Efficient Privacy-Preserving Matrix Factorization via Fully Homomorphic Encryption[*]

Sungwook Kim[†1], Jinsu Kim[‡1], Dongyoung Koo[§2], Yuna Kim[¶1], Hyunsoo Yoon[‖2], and Junbum Shin[**1]

[1]Software R&D Center, Samsung Electronics, South Korea
[2]School of Computing, Korea Advanced Institute of Science and Technology (KAIST), South Korea

Abstract

Recommendation systems become popular in our daily life. It is well known that the more the release of users' personal data, the better the quality of recommendation. However, such services raise serious privacy concerns for users. In this paper, focusing on matrix factorization-based recommendation systems, we propose the first privacy-preserving matrix factorization using fully homomorphic encryption. On inputs of encrypted users' ratings, our protocol performs matrix factorization over the encrypted data and returns encrypted outputs so that the recommendation system knows nothing on rating values and resulting user/item profiles. It provides a way to obfuscate the number and list of items a user rated without harming the accuracy of recommendation, and additionally protects recommender's tuning parameters for business benefit and allows the recommender to optimize the parameters for quality of service. To overcome performance degradation caused by the use of fully homomorphic encryption, we introduce a novel data structure to perform computations over encrypted vectors, which are essential operations for matrix factorization, through secure 2-party computation in part. With the data structure, the proposed protocol requires dozens of times less computation cost over those of previous works. Our experiments on a personal computer with 3.4 GHz 6-cores 64 GB RAM show that the proposed protocol runs in 1.5 minutes per iteration. It is more efficient than Nikolaenko et al.'s work proposed in CCS 2013, in which it took about 170 minutes on two servers with 1.9 GHz 16-cores 128 GB RAM.

1 INTRODUCTION

A lot of recommendation services, including mobile ads and movie recommendations, have been used in our daily life, and a collaborative filtering (CF) is one of commonly used algorithms for such services [2, 43]. The principle of CF algorithm is to infer user's preference from gathered other users' history and to give a recommendation for new items similar users prefer.

To address privacy issues caused by CF [33, 38, 48], there have been numerous works to propose privacy-preserving CF algorithms [9, 28, 31, 35, 39]. It has been discovered that anonymization of

[*]An extended abstract of this paper appears in ASIACCS 2016.
[†]sw14.kim@samsung.com
[‡]jinsu86.kim@samsung.com
[§]dykoo@nslab.kaist.ac.kr
[¶]yuna1.kim@samsung.com
[‖]hyoon@kaist.ac.kr
[**]junbum.shin@samsung.com

users' data is not sufficient to protect the privacy [9, 33, 44]. Differential privacy [28, 31, 39] has been proposed for minimizing the chances of identifying any single user's data. However it takes trade-off between privacy and accuracy of estimated prediction and does not offer confidentiality of user's rating values by definition.

Recently, a remarkable approach based on cryptographic techniques, that is garbled circuits, was proposed by Nikolaenko et al. [35]. They targeted a *matrix factorization*, mainly used among several CF algorithms. It addresses how to protect both user's rating values and which items are rated by the user. However it seriously requires improvement of efficiency and utility for practical use. In addition, it only supports a fixed number of iteration of matrix factorization, which does not guarantee the quality of matrix factorization.

In this paper, we focus our interest on how to efficiently perform matrix factorization to compute user and item profiles from the ratings given by users in a privacy-preserving way with full functionality. We develop a fully homomorphic encryption-based matrix factorization protocol among users, a recommendation system (*RecSys*), and crypto-service pro-vider (*CSP*).

We revisit the following three security requirements for secure matrix factorization for practical use: (i) *Privacy*, (ii) *Accuracy*, and (iii) *Completeness*. Privacy consists of considerations for both users and the RecSys. On a user's side, it typically comprises a user's rating values, which items a user has rated, the number of items a user has rated, and user/item profiles. On RecSys's side, tuning parameters for matrix factorization can be considered as sensitive information, because the efficacy of the outputs depends on the parameters that the RecSys chooses [45]. Accuracy points out that a privacy protection mechanism should not weaken the accuracy of the recommendation as far as possible. Completeness means a mechanism needs to support a full learning process for recommendation, repeatedly training and validation until it converges. During the process, the tuning parameters can be optimized for faster convergence [12, 51].

To the best of our knowledge there is no known mechanism for privacy-preserving matrix factorization which meets all of previously mentioned requirements. In this work we initiate a study in the direction of efficient privacy-preserving matrix factorization to take all requirements into account.

1.1 Contributions

Main contributions and features of the proposed protocol are as follows:

New construction from fully homomorphic encryption. We first construct a privacy-preserving matrix factorization via *fully homomorphic encryption* (FHE). Our construction performs matrix factorization to compute user/item profiles using approximation techniques (e.g. gradient descent) as described in Section 3.3. Our protocol runs a secure two-party computation protocol between the RecSys and CSP, when it is too burdensome to perform the above operations by FHE solely. The efficient use of FHE is quite non-trivial; authors in [35] introduced garbled circuit instead of FHE, pointing out that FHE schemes for simpler algebraic computations are not as efficient as garbled circuit approaches.

The main obstacles for gradient descent computation using FHE are (i) the inner product operation of vectors according to the index, which is induced from the user-item matrix and (ii) the fixed point operation due to vectors over real numbers. For inner product operation, a naive use of Single Instruction Multiple Data (SIMD) operation supported by FHE [41] does not help for reducing computation cost since it gets to perform one homomorphic multiplication for one inner product operation only, not multiple inner products, after all (refer to Section 4.1 for details). For the fixed point arithmetic, it requires the division-like operation over encrypted data and as far as we know there is no efficient method to evaluate it homomorphically. Thus naive approach using FHE does not work efficiently in the sense that the computation and communication costs linearly depend on the number of ratings.

We resolve the above limitations of FHE in gradient descent by introducing a novel data structure. It enables to enjoy the full use of slots supported by the SIMD operation in FHE ciphertext while allowing one homomorphic operation for multiple operations over vectors during gradient descent. As a result, both the running time and the communication overhead of the proposed protocol are linear in M/L, where M and L are the number of ratings and slots which FHE supports, respectively. Since L is typically a number in thousands in our implementation, the data structure reduces computation cost by dozens of times. For the fixed point arithmetic on encrypted data, we use secure 2-party computation between the RecSys and CSP. That is, the CSP performs fixed point arithmetic over masked plaintexts without learning any information about actual plaintexts and then returns the encrypted results to the RecSys. We believe that our technique can be generally applied to various privacy-preserving data analytics protocols using gradient descent.

Additionally, we provide an optimization for the ciphertext size of FHE itself. In most FHE constructions [14, 16, 22], the size of ciphertext grows quadratically in the size of message space, and it causes large computational and communication overhead. To obtain small ciphertexts for the case of large message space, we represent the message in several smaller spaces using Chinese remainder theorem. This technique yields linear growth in the ciphertext size.

Privacy and security. Our protocol guarantees the secrecy of rating values from users and all intermediate computation results during joint computation between the RecSys and CSP. The final results, user/item profiles, are also returned under FHE encryption so that the RecSys and CSP cannot learn both user and item profiles in the clear. On RecSys's side, the optimized parameters for quality recommendation can be kept secret, because the RecSys itself can set tuning parameters for matrix factorization without disclosure from the use of FHE.

Finally the extension of the proposed protocol delivers the ability of launching various techniques to obfuscate the number of items and the list of items a user rated. Among the obfuscation techniques, we describe how to apply an injection of fake ratings considered in differential privacy approach [13, 39], and the description can be helpful for applying another techniques easily.

Accuracy and completeness. As mentioned previously, the use of differential privacy or the injection of fake ratings suffer from distortion of results of data analysis. We ensure the accuracy of the recommendation by introducing the encrypted indicator of rated items. The indicator sets 0 or 1 by users, depending on whether the item is rated or not. Through the protocol, the indicator vector works correctly without revealing the value at all to both the RecSys and CSP. If the rating is real, it gets to involve to the computation for analysis. Otherwise its influence on the computation is removed.

Typically, matrix factorization iteratively computes user/ item profiles using gradient descent. The iteration ends up with convergence of the profiles. In our protocol, the RecSys can check the convergence by verifying a stopping criteria without knowing the raw values of users' ratings or user/item profiles. Thus it provides the ability to complete a learning process by optimizing the gradient descent parameters.

Efficiency. We implement our protocol and evaluate the performance. We use the MovieLens dataset for our experiment. Our matrix factorization protocol without obfuscation of rating information takes about 1.5 minutes with 580 MB communication between the RecSys and CSP per iteration for 14K ratings dataset. This improves execution time and communication overhead drastically over the work in [35], which takes 2.9 hours with 128 GB. The extended protocol takes 1.9 minutes with 623 MB for 10% fake ratings, and 2.5 minutes with 910 MB for 50%.

We further estimate performance with taking into account parallelism and compare to recent progress on parallelization of Nikolaenko et. al's protocol using GraphSC [34] and optimization technique of circuits for the garbled circuit protocol called TinyGarble [42]. The estimate shows that parallelized version of our construction is expected to be about 33 times faster than that of Nikolaenko et al.'s protocol together with TinyGarble.

The paper is organized as follows. Related work and some preliminaries will be summarized in Section 2 and 3. We present data structure for gradient descent in Section 4 and then propose our protocol in Section 5. We analyze security and computation/communication costs of the proposed protocol in Section 6. Several optimization efforts with implementation will be described in Section 7. We evaluate the performance in Section 8 and give a conclusion in Section 9.

2 RELATED WORK

Our work is to present an efficient and practical matrix factorization for collaborative filtering based-recommendation while preserving user privacy about item ratings and the analyzed preference throughout the two-party protocol between the RecSys and CSP.

Privacy with cryptography. A lot of works have been done to solve the privacy issues for various machine learning algorithms, and some of their basic operations can be protected using cryptography efficiently: classification [4], regression [3, 36].

The first work for privacy-preserving matrix factorization is proposed by Nikolaenko et. al. [35], which used a mixed protocol of additive homomorphic encryption [11, 37] and garbled circuits [50] for secure two-party computation, and applied oblivious sorting network for supporting sparsity of rating data. They tried to improve efficiency of the implementation using multi-threaded FastGC [29], and parallelization of the sorting network. However, the excessive computation and communication cause impracticality, for example, it takes 2.9 hours per iteration for 14K ratings. The performance comparisons between [35] and ours will be given in Section 8.

Nayak et al. [34] proposed more practical approach, named GraphSC, by applying both a GraphLab [13] for a parallelism and oblivious approach [25] for the security. They have shown that the previous work proposed by Nikolaenko et al. [35] can be extended using GraphSC and showed that it can make overall parts of secure matrix factorization parallel on large dataset of 1M ratings, which is a large improvement from the original one [35]. Such a parallelism for our work is analyzed in Section 8.3.

Besides matrix factorization, crypto-based privacy protection techniques have been applied to other collaborative filtering methods [18, 47]. Erkin et al. [18] applied homomorphic encryption to user-based collaborative filtering using cosine similarity between users' ratings. For reducing computational and communication overhead, multiple numerical values are packed in a single ciphertext. It differs from our data packing in that their packing can be applied only to vector scalar multiplications, not batch inner products needed for matrix factorization.

Veugen et al. [47] proposed a generic framework for secure multi-party computations based on secret sharing, especially focusing on the recommendation systems. It provides security in the malicious model, where one of servers may not follow the protocol specifications and may control a couple of users to deduce more personal data. It also focuses on efficient online recommendation when a user asks for recommendation with updated ratings. Differently from their work, we focus on efficient computation of matrices for user and item profiles without leakage of private data before recommendation.

