Efficient Privacy-Preserving Collaborative Filtering Based on the Weighted Slope One Predictor[*]

Anirban Basu
Graduate School of Engineering, Tokai University,
2-3-23 Takanawa, Minato-ku, Tokyo 108-8619, Japan
abasu@cs.dm.u-tokai.ac.jp

Jaideep Vaidya
MSIS Department, Rutgers, The State University of New Jersey
1, Washington Park, Newark, New Jersey, 07102-1897, USA
jsvaidya@business.rutgers.edu

Hiroaki Kikuchi
Graduate School of Engineering, Tokai University,
1117, Kitakaname, Hiratsuka, Kanagawa, 259-1292, Japan
kikn@tokai.ac.jp

Abstract

Rating-based collaborative filtering (CF) predicts the rating that a user will give to an item, derived from the ratings of other items given by other users. Such CF schemes utilise either user neighbourhoods (i.e. user-based CF) or item neighbourhoods (i.e. item-based CF). Lemire and MacLachlan [19] proposed three related schemes for an item-based CF with predictors of the form $f(x) = x + b$, hence the name "slope one". Slope One predictors have been shown to be accurate on large datasets. They also have several other desirable properties such as being updatable on the fly, efficient to compute, and work even with sparse input. In this paper, we present a privacy-preserving item-based CF scheme through the use of an additively homomorphic public-key cryptosystem on the weighted Slope One predictor; and show its applicability on both horizontal and vertical partitions, and include a discussion on arbitrary partitions as well. We present an evaluation of our proposed scheme in terms of communication and computation complexity, performance of cryptographic primitives and performance of a single-partition, single machine implementation in 64-bit Java.

Keywords: Privacy Preserving, Slope One, Collaborative Filtering

1 Introduction

The information available over the World Wide Web (e.g. from social networks, e-commerce catalogs, amongst others) has reached insurmountable levels over the last two decades, and users are thus accosted with the problem of information overload [30]. Recently, more web-based services offered through cloud computing have only exacerbated the problem. User-tailored *recommendation systems* have been seen as the rescue that are expected to help users weed out the unnecessary information from the essential ones.

Most automated recommendation systems are generally classified according to two main techniques: *profile-based* and *collaborative filtering* (CF). The former involves relevant details about users (i.e. information that relate to their tastes), which are collected in order to match the items to be recommended to

[*]This paper is an extended journal version of the workshop paper [4] published in the proceedings of TP-DIS 2011, co-located with the IFIPTM 2011 in København, Denmark.

them. In contrast, prediction through CF results from the recorded preferences of the community. While profile-based recommendation for a user with rich profile information can be thorough, CF is fairly accurate, without the need for the user's preferential history. CF has, thus, positioned itself as one of the predominant means of generating recommendations.

Based on filtering techniques, CF is broadly classified into: *memory-based* or *neighbourhood-based* and *model-based*. In *memory-based* approaches, recommendations are developed from user or item neighbourhoods, based on some sort of proximity (or deviation) measures between opinions of the users, or the ratings of the items, e.g. cosine similarity, Euclidean distance and various statistical correlation coefficients. Memory-based CF can also be distinguished into: *user-based* and *item-based*. In the former, CF is performed using neighbourhood between users computed from the ratings provided by the different users. The latter is item-based where prediction is obtained using item neighbourhoods, i.e. proximity (or deviation) of ratings between various items.

Model-based approaches, in contrast, are sometimes more applicable on large datasets for which some memory-based approaches do not scale well. In model-based approaches, the original user-item ratings dataset is used to *train* a compact model, which is then used for prediction. The model is developed by methods borrowed from artificial intelligence, such as Bayesian classification, latent classes and neural networks; or, from linear algebra, e.g. singular value decomposition (SVD), latent semantic indexing (LSI) and principal component analysis (PCA). Model-based algorithms are usually fast to query but relatively slow to update.

Collaborative filtering based approaches attain better accuracy with the availability of more data. Sometimes, it may be possible to perform cross domain (e.g. between two sites) recommendations, if the corresponding data can be utilised to relate contextual information (e.g. a person with a strong interest in horror movies may also rate certain Halloween products highly). However, sharing user-item preferential data for use in CF poses significant privacy and security challenges. Competing organisations, e.g. Netflix and Blockbuster may not wish to share specific user information, even though both may benefit from such sharing. Users themselves might not want detailed information about their ratings and buying habits known to any single organisation. To overcome this, there has been active recent work in privacy-preserving collaborative filtering (PPCF) that enable CF without leaking private information. However, in CF, achieving accuracy and preserving privacy are orthogonal problems.

The two main directions of research in privacy-preserving collaborative filtering are: *encryption-based* and *randomisation-based*. In encryption-based techniques, prior to sharing individual user-item ratings data are encrypted using cryptosystems that support homomorphic properties. Therefore, third party collaborative filtering servers can identify a user but cannot see their data. In randomisation-based privacy preserving techniques, the ratings data is randomised either through random data swapping or data perturbation or anonymisation. In this technique, the third party CF servers can sometimes identify the user and read the data but the data is not the real data due to perturbation; in other cases, the third party is unable to accurately link the user from the available data (e.g. in k-anonymity [31], l-diversity [22]). We believe that the existing schemes are either impractical from the efficiency standpoint (especially, from the perspective of updating) or from the security standpoint. To improve on this, in this paper, we propose a privacy-preserving (with encryption strategy) item-based collaborative filtering scheme extended from the well-known weighted Slope One predictor [19].

1.1 Our contribution

The contributions of this paper can be summarised as follows: (i) this is an extension of our earlier work [4], in which the previously proposed privacy-preserving collaborative filtering scheme using the weighted Slope One predictor is extended to arbitrary dataset partitions; (ii) our proposal retains the high level of accuracy of Slope One predictors while also providing a high level of privacy – since we are using encryption and security primitives, we can prove that our protocols do not leak any information other than the result; (iii) our proposal has relatively low computation and communication complexity. (iv) this extended paper also contains implementation results using a single machine, and single dataset partition.

The rest of the paper is organised as follows: we briefly present background and overview of preliminaries including the key related work in this area in §2. We summarise our problem statements in §3. In §4, we propose a PPCF scheme based on the weighted Slope One predictor and apply it on horizontal and vertical partitions. In §5, we present evaluations and implementation results followed by a conclusion and promising future directions in §6.

2 Background and Preliminaries

2.1 The weighted Slope One predictor

Lemire and MacLachlan proposed [19] a CF scheme based on predictors of the form $f(x) = x + b$, hence the name "slope one". However, to our knowledge, no one has applied privacy preserving techniques on slope one based CF. Before delving further into PPCF, we present a brief overview of the Slope One predictors. Conforming with realistic datasets, in the following example, we will use the discrete integral range of ratings $[1-5]$ with "0" or "-" or "?" representing absence of ratings. Table 2.1 shows a simple user-item ratings matrix of users rating airlines companies.

The simplest Slope One prediction of rating for any user for an item i_1 given the user's rating for i_2 (i.e. r_{i_2}), is of the form $r_{i_1} = \overline{\delta_{i_1,i_2}} + r_{i_2}$ where $\overline{\delta_{i_1,i_2}}$ is the average deviation of the ratings of item i_1 from those of item i_2 while r_{i_2} is the rating the user has given to item i_2. The average deviation of ratings between a pair of items is calculated using only those ratings where both items have been rated by the same user.

Table 2.1: A simple three users, three items rating matrix.

	British Airways	Emirates	Cathay Pacific
Kikuchi Hiroaki	2	4	4
Jaideep Vaidya	2	5	4
Basu Anirban	1	?	4

Using the *unweighted* Slope One predictor, we derive the missing rating as:

$$? = \frac{(\frac{(4-2)+(5-2)}{2} + 1) + (\frac{(4-4)+(5-4)}{2} + 4)}{2} = 4.0$$

The unweighted scheme estimates a missing rating using the average deviation of ratings between pairs of items with respect to their *cardinalities*. Slope One CF can be evaluated in two stages: pre-computation and prediction of ratings. In the pre-computation stage, the average deviations of ratings from item a to item b is given as:

$$\overline{\delta_{a,b}} = \frac{\Delta_{a,b}}{\phi_{a,b}} = \frac{\sum_i \delta_{i,a,b}}{\phi_{a,b}} = \frac{\sum_i (r_{i,a} - r_{i,b})}{\phi_{a,b}} \tag{2.1}$$

where $\phi_{a,b}$ is the count of the users who have rated both items while $\delta_{i,a,b} = r_{i,a} - r_{i,b}$ is the deviation of the rating of item a from that of item b both given by user i.

