanced familiarity-based recommender system. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 400–417, 2013.

[Ler07] Kristina Lerman. Social networks and social information filtering on digg. In *Proceedings of the First International Conference on Weblogs and Social Media, ICWSM 2007, Boulder, Colorado, USA, March 26-28, 2007*, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013.

[LV11] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Transactions on Information Theory*, 57(3):1786–1802, 2011.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124, 2011.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 448–457, 2001.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.

[PTK13] 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.

[RS10] S. Rane and W. Sun. Privacy preserving string comparisons based on levenshtein distance. 2010.

[SS01] Rashmi R. Sinha and Kirsten Swearingen. Comparing recommendations made by online systems and friends. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 24–43, 2010.

[VdHCM15] Thijs Veugen, Robbert de Haan, Ronald Cramer, and Frank Muller. A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. *IEEE Transactions on Information Forensics and Security*, 10(3):445–457, 2015.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.