Privacy with non-cryptography. Differential privacy is mainly considered for non crypto-based privacy protection [17, 32], including the recommendation privacy [28, 31, 39, 49]. It prevents attacker from the inference of users' privacy by adding fake users' ratings as long as it does not affect significantly to the distribution of the system's output. However, differential privacy approach does not guarantee confidentiality of users' data to the RecSys, and reduce accuracy of recommendation as much as fake ratings added. In our work, users can add fake ratings along with indicator vectors, 1 for real rating and 0 for fake to obfuscate which items are rated. This does not affect the accuracy of recommendation, since the fake ratings are removed when matrix factorization runs.

Fully homomorphic encryption. In 2009, Gentry introduced the first FHE scheme based on ideal lattice that allows arbitrary many additions and multiplications on encrypted bit [19]. Since Gentry's breakthrough result, many improvements have been made, introducing new variants [7,

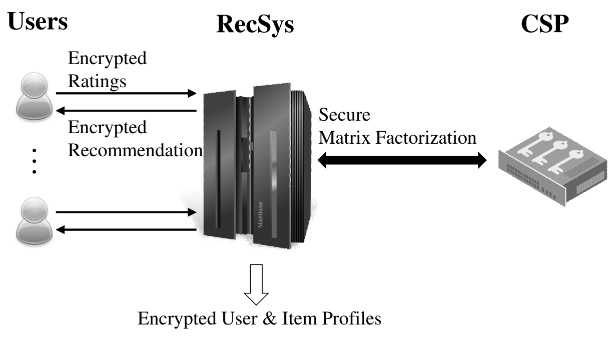

Figure 1: Three Parties in Our Protocol Design.

8, 46], improving efficiency [10, 14, 15, 16, 21], and providing new features [5, 6, 23]. Recently, Halevi and Sahai provide the state-of-the-art FHE library (HElib) based on ring learning with errors problem (RLWE) [27]. The library allows SIMD operations on encrypted data [20, 41] and includes a new efficient key switching and bootstrapping method [22].

Even though HElib provides efficient implementation, the main operations in matrix factorization such as inner product and floating-point arithmetics are still extremely time and space consuming job when FHE is solely used. To resolve this, we propose a novel data structure and design a new matrix factorization protocol by merging the FHE with secure 2-party computation.

3 PRELIMINARIES

3.1 Setting

The proposed protocol and system consists of three ends: Users, the Recommendation System (RecSys), and Crypt-Service Provider (CSP), as shown in Figure 1. User's rating information is uploaded to the RecSys, which provides a recommendation service based on users' uploaded information. In our model, we will use a matrix factorization algorithm to infer the value of unrated item: the details will be explained in Section 3.3.

Users' rating information will be encrypted using a public key homomorphic encryption, so it is very important to protect such a key. Therefore, we will use the CSP, which is secure server and the key will be stored and used only inside the CSP to minimize the risk of key exposure.

Regarding the security, we assume the standard adversary model used in [35]: Following the definition of honest-but-curious adversary model [24], either the RecSys or CSP can be compromised by an adversary, but they will follow the prescribed protocol correctly. In addition, the adversary might look at the data exchanged with the other parties, and try to uncover users' data including the ratings and to infer recommendation and matrix factorization results including both user and item profiles. However, the RecSys does not misbehave with users' ratings such as altering, dropping, or copying them to control the output. The CSP can also always keep the secrets. We also assume the RecSys and CSP do not collude, and such an assumption makes sense in business environment because they value the reputation. In addition, the CSP usually could be a governmental organization guarding the privacy protection of users [47].

3.2 Tools and Notations

In the protocol we use somewhat (possibly fully) homomorphic encryption and additively homomorphic encryption schemes, denoted by HE and AHE, respectively. With public keys hpk for HE, we call $\mathsf{HE}(\mathsf{hpk}, \mathsf{msg})$ a HE-ciphertext or a HE-encryption of a plaintext msg. Given two HE-ciphertexts

$\mathsf{HE}(\mathsf{hpk}, \mathsf{msg}_1)$ and $\mathsf{HE}(\mathsf{hpk}, \mathsf{msg}_2)$, we call $C_1 + (\times) C_2$ *homomorphic* addition (multiplication) which is a HE-encryption of $\mathsf{msg}_1 + (\times) \mathsf{msg}_2$. When it is clear in the context, we just say addition or multiplication and omit the public key part in encryption algorithm, i.e., $\mathsf{HE}(\mathsf{msg})$. Similar notations are used for AHE with a public key apk. .

In [41], Smart et al. suggested homomorphic encryption supporting Single Instruction Multiple Data (SIMD) operation, which enables to pack many plaintexts into each ciphertext. We say HE has #slot L if L plaintexts can be packed.

A vector is written in bold, i.e., $\mathbf{a}$. Given two vectors $\mathbf{a} = (a_1, \ldots, a_k)$ and $\mathbf{b} = (b_1, \ldots, b_k)$, we define

$$\mathbf{a} \times_{\mathsf{c}} \mathbf{b} = (a_1 b_1, \ldots, a_k b_k)$$

and $\mathrm{sum}(\mathbf{a}) = \sum_{i=1}^{k} a_i$. We then have $\langle \mathbf{a}, \mathbf{b} \rangle = \mathrm{sum}(\mathbf{a} \times_{\mathsf{c}} \mathbf{b})$.

For a vector $\mathbf{a}$ we denote $\mathsf{HE}(\mathbf{a})$ by a HE-encryption of $\mathbf{a}$, where each entry of $\mathbf{a}$ is encrypted to each slot of a ciphertext. Then $\mathsf{HE}(\mathbf{a})$ consists of $k/\#\mathrm{slot}$ HE-ciphertexts. Note that $\mathsf{HE}(\mathbf{a}) + \mathsf{HE}(\mathbf{b})$ is a HE-encryption of $\mathbf{a} + \mathbf{b}$ and $\mathsf{HE}(\mathbf{a}) \times \mathsf{HE}(\mathbf{b})$ is a HE-encryption of $\mathbf{a} \times_{\mathsf{c}} \mathbf{b}$.

A fixed point representation over a real number f is denoted by $\overline{f}$. For a vector $\mathbf{a} = (a_1, \ldots, a_k)$, we denote $\overline{\mathbf{a}}$ by component-wise fixed point representation. In this work, we use the binary fixed point representation and denote α by the bit length of scaling factor.

3.3 Matrix Factorization

We briefly introduce a matrix factorization method to be used in our system, which is also considered in [35]. Suppose n users rate a subset of m possible items. We use the following notations:

- $[n] = \{1, \ldots, n\}$, $[m] = \{1, \ldots, m\}$: the sets of users and items, respectively,
- $r_{ij} \in \mathbb{R}$: the user i's rating for the item j,
- $\mathcal{M} \subset [n] \times [m]$: a set of the user/item pairs for which a rating has been generated,
- M: the total number of ratings, i.e., $M = |\mathcal{M}|$,
- $\mathbf{u}_i, \mathbf{v}_j \in \mathbb{R}^d$: profiles for the user i and item j, respectively, where d is the dimension of profiles
- $U \in \mathbb{R}^{n \times d}$: the user-profile matrix (user profiles) whose i-th row is $\mathbf{u}_i$,
- $V \in \mathbb{R}^{m \times d}$: the item-profile matrix (item profiles) whose j-th row is $\mathbf{v}_j$

Given the ratings $\{r_{ij} : (i,j) \in \mathcal{M}\}$, matrix factorization computes the user profiles U and item profiles V. The resulting profiles are used to predict the user i's rating for item j, that is, $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$, for $(i,j) \notin \mathcal{M}$. This can be done by fitting $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$ on the existing ratings r_{ij} for $(i,j) \in \mathcal{M}$, that is, solving the regularized least squares minimization:

$$\min_{U,V} \frac{1}{M} \sum_{(i,j) \in \mathcal{M}} (r_{ij} - \langle \mathbf{u}_i, \mathbf{v}_j \rangle)^2 + \lambda \sum_{i \in [n]} \|\mathbf{u}_i\|_2^2 + \mu \sum_{j \in [m]} \|\mathbf{v}_j\|_2^2$$

for some $\lambda, \mu > 0$.

Among methods to solve the above, we use gradient descent [30], a popular method in practice. It iteratively adopts the profiles U and V through the following adaptation rule:

$$\begin{aligned} \mathbf{u}_i(t) &= \mathbf{u}_i(t-1) - \gamma \nabla_{\mathbf{u}_i} F(U(t-1), V(t-1)), \\ \mathbf{v}_j(t) &= \mathbf{v}_j(t-1) - \gamma \nabla_{\mathbf{v}_j} F(U(t-1), V(t-1)), \end{aligned} \tag{1}$$

where $\gamma > 0$ a small gain factor and

$$\begin{aligned}\nabla_{\mathbf{u}_i} F(U,V) &= \nabla_{\mathbf{u}_i} \\ &= \sum_{j:(i,j)\in\mathcal{M}} \mathbf{v}_j(\langle \mathbf{u}_i, \mathbf{v}_j\rangle - r_{ij}) + \lambda \mathbf{u}_i, \\ \nabla_{\mathbf{v}_j} F(U,V) &= \nabla_{\mathbf{v}_j} \\ &= \sum_{i:(i,j)\in\mathcal{M}} \mathbf{u}_i(\langle \mathbf{u}_i, \mathbf{v}_j\rangle - r_{ij}) + \mu \mathbf{v}_j,\end{aligned} \tag{2}$$

where $U(0)$ and $V(0)$ consist of uniformly random norm 1 rows.

The number of iterations depends on stopping criteria. One of typical criteria is to set a (small) threshold for gradient norms, which is applied to the proposed protocol in Section 4. The protocol stops when the number of iterations exceeds some pre-defined number or when both

$$\sum_{i:(i,j)\in\mathcal{M}} \| \nabla_{\mathbf{u}_i} F(U,V) \|_2^2 \text{ and } \sum_{j:(i,j)\in\mathcal{M}} \| \nabla_{\mathbf{v}_j} F(U,V) \|_2^2$$

become smaller than some threshold values, respectively.

Note that the values of d, γ, λ, μ, and stopping criteria can be a know-how for a recommendation service provider [45].

4 DESIGN COMPONENTS

4.1 Naive Approach and Challenges

Fully homomorphic encryption. We can consider a naive way to perform gradient descent using fully homomorphic encryption. The simplest way is to encrypt each component in vectors $\mathbf{u}_i, \mathbf{v}_j$ independently. In this case, we need to deal dM HE-ciphertexts to compute gradient descent, which is computational burden.

Instead, the SIMD operation proposed in [41] can be used to improve the inner product algorithm, that is, if we pack elements in two vectors of length d into two ciphertexts, respectively, the computation overhead is reduced by a factor of $(2\log d/d)$ [1]. In our case, on the other hand, encrypting one vector in one ciphertext is quiet inefficient, since the dimension of $\mathbf{u}_i, \mathbf{v}_j$ is quite small compared to the number of slots underlying HE and the number of ciphertexts grows in M.

To make use of full message slots, one may consider packing several user and item profile vectors into a single ciphertext, respectively. Let $c_{\mathbf{u}} = \mathsf{HE}(\mathbf{u}), c_{\mathbf{v}} = \mathsf{HE}(\mathbf{v})$ for $\mathbf{u} = (\mathbf{u}_1 || \cdots || \mathbf{u}_n)$ and $\mathbf{v} = (\mathbf{v}_1 || \cdots || \mathbf{v}_m)$ where $||$ means vector concatenation. To compute $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$ with $i < j$, we operate $d \cdot (j - i)$ left-shift of plaintext slots on $c_{\mathbf{v}}$, say $c'_{\mathbf{v}}$, and then evaluate homomorphically multiplication of $c_{\mathbf{v}}$ and $c'_{\mathbf{v}}$. Even though we have $c_{\mathbf{u}} \times c'_{\mathbf{v}}$ whose slots from $d \cdot (i-1)$ to $d \cdot i$ are $\mathbf{u}_i \times_c \mathbf{v}_j$, the rest of slots is hard to be exploited in computing gradient descent. This means that we need to evaluate one homomorphic multiplication for only one inner product, which does not enjoy the benefit of SIMD at all.