In the prediction stage, the rating for user u and item x using the *weighted* Slope One is predicted as:

$$r_{u,x} = \frac{\sum_{a|a\neq x}(\overline{\delta_{x,a}} + r_{u,a})\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}} = \frac{\sum_{a|a\neq x}(\Delta_{x,a} + r_{u,a}\phi_{x,a})}{\sum_{a|a\neq x}\phi_{x,a}}. \tag{2.2}$$

Thus, we can precompute the *difference (or deviation) matrix*[1] $\Delta = \{\Delta_{a,b}\}$ and the *cardinality matrix* $\phi = \{\phi_{a,b}\}$. Note that for space efficiency, we only need to calculate the upper triangulars of those matrices because the lower triangulars can be easily derived from the upper ones, and the leading diagonals are irrelevant. The weighted Slope One has been found to be efficient, e.g. achieving a mean absolute error (MAE) rate close to 0.7 on the MovieLens 100K dataset[2], which is better than CF schemes using cosine similarity or the Singular Value Decomposition, using a reference implementation in Apache Mahout[3].

2.2 Related work

In recent years, privacy has attracted a lot of attention. There has been extensive work in privacy-preserving data mining, most of which is either based on randomization or on cryptographic techniques. In the randomization approach[3], "noise" is added to the data before the data analysis process. Techniques are then used to remove the noise from the data mining results. Several privacy-preserving data mining algorithms have since been proposed [2, 12, 29] using this approach. However, since the original data is still accessible (even though it is perturbed), estimates of the original values can be obtained using noise removal techniques, giving rise to debate about the security properties of such algorithms![18, 16]. The alternative approach to protecting privacy of distributed sources using cryptographic techniques was first applied in the area of data mining for the construction of decision trees by Lindell and Pinkas [20, 21]. This work falls under the framework of secure multiparty computation [32, 13], achieving "perfect" privacy, i.e. nothing is learned that could not be deduced from one's own data and the resulting tree. The key insight was to trade off computation and communication cost for accuracy, improving efficiency over the generic secure multiparty computation method. However, the proposed solution is still too inefficient for practical usage. Therefore, much of the following work has concentrated on creating efficient techniques for specific problems. A general survey of privacy preserving data mining in presented in [1].

There are a number of existing works on privacy-preserving collaborative filtering (PPCF). One of the earliest such efforts is due to [7] which uses a partial Singular Value Decomposition (SVD) model and homomorphic encryption to devise a multi-party PPCF scheme. Canny [8] also proposes a new solution based on a probabilistic factor analysis model, that can handle missing data, and provides privacy through a peer-to-peer protocol. Polat and Du have also investigated this problem from several perspectives. In [24] a randomised perturbation technique is proposed to protect individual privacy while still producing accurate recommendations results; [25] presents a privacy-preserving protocol for collaborative filtering over vertically partitioned data, while [26] presents a scheme for providing top-N recommendations over horizontally partitioned data. Also, [27] presents a randomisation based SVD approach, while [28] enables recommendations via item-based algorithms using randomized response techniques. Berkovsky et. al. [5] propose a decentralized method that stores most of the data only on the client side and transfers very limited data over the network to mitigate the privacy concerns of collaborative filtering. In [6], they present a decentralized distributed storage of user data combined with data modification techniques to mitigate privacy issues. Cissée and Albayrak [9] develop a method for privacy-preserving recommender systems based on multi-agent technology which enables applications to generate recommendations via

[1] Note that we do not need to compute average differences according to equation 2.2.

[2] http://www.grouplens.org/node/73

[3] http://mahout.apache.org/

various filtering techniques while preserving the privacy of all participants. In [17], the authors propose a naïve Bayesian classifier based CF over a P2P topology where the users protect the privacy of their data using masking, which is comparable to randomisation. Another homomorphic encryption based SVD scheme has been proposed in [15] but the authors also describe that their scheme does not scale well for realistic datasets; Gong et. al.[14] presents a new collaborative filtering technique based on randomized perturbation and secure multiparty computation. However, it is not as efficient as our technique, and also does not guarantee completely accurate results. Our work follows the cryptographic approach, is efficient, and works for all types of data partitions.

2.3 An additively homomorphic public-key cryptosystem – Paillier

Paillier introduced a public-key cryptosystem [23] with additively homomorphic properties. Denoting encryption and decryption functions as $\mathscr{E}()$ and $\mathscr{D}()$ respectively, the encryption of the sum of two plaintext messages m_1 and m_2 is the modular product of their individual ciphertexts:

$$\mathscr{E}(m_1 + m_2) = \mathscr{E}(m_1) \cdot \mathscr{E}(m_2) \tag{2.3}$$

and, the encryption of the product of one plaintext messages m_1 and a plaintext integer multiplicand π is the modular exponentiation of the ciphertext of m_1 with π as the exponent:

$$\mathscr{E}(m_1 \cdot \pi) = \mathscr{E}(m_1)^{\pi}. \tag{2.4}$$

Paillier cryptosystem in described in three steps: *key generation* (algorithm 2.1), *encryption* (algorithm 2.2) and *decryption* (algorithm 2.3).

Algorithm 2.1 Paillier cryptosystem key generation.

```
1: Generate two large prime numbers $p$ and $q$, each with half the specified modulus bit length for the cryptosystem.
Ensure: $gcd(pq,(p-1)(q-1)) = 1$ and $p \neq q$.
2: Modulus $n = pq$.
3: Pre-compute $n^2$.
4: Compute $\lambda = lcm(p-1,q-1) = \frac{(p-1)(q-1)}{gcd(p-1,q-1)}$.
5: $g \leftarrow (1+n)$. {Optimised here but originally: select random $g \in Z^*_{n^2}$ such that $n$ divides the order of $g$.}
Ensure: $gcd(L(g^{\lambda} \mod n^2), n) = 1$ where $L(u) = \frac{u-1}{n}$. {Optimisation: $g^{\lambda} \mod n^2 = (1+n\lambda) \mod n^2$.}
6: Pre-compute the modular multiplicative inverse $\mu = L(g^{\lambda} \mod n^2)^{-1} \mod n$.
7: return Public key: $(n, n^2, g)$ and Private key: $(\lambda, \mu)$.
```

Algorithm 2.2 Paillier encryption algorithm.

```
Require: Plaintext $m \in Z_n$.
1: Choose random $r \in Z^*_n$.
2: return Ciphertext $c \leftarrow (1+mn)\, r^n \mod n^2$. {Optimised here but originally: $c \leftarrow g^m r^n \mod n^2$.}
```

2.3.1 Generalised threshold variant – the Damgärd-Jurik cryptosystem

The Damgärd-Jurik cryptosystem [11] proposes a generalisation and a threshold variant of the Paillier cryptosystem using modulo n^{s+1} computations for any natural number $s \geq 1$ (with $s = 1$ for Paillier).

Algorithm 2.3 Paillier decryption algorithm.

```
Require: Ciphertext $c \in Z^*_{n^2}$.
1: return Plaintext $m \leftarrow L(c^\lambda \mod n^2)\mu \mod n$.
```

It allows for private key sharing between k parties. The ciphertext can be decrypted only after combining all k partial decryptions. In terms of performance, it is slightly slower than Paillier. Security of this cryptosystem is the same as that of the Paillier cryptosystem based on the *decisional composite residuosity assumption*. Note that in our proposed scheme, we assume a *semi-honest* model for the participating sites. Hence, we do not require Damgärd-Jurik zero-knowledge proofs (ZKPs) for the various cryptographic operations from the participating sites.

3 Problem statement

We first define our problem statement in generic terms, and then make it more specific.

*[*Privacy-Preserving Slope One Predictors] Given a dataset consisting of m users $u_1, \ldots, u_m$ and n items $it_1, \ldots, it_n$ distributed between a set of k sites, $\mathscr{S}_0, \ldots, \mathscr{S}_{k-1}$ in some fashion, build the weighted Slope One predictor for each item without leaking the privacy of the data owned by site $\mathscr{S}_i$ to any other site $\mathscr{S}_j$.

Depending on the situation at hand, the collaborative protocol to build the Slope One predictor might result in all of the sites holding the predictor or only some master site holding it, which is still trusted to see the raw data. While data may be arbitrarily partitioned between the sites, in general, it is much more likely that the data is partitioned in a particular fashion in real life. Therefore, we discuss two specific partitions – horizontal partitioning and vertical partitioning, and present a specific problem statement for each case as well.

3.1 Horizontal Partitioning of Data

A perfect horizontal partition of a user-item ratings dataset is such that multiple sites (or organisations) contain completely disjoint sets of users but the same set of items. In figure 3.1, a partition of a multiple user, two-items dataset is shown where the data is present in three sites.

In this case, while each site can independently build the Slope One predictor for each item, the predictors are likely to much more accurate when built over the entire data set.

*[*Privacy-Preserving Slope One Predictors for Horizontally Partitioned Data] Given a set of k sites, $\mathscr{S}_0, \ldots, \mathscr{S}_{k-1}$, where each site $\mathscr{S}_i$ owns the data about m_i users $u_{i1}, \ldots, u_{im_i}$, such that $\sum_i m_i = m$ and all sites collect information about the same n items $it_1, \ldots, it_n$, build the weighted Slope One predictor for each item without leaking the privacy of the data owned by site $\mathscr{S}_i$ to any other site $\mathscr{S}_j$.