Another concern in adapting HE is to perform fixed point arithmetic in encrypted form. Since there is no efficient homomorphic encryption supports rational number arithmetic, we need to transform data into integer form. In this case, it essentially needs additional post-process like integer division to maintain the same scaling factor. Under HE, this post-process seems heavy computations as in [20].

Stopping criteria. In general, the matrix factorization for recommendation runs several iterations while tuning some parameters such as γ, λ and μ until one gets meaningful user/item profiles.

[1]The inner product on packed two vectors of length d can be carried out through $\log d$ shift in plaintext slots and $\log d$ multiplications instead of d.

Furthermore, these tuning parameters might be know-how of the recommendation service provider. Therefore, it is important to construct recommendation system that can adjust tuning parameters during matrix factorization without leakage of the private parameters.

4.2 Data Structure for Gradient Descent

To overcome challenges in the previous section, we come up with an elaborate data structure to exploit slots fully so that the protocol requires only M/L HE-ciphertexts, instead of M, for #slot L. The idea behind our approach is to set a long vector whose segments are profile vectors preserving index information induced from the user-item matrix and perform joint computation with the CSP. If inner products and fixed point operations are required, the RecSys calls the CSP as encryption/decryption oracles and computes a part of operations with the CSP.

We define a total order on a set $\mathcal{M}$ as follows: for (i_1, j_1) and $(i_2, j_2) \in \mathcal{M}$,

$$\begin{cases} (i_1, j_1) > (i_2, j_2) & \text{if } i_1 > i_2 \text{ or } j_1 > j_2 \text{ when } i_1 = i_2, \\ (i_1, j_1) = (i_2, j_2) & \text{if } i_1 = i_2 \text{ and } j_1 = j_2. \end{cases}$$

From now on we assume elements in $\mathcal{M}$ are sorted in ascending order. Then for a set $\mathcal{M}$, we define 4 vectors as follows:

- $\mathbf{U} = \|_{(i,j)\in\mathcal{M}}\ \mathbf{u}_i$ and $\mathbf{V} = \|_{(i,j)\in\mathcal{M}}\ \mathbf{v}_j$, where $\|$ means vector concatenation. Note that, depending on its order, every d-dimensional vector $\mathbf{u}_i$ ($\mathbf{v}_j$) in $\mathbf{U}$ ($\mathbf{V}$) corresponds to the unique pair (i, j) in $\mathcal{M}$.
- $\hat{\mathbf{U}} = \|_{(i,j)\in\mathcal{M}}\ \hat{\mathbf{u}}_i$, where $\hat{\mathbf{u}}_i = \mathbf{u}_i$ if i of the corresponding index (i, j) appears first and the d-dimensional zero vector otherwise. We define $\hat{\mathbf{V}}$ similarly, according to the index j from $(i, j) \in \mathcal{M}$.

Example 1. Let $\mathcal{M} = \{(2, 3), (2, 4), (5, 3)\}$ (sorted in ascending order), $\mathbf{u}_i = (u_{i,1}, \ldots, u_{i,d})$, and $\mathbf{v}_j = (v_{i,1}, \ldots, v_{i,d})$. Then

$$\begin{aligned} \mathbf{U} &= \mathbf{u}_2 \parallel \mathbf{u}_2 \parallel \mathbf{u}_5 \\ &= (u_{2,1}, \ldots, u_{2,d}, u_{2,1}, \ldots, u_{2,d}, u_{5,1}, \ldots, u_{5,d}), \\ \mathbf{V} &= \mathbf{v}_3 \parallel \mathbf{v}_4 \parallel \mathbf{v}_3, \\ \hat{\mathbf{U}} &= \mathbf{u}_2 \parallel \vec{\mathbf{0}} \parallel \mathbf{u}_5,\ \hat{\mathbf{V}} = \mathbf{v}_3 \parallel \mathbf{v}_4 \parallel \vec{\mathbf{0}}. \end{aligned}$$

Suppose $\mathcal{M}_I = \{i_1, \ldots, i_{\ell_u}\}$ and $\mathcal{M}_J = \{j_1, \ldots, j_{\ell_v}\}$ are the sets of distinct user index i's and item index j's of $(i, j) \in \mathcal{M}$ in ascending order, respectively. Consider a dM-dimensional vector $\mathbf{A} = \mathbf{a}_1 \parallel \cdots \parallel \mathbf{a}_M$, where $\mathbf{a}_k$ is a d-dimensional vector. Then we can match $\mathbf{a}_k$ to $(i, j) \in \mathcal{M}$ according to order, which is a 1-1 correspondence. With this correspondence, we define additional 4 operations on $\mathbf{A}$ as follows:

- $\mathsf{agg}_u, \mathsf{agg}_v$: We define aggregations of the user and item profiles in $\mathbf{A}$ according to $\mathcal{M}$ in agg_u and agg_v:
$$\begin{aligned} \mathsf{agg}_u(\mathbf{A}, \mathcal{M}) &= \sum_{j:(i_1,j)\in\mathcal{M}} \mathbf{a}_j \parallel \cdots \parallel \sum_{j:(i_{\ell_u},j)\in\mathcal{M}} \mathbf{a}_j, \\ \mathsf{agg}_v(\mathbf{A}, \mathcal{M}) &= \sum_{i:(i,j_1)\in\mathcal{M}} \mathbf{a}_i \parallel \cdots \parallel \sum_{i:(i,j_{\ell_v})\in\mathcal{M}} \mathbf{a}_i. \end{aligned}$$
- $\mathsf{rec}_u, \mathsf{rec}_v$: We define reconstitutions of the user/item profiles by
$$\begin{aligned} \mathsf{rec}_u(\mathsf{agg}_u, \mathcal{M}) &= \|_{(i,j)\in\mathcal{M}}\ \mathbf{A}'_i, \\ \mathsf{rec}_v(\mathsf{agg}_v, \mathcal{M}) &= \|_{(i,j)\in\mathcal{M}}\ \mathbf{A}''_j \end{aligned}$$

where

$$\mathsf{agg}_u(\mathbf{A}, \mathcal{M}) = \mathbf{A}'_1 \parallel \cdots \parallel \mathbf{A}'_{\ell_u},$$
$$\mathsf{agg}_v(\mathbf{A}, \mathcal{M}) = \mathbf{A}''_1 \parallel \cdots \parallel \mathbf{A}''_{\ell_v}.$$

When it is clear in the context we omit $\mathcal{M}$ of agg and rec.

Example 2. Suppose $\mathcal{M} = \{(2, 3), (2, 4), (5, 3)\}$. Given $\mathbf{A} = \mathbf{a}_1 \parallel \mathbf{a}_2 \parallel \mathbf{a}_3$,

$$\begin{aligned}
\mathsf{agg}_u(\mathbf{A}, \mathcal{M}) &= \mathbf{a}_1 + \mathbf{a}_2 \parallel \mathbf{a}_3,\\
\mathsf{rec}_u(\mathsf{agg}_u, \mathcal{M}) &= \mathbf{a}_1 + \mathbf{a}_2 \parallel \mathbf{a}_1 + \mathbf{a}_2 \parallel \mathbf{a}_3,\\
\mathsf{agg}_v(\mathbf{A}, \mathcal{M}) &= \mathbf{a}_1 + \mathbf{a}_3 \parallel \mathbf{a}_2,\\
\mathsf{rec}_v(\mathsf{agg}_v, \mathcal{M}) &= \mathbf{a}_1 + \mathbf{a}_3 \parallel \mathbf{a}_2 \parallel \mathbf{a}_1 + \mathbf{a}_3.
\end{aligned}$$

4.3 Fixed Point Arithmetic on Encrypted Data

We use fixed point representation (FPR) of real number instead of floating point representation at the cost of small errors as in [35]. We use the integral version of FPR and denote the FPR with fixed α-bit precision by $\overline{a}$ for a real number $a \in \mathbb{R}$. Here, we just say the FPR for integral version of that. The operations on FPR are as follows:

- Addition/Subtraction: $\overline{\overline{a} \pm \overline{b}} = \overline{a \pm b}$
- Multiplication: $\overline{\overline{a} \cdot \overline{b}} = \left\lfloor \overline{a} \cdot \overline{b} / 2^{\alpha} \right\rfloor$

where $\lfloor \cdot \rfloor$ is a round down function. In our construction, we encrypt the FPR of user/item profiles and perform homomorphic additions/subtractions and multiplications on them. When encrypting this type of data, the main obstacle is multiplication because of integer division by 2^{α} and rounding, which can be considered as the α-bit right shift. In [21], the authors proposed homomorphic computation of right shift on integers using bit-extraction. On the other hand, α multiplicative depth of underlying FHE is consumed and it needs α^2 homomorphic multiplications, which results in huge computational overhead.

We design the RecSys to work jointly with the CSP to enhance the efficiency when performing multiplications on FPR values. We need one encryption and one decryption with one interaction in handling fixed point multiplication as follows: (i) the RecSys sends masked encryption of multiplication result to the CSP to protect the privacy of data. (ii) the CSP decrypts the masked ciphertext and computes right shift on the plaintext. (iii) the CSP encrypts resulting data and send it to the RecSys. (iv) the RecSys removes the mask and obtain the result. If the right shift is applied to masked integer, the overflow in the LSB of integer may occur. However it does not affect to the quality of matrix factorization significantly because of its small magnitude. The detail analysis and experimental results will be presented in Section 8.1.

5 Our Protocol

We present our privacy-preserving matrix factorization protocol. Our protocol consists of the setup, rating-upload, and matrix factorization phases. We additionally describe the recommendation phase after finishing matrix factorization in Appendix B.

5.1 Setup

In the setup phase, the CSP generates its HE-public/secret key pairs and AHE-public/secret key pairs. For any pubic key encryption PE, the users also generate their PE-public/secret key pairs.

The RecSys specifies the following public parameters: (i) the dimension of profiles d, (ii) the number of items m, the number of users n, (iii) the number of bits used to represent the integer

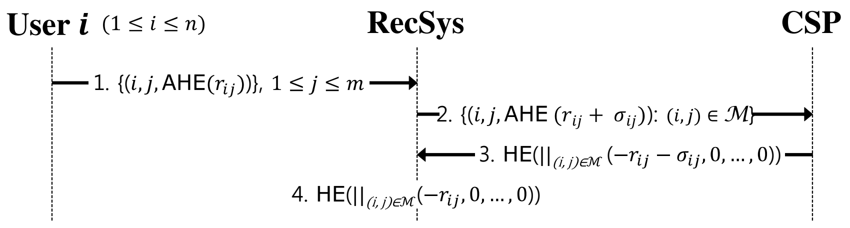
User i $(1 \le i \le n)$
RecSys
CSP
1. $\{(i, j, \mathsf{AHE}(r_{ij}))\}$, $1 \le j \le m$
2. $\{(i, j, \mathsf{AHE}\ (r_{ij} + \sigma_{ij})) : (i, j) \in \mathcal{M}\}$
3. $\mathsf{HE}(||_{(i,j)\in\mathcal{M}}\ (-r_{ij} - \sigma_{ij}, 0, \ldots, 0))$
4. $\mathsf{HE}(||_{(i,j)\in\mathcal{M}}(-r_{ij}, 0, \ldots, 0))$

Figure 2: The Rating-Upload Phase

and fractional parts of a real number in ratings as well as the fractional part of real numbers λ, μ, and γ in gradient descent computation.