3.2 Vertical Partitioning of Data

A perfect vertical partition is one in which the multiple sites contain the same set of users but completely disjoint sets of items. In figure 3.2, a partition of a multi-user multi-item dataset is shown where there are three item sets across three sites.

In this case, while each party can build a Slope One predictor for its items based on the other items it possesses, the global Slope One predictor built using all of the other items is likely to be significantly more accurate. Therefore, the parties must collaboratively build the predictor.

*[*Privacy-Preserving Slope One Predictors for Vertically Partitioned Data] Given a set of k sites, $\mathscr{S}_0, \ldots, \mathscr{S}_{k-1}$, where each site $\mathscr{S}_i$ owns the data about n_i items $it_{i1}, \ldots, it_{nm_i}$, such that $\sum_i n_i = n$ and all

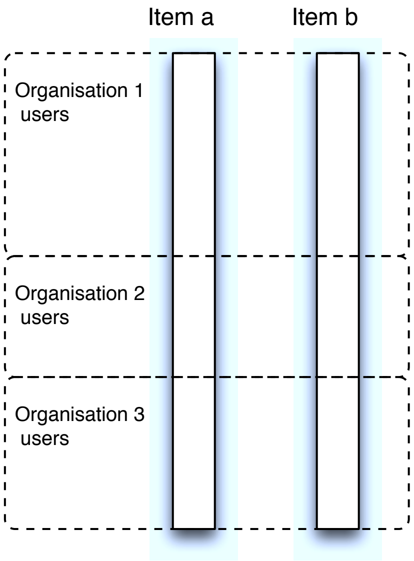


Figure 3.1: A visualisation of a horizontal partition.

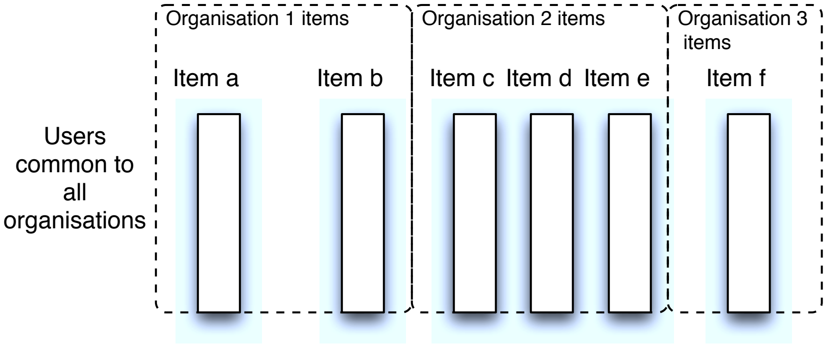


Figure 3.2: A visualisation of a vertical partition.

sites collect information about the same m users $u_1, \ldots, u_m$, build the Slope One predictor for each item without leaking the privacy of the data owned by site $\mathscr{S}_i$ to any other site $\mathscr{S}_j$.

3.3 Arbitrary Partitioning of Data

As discussed above, in arbitrary partitioning neither the users nor the items are necessarily split in disjoint partitions. Thus, any site may know an arbitrary set of ratings for an arbitrary set of users. However, the only assumption is that the global set of user-rating pairs is both unique and complete – every user-rating is collected by exactly one site (i.e., all ratings for all users are collected and are not duplicated). In figure 3.3, a partition of a multi-user multi-item dataset is shown where both users and ratings are split across three sites.

Note that in all of the above cases, since the Slope One predictor can be easily built (see equation 2.2) given the deviation matrix and the cardinality matrix, we limit our attention to computing these two matrices in a privacy-preserving fashion.

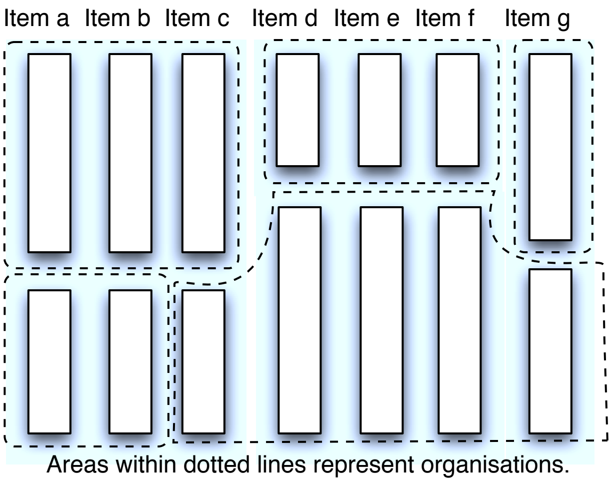


Figure 3.3: A visualisation of an arbitrary partition.

4 Privacy-preserving Slope One

Below, we show that prediction of a rating, given in equation 2.2, can be derived from encrypted deviations and plaintext cardinalities. Encrypted deviations can be completely decrypted by combining the partial decryptions sites by k sites using the Damgård-Jurik cryptosystem[4].

$$\begin{aligned} \sum_{a|a \neq x} (\Delta_{x,a} + r_{u,a}\phi_{x,a}) &= \mathscr{D}(\prod_{a|a \neq x} (\mathscr{E}(\Delta_{x,a})(\mathscr{E}(r_{u,a})^{\phi_{x,a}}))) && (4.1) \\ \implies r_{u,x} &= \frac{\mathscr{D}(\prod_{a|a \neq x}(\mathscr{E}(\Delta_{x,a})(\mathscr{E}(r_{u,a})^{\phi_{x,a}})))}{\sum_{a|a \neq x} \phi_{x,a}}. && (4.2) \end{aligned}$$

We now look at how to compute the deviation (Δ) and cardinality (ϕ) matrices when the data is distributed. Note that although local cardinalities at each site are encrypted, the global cardinalities should be decrypted to fit into the prediction equation 4.2. Also, the prediction equation contains a number of $\mathscr{E}(r_{u,a})$ values, which are provided encrypted in a query to obtain the prediction.

4.1 Horizontal Partition

If we denote the deviation matrix for all item pairs as Δ and the cardinality matrix as ϕ, and denote organisational sites as $\mathscr{S}_{i|i=0 \ldots k}$ then site $\mathscr{S}_i$ owns $\Delta^{\mathscr{S}_i}$ and $\phi^{\mathscr{S}_i}$ respectively. An element in the deviation matrix between items "a" and "b" is given as $\Delta_{a,b}^{\mathscr{S}_i}$ owned by $\mathscr{S}_i$, if any. Since the user sets are completely disjoint between the organisations, the following two equations can be easily inferred:

$$\Delta = \sum_i \Delta^{\mathscr{S}_i} \implies \text{any } \Delta_{a,b} = \mathscr{D}(\prod_i \mathscr{E}(\Delta_{a,b}^{\mathscr{S}_i})) \text{ and} \tag{4.3}$$

$$\phi = \sum_i \phi^{\mathscr{S}_i} \implies \text{any } \phi_{a,b} = \mathscr{D}(\prod_i \mathscr{E}(\phi_{a,b}^{\mathscr{S}_i})). \tag{4.4}$$

[4] In the Paillier and the Damgård-Jurik cryptosystems, encryption of a negative integer can be calculated by using its modular additive inverse, i.e. $E(-x) = E(n-x)$, so we can treat all $x > \frac{n}{2}$ to be negative.

4.1.1 Initial matrix aggregation

Each site independently calculates its own deviation and cardinality matrices ranging over all pairs of items from the user set it has[5]. To aggregate the matrices, any site $\mathscr{S}_i$ should start a k-cycle (k being the number of collaborating sites) by sending its neighbour, $\mathscr{S}_{i+1}$, the upper triangulars of $\mathscr{E}(\Delta^{\mathscr{S}_i})$ and $\mathscr{E}(\phi^{\mathscr{S}_i})$. Site $\mathscr{S}_{i+1}$ can then aggregate these with its own deviation and its own cardinality matrices by homomorphic addition and pass the results on to its neighbour $\mathscr{S}_{i+2}$. Once the cycle finishes, i.e. the encrypted matrices reach $\mathscr{S}_i$ in a cycle, it can then forward the updated final encrypted matrices in one more k-cycle so that its neighbours can all maintain the same up-to-date matrix. This process is described in algorithm 4.1.

Algorithm 4.1 Initial matrix aggregation in the horizontal partitioning scheme.

```
Require: Each site $\mathscr{S}_i$ independently calculates its own deviation matrix $\Delta^{\mathscr{S}_i}$ and its own cardinality matrix $\phi^{\mathscr{S}_i}$.
for each site $\mathscr{S}_i$ in the $k$ sites do
    $\mathscr{S}_i$ sends $\mathscr{E}(\Delta^{\mathscr{S}_i})$ and $\mathscr{E}(\phi^{\mathscr{S}_i})$ to its neighbour $\mathscr{S}_{i+1}$.
    $\mathscr{S}_{i+1}$ computes homomorphic addition: $\mathscr{E}(\Delta^{\mathscr{S}_i})\mathscr{E}(\Delta^{\mathscr{S}_{i+1}})$ and $\mathscr{E}(\phi^{\mathscr{S}_i})\mathscr{E}(\phi^{\mathscr{S}_{i+1}})$; then sends this to its neighbour $\mathscr{S}_{i+2}$ until all sites are reached in this $k$-cycle.
end for
{In $k$ such forwarding operations, the complete encrypted matrices $\mathscr{E}(\Delta) = \prod_i \mathscr{E}(\Delta^{\mathscr{S}_i})$ and $\mathscr{E}(\phi) = \prod_i \mathscr{E}(\phi^{\mathscr{S}_i})$ will reach $\mathscr{S}_i$.}
for each site $\mathscr{S}_i$ in the $k$ sites do
    $\mathscr{S}_i$ forwards $\mathscr{E}(\Delta)$ and $\mathscr{E}(\phi)$ to $\mathscr{S}_{i+1}$ until all sites are reached.
end for
{In this cycle, every $\mathscr{S}_i$ gets $\mathscr{E}(\Delta)$ and $\mathscr{E}(\phi)$, where $\mathscr{E}(\Delta) = \mathscr{E}(\sum_i \Delta^{\mathscr{S}_i})$ and $\mathscr{E}(\phi) = \mathscr{E}(\sum_i \phi^{\mathscr{S}_i})$.}
```

4.1.2 Matrix update

Any changes in individual ratings will change the individual pairwise deviations (and also cardinalities if new users are added or old users are removed), which will trigger an update in a cycle akin to the matrix aggregation process. If $\Delta_{a,b}$ is changed to $\Delta^{'}_{a,b}$ at site $\mathscr{S}_i$ then it can pass it on encrypted to its neighbour $\mathscr{S}_{i+1}$ in a k-cycle (k being the number of collaborating servers), which in turn can pass it on to its neighbour, $\mathscr{S}_{i+2}$ so that each site can update its $\mathscr{E}(\Delta_{a,b})$ to $\mathscr{E}(\Delta^{'}_{a,b})$. The cardinality matrix elements can be updated in the same way. This update process is described in algorithm 4.2.

Algorithm 4.2 Matrix update in the horizontal partitioning scheme.

```
Require: $\Delta_{a,b}$ is changed to $\Delta^{'}_{a,b}$ at site $\mathscr{S}_i$ because of change of ratings for item "a" or item "b" or additional or removal of users at site $\mathscr{S}_i$. Alternatively, $\phi$ may have changed to $\phi^{'}$ at site $\mathscr{S}_i$.
for each site $\mathscr{S}_i$ in the $k$ sites do
    If the update is required, $\mathscr{S}_i$ forwards $\mathscr{E}(\Delta^{'}_{a,b})$ to $\mathscr{S}_{i+1}$ until all sites are reached.
    If the update is required, $\mathscr{S}_i$ forwards $\mathscr{E}(\phi^{'}_{a,b})$ to $\mathscr{S}_{i+1}$ until all sites are reached.
end for
```

We will have to decrypt the global cardinality matrix $\phi = \mathscr{D}(\prod_i \mathscr{E}(\phi^{\mathscr{S}_i})) = \sum_i \phi^{\mathscr{S}_i}$ before we can apply the prediction equation 4.2 but we can also use the secure sum protocol [10] to add up the local

[5] Recall the need to calculate only upper triangulars.

cardinality, i.e. $\phi^{\mathscr{S}_i}$ matrices without encrypting, which will reduce the computation cost but increase the communication cost.

4.2 Vertical Partition

Vertical partitions are not trivial as this involves expansions of the individual servers' deviation and cardinality matrices. This will involve sharing of ratings for each pair of items in which one item (in the pair) exists in one partition and the other item exists in another partition. Clearly, if both items are completely held by any one server, it is easy to calculate the average deviation. However, when both items are held by different servers, a privacy-preserving protocol is needed to compute the average deviation. This can be done as follows. Assume site $\mathscr{S}_x$ holds item a and site $\mathscr{S}_y$ holds item b. Now, $\mathscr{S}_x$ can compute $s_a = \sum_i r_{i,a}$ for all its users and $\mathscr{S}_y$ can compute $s_b = \sum_i r_{i,b}$. Note that $s_a - s_b = \Delta_{a,b} + overcount_a - overcount_b$, where $overcount_a = \sum_i r_{i,a}$, where $r_{i,b}$ does not exist. Similarly, $overcount_b = \sum_i r_{i,b}$, where $r_{i,a}$ does not exist. Since, s_a, s_b can be computed locally, given a way to compute $overcount_a$, $overcount_b$, $\Delta_{a,b}$ can be computed in a privacy-preserving fashion.

We now show how to compute $overcount_a$, $overcount_b$. Essentially, this can be done using a secure scalar product protocol. Basically, $\mathscr{S}_x$ forms the vector $\vec{xa}$, ($xa_i = r_{i,a}$, if $r_{i,a}$ exists, otherwise 0). Correspondingly $\mathscr{S}_y$ forms the boolean vector $\vec{ya}$, ($ya_i = 1$, if $r_{i,b}$ does *not* exist, otherwise 0). Now, $overcount_a = \vec{xa} \cdot \vec{ya}$. This can be seen as follows – the crucial part is the correct formulation of the boolean vector. Note that the boolean vector is essentially the inverse of the existence vector for item b. Thus, any rating of a without a corresponding rating of b will of necessity be now added in to the scalar product. This is exactly $overcount_a$.

Similarly, $\mathscr{S}_x$ creates the boolean vector $\vec{xb}$, ($xb_i = 1$, if $r_{i,a}$ does *not* exist, otherwise 0), while $\mathscr{S}_y$ forms the vector $\vec{yb}$, ($yb_i = r_{i,b}$, if $r_{i,b}$ exists, otherwise 0). Now, $overcount_b = \vec{xb} \cdot \vec{yb}$. The reasoning is exactly as before – any rating of b without a corresponding rating of a will now be added into the scalar product.

We can easily use the additively homomorphic cryptosystem to do the scalar product. Basically, $\mathscr{S}_x$ can encrypt $\vec{xa}$ and send it to $\mathscr{S}_y$ who can multiply the correct encryptions together to get the encrypted form of $overcount_a$. Similarly, $\mathscr{S}_y$ can encrypt $\vec{yb}$ and send it to $\mathscr{S}_x$ who can multiply the correct encryptions together to get the encrypted form of $overcount_b$. Deriving $\phi_{a,b}$ is similarly easy – both $\mathscr{S}_x$ and $\mathscr{S}_y$ simply encode the existence vectors for items a and b respectively. The scalar product of these two vectors is $\phi_{a,b}$. Note that typically, the secure scalar product is computed in split fashion. i.e., for $overcount_a$ the two sites get o_x and o_y respectively such that $o_x + o_y = overcount_a$. This is to ensure that the scalar product itself does not directly reveal any information. This works well for us since, when computing $\Delta_{a,b}$ both sites can locally sum up their shares and finally sum up the whole in encrypted form. For the sake of simplicity, in the algorithm, we just note that the scalar products are computed securely. The overall matrix aggregation process is described in algorithm 4.3.

4.2.1 Matrix update

As earlier, any changes in individual ratings will change the individual pairwise deviations (and also cardinalities if new users are added or old users are removed[6]). The addition of new users is easy to handle. We simply need to carry out the prior aggregration process only for the added new users – these can then be added in encrypted form to the existing $\Delta_{a,b}$ and $\phi_{a,b}$. Similarly, for deleted users, we can carry out the prior aggregation process only for the deleted users and then subtract those from the existing $\Delta_{a,b}$ and $\phi_{a,b}$. The case for updated users is a little more complicated. However, a simple solution is to

[6] Note that an user can only be added or deleted by all of the sites together to maintain perfect vertical partitions. We also leave the case of addition or deletion of items to future work.