We denote by α and β the number of bits used to represent the fractional part of real numbers in ratings and γ, respectively. We do not set such numbers for λ, μ assuming that they are smaller than α (see Step 6 of Figure 8). Note that λ, μ, and γ are privately selected by the RecSys and kept secret from users and the CSP.

5.2 Rating-Upload

In the rating-upload phase (Figure 2), the users upload their ratings so that the RecSys performs matrix factorization to obtain the user and item profiles. Since users' ratings are secret information, the RecSys are given encrypted ratings via the CSP. During the phases no one learns r_{ij} except the user i. In the below protocol, all public key encryption schemes use only public/secret keys of the CSP. So we omit key argument (the 1st argument) in encryption. The detailed description is as follows:

1. For $i \in [n]$ and $j \in [m]$, each user i encrypts ratings r_{ij}'s with AHE under the public key of the CSP. The users send $(i, j, \mathsf{AHE}(r_{ij}))$'s to the RecSys.

2. Receiving
$$\{(i, j, \mathsf{AHE}(r_{ij})) : (i, j) \in \mathcal{M}\},$$
the RecSys generates random masks σ_{ij}'s. It then computes and sends
$$\{(i, j, \mathsf{AHE}(r_{ij} + \sigma_{ij})) : (i, j) \in \mathcal{M}\}$$
to the CSP.

3. The CSP decrypts AHE-ciphertexts to obtain
$$\{(i, j, r_{ij} + \sigma_{ij}) : (i, j) \in \mathcal{M}\}.$$
It then sets the set of d-dimensional vectors
$$\{(-r_{ij} - \sigma_{ij}, 0, ..., 0) : (i, j) \in \mathcal{M}\}.$$
Finally it computes and sends
$$\mathsf{HE}(\|_{(i,j)\in\mathcal{M}}\ (-r_{ij} - \sigma_{ij}, 0, ..., 0))$$
to the RecSys.

4. From the knowledge of $\sigma_{ij}'s$ the RecSys can build the set of d-dimensional vectors
$$\{(\sigma_{ij}, 0, ..., 0) : (i, j) \in \mathcal{M}\}.$$

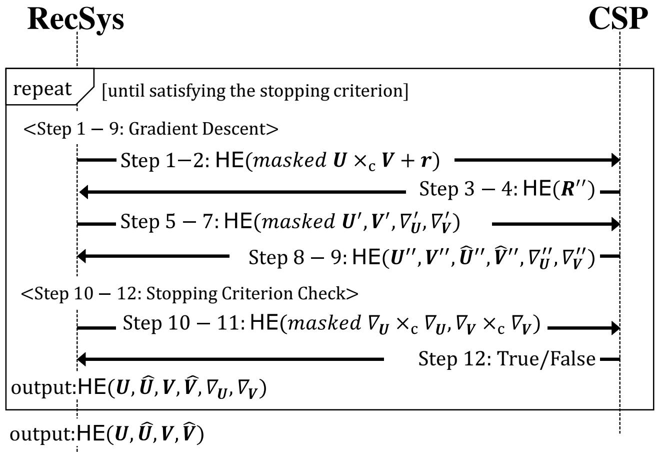
RecSys
CSP
repeat
[until satisfying the stopping criterion]
<Step 1 – 9: Gradient Descent>
Step 1–2: HE(masked U ×c V + r)
Step 3 – 4: HE(R'')
Step 5 – 7: HE(masked U', V', ∇'U, ∇'V)
Step 8 – 9: HE(U'', V'', Û'', V̂'', ∇''U, ∇''V)
<Step 10 – 12: Stopping Criterion Check>
Step 10 – 11: HE(masked ∇U ×c ∇U, ∇V ×c ∇V)
Step 12: True/False
output: HE(U, Û, V, V̂, ∇U, ∇V)
output: HE(U, Û, V, V̂)

Figure 3: The Matrix Factorization Phase

It then adds

$$\mathsf{HE}(\|_{(i,j)\in\mathcal{M}}\ (\sigma_{ij}, 0, ..., 0))$$

to a given

$$\mathsf{HE}(\|_{(i,j)\in\mathcal{M}}\ (-r_{ij} - \sigma_{ij}, 0, ..., 0)).$$

The result is a HE-encryption of

$$\|_{(i,j)\in\mathcal{M}}\ (-r_{ij}, 0, ..., 0).$$

Let $\mathbf{r} = \|_{(i,j)\in\mathcal{M}}\ (-r_{ij}, 0, ..., 0)$. At last the RecSys selects uniformly random norm 1 vectors $\mathbf{u}_i(0)$ and $\mathbf{v}_j(0)$ for $i \in [n]$ and $j \in [m]$ and sets $\mathbf{U}(0)$, $\hat{\mathbf{U}}(0)$, $\mathbf{V}(0)$, and $\hat{\mathbf{V}}(0)$.

5.3 Matrix Factorization

Given $\mathcal{M}$, let

$$\mathbf{U}(t-1) = \|_{(i,j)\in\mathcal{M}}\ \mathbf{u}_i(t-1),\ \ \mathbf{V}(t) = \|_{(i,j)\in\mathcal{M}}\ \mathbf{v}_j(t-1)$$

from $\mathbf{u}_i(t-1)$'s and $\mathbf{v}_j(t-1)$'s in (1). $\hat{\mathbf{U}}(t-1)$ and $\hat{\mathbf{V}}(t-1)$ are induced vectors from $\mathbf{U}(t-1)$ and $\mathbf{V}(t-1)$, respectively as defined in the previous section. For items which nobody rates, item profiles are directly computed by the RecSys, since no rating r_{ij} is involved in the gradient descent computation (1) and (2). For items which are rated, we consider the vector representation of ∇_{u_i} and ∇_{v_j} for $i \in \mathcal{M}_I$ and $j \in \mathcal{M}_J$ in (2) as

$$\begin{aligned}\nabla_{\mathbf{U}}(t-1) &= \nabla_{u_{i_1}}(t-1) \parallel \cdots \parallel \nabla_{u_{i_{\ell_u}}}(t-1) \quad (i_k \in \mathcal{M}_I),\\ \nabla_{\mathbf{V}}(t-1) &= \nabla_{v_{j_1}}(t-1) \parallel \cdots \parallel \nabla_{v_{j_{\ell_v}}}(t-1) \quad (j_k \in \mathcal{M}_J).\end{aligned}$$

Now we are in a position to present how to compute HE-encryption of $\{\mathbf{U}, \mathbf{V}, \hat{\mathbf{U}}, \hat{\mathbf{V}}, \nabla_{\mathbf{U}}, \nabla_{\mathbf{V}}\}$ at t from HE-encryption at $t-1$ for a positive integer t. Note that the RecSys can initialize each vector since it does not require rating information at $t = 0$.

In the protocol, random masking vectors are required when the RecSys sends HE-ciphertexts to the CSP. Given $\mathcal{M}$ and the message space $\mathcal{B}$ of HE, we define the distribution $\mathcal{D}(\mathcal{M}, \mathcal{B})$, which works as follows:

1. chooses M making vectors $\mathbf{ev}^{(i,j)} = (\mathsf{ev}_1^{(i,j)}, ..., \mathsf{ev}_d^{(i,j)})$ for $(i,j) \in \mathcal{M}$, where $\mathsf{ev}_k^{(i,j)}$'s are random elements from $\mathcal{B}$.

2. outputs $\mathbf{ev} = \|_{(i,j)\in\mathcal{M}} \mathbf{ev}^{(i,j)}$.

The protocol consists of the secure 2-party computation between the RecSys and CSP, as shown in Figure 3. To keep intermediate values secret from the CSP, the RecSys mostly sends the HE-encryption of masked values. Then the CSP decrypts them. The CSP performs fixed point arithmetic and gets back them to the RecSys in HE-encrypted form. Given HE-ciphertexts the RecSys can remove masks since it creates them. It is important to note that this mask-removing approach causes small errors to the accuracy of the resulting profiles. However it is easy to see that errors are limited and negligible. For details refer to Section 7.

The detailed procedure for the matrix factorization phase is described in Figure 8 in Appendix C. In the below we briefly present what information are transmitted between 2 parties. Note that all operations by the RecSys are performed over HE-ciphertexts under the HE-public key of the CSP, i.e., $\mathsf{hpk}_{\mathsf{csp}}$.

Step 1–2. Multiply component-wise & add rating: The RecSys determines parameters λ, μ, and γ within the ranges defined in the setup phase. It computes $\langle \mathbf{u}_i, \mathbf{v}_j\rangle - r_{ij}$'s for $(i,j) \in \mathcal{M}$ in (2). The results are presented in a vector, that is, each $\langle \mathbf{u}_i, \mathbf{v}_j\rangle - r_{ij}$ is structured to the vector

$$\mathbf{f}_{ij} = (u_{i,1}v_{j,1} - r_{ij}, u_{i,2}v_{j,2}, \ldots, u_{i,d}v_{j,d}).$$

These vectors are concatenated to $\|_{(i,j)\in\mathcal{M}} \mathbf{f}_{ij}$ which equals to $\mathbf{R}'(t-1)$ in Figure 8. Added by a masking vector ϵ, it is sent to the CSP.

Step 3–4. Add for inner product: The CSP decrypts and performs fixed point arithmetic on the masked $\mathbf{f}_{ij}$. It computes the masked version R_{ij} of $\mathrm{sum}(\mathbf{f}_{ij}) = \langle \mathbf{u}_i, \mathbf{v}_j\rangle - r_{ij}$. It sends a HE-encryption of $\|_{(i,j)\in\mathcal{M}} (R_{ij}, \ldots, R_{ij})$ to the RecSys ($\mathbf{R}''(t-1)$ in Figure 8).

Step 5–7. Compute gradient descent in concatenated form: The RecSys can remove masks in $\mathbf{R}''(t-1)$ from the knowledge of the mask vector ϵ, which returns $\mathbf{R}(t-1)$ in Figure 8. Through this stage the RecSys computes the vector representation of $\nabla_{\mathbf{u}_i}$'s and $\nabla_{\mathbf{v}_j}$'s. More precisely, instead of computing the sum of vectors in summations of (2), it first concatenates vectors according to $\mathcal{M}$, i.e.,

$$\|_{(i,j)\in\mathcal{M}} \mathbf{v}_j(\langle \mathbf{u}_i, \mathbf{v}_j\rangle - r_{ij}), \quad \|_{(i,j)\in\mathcal{M}} \mathbf{u}_i(\langle \mathbf{u}_i, \mathbf{v}_j\rangle - r_{ij}).$$

Then it adds $\lambda\mathbf{u}_i$'s and $\mu\mathbf{v}_j$'s in the last terms in (2) to the above two vectors with appropriate positions, which are structured in $\lambda\hat{\mathbf{U}}(t-1)$ and $\mu\hat{\mathbf{V}}(t-1)$, respectively. The results are $\nabla'_{\mathbf{U}}(t)$ and $\nabla'_{\mathbf{V}}(t)$ in Figure 8.

In addition, it computes (1) in the concatenated vector form. First it computes $\gamma\nabla'_{\mathbf{U}}(t)$ and $\gamma\nabla'_{\mathbf{V}}(t)$. It then subtracts them from the concatenated representation of $\mathbf{u}_i(t-1)$'s and $\mathbf{v}_j(t-1)$'s (the 1st terms in (1)) to appropriate positions which are structured in $\hat{\mathbf{U}}(t-1)$ and $\hat{\mathbf{V}}(t-1)$, respectively. This results in $\mathbf{U}'(t)$ and $\mathbf{V}'(t)$ at Step 6 in Figure 8. Finally it sends HE-encryptions of masked $\{\nabla'_{\mathbf{U}}(t), \nabla'_{\mathbf{V}}(t), \mathbf{U}'(t), \mathbf{V}'(t)\}$ to the CSP.