Algorithm 4.3 Initial matrix aggregation in the vertical partitioning scheme.

```
1: {As before note that only upper triangulars of the deviation and relative occurence matrix needs to be
   computed as the lower triangulars can be easily deduced from the upper ones. The leading diagonal
   is irrelevant.}
2: for each cell (a,b) in the upper triangular matrix do
3:     if both item $a$ and item $b$ are held by the same party $\mathscr{S}_k$ then
4:         $\mathscr{S}_k$ computes $\Delta_{a,b}$ and $\phi_{a,b}$
5:     else
6:         Assume site $\mathscr{S}_x$ owns item $a$ while site $\mathscr{S}_y$ owns item $b$
7:         $\mathscr{S}_x$ computes $s_a = \sum_i r_{i,a}$
8:         $\mathscr{S}_y$ computes $s_b = \sum_i r_{i,b}$
9:         $\mathscr{S}_x$ creates the vector $\vec{xa}$, ($xa_i = r_{i,a}$, if $r_{i,a}$ exists, otherwise 0)
10:        $\mathscr{S}_x$ creates the boolean vector $\vec{xb}$, ($xb_i = 1$, if $r_{i,a}$ does *not* exist, otherwise 0)
11:        $\mathscr{S}_y$ creates the vector $\vec{yb}$, ($yb_i = r_{i,b}$, if $r_{i,b}$ exists, otherwise 0)
12:        $\mathscr{S}_y$ creates the boolean vector $\vec{ya}$, ($ya_i = 1$, if $r_{i,b}$ does *not* exist, otherwise 0)
13:        $\mathscr{S}_x$ and $\mathscr{S}_y$ securely compute $overcount_a = \vec{xa} \cdot \vec{ya}$
14:        $\mathscr{S}_x$ and $\mathscr{S}_y$ securely compute $overcount_b = \vec{xb} \cdot \vec{yb}$
15:        $\Delta_{a,b} = s_a - s_b - overcount_a + overcount_b$
16:        $\mathscr{S}_x$ creates the boolean vector $\vec{va}$, ($va_i = 1$, if $r_{i,a}$ exists, otherwise 0)
17:        $\mathscr{S}_y$ creates the boolean vector $\vec{vb}$, ($vb_i = 1$, if $r_{i,b}$ exists, otherwise 0)
18:        $\mathscr{S}_x$ and $\mathscr{S}_y$ securely compute $\phi_{a,b} = \vec{va} \cdot \vec{vb}$
19:    end if
20: end for
```

treat an update as a deletion of the old user and addition of a new user with the updated values. Now, the prior process can be used to perform the updates. This update process is described in algorithm 4.4.

Algorithm 4.4 Matrix update in the vertical partitioning scheme. Note that the update scheme does not mention the process of updating ϕ but it is similar to that of updating Δ.

```
1: for each cell (a,b) in the upper triangular matrix do
2:     if new users are added then
3:         Compute $ND_{a,b} = \Delta_{a,b}$ over only the new users using Algorithm 4.3
4:         Compute $\Delta'_{a,b} = \Delta_{a,b} + ND_{a,b}$
5:     else if some users are deleted then
6:         Compute $ND_{a,b} = \Delta_{a,b}$ over only the deleted users using Algorithm 4.3
7:         Compute $\Delta'_{a,b} = \Delta_{a,b} - ND_{a,b}$
8:     else
9:         Compute $NDO_{a,b} = \Delta_{a,b}$ over only the modified users (using original values) using Algo-
           rithm 4.3
10:        Compute $NDN_{a,b} = \Delta_{a,b}$ over only the modified users (using new values) using Algorithm 4.3
11:        Compute $\Delta'_{a,b} = \Delta_{a,b} - NDO_{a,b} + NDN_{a,b}$
12:    end if
13: end for
14: {Note that the process for computing $\phi'$ from $\phi$ is exactly the same as above.}
```

Note that, for the sake of simplicity, the derivation of the Δ matrix is described in plaintext form. However, if step 15 of algorithm 4.3, and steps 4, 7, and 11 of algorithm 4.4 is carried out in encrypted form, we can derive the matrix in encrypted form.

4.3 Arbitrary Partition

Arbitrarily partitioned data is more complex to handle. Essentially, the computation of each element of the deviation matrix and the cardinality matrix must be done in a fully distributed manner. Since the computation of each element is in effect independent, we can consider each in isolation. The computation required can actually be visualized as a combination of both the processing of horizontally partitioned data and the processing of vertically partitioned data. Consider a general element in the deviation matrix $\Delta_{i,j}$ and the corresponding element $\phi_{i,j}$. Now, each site $\mathscr{S}_x$ owns some number of users (potentially 0) for whom it owns both item i and item j. $\mathscr{S}_x$ can compute the local deviation matrices $\Delta_{i,j}^{\mathscr{S}_x}$ and $\phi_{i,j}^{\mathscr{S}_x}$ for those users. Every pair of sites $\mathscr{S}_x$ and $\mathscr{S}_y$ together own some number of users (potentially 0) for whom one site owns item i while the other owns item j. Now, the protocol for vertical partitioning (Algorithm 4.3) can be used to compute $\Delta_{i,j}^{\mathscr{S}_{x,y}}$ and $\phi_{i,j}^{\mathscr{S}_{x,y}}$ for those users, where $\mathscr{S}_x$ owns item i and $\mathscr{S}_y$ owns item j. Since every user either has both item i and j owned by one site or by two separate sites, each user is uniquely counted in the above computation. Therefore, $\Delta_{i,j} = \sum_x \Delta_{i,j}^{\mathscr{S}_x} + \sum_{x,y} \Delta_{i,j}^{\mathscr{S}_{x,y}}$. Similarly, $\phi_{i,j} = \sum_x \phi_{i,j}^{\mathscr{S}_x} + \sum_{x,y} \phi_{i,j}^{\mathscr{S}_{x,y}}$ The detailed procedure is given in Algorithm 4.5.

Algorithm 4.5 Initial matrix aggregation in the arbitrary partitioning scheme.

```
for all items $i$ and $j$, $i \neq j$ do
    {Compute $\Delta_{i,j}$ and $\phi_{i,j}$}
    for $x \leftarrow 0 \ldots k-1$ do
        Site $\mathscr{S}_x$ calculates the local deviation matrix $\Delta_{i,j}^{\mathscr{S}_x}$ and corresponding cardinality matrix $\phi_{i,j}^{\mathscr{S}_x}$ for all users that it owns both items $i$ and $j$
    end for
    for $x, y \in 0 \ldots k-1, x \neq y$ do
        Site $\mathscr{S}_x$ and $\mathscr{S}_y$ use Algorithm 4.3 to compute the deviation matrix $\Delta_{i,j}^{\mathscr{S}_{x,y}}$ and corresponding cardinality matrix $\phi_{i,j}^{\mathscr{S}_{x,y}}$ for all users for whom $\mathscr{S}_x$ owns item $i$ and $\mathscr{S}_y$ owns $j$
    end for
    for $x \leftarrow 0 \ldots k-1$ do
        At Site $\mathscr{S}_x$: Compute partial total $ptdelta_x \leftarrow \Delta_{i,j}^{\mathscr{S}_x} + (\forall y, \Delta_{i,j}^{\mathscr{S}_{x,y}})$
        At Site $\mathscr{S}_x$: Compute partial total $ptphi_x \leftarrow \phi_{i,j}^{\mathscr{S}_x} + (\forall y, \phi_{i,j}^{\mathscr{S}_{x,y}})$
    end for
    All sites together go through a k-cycle to compute $\Delta_{i,j} \leftarrow \sum_x ptdelta_x$ and $\phi_{i,j} \leftarrow \sum_x ptphi_x$
end for
```

Note that in the above algorithm, when each pair of sites uses the vertical partitioning algorithm to compute the pair element, it is split between them, as earlier – but this works fine, since they can all be added up in encrypted form to get the correct final values.

4.3.1 Matrix update

Again as earlier, any changes in individual ratings will change the individual pairwise deviations (and also cardinalities if new users are added or old users are removed[7]). The addition of new users is easy to handle. We simply need to carry out the prior aggregration process only for the added new users –

[7] Note that, as earlier, when a user's ratings are vertically partitioned, the user can only be added or deleted by all of the sites together to maintain the vertical partition. We again leave the case of addition or deletion of items to future work.

these can then be added in encrypted form to the existing $\Delta_{a,b}$ and $\phi_{a,b}$. Similarly, for deleted users, we can carry out the prior aggregation process only for the deleted users and then subtract those from the existing $\Delta_{a,b}$ and $\phi_{a,b}$. The case for updated users is more complicated. However, the same prior solution works - simply treat the update as a set of deletions and additions. Now, the prior process can be used to perform the updates. This update process is described in algorithm 4.6.

Algorithm 4.6 Matrix update in the arbitrary partitioning scheme. Note that the update scheme does not mention the process of updating ϕ but it is similar to that of updating Δ.

```
for each cell (a,b) in the upper triangular matrix do
    if users are added then
        Compute $ND_{a,b} = \Delta_{a,b}$ over only the new users using Algorithm 4.5
        Compute $\Delta'_{a,b} = \Delta_{a,b} + ND_{a,b}$
    else if users are deleted then
        Compute $ND_{a,b} = \Delta_{a,b}$ over only the deleted users using Algorithm 4.5
        Compute $\Delta'_{a,b} = \Delta_{a,b} - ND_{a,b}$
    else
        {Values are updated}
        Compute $NDO_{a,b} = \Delta_{a,b}$ over only the modified users (using original values) using Algorithm 4.5
        Compute $NDN_{a,b} = \Delta_{a,b}$ over only the modified users (using new values) using Algorithm 4.5
        Compute $\Delta'_{a,b} = \Delta_{a,b} - NDO_{a,b} + NDN_{a,b}$
    end if
end for
{Note that the process for computing $\phi'$ from $\phi$ is exactly the same as above.}
```

5 Implementation and evaluation

We first analyse the computation and communication complexity of our algorithms, then look at the security provided through our scheme, and finally present a performance evaluation of the cryptographic primitives.

5.1 Computation and Communication Complexity

In the following, note that, as defined in Definition 3, m represents the global number of users and n represents the global number of items.

5.1.1 Horizontal partition

Considering the case of horizontal partitioning, the cost of matrix aggregation in Algorithm 4.1 is significant in comparison with cost of computing the local deviation matrix and cardinality matrix. First, every site needs to encrypt its own deviation and cardinality matrix. Given that only the upper triangulars are computed, this gives $\sum_{i=0}^{n-1} i = n(n-1)/2$ encryptions for each matrix at each site. With k sites, there are a total of $kn(n-1)/2 = O(kn^2)$ encryptions for each matrix. Each site, excepting the first one, also needs to homomorphically add its own matrix to the running aggregate. Thus $O(kn^2)$ multiplications are also required. Finally, the cardinality matrix is decrypted (unless we are using secure sum), giving $n(n-1)/2)$ decryptions. Since the cost of the encryptions and decryptions dominate, the overall computation cost is $O(kn^2)$. Now, let us consider the communication cost. If we assume that each matrix is forwarded in a single message, there are a k messages in the first round (to aggregate), and then another k messages to forward, thus giving a total of $2k$ messages. Finally, for the cardinality matrix, the decryptions require another k messages. Since the upper triangular is transmitted in each message, giving a total of $O(kn^2)$ bits of communication. For updating the matrices (Algorithm 4.2), $\mathscr{S}_i$ forwards the updated element(s) of the deviation matrix and/or the cardinality matrix in a k cycle. Since the updated element in each matrix is simply forwarded as an encrypted message, there is no further cost of encryption in the subsequent sites apart from the first site $\mathscr{S}_i$. If the number of items updated is η then the number of encryptions will be $O(\eta^2)$. The number of messages forwarded is now k (because of one cycle), also requiring $O(k\eta^2)$ bits of communication. We will still have to decrypt the entire cardinality matrix, hence another $O(kn^2)$ bits of communication are involved.

5.1.2 Vertical partition

Now, consider the vertical partitioning case. This takes significantly larger computation and communication effort. In algorithm 4.3, for every pair of items (a,b) that are held by two different sites, two secure scalar products need to be computed to get the deviation for that cell, while one secure scalar product needs to be computed to get the cardinality. Each scalar product requires m encryptions and m multiplications. Therefore the total cost is $O(m)$. Since there are $n(n-1)/2$ cells, even though all cells do not require secure operations, overall the computation complexity is $O(mn^2)$. Since, typically, $m >> k$, this cost is far larger than the horizontally partitioned data case. Each scalar product requires two rounds of communication, and $O(m)$ bits. Therefore, the overall communication cost is $O(n^2)$ messages, and $O(mn^2)$ bits. In algorithm 4.4, the process is exactly the same as earlier, but only carried out over the changed users. Assuming the number of changed users is given by m', the computation cost is $O(m'n^2)$, and the communication cost is $O(n^2)$ messages and $O(m'n^2)$ bits of communication.

5.1.3 Arbitrary partition

When data is arbitrarily partitioned, the worst case occurs when the data is completely fragmented between all of the sites. Now, like vertical partitioning, for every pair of items (a,b), the vertical partitioning algorithm needs to be invoked for every pair of sites co-owning some users for those pair of items, and then the partial totals are all summed up. Therefore, in the worst case, for every pair of items (a,b), two secure scalar products need to be computed by every pair of sites collecting that data for some set of users to get the deviation for that cell, while one secure scalar product needs to be computed to get the cardinality. However, unlike vertical partitioning, the size of the scalar product changes based on the number of users for which the sites co-own the item ratings. Infact, since the global number of users is fixed (m), in total, the same number of encryptions and multiplications are carried out. The added computation complexity of adding the partial totals is negligible. Therefore, the overall complexity in the worst case is still $O(mn^2)$. Similarly, the overall communication cost is $O(n^2)$ messages, and $O(mn^2)$ bits. As earlier, the matrix update process is exactly the same as earlier, but only carried out over the changed users. Assuming the number of changed users is given by m', the computation cost of matrix update is $O(m'n^2)$, and the communication cost is $O(n^2)$ messages and $O(m'n^2)$ bits of communication.

5.2 Security Analysis

We now briefly analyse the security provided by our scheme. Essentially, the security of our algorithms is dependent on the security of the underlying primitives. Since both the deviation matrix and the cardinality matrix are kept in encrypted form, their confidentiality is preserved as long as the underlying encryption is secure. Furthermore, consider Algorithm 4.1: since all matrices are exchanged in encrypted form, where the underlying cryptosystem is a threshold cryptosystem, nothing is leaked unless all parties collude. The same is true of the matrix update process for horizontally partioned data. In the vertical partitioning case, Algorithm 4.3 users the secure scalar product as the underlying primitive. Since this is known to be secure and only random shares of the final result are exchanged, the entire procedure can also be shown to be secure (using the Composition theorem). This is also true of the update process. Finally, arbitrary partitioning requires usage of Algorithm 4.3 as well as secure sum. Since both can be shown to be secure, the entire process can again be shown to be secure under the framework of secure multiparty computation. Again, this is also true of the updating process.

5.3 Cryptographic primitives of the Paillier cryptosystem

In the workshop version of this paper, we showed the performance results of cryptographic primitives for which we had implemented the Paillier cryptosystem with homomorphic addition and multiplication functions. The cryptographic primitives of our implementation had been tested on a hardware consisting of a 2.53 GHz Intel Core i5 processor with 8GB DDR3 (1.07GHz bus speed) RAM running 64-bit Mac OS X 10.6.7 and Java 1.6.0_22 64-bit HotSpot Server VM[8]. The results are shown in table 5.1[9]. The results have been generated taking averages over 943 iterations (which happens to be the number of users in the MovieLens 100K dataset).

Table 5.1: Comparison of a Java implementation of Paillier cryptosystem with different bit lengths for the public key (i.e. modulus n). Plaintext is random and is in $\{1,2,3,4,5\}$ and integer multiplicand is random and is in $\{0,\ldots,999\}$ with about 80% of them being zero. Note that the bit length of the plaintext does not have much effect on encryption time due to the optimisation $c = (1+mn)\, r^n \, mod \, n^2$.

Paillier cryptosystem	**512-bits**	**1024-bits**	**2048-bits**
Average encryption time (ms)	2.656	17.262	124.544
Average decryption time (ms)	2.719	17.127	123.703
Average homomorphic addition time (ms)	0.008	0.027	0.099
Average homomorphic multiplication time (ms)	0.013	0.039	0.132

5.4 Single partition, single machine implementation

In this paper, we developed an optimised single partition, single machine implementation. Despite being a single machine implementation, we simulated a client server model and also used a Java implementation[10] threshold version of the Damgärd-Jurik cryptosystem. Therefore, for the same bit lengths as those shown in table 5.1, the decryption performance was slower with the increase of the number of decrypting server entities.

Figure 5.1 shows the overview of our implementation. Note that the pre-computation and prediction stages are separated. Pre-computation is done by the CF server while the query server handles the prediction. Both of these servers are multi-threaded. Communications between the servers and the client happen over TCP sockets, and data is transferred through serialized TCP streams.

One of the fundamental aspects for achieving high-efficiency pre-computation as well as prediction is the storage of the three matrices: (1) the user-item ratings matrix, (2) the item-item deviation matrix,

[8] Arbitrary precision integers are represented using the `java.math.BigInteger` class.

[9] For better accuracy, profiling has been done using `ThreadMXBean::getCurrentThreadUserTime()` method instead of the de-facto `System.nanoTime()`.

[10] University of Texas (Dallas) paillierp library.

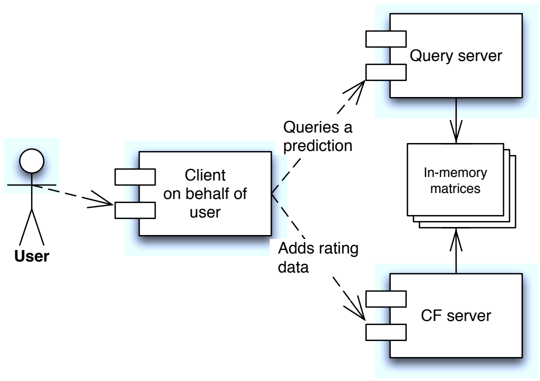


Figure 5.1: The overview of the implementation.

and (3) the item-item cardinality matrix. Whether the data stored in those matrices is in plaintext or in ciphertext, the choice of an efficient data structure holds the key to performance. In order to choose a data structure, we ought to remember that there is significant level of sparseness in the stored data. Even though the deviation and cardinality matrices are not as sparse as the user-item ratings matrix, recall that only the upper triangulars of the deviation and cardinality matrices are stored.