Step 8–9. Compute gradient descent and reconstitute data: The CSP decrypts HE-ciphertexts and performs fixed point arithmetic on them. As a result, it obtains the masked values of $\mathbf{u}_i(t)$, $\mathbf{v}_j(t)$, $\nabla_{\mathbf{u}_i}(t)$, and $\nabla_{\mathbf{v}_j}(t)$ in (1) and (2). It then sets the masked values of $\mathbf{U}(t)$, $\mathbf{V}(t)$, $\nabla_{\mathbf{U}}(t)$ and $\nabla_{\mathbf{V}}(t)$ as defined above.

Step 10–11. Recover user/item profiles and generate threshold: The RecSys recovers HE-encryptions of $\{\mathbf{U}, \mathbf{V}, \hat{\mathbf{U}}, \hat{\mathbf{V}}, \nabla_{\mathbf{U}}, \nabla_{\mathbf{V}}\}$ from the knowledge of mask vectors. It computes the vector representation of $\| \nabla_{u_i}(t) \|_2^2$, i.e., component-wise square of $\nabla_{u_i}(t)$ for $i \in \mathcal{M}_I$. This can be done via $\nabla_{\mathbf{U}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{U}}(t)$ with our notation. Similarly, it computes $\nabla_{\mathbf{V}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{V}}(t)$.

The RecSys generates threshold values ω_u and ω_v for the user and item profiles, respectively. It then sends HE-encryptions of $\nabla_{\mathbf{U}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{U}}(t)$ and $\nabla_{\mathbf{V}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{V}}(t)$ after adding mask vectors to the CSP. It also sends masked ω_u and ω_v, where the added masks are equal to sums of entries of mask vectors for $\nabla_{\mathbf{U}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{U}}(t)$ and $\nabla_{\mathbf{V}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{V}}(t)$, respectively.

Step 12. Verify stopping criteria: The CSP verifies the stopping criteria by computing

$$\omega_u - \mathrm{sum}(\nabla_{\mathbf{U}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{U}}(t)) = \omega_u - \sum_{i:(i,j)\in\mathcal{M}} \| \nabla_{\mathbf{u}_i} F(U,V) \|_2^2 \, .$$

The same process goes to the item profile case.

5.4 Enhancing Privacy of Rated Items

Besides users' ratings on items, other information can be used to infer users' partial identities. In [48], it has been shown that one can infer user's gender from which items the user rated. To address the issue, the use of differential privacy or injection of fake ratings has been suggested. However these methods distort results of data analysis, i.e., U and V of matrix factorization.

The protocol presented in the previous section permits all the above approaches, that is, in the rating-upload phase users can blur their rating information. Note that noising users' ratings is unnecessary in the protocol since all ratings are encrypted, hence, kept secret from the RecSys and CSP. Thus we extend the previous protocol by taking the injection of fake ratings in this section. To prevent the distortion of resulting profiles, the protocol removes the effects of fake ratings in the matrix factorization phase from the use of input indicators produced by the users in the encrypted form[2].

The protocol then is able to additionally hide the following information from the above approach: (i) which items the users rated exactly and (ii) the number of ratings given by the users. Note that this approach increases the number of ratings in encrypted form. However it increases the running time of the protocol only linearly with the number of fake items.

There are slight differences in both the rating-upload and matrix factorization phases. We give the description of the extended protocol focusing on the difference.

The rating-upload phase. In the rating-upload phase the users generate their fake and real ratings attached with indicators $\tau_{ij} \in \{0,1\}$. 0 indicates the fake and 1 the real. Users send the AHE-encryption of the ratings under CSP's public key. After the CSP decrypts them, it figures out the HE-encryption of the masked rating vector as it does in the original protocol. In addition, it reconstructs the masked indicator vector τ according to $\mathcal{M}$ so that every u_i in $\mathbf{U}(t)$ corresponds to $(1,\ldots,1)$ if it is real or $(0,\ldots,0)$ otherwise in the clear forms. The detailed procedures are as follows (all encryptions are made under CSP's public key):

1. User i sends $(i, j, \mathsf{AHE}(r_{ij}), \mathsf{AHE}(\tau_{ij}))$ to the RecSys.
2. Adding errors σ_{ij} and σ'_{ij} to AHE-encryptions, the RecSys sends $\{(i, j, \mathsf{AHE}(r_{ij}+\sigma_{ij}), \mathsf{AHE}(\tau_{ij}+\sigma'_{ij}))\}$ to the CSP.
3. The CSP sets two dM-dimensional vectors

$$\|_{(i,j)\in\mathcal{M}} \, (-r_{ij} - \sigma_{ij}, 0, ..., 0),$$

$$\|_{(i,j)\in\mathcal{M}} \, (\tau_{ij} + \sigma'_{ij}, \cdots, \tau_{ij} + \sigma'_{ij}),$$

   where all vectors in concatenation are d-dimensional. The CSP encrypts them with HE and sends the ciphertexts to the RecSys.
4. The RecSys removes errors in the encrypted vectors as it does in the original protocol. The rest of the phase is identical.

[2]This approach has been mentioned in [35], however, was not included in the design of the protocol for efficiency issue.

		RecSys	CSP	Communication Cost
RU	AHE.dec	-	M	$M\lvert\mathsf{AHE}\rvert + \frac{dM}{L}\lvert\mathsf{HE}\rvert$
	HE.enc	-	dM/L	
MF	HE.enc	-	$7dM/L$	$\frac{14dM}{L}\lvert\mathsf{HE}\rvert$
	HE.dec	-	$7dM/L$	
	HE.add	$5dM/L$	-	
	HE.smul	$7dM/L$	-	
	HE.mul	$5dM/L$	-	

Table 1: Computation and Communication Complexities (RU: rating-upload, MF: matrix factorization, enc and dec: encryption and decryption, resp., add, smul, mul: addition, scalar multiplication, and multiplication over encrypted data, resp., |AHE| and |HE|: the sizes of AHE and HE-ciphertexts, resp., M: # of ratings, d: the dimension of profile vectors, L: # of slots)

The matrix factorization phase. Let τ be a HE-encryption of the error-free indicator vectors obtained in the previous phase, i.e, $\tau = \mathsf{HE}(\|_{(i,j)\in\mathcal{M}} (\tau_{ij}, \cdots, \tau_{ij}))$. The only difference happens when computing $\nabla'_{\mathbf{U}}(t)$ and $\nabla'_{\mathbf{V}}(t)$, that is, the multiplication of $\mathbf{R}$ vector to $\mathbf{V}$ and $\mathbf{U}$ vectors at Step 6 in Figure 8. That is, the RecSys computes

$$\mathsf{HE}(\tau \times_{\mathsf{c}} \mathbf{V}(t-1) \times_{\mathsf{c}} \mathbf{R}(t-1) + \alpha\lambda\hat{\mathbf{U}}(t-1)),$$

$$\mathsf{HE}(\tau \times_{\mathsf{c}} \mathbf{U}(t-1) \times_{\mathsf{c}} \mathbf{R}(t-1) + \alpha\mu\hat{\mathbf{V}}(t-1)).$$

Note that multiplying $\mathsf{HE}(\tau)$ requires the appropriate set-up of multiplication depth for HE, but does not the enlarged message space.

6 Analysis of the Protocol

In this section we give an analysis of our protocol in Section 5.

6.1 Complexity

We analyze computation and communication complexities of the proposed protocol in Table 1. We only counted the number of cryptographic operations performed by each party. Given fixed d, the table shows that the proposed protocol requires $O(M/L)$ computation and communication complexities.[3]

RU and MF are abbreviations for rating-upload, matrix factorization, respectively (for recommendation, refer to Appendix B). enc and dec denote encryption and decryption algorithm, respectively. add, smul, and mul point out addition, scalar multiplication, and multiplication over encrypted data, respectively. |AHE| and |HE| mean the sizes of AHE and HE-ciphertexts, respectively. We remark that masks themselves are encryption, so we do not count adding and removing masks by the RecSys as a cryptographic operation.

In MF, workloads for both the RecSys and CSP are comparable. In fact we observe that HE.dec is more expensive than any other homomorphic operations (see Section 8.3) because it requires an evaluation of polynomials to recover plaintext in each slot. However our evaluation in Section 8.2 shows that the workload for the CSP in our construction is quite small compared to that from garbled circuit approaches.

[3] M includes the number of fake ratings in the case of the extended protocol in Section 5.4.

6.2 Security

As mentioned in Section 1, the purpose of privacy preserving matrix factorization is to protect the followings: user's rating values, the list and number of items a user has rated, tuning parameters of the RecSys side, intermediate values and user/item profiles.

Our protocol protects all user's rating values, intermediate values, and resulting user/item profiles using homomorphic encryptions (AHE and HE) and random mask all in the three phases. In a rating uploading phase, users' ratings are AHE-encrypted under CSP's public key, so the RecSys cannot reveal the data; and the CSP can obtain no information about users' ratings even if it decrypts AHE-ciphertexts, because the rating values are masked by the RecSys. Note that mask works if there is no colluding attack of the RecSys and CSP, which is already assumed in Section 3. The similar argument works in the matrix factorization and recommendation phases. All intermediate values are kept secret, since the RecSys computes gradient descent on HE-encrypted data homomorphically and the values are masked when it needs help of the CSP. Furthermore, user/item profiles, output of matrix factorization, are obtained in HE-encrypted form and used in recommendation phase with masking for an individual user.

Regarding protection for the rated item list and number of ratings each user generates, our construction adopts an injecting method of fake ratings along with indicators. The construction without hiding the item profile in [35] has inferior security level than that of ours, since the adversary can infer whether a target user rate on a target item or not as described in [35, Section 4.1]. They extended their construction to protect the rated item list using random mask provided by the CSP at the cost of additional computation and communication overhead [35, Section 4.2]. This extended version seems to have higher security level with respect to the rated item list than that of ours.

However there are some applications where only a specific item set, possibly small, is sensitive to leak whether rated or not. In such applications, we only obfuscate the small portion of the whole item set in our construction, hence achieve the same security with the extended version of [35]. For example, we can apply the collaborative filtering to medical check-up to predict patients' medical condition. The fact whether patients have checked typical medical condition such as blood pressure and sugar does not leak any private information on the patients. On the other hand, the fact that they have checked HIV infection or mental status is significantly related to privacy. In this case, we obfuscate only these sensitive checkup items to keep the user privacy.

When computing gradient descent in matrix factorization phase, the RecSys itself computes constant HE-multiplications with tuning parameters (γ, μ, λ) without help of the CSP. Therefore the optimized tuning parameters for quality recommendation can be kept secret from the CSP.

7 IMPLEMENTATION

In this section, we describe an implementation of our construction. In our protocol, we use two encryption scheme, an additively homomorphic encryption (AHE) and a fully (or somewhat) homomorphic encryption (FHE). For AHE in our protocol, a *partially* additive homomorphic property, which allows a constant addition to encrypted data without knowing any secret key, is enough, since the additive property is only used for large random masking. Therefore, we use hash-ElGamal encryption scheme [11]. The description of hash-ElGamal scheme is presented in Appendix A. We use a RLWE-based homomorphic encryption for FHE, which provides several optimizations such as efficient key switching, message packing and message slot movement [6, 21, 22, 41]. We implement our protocol using HELib [27], a homomorphic encryption library based on RLWE. The library is written in C++ and uses the Gnu MP library [26] and the NTL mathematical library [40].