As of now, we considered in-memory storage of those matrices. In future, we will consider stream-based (e.g. disk file, TCP socket stream) and database storage options later. With in-memory storage, the sparseness requirement immediately implies that a 2-dimensional array (e.g. `long[][]`) is an unsuitable storage data structure. While the 2-D array provides constant time, i.e. $O(1)$ lookup performance, it is wasteful in storage space. In addition, resizing it is inefficient. Even if `ArrayList` implementations provide $O(1)$ lookup performance, the addition operation time complexity is $O(n)$. In our implementation, the pre-computation stage requires several concurrent read/write access to the matrices, `ArrayLists` are not suitable. A Trie data structure[11] is not part of the standard Java API, but it is also not suitable in our context as the keys used to store the data in the matrices are simply integer-based indexes of matrix cell positions.

Assuming access to a matrix by either row-major order or column-major order but not both, a 2-D matrix can be represented as a `Map<K1, Map<K2, V>>`. Java's `TreeMap` implementation provides $O(log\,n)$ lookup and storage performance while `Hashtable` and `HashMap` both provide constant time lookup and storage. Having tested Oracle (Sun) JVM's `HashMap` and `Hashtable` implementations, we found that `HashMap` is faster although there is no theoretical basis supporting this view and may be purely JVM implementation specific. Both `Hashtable` and `HashMap` are very similar except that `HashMap` allows `null` values for keys and objects, which is irrelevant in our context. `HashMap` iterator is fail-safe, which means changes made to the map get reflected in its iterator.

Having chosen an efficient data structure, i.e. `HashMap`, we looked at the data types for `K1`, `K2` and `V` because that affects performance too. Both `K1` and `K2` are integers (perhaps `long`) while `V` could contain arbitrary precision integers for storage of cipher texts. Realistically, the value of the keys is well expressed by Java's primitive 32-bit `int` with a positive range of $[0 \quad (2^{31}-1)]$. This is enough for indexing rows and columns: a $(2^{31}-1)$ by $(2^{31}-1)$ square matrix is extremely large! There is also another advantage of a `HashMap<Integer,V>` because `Integer` provides "a hash code value for this object, equal to the primitive int value represented by this Integer object"[12], which is faster to compute

[11] See: `http://en.wikipedia.org/wiki/Trie`.

[12] See: `http://download.oracle.com/javase/6/docs/api/java/lang/Integer.html`.

than that for a Long, i.e. "(int)(this.longValue()^(this.longValue()>>>32))"[13].

When storing user-item ratings, we used Long in conformance with the range of rating values in the MovieLens dataset, for example. Note that we can use Double to work with ratings but that will require our homomorphic cryptosystem to handle double precision floating point numbers. It is possible but not necessary for the current proof-of-concept. Ciphertexts are represented using Java's BigInteger that allows storing arbitrary precision integers. Since the deviation matrix stores encrypted deviations while the cardinality matrix stores plaintext cardinalities (expressed as Long), we avoid the overheads of having two matrices by combining the cardinality and deviation matrix into one and representing the V as a deviation-cardinality tuple.

The semantics of an "empty" deviation-cardinality tuple must be understood in order to ensure that the deviation-cardinality matrix allows sparseness by not storing empty tuples. Since the deviation value of zero (precisely, encrypted zero) does not indicate an absence of deviation, "emptiness" is determined by the cardinality value of zero. There is also another point that aids the sparse nature of the matrix – the storage of the upper triangular of the matrix only, discarding the lower triangular and the leading diagonal. While this behaviour is controllable through the matrix access methods, the tuple itself in our implementation enables access to the not-stored lower triangular by inverting the stored value in the upper triangular. Note that the cardinality is not inverted but the deviation (Δ) is, i.e. $\Delta_{i,j} = -\Delta_{j,i}$. This is achieved through a homomorphic multiplication, i.e. $\mathscr{E}(-\Delta_{i,j}) = \mathscr{E}(\Delta_{i,j}(-1)) = \mathscr{E}(\Delta_{i,j})^{-1} = \mathscr{E}(\Delta_{j,i})$.

For the sake of concurrency, all map-based matrix representations are derived from ConcurrentMap and ConcurrentHashMap.

5.4.1 Multi-threaded pre-computation

In the weighted Slope One predictor, the task of pre-computation of the deviations and cardinalities between pairs of items can be split in parallel across the different users. This means, the deviations for item pairs can be calculated for one user, independently from another user. This allows us to implement the pre-computation as a multi-threaded operation, which is a performance improvement over sequential pre-computation. However, it is important to note that the performance gain achieved through multi-threading can be offset by too many threads that can lead to resource starvation over the shared resources (e.g. the matrices storing user-item ratings and deviation-cardinality tuples). For this reason, we use a fixed pool of threads equal to the number of logical processors visible to the JVM. We have tested that using a cached thread pool, i.e. Executors.newCachedThreadPool(), for pre-computation leads to shared resource starvation when the number of users is large because too many threads (one for each user) get started and they all try to access the same shared matrices.

Since we start the pre-computation in a multi-threaded fashion, we will still be able to predict while pre-computation continues. Predicted values will eventually converge to stable values as the pre-computation finishes off. However, the ability to predict during pre-computation reduces turn-around time for prediction queries. Users can still get some predictions without having to wait for the entire pre-computation to finish, or wait for any subsequent updates. However, depending on how long the pre-computation takes, it may not always be possible to predict for a user at a particular time until a substantial chunk of pre-computation completes. In essence, if a query comes in when none of the pre-computation threads (one for each user) has finished then prediction may not be possible, or most likely to be error prone.

The pre-computation performance can be improved with multi-core or multi-processor CPU architecture. Since we are using fixed pool of threads, it is important to note that the pre-computation does not have a large memory footprint, which will mostly be dominated by the storage size of ciphertexts.

[13] See: http://download.oracle.com/javase/6/docs/api/java/lang/Long.html.

Threads are re-used from the fixed pool of threads. If a thread exits with an exception and the thread object is destroyed then a new thread gets created to replace the destroyed thread in the pool. Therefore, for the entire pre-computation, memory leaks do not happen with unexpected thread failures. All `Runnable` tasks are submitted at the start of pre-computation and they remain in an unbounded queue, which is serviced by the fixed number of threads. Thus, the total amount of memory required to run those tasks is claimed at the start of pre-computation.

5.4.2 Multi-threaded client-server architecture

The prediction of rating is achieved over a client-server architecture, where each site starts a multi-threaded server to which clients connect to request predictions. In the single partition, single site model, there is only one such site and the entire rating dataset is available as a single partition to the site. The site *simulates* shared decryptions of the threshold cryptosystem itself. At start, the site generates threshold keys for the homomorphic cryptosystem (or it may read from a pre-generated file) and reads in the rating dataset, e.g. the MovieLens 100K dataset. Then it starts the multi-threaded pre-computation as well as spawns out threads to listen on a server socket bound to a port on the IP addresses assigned to the network adapters of the machine. From this point, the client can connect to the server for prediction while pre-computation continues in the background.

The client-server protocol is based on character streams used to serialize objects. For each client connection, the server maintains an input (i.e. from the client) and an output (i.e. to the client) message queue. Objects in the input queue are enqueued by constantly reading objects from the socket input stream (using `ObjectInputStream`); while objects from the output queue are dequeued into the socket output stream (using `ObjectOutputStream`). The queue enqueue and dequeue operations are also multi-threaded so that independent threads can poll the input stream and the output message queue for deserialization and serialization operations respectively.

Prediction vs. pre-computation It is important to note that encrypting all deviations during pre-computation is essentially a single-machine *simulation* of a horizontally partitioned dataset with each user in one partition. In reality, there is no need to encrypt deviations during pre-computation in any one partition. The encryption is only required when the deviation data is shared with other partitions in other sites. The approximate cost of encrypted pre-computation is shown in table 5.2.

On the other hand, if we use unencrypted deviations and encrypt them only at the time of answering the prediction query then the pre-computation phase is very fast but each prediction task takes significantly longer, e.g. up to about 30 times more! The time taken will depend on how many items are being compared because that will determine the number of encryptions and homomorphic multiplications. However, in a realistic scenario, the encryptions will not need to be performed for every query. We can use on-demand encryptions. In that case, when most of the deviation matrix has been encrypted from several predictions, the prediction timing will eventually drop, helping us to achieve a balance of performance between pre-computation and prediction. Or there could be an additional step of encrypting the entire deviation matrix once the pre-computation is done: that helps speedup by eliminating homomorphic addition that is encountered while pre-computing with encrypted deviations.

6 Conclusion and future work

In this paper, we have proposed an efficient privacy-preserving solution for the problem of collaborative filtering over distributed data. Our solution is based on the weighted Slope One predictor from Lemire and MacLachlan [19] and uses homomorphic encryption to carry out the necessary computations. We

Table 5.2: Comparison of pre-computation and prediction times on a 2.53GHz Intel Core 2 Duo processor architecture with 8GB RAM running Mac OS X 10.6.7 and 64-bit Java 1.6 with a fixed thread pool executor service with a pool size equal to the number of logical CPU cores available to the JVM.

Keys[a]	Mean pre-computation[b]	Total pre-computation[c]	Prediction
512 bits	30s	2h	500ms
1024 bits	90s	6h	1.5s
2048 bits	270s	18h	4.5s

[a] This refers to modulus bit length of the cryptosystem.

[b] This is the mean of the pre-computation time taken for each user and all item pairs.

[c] This is an approximate linear estimate. Note that the total is not equal to the mean time for pre-computation multiplied by the number of users because the pre-computation tasks run in parallel.

have also presented test results of our CF scheme on a single machine, single dataset partition. Currently, we assume a PKI infrastructure, and complete collaborative environment. In the future, we are planning to reduce these trust assumptions, by leveraging P2P based topology for the participating sites, and reducing the need for cryptographic operations. One way to do this is to explore the possibility of aggregating partial deviation and cardinality matrix for a subgroup at a higher level aggregator using secure sum based techniques and then propagate it upwards. We will explore this in the future.

References

[1] C. C. Aggarwal and P. S. Yu. *A General Survey of Privacy-Preserving Data Mining Models and Algorithms*, chapter 2, pages 11–52. Springer-Verlag, 2008.

[2] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'01), Santa Barbara, California, USA*, pages 247–255. ACM, May 2001.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the 2000 ACM SIGMOD Conference on Management of Data (SIGMOD'00), Dallas, Texas, USA*, pages 439–450. ACM, May 2000.

[4] A. Basu, H. Kikuchi, and J. Vaidya. Privacy-preserving weighted Slope One predictor for Item-based Collaborative Filtering. In *Proc. of the International Workshop on Trust and Privacy in Distributed Information Processing (TP-DIS'11), Copenhagen, Denmark*, July 2011.

[5] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Privacy-enhanced collaborative filtering. In *Proc. of the User Modeling Workshop on Privacy-Enhanced Personalization (PEP'05), Edingburgh, UK*, pages 75–83, July 2005.

[6] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *Proc. of the ACM Conference on Recommender Systems (RecSys'07), Minneapolis, Minnesota, USA*, pages 9–16. ACM, October 2007.

[7] J. Canny. Collaborative filtering with privacy. In *Proc. of the 2002 IEEE Symposium on Security and Privacy, Oakland, California, USA*, pages 45–57. IEEE, May 2002.

[8] J. Canny. Collaborative filtering with privacy via factor analysis. In *Proc. of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02), Tampere, Finland*, pages 238–245. ACM, August 2002.

[9] R. Cissée and S. Albayrak. An agent-based approach for privacy-preserving recommender systems. In *Proc. of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07), Honolulu, Hawaii, USA*, pages 182:1–182:8. ACM, May 2007.

[10] C. Clifton, M. Kantarcioglu, X. Lin, J. Vaidya, and M. Zhu. Tools for Privacy Preserving Distributed Data Mining. *SIGKDD Explorations*, 4(2):28–34, January 2003.

[11] I. Damgärd and M. Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Proc. of the 4th International Workshop on Public Key Cryptography (PKC'01), Cheju Island, Korea. LNCS*, volume 1992, pages 119–136. Springer-Verlag, February 2001.

[12] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02), Edmonton, Alberta, Canada*, pages 217–228. ACM, July 2002.

[13] O. Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols, pages 599–764. Cambridge University Press, Cambridge, UK, 2004.

[14] S. Gong. Privacy-preserving collaborative filtering based on randomized perturbation techniques and secure multiparty computation. *International Journal of Advancements in Computing Technology (IJACT)*, 3(4):89–99, 2011.

[15] S. Han, W. K. Ng, and P. S. Yu. Privacy-Preserving Singular Value Decomposition. In *Proc. of the 25th IEEE International Conference on Data Engineering (ICDE'09), Shanghai, China*, pages 1267–1270. IEEE, March-April 2009.

[16] Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *Proc. of the ACM International Conference on Management of Data (SIGMOD'05), Baltimore, Maryland, USA*, pages 37–48. ACM, June 2005.

[17] C. Kaleli and H. Polat. P2P collaborative filtering with privacy. *Turkish Journal of Electric Electrical Engineering and Computer Sciences*, 18(1):101–116, 2010.

[18] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proc. of the 3rd IEEE International Conference on Data Mining (ICDM'03), Melbourne, Florida, USA*, pages 99–106. IEEE, November 2003.

[19] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *Proc. of the SIAM Data Mining (SDM'05), Newport Beach, California, USA*, April 2005.

[20] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Proc. of the Advances in Cryptology (CRYPTO'00), Santa Barbara, California, USA, LNCS*, volume 1880, pages 20–24. Springer-Verlag, August 2000.

[21] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[22] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), March 2007.

[23] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of the 17th International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT'99), Prague, Czech Republic. LNCS*, volume 1592, pages 223–238. Springer-Verlag, May 1999.

[24] H. Polat and W. Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *Proc. of the 3rd IEEE International Conference on Data Mining (ICDM'03), Melbourne, Florida, USA*, pages 625–628. IEEE, November 2003.

[25] H. Polat and W. Du. Privacy-preserving collaborative filtering on vertically partitioned data. In *Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05), Porto, Portugal. LNCS*, volume 3271, pages 651–658. Springer-Verlag, October 2005.

[26] H. Polat and W. Du. Privacy-preserving Top-N recommendation on horizontally partitioned data. In *Proc. of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05), Compiegne, France*, pages 725–731. IEEE, September 2005.

[27] H. Polat and W. Du. SVD-based collaborative filtering with privacy. In *Proc. of the 20th ACM Symposium on Applied Computing (SAC'05), Santa Fe, New Mexico, USA*, pages 791–795. ACM, March 2005.

[28] H. Polat and W. Du. Achieving private recommendations using randomized response techniques. In *Proc. of the 10th Asia-Pacific Conference on Advances in Knowledge Discovery and Data Mining (PAKDD'06), Singapore, LNCS*, volume 3918, pages 637–646. Springer-Verlag, April 2006.

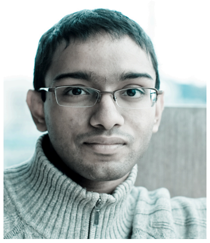
[29] S. J. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *Proc. of the 28th International Conference on Very Large Databases (VLDB'02), Hong Kong SAR, China*, pages 682–693,

August 2002.

[30] J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *Proc. of the 1st ACM Conference on Electronic Commerce (EC'99), Denver, Colorado, USA*, pages 158–166. ACM, November 1999.

[31] L. Sweeney. *k*-anonymity: a model for protecting privacy. *International Journal on Uncertainty Fuzziness Knowledge-Based Systems*, 10(5):557–570, October 2002.

[32] A. C. Yao. How to generate and exchange secrets. In *Proc. of the 27th IEEE Symposium on Foundations of Computer Science (FOCS'86), Toronto, Canada*, pages 162–167. IEEE, October 1986.



Anirban Basu is a Post-doctoral Researcher at Kikuchi lab at Tokai University working on a Japanese Ministry of Internal Affairs and Communications funded project in collaboration with Waseda University, Hitachi, NEC and KDDI. He is also a Visiting Research Fellow at the University of Sussex working with the Foundations of Software Systems group. He holds a Ph.D. in Computer Science and a Bachelor of Engineering (Hons.) in Computer Systems Engineering from the University of Sussex. His research interests are in computational trust management, privacy and security and peer-to-peer networks. He has several years of experience with academic research at the University of Sussex where he was involved with two EPSRC funded and one EU IST FP5 funded research projects alongside his doctoral research.

Jaideep Vaidya is an Associate Professor of Computer Information Systems at Rutgers University. He received his Masters and Ph.D. in Computer Science from Purdue University and his Bachelors degree in Computer Engineering from the University of Mumbai. His research interests are in Privacy, Security, Data Mining, and Data Management. He has published over 60 papers in international conferences and archival journals, and has received three best paper awards from the premier conferences in data mining, databases, and digital government research. He is also the recipient of a NSF Career Award and a Rutgers Board of Trustees Research Fellowship for Scholarly Excellence.

Hiroaki Kikuchi received B.E., M.E. and Ph.D. degrees from Meiji University. He is currently a Professor in the Department of Communication and Network Engineering, School of Information and Telecommunication Engineering, Tokai University. He was a visiting researcher of the School of Computer Science, Carnegie Mellon University in 1997. His main research interests are fuzzy logic, cryptographic protocols, and network security. He is a member of the Institute of Electronics, Information and Communication Engineers of Japan (IEICE), the Information Processing Society of Japan (IPSJ), the Japan Society for Fuzzy Theory and Systems (SOFT), IEEE and ACM. He is an IPSJ Fellow.