The main bottleneck in deploying FHE is huge computation and communication overhead due to its large ciphertext. There are several factors that determines the size of ciphertext: multiplicative depth, security level and message space $(\mathbb{Z}_p)$. If the bit length of message space becomes k times larger, then the size of ciphertext is increased k^2 times.[4] Since the computational overhead heavily depends on the size of ciphertext, the reduction of the size is the most important to enhance the efficiency. We consider chinese remainder theorem (CRT) to represent a large message. When applying CRT to message space, we obtain only linear overhead increase in ciphertext size. For a large message msg, we compute $\mathsf{HE.enc}(\mathsf{msg}_1), \ldots, \mathsf{HE.enc}(\mathsf{msg}_k)$ where $\mathsf{msg}_i = \mathsf{msg} \bmod p_i$ for relatively primes $p_1, \ldots, p_k$ of the same size. Since the Chinese remainder algorithm is a ring isomorphism, this optimization does not affect the correctness of our construction.

8 EVALUATION

In this section we discuss accuracy of computation results including user and item profiles in the proposed protocol and provide experiment results.

8.1 Error Estimation

Compared to the work in [35], our protocol introduces more error in profiles depending on d, the dimension of profiles, due to fixed point arithmetic between the RecSys and CSP described in 4.3. The Recsys computes

$$\sum_{k=1}^{d} (\overline{f_{ijk} + \epsilon_k} - \overline{\epsilon_k})$$

not $\sum_{k=1}^{d} \overline{f_{ijk}}$ for $\mathbf{f}_{ij} = (f_{ijk})_{k=1,\ldots,d}$ and a mask vector $(\epsilon_k)_{k=1,\ldots,d}$ in Step 3–4 of matrix factorization phase. This computation increases the error size in profiles. For $a, \epsilon \in \mathbb{R}$, let us consider the difference of $\overline{a}$ and $(\overline{a+\epsilon} - \overline{\epsilon})$. We have

$$\left| \overline{a} - \left(\overline{a+\epsilon} - \overline{\epsilon} \right) \right| \leq 1,$$

which means the overflow is bounded by $2^{-\alpha}$. Therefore the error in our protocol becomes at most $d \cdot 2^{-\alpha}$ by triangle inequality and this does not affect significantly on the accuracy of matrix factorization.

We evaluated the relative error of our matrix factorization to the original matrix factorization that operates in the clear with the float point representation. We use the relative error, which is used in [35]. Given user profiles U and item profiles V, the squared error is

$$E(U, V) = \sum_{(i,j) \in \mathcal{M}} (r_{ij} - \langle \mathbf{u}_i, \mathbf{v}_j \rangle)^2.$$

The relative error is then defined as

$$\frac{|E(U^*, V^*) - E(U, V)|}{|E(U, V)|},$$

where U^* and V^* are computed using our protocol with fixed-point representation and U and V are computed using gradient descent executed in the clear over floating point arithmetic. Figure 4 shows the relative error of our matrix factorization for the 100K MovieLens dataset. It shows that the relative errors occurred during 10 iterations is small (less than 10^{-4}) for more than 20-bit fractional part.

[4]This can be easily verified from the parameter selection of [22]

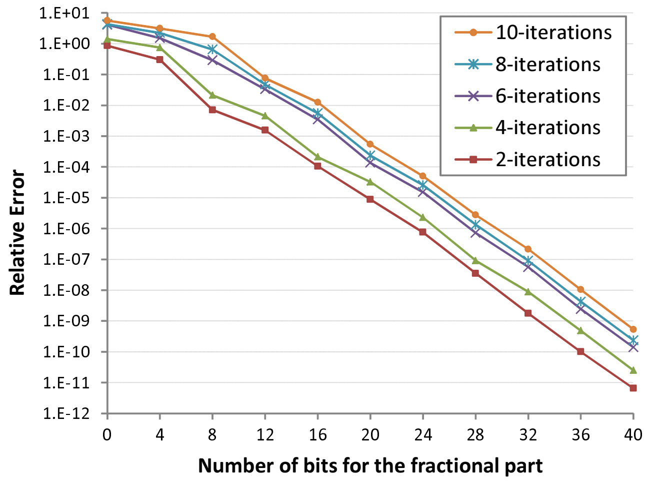

Figure 4: Relative Errors due to the Fixed Point Representation

8.2 Experiment

Parameters and encoding of message. Let us consider message space of the i-th FHE, $R_{p_i^{r_i}} = \mathbb{Z}_{p_i^{r_i}}[x]/\Phi_{k_i}(x)$, where $\Phi_{k_i}(x)$ is the k_i-th cyclotomic polynomial. Since the bit length of message space for the dataset is 36, we use three FHE whose message space moduli are relatively coprime p_i's of size 16-bit to represent 48-bit message space. In this case, we fix $r_i = 1$ for all i and find p_i and k_i to have large enough message slots in R_{p_i}. We choose p_i and k_i such that the number of message slots of underlying FHE is at least 6000 for all i. To evaluate our protocol we need FHE that allows only 3 multiplicative depth. Putting them together, the FHE-ciphertext size is 700 KB. For AHE, we use hash-ElGamal from OpenSSL [1] with slight customization (refer to Appendix A). Every encryption scheme supports 80-bit security.

Experiment results. We implement the proposed protocol and test the performance. We use MovieLens 100K dataset for experiment, which is popular real dataset in the research area. It contains 100K ratings on 1682 movies made by 943 users. For comparison with [35], we evaluate the time and communication performance for the above dataset restricted to 40 most popular movies. The restricted dataset contains 14683 ratings generated by 940 users. The rating-upload phase finishes in 4.3 seconds. For the matrix factorization, we measure the run time for one iteration to compare the result of the previous protocol [35]. Our matrix factorization protocol takes 1.5 minutes with 580 MB for communication. Note that our matrix factorization protocol includes the procedure to verify the stopping criterion.

The test is performed on a single machine with 3.4GHz 6-cores 64GB RAM, where the operation system is Ubuntu and the program language is C++. Our platform equips about 2 times faster CPU than that of [35]. After normalizing the factor, we can estimate our matrix factorization protocol runs at least 50 times faster and consumes much less communication cost[5], compared to [35]. Figure 5 and 6 compare the execution time and the communication cost of our matrix factorization to those of [35] for the same sets, respectively.

As describe in Section 5.4, each user can add fake ratings to enhance his own privacy. Our extended protocol takes 1.8 minutes with 623 MB for 10% fake ratings and 2.3 minutes with 910 MB for 50%, which is almost linear growth to the number of ratings.

[5]When $M = 4096$, [35] already consumes 40 GB.

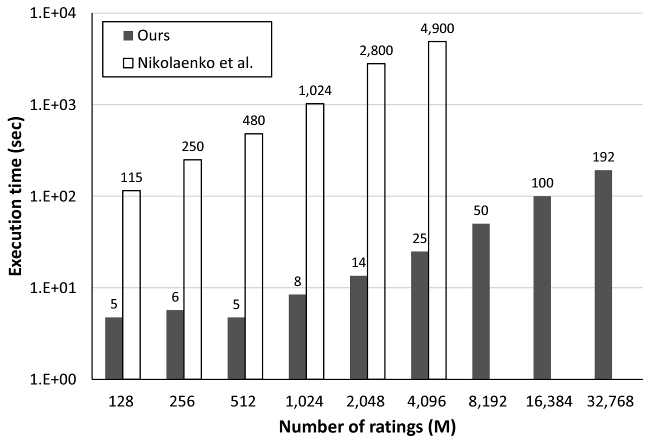

Figure 5: Comparison of Execution Time to [35]

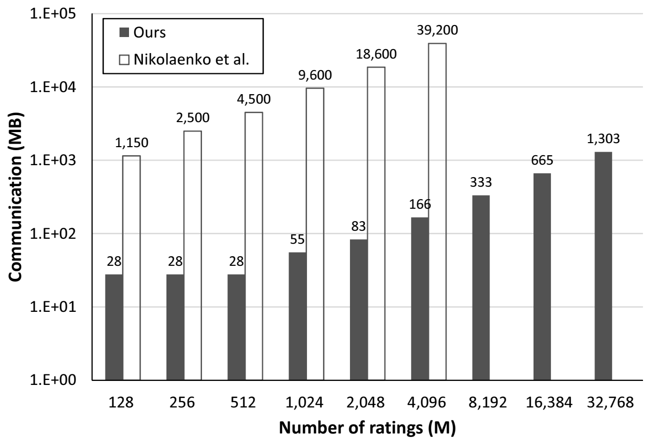

Figure 6: Comparison of Communication Cost to [35]

8.3 Further Analysis and Comparison

We consider further potential improvement of the proposed protocol toward parallelism. We also give comparison to recent results on parallelized privacy-preserving matrix factorization [34] and optimization of circuits for the garbled circuit protocol [42].

Parallelism. In the matrix factorization phase of our protocol, all operations over ciphertexts such as $\mathsf{HE.enc}$, $\mathsf{HE.dec}$, $\mathsf{HE.add}$, $\mathsf{HE.smul}$, and $\mathsf{HE.mul}$ can be parallelized since ciphertexts are treated independently. For example, consider Step 1 of the matrix factorization in Figure 8 to compute

$$\mathsf{HE}(\mathbf{U}(t-1)) \times \mathsf{HE}(\mathbf{V}(t-1)) - 2^{\alpha} \cdot \mathsf{HE}(\mathbf{r}).$$

Suppose there are N' machines acting as the RecSys and each $\mathsf{HE}(\mathbf{U}(t-1))$, $\mathsf{HE}(\mathbf{V}(t-1))$, and $\mathsf{HE}(\mathbf{r})$ consists of N HE-ciphertexts, say $\mathsf{HE}(\mathbf{U}(t-1))_i$, $\mathsf{HE}(\mathbf{V}(t-1))_i$, and $\mathsf{HE}(\mathbf{r})_i$ for $i = 1, \ldots, N$. Parallelism then can be achieved by sending

$$\{\mathsf{HE}(\mathbf{U}(t-1))_i, \mathsf{HE}(\mathbf{V}(t-1))_i, \mathsf{HE}(\mathbf{r})_i\}_{i=j+1,\ldots,j+N/N'}$$

to the j-th machine and gathering the result

$$\{\mathsf{HE}(\mathbf{U}(t-1))_i \times \mathsf{HE}(\mathbf{V}(t-1))_i - 2^{\alpha} \cdot \mathsf{HE}(\mathbf{r})_i\}_{i=j+1,\ldots,j+N/N'}$$

computed by the j-th machine for $j = 1, \ldots, N'$. The CSP side can be parallelized in a similar way.

On the contrary to the case of ciphertexts, operations over plaintexts should be performed by a single machine since they include reconstitution of data according to the order defined on $\mathcal{M}$. However run time for operations over plaintexts is negligible compared to that over ciphertexts; From the experiment for 14K dataset, we observe that the RecSys and CSP totally spend 15.5 and 73.8 seconds for matrix factorization (1.5 minutes overall), respectively. Run times over plaintexts are 1.57 seconds for the RecSys and 0.16 for the CSP. Run time over plaintexts has only 1.9% of total run time, which implies that time complexity of our matrix factorization is almost reduced by a factor of the number of machines involved in parallel operations.

Further comparison from the estimate. In 2015 Songhori et al. presented techniques, called TinyGarble, for optimizing circuits for protocols with garbled circuits [42]. It minimizes the number of non-XOR gates in a circuit. Their proof-of-concept implementation showed that the technique decreases the number of non-XOR gates by 80%, which results in 5 times speed-up in generating and evaluation of a garbled circuit. The run time of Nikolaenko et al.'s protocol together with TinyGarble is estimated to about 34.8 minutes. It is still 10 times slower than our protocol (3 minutes after normalization due to CPU clock).

At the same time Nayak et al. proposed a parallel secure computation framework called GraphSC. It focuses on secure 2-party computation protocol based on garbled circuit [34]. They evaluated matrix factorization over 1M ratings on GraphSC, which takes 13 hours using a cluster, which consists of 7 machines of 128 processors (1.9–2.6 GHz CPUs) with 2048 cores overall. From the experiment result, estimated run time for 14K ratings using GraphSC is about 10 minutes. Since our experiment takes 1.5 minutes (3 minutes after normalization from CPU clock), our protocol even on a single machine with 6 cores is still about 3 times faster.

GraphSC together with TinyGarble may reduce the run time of matrix factorization further, that is, 13 hours for 1M rating to 2.6 hours. We estimate our protocol under the same environment of GraphSC, that is, 7 machines with 2048 cores. Our experiment on 14K ratings over a single machine with 6 cores (3.4 GHz) shows that it takes about 175.3 seconds for operations over ciphertexts and 3.4 seconds over plaintexts with CPU clock normalization. Thus, for 1M dataset, parallelization of our protocol is expected to take 36.7 seconds for operation over ciphertexts and 244.2 seconds over plaintexts since operations over plaintexts is not parallelizable. Thus, by adapting parallelization, the run time of our protocol is estimated to 4.7 minutes, which is about 33 times faster than the expected run time of GraphSC together with TinyGarble for 1M dataset.

9 CONCLUSION

A new efficient and practical privacy preserving matrix factorization protocol is proposed in this paper. The protocol protects users' rating values, user/item profiles and service's tuning parameters. Using various obfuscation techniques, we reasonably protect the rated item list and number users rated. We improved the efficiency by optimizing the use of fully homomorphic encryption, so the performance of ours is at least 50 times faster than that of the previous work [35]. Furthermore, ours satisfies both accuracy and completeness requirements.

References

[1] *OpenSSL: The Open Source toolkit for SSL/TLS.*

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng*, 17(6):734–749, 2005.

[3] J. W. Bos, K. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. Cryptology ePrint Archive, Report 2014/336, 2014.

[4] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS 2015*, 2015.

[5] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO 2012*, pages 868–886, 2012.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325, 2012.

[7] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS 2011*, pages 97–106, 2011.

[8] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *CRYPTO 2011*, pages 505–524, 2011.

[9] J. Canny. Collaborative filtering with privacy. In *IEEE Security Privacy 2002*, pages 45–57, 2002.

[10] J. H. Cheon, J. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. Batch fully homomorphic encryption over the integers. In *EUROCRYPT 2013*, pages 315–335, 2013.

[11] B. Chevallier-Mames, P. Paillier, and D. Pointcheval. Encoding-free elgamal encryption without random oracles. In *PKC 2006*, pages 91–104, 2006.

[12] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In *PAKDD 2015*, pages 442–455, 2015.

[13] R. Chow, M. A. Pathak, and C. Wang. A practical system for privacy-preserving collaborative filtering. In *ICDM 2012*, pages 547–554, 2012.

[14] J. Coron, T. Lepoint, and M. Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *PKC 2014*, pages 311–328, 2014.

[15] J. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *CRYPTO 2011*, pages 487–504, 2011.

[16] J. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *EUROCRYPT 2012*, pages 446–464, 2012.

[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC 2006*, pages 265–284, 2006.

[18] Z. Erkin, T. Veugen, T. Toft, and R. Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE Trans. Inf. Forensics Security*, 7(3):1053–1066, 2012.

[19] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC 2009*, pages 169–178, 2009.

[20] C. Gentry, S. Halevi, and N. P. Smart. Better bootstrapping in fully homomorphic encryption. In *PKC 2012*, pages 1–16, 2012.

[21] C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT 2012*, pages 465–482, 2012.

[22] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, pages 850–867, 2012.

[23] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013*, pages 75–92, 2013.

[24] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[25] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[26] T. Granlund and the GMP development team. *GMP – The GNU Multiple Precision Arithmetic Library*.

[27] S. Halevi and V. Shoup. Algorithms in helib. In *CRYPTO 2014*, pages 554–571, 2014.

[28] J. Hua, C. Xia, and S. Zhong. Differentially private matrix factorization. In *IJCAI 2015*, pages 1763–1770, 2015.

[29] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security 2011*, 2011.

[30] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[31] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *KDD 2009*, pages 627–636, 2009.

[32] N. Mohammed, R. Chen, B. Fung, and P. S. Yu. Differentially private data release for data mining. In *KDD 2011*, pages 493–501, 2011.

[33] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Security Privacy 2008*, pages 111–125, 2008.

[34] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. GraphSC: Parallel secure computation made easy. In *IEEE Security Privacy 2015*, pages 377–394, 2015.

[35] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *ACM CCS 2013*, pages 81–8121–812.

[36] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE Security Privacy 2013*, pages 334–348, 2013.

[37] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT 1999*, pages 223–238, 1999.

[38] N. Ramakrishnan, B. Keller, B. Mirza, A. Y. Grama, and G. Karypis. Privacy risks in recommender systems. *IEEE Internet Comput.*, 5(6):54–63, 2001.

[39] Y. Shen and H. Jin. Privacy-preserving personalized recommendation: An instance-based approach via differential privacy. In *ICDM 2014*, pages 540–549, 2014.

[40] V. Shoup. *NTL: A Library for doing Number Theory.*

[41] N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.

[42] E. M. Songhori, S. U. Hussain, A. Sadeghi, T. Schneider, and F. Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *IEEE Security Privacy 2015*, pages 411–428, 2015.

[43] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:1–19, 2009.

[44] L. Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[45] G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. In *ICDMW 2008.*, pages 553–562, 2008.

[46] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT 2010*, pages 24–43, 2010.

[47] T. Veugen, R. de Haan, R. Cramer, and F. Muller. A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. *IEEE Trans. Inf. Forensics Security*, 10(3):445–457, 2015.

[48] U. Weinsberg, S. Bhagat, S. Ioannidis, and N. Taft. Blurme: inferring and obfuscating user gender based on ratings. In *RecSys 2012*, pages 195–202, 2012.

[49] Y. Xin and T. Jaakkola. Controlling privacy in recommender systems. In *NIPS 2014*, pages 2618–2626, 2014.

[50] A. C. Yao. Protocols for secure computations. In *FOCS 1982*, pages 160–164, 1982.

[51] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon. Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *VLDB 2014*, 7(11):975–986, 2014.

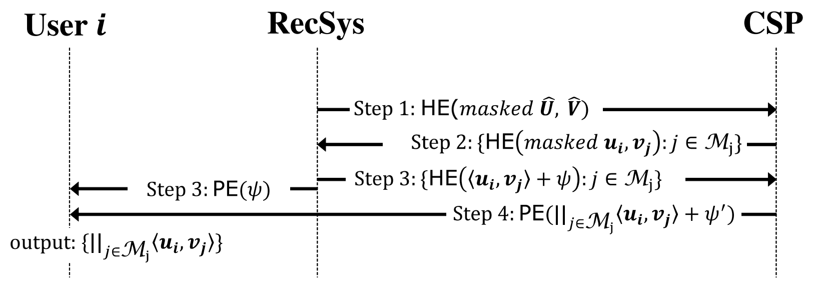
User i
RecSys
CSP
Step 1: HE(masked $\hat{U}$, $\hat{V}$)
Step 2: {HE(masked u_i, v_j): $j \in \mathcal{M}_j$}
Step 3: {HE($\langle u_i, v_j \rangle + \psi$): $j \in \mathcal{M}_j$}
Step 3: PE(ψ)
Step 4: PE($||_{j \in \mathcal{M}_j} \langle u_i, v_j \rangle + \psi'$)
output: {$||_{j \in \mathcal{M}_j} \langle u_i, v_j \rangle$}

Figure 7: The Recommendation Phase

A Hash-ElGamal Encryption

Hash-ElGamal is a variant of ElGamal cryptosystem that provides additive homomorphic property. Let $\mathbb{G}$ be a multiplicative cyclic group of prime order p with a generator g and $H : \mathbb{G} \to \mathbb{Z}_M$ for a sufficiently large integer $M \in \mathbb{N}$ be a cryptographic hash function. When a public key of this cryptosystem is $\mathbf{pk} = (g, y)$ where $y = g^{\mathbf{sk}} \in \mathbb{G}$ for a randomly chosen secret key $\mathbf{sk} \in \mathbb{Z}_p$, a ciphertext ct can be generated from a message $m \in \mathbb{Z}_M$ via the following encryption algorithm

$$ct = (g^\rho, (m + H(y^\rho)) \bmod M)$$

for a uniform random value $\rho \in \mathbb{Z}_p$. Then, the plaintext can be restored from the ciphertext $ct = (ct_1, ct_2)$ via the decryption algorithm

$$ct_2 - H(ct_1^{\mathbf{sk}}) \bmod M.$$

The ciphertext $ct = (ct_1, ct_2)$ can be masked by anyone with any uniform randomly chosen mask $\mu \in \mathbb{Z}_M$ such that

$$\hat{ct} = (ct_1, (ct_2 + \mu) \bmod M) = (\hat{ct}_1, \hat{ct}_2).$$

This corresponds to a masked plaintext by μ. Specifically, when it comes to decryption of $\hat{ct}$, we have

$$\begin{aligned}\hat{ct}_2 - H(\hat{ct}_1^{\mathbf{sk}}) \bmod M &= (ct_2 + \mu) - H(ct_1^{\mathbf{sk}}) \\ &= \{(m + H(g^{\mathbf{sk}\rho})) + \mu\} - H(g^{\mathbf{sk}\rho}) \\ &= m + \mu.\end{aligned}$$

It is noteworthy that the above Hash-ElGamal encryption scheme is slightly modified version of the one described in [35], which supports bit-wise AND operation, while above encryption scheme supports integer addition.

B Recommendation

In the recommendation phase (Figure 7), the user i, who has uploaded at least one rating, calls for her estimated ratings for items which have been rated by some users, i.e., $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$ for $j \in \mathcal{M}_J$. Since for $j \notin \mathcal{M}_J$ the RecSys computes $\mathbf{v}_j$ without any rating information on those items, $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$ does not reflect on the result of collaborative filtering.

In brief, the RecSys obtains encryptions of $\mathbf{u}_i$'s and $\mathbf{v}_j$'s from $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$ with the help of the CSP. Then the RecSys computes $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$ for $j \in \mathcal{M}_J$ in the HE encrypted form. Noising them, the user i requires the CSP to decrypt them and transfer to the user i. At the same time the RecSys sends the mask vectors to the user i with her public key of PE so that the user i can remove masks and obtain $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$ for $j \in \mathcal{M}_J$ in the plain form. The detailed procedures are as follows:

		RecSys	CSP	Communication Cost
RC	HE.enc	-	$2dM/L$	$\frac{5dM}{L}\lvert\mathsf{HE}\rvert + M\lvert\mathsf{PE}\rvert$
	HE.dec	-	$3dM/L$	
	HE.mul	dM/L	-	
	PE.enc	M	M	

Table 2: Computation and Communication Complexities for the Recommendation Phase

1. For $j \in \mathcal{M}_J$, the ResSys samples a mask vector $\varpi_u, \varpi_v \xleftarrow{\$} \mathcal{D}(\mathcal{M}, \mathcal{B})$ and computes $\mathsf{HE}(\hat{\mathbf{U}}(t)+\varpi_u)$ and $\mathsf{HE}(\hat{\mathbf{V}}(t)+\varpi_v)$. The RecSys sends all computed ciphertexts to the CSP.
2. The CSP decrypts all ciphertexts. It then computes and sends the RecSys the following ciphertexts:
   (a) $\mathsf{HE}(\|_{j\in\mathcal{M}_J} (\mathbf{u}_i + \zeta_i))$, where $\mathbf{u}_i + \zeta_i$ is concatenated m times and ζ_i is a mask vector of dimension d coming from the $\mathbf{u}_i$ part of ϖ_u.
   (b) $\mathsf{HE}(\|_{j\in\mathcal{M}_J} (\mathbf{v}_j + \xi_j))$, where ξ_j 's are mask vectors of dimension d coming from vectors η_j's and ϖ_v.

   Note that the CSP knows where $\mathbf{u}_i + \zeta_i$ from $\hat{\mathbf{U}}(t) + \varpi_u$ and how to concatenate vectors in right order from the knowledge of $\mathcal{M}$.
3. Given ciphertexts, the RecSys removes masks to obtain $\mathsf{HE}(\|_{j\in\mathcal{M}_J} \mathbf{u}_i)$ and $\mathsf{HE}(\|_{j\in\mathcal{M}_J} \mathbf{v}_j)$. The RecSys generates dm-dimensional random mask vector ψ and computes
$$\mathsf{HE}(\|_{j\in\mathcal{M}_J} \mathbf{u}_i \times_{\mathsf{c}} \|_{j\in\mathcal{M}_J} \mathbf{v}_j + \psi).$$
   The RecSys sends ciphertexts to the CSP. Let $\mathsf{pk_i}$ be PE-public key of the user i. Then the RecSys computes and sends $\mathsf{PE}(pk_i, \psi)$ to the user i.
4. The CSP decrypts given ciphertexts. Then it computes
$$\|_{j\in\mathcal{M}_J} \langle \mathbf{u}_i, \mathbf{v}_j\rangle + \psi',$$
   where ψ' comes from ψ. This can be done by performing sum for every d-dimensional vectors of plaintexts in a row. At last the CSP sends
$$\mathsf{PE}(pk_i, \|_{j\in\mathcal{M}_J} \langle \mathbf{u}_i, \mathbf{v}_j\rangle + \psi')$$
   to the user i.
5. The user i computes ψ' from ψ. Finally the user i obtains $\|_{j\in\mathcal{M}_J} \langle \mathbf{u}_i, \mathbf{v}_j\rangle$.

The computation and communication complexities of the recommendation phase are presented in Table 2. To transfer recommendation results, the RecSys and CSP perform PE-encryption M times and transfer M PE-ciphertexts. This number can be reduced further, that is, we do not take optimization techniques for PE (eg. message packing) into account. We denote by |PE| the size of PE-ciphertexts.

C Matrix Factorization

The full description of our matrix factorization protocol in Section 5 is presented in Figure 8.

On inputs r, HE-encryption of $\{\mathbf{U}(t-1), \hat{\mathbf{U}}(t-1), \mathbf{V}(t-1), \hat{\mathbf{V}}(t-1)\}$, the RecSys and CSP jointly and securely compute HE-encryption of $\{\mathbf{U}(t), \hat{\mathbf{U}}(t), \mathbf{V}(t), \hat{\mathbf{V}}(t)\}$. At the end of the below procedure, the RecSys confirms the stopping criteria and decides to continue the next iteration.

1. The RecSys determines parameters λ, μ, and γ within the ranges defined in the setup phase. It computes
$$\mathsf{HE}(\mathbf{U}(t-1)) \times \mathsf{HE}(\mathbf{V}(t-1)) - 2^{\alpha} \cdot \mathsf{HE}(\mathbf{r}),$$
where a positive integer α is used to match fixed point representation of entries in $\mathbf{r}$ to those in $\mathbf{U}(t-1) \times_{\mathsf{c}} \mathbf{V}(t-1)$ (See Section 4.3 for details). Let $\mathbf{R}'(t-1) = \mathbf{U}(t-1) \times_{\mathsf{c}} \mathbf{V}(t-1) - 2^{\alpha}\mathbf{r}$.
2. The RecSys samples $\epsilon(t-1) \xleftarrow{\$} \mathcal{D}(\mathcal{M}, \mathcal{B})$ and stores it. The RecSys then computes
$$\mathsf{HE}(\mathbf{U}(t-1)) \times \mathsf{HE}(\mathbf{V}(t-1)) - 2^{\alpha} \cdot \mathsf{HE}(\mathbf{r}) + \mathsf{HE}(\epsilon(t-1)),$$
and sends HE-ciphertexts to the CSP.
3. The CSP decrypts given HE-ciphertexts from the RecSys. The CSP then performs significant arithmetic over it, which yields
$$\|_{(i,j)\in\mathcal{M}} \{\overline{\mathbf{u}_i(t-1) \times_{\mathsf{c}} \mathbf{v}_j(t-1) - (r_{ij}, 0, \ldots, 0) + \epsilon^{(i,j)}(t-1)}\}.$$
4. The CSP computes $R_{ij} = \mathrm{sum}(\overline{\mathbf{u}_i(t-1) \times_{\mathsf{c}} \mathbf{v}_j(t-1) - (r_{ij}, 0, \ldots, 0) + \epsilon^{(i,j)}(t-1)})$'s for $(i,j) \in \mathcal{M}$, and sets $\mathbf{R}''(t-1) =\|_{(i,j)\in\mathcal{M}} (R_{ij}, \ldots, R_{ij})$, where each vector in concatenation is d-dimensional. At last the CSP computes $\mathsf{HE}(\mathbf{R}''(t-1))$, and sends them to the RecSys.
5. The RecSys computes $\epsilon''^{(i,j)}(t-1) = \mathrm{sum}(\overline{\epsilon^{(i,j)}(t-1)})$, sets the concatenation of dimension d vectors $\epsilon''(t-1) =\|_{(i,j)\in\mathcal{M}} (\epsilon''^{(i,j)}(t-1), \ldots, \epsilon''^{(i,j)}(t-1))$. It then computes $\mathsf{HE}(\mathbf{R}''(t-1)) - \mathsf{HE}(\epsilon''(t-1))$, which yields a HE-encryption of
$$\mathbf{R}(t-1) =\|_{(i,j)\in\mathcal{M}} (\overline{\langle \mathbf{u}_i(t-1), \mathbf{v}_j(t-1)\rangle - r_{ij}}, \ldots, \overline{\langle \mathbf{u}_i(t-1), \mathbf{v}_j(t-1)\rangle - r_{ij}}),$$
where each vector in concatenation is d-dimensional.
6. The RecSys computes HE-encryptions of
$$\nabla'_{\mathbf{U}}(t) = \mathbf{V}(t-1) \times_{\mathsf{c}} \mathbf{R}(t-1) + 2^{\alpha}\lambda\hat{\mathbf{U}}(t-1), \quad \nabla'_{\mathbf{V}}(t) = \mathbf{U}(t-1) \times_{\mathsf{c}} \mathbf{R}(t-1) + 2^{\alpha}\mu\hat{\mathbf{V}}(t-1).$$
It then computes HE-encryptions of
$$\mathbf{U}'(t) = 2^{\alpha+\beta} \cdot \hat{\mathbf{U}}(t-1) - \gamma 2^{\beta} \cdot \nabla'_{\mathbf{U}}(t), \quad \mathbf{V}'(t) = 2^{\alpha+\beta} \cdot \hat{\mathbf{V}}(t-1) - \gamma 2^{\beta} \cdot \nabla'_{\mathbf{V}}(t).$$
7. The RecSys samples $\delta_{\mathbf{U}}(t), \delta_{\mathbf{V}}(t), \theta_{\mathbf{U}}(t), \theta_{\mathbf{V}}(t) \xleftarrow{\$} \mathcal{D}(\mathcal{M}, \mathcal{B})$. It then computes and sends the CSP
$$\{\mathsf{HE}(\mathbf{U}'(t) + \delta_{\mathbf{U}}(t)), \mathsf{HE}(\mathbf{V}'(t) + \delta_{\mathbf{V}}(t))\}, \quad \{\mathsf{HE}(\nabla'_{\mathbf{U}}(t) + \theta_{\mathbf{U}}(t)), \mathsf{HE}(\nabla'_{\mathbf{V}}(t) + \theta_{\mathbf{V}}(t))\}.$$
8. After decrypting HE-ciphertexts the CSP performs fixed point arithmetic on them to get
$$\overline{\mathbf{U}'(t) + \delta_{\mathbf{U}}(t)},\ \overline{\mathbf{V}'(t) + \delta_{\mathbf{V}}(t)},\ \overline{\nabla'_{\mathbf{U}}(t) + \theta_{\mathbf{U}}(t)},\ \overline{\nabla'_{\mathbf{V}}(t) + \theta_{\mathbf{V}}(t)}.$$
It then computes $\mathbf{U}''(t) = \mathsf{rec}(\mathsf{agg}_u(\overline{\mathbf{U}'(t) + \delta_{\mathbf{U}}(t)})$ and $\hat{\mathbf{U}}''(t)$ in turn. Similarly it computes $\mathbf{V}''(t) = \mathsf{rec}(\mathsf{agg}_v(\overline{\mathbf{V}(t) + \delta_{\mathbf{V}}(t)})$ and $\hat{\mathbf{V}}''(t)$.
9. The CSP computes
$$\nabla''_{\mathbf{U}}(t) = \mathsf{agg}_u(\overline{\nabla'_{\mathbf{U}}(t) + \theta_{\mathbf{U}}(t)}), \quad \nabla''_{\mathbf{V}}(t) = \mathsf{agg}_v(\overline{\nabla'_{\mathbf{V}}(t) + \theta_{\mathbf{V}}(t)}).$$
Note that when mask vectors in $\mathbf{U}''(t)$, $\hat{\mathbf{U}}''(t)$, $\mathbf{V}''(t)$, $\hat{\mathbf{V}}''(t)$, $\nabla''_{\mathbf{U}}(t)$, and $\nabla''_{\mathbf{V}}(t)$ are removed, those vectors equal to $\mathbf{U}(t)$, $\hat{\mathbf{U}}(t)$, $\mathbf{V}(t)$, $\hat{\mathbf{V}}(t)$, $\nabla_{\mathbf{U}}(t)$, and $\nabla_{\mathbf{V}}(t)$, respectively. At last it sends HE-encryption of $\mathbf{U}''(t)$, $\hat{\mathbf{U}}''(t)$, $\mathbf{V}''(t)$, $\hat{\mathbf{V}}''(t)$, $\nabla''_{\mathbf{U}}(t)$, and $\nabla''_{\mathbf{V}}(t)$
10. From the knowledge of $\delta_{\mathbf{U}}(t)$, $\delta_{\mathbf{V}}(t)$, $\theta_{\mathbf{U}}(t)$, and $\theta_{\mathbf{V}}(t)$, the RecSys removes aggregated mask vectors in given HE-ciphertexts to get HE-encryptions of desired vectors $\mathbf{U}(t)$, $\hat{\mathbf{U}}(t)$, $\mathbf{V}(t)$, $\hat{\mathbf{V}}(t)$, $\nabla_{\mathbf{U}}(t)$, and $\nabla_{\mathbf{V}}(t)$.
11. The RecSys sets threshold values for the user and item profiles, say ω_u and ω_v. It generates mask vectors $\mathbf{w}_u$ and $\mathbf{w}_v$, whose dimensions are equal to those of $\nabla_{\mathbf{U}}(t)$ and $\nabla_{\mathbf{V}}(t)$, respectively. It computes and sends the CSP
$$\mathsf{HE}(\nabla_{\mathbf{U}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{U}}(t) + \mathbf{w}_u), \quad \mathsf{HE}(\nabla_{\mathbf{V}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{V}}(t) + \mathbf{w}_v)$$
together with $s_u = \omega_u + \mathrm{sum}(\mathbf{w}_u)$ and $s_v = \omega_v + \mathrm{sum}(\mathbf{w}_v)$.
12. The CSP computes
$$s'_u = \mathrm{sum}(\nabla_{\mathbf{U}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{U}}(t) + \mathbf{w}_u), \quad s'_v = \mathrm{sum}(\nabla_{\mathbf{V}}(t) \times_{\mathsf{c}} \nabla_{\mathbf{V}}(t) + \mathbf{w}_v).$$
Finally, it returns a boolean vector $(s_u - s'_u \geq 0?, s_v - s'_v \geq 0?)$ to the RecSys.

Figure 8: Matrix Factorization