Protect both Integrity and Confidentiality in Outsourcing Collaborative Filtering Computations

Qiang Tang
University of Luxembourg
Luxembourg
Email: qiang.tang@uni.lu

Balázs Pejó
University of Luxembourg
Luxembourg
Email: balazs.pejo@uni.lu

Husen Wang
University of Luxembourg
Luxembourg
Email: husen.wang@uni.lu

***Abstract*—In the cloud computing era, in order to avoid the computational burdens, many recommendation service providers tend to outsource their collaborative filtering computations to third-party cloud servers. In order to protect service quality, the integrity of computation results needs to be guaranteed. In this paper, we analyze two integrity verification approaches by Vaidya et al. and demonstrate their performances. In particular, we analyze the verification via auxiliary data approach which is only briefly mentioned in the original paper, and demonstrate the experimental results. We then propose a new solution to outsource all computations of the weighted Slope One algorithm in two-server setting and provide experimental results.**

***Keywords*-Collaborative filtering; Integrity; Confidentiality.**

I. Introduction

Collaborative filtering is a general technique for recommender systems to make automatic predictions about the interests of a user by collecting preferences or taste information from many users. Numerous organizations, small or big, have deployed it to provide personalized services to their customers. But the benefits do not come for free because collaborative filtering algorithms are often computation-intensive, especially when more data are desired to get better results. In the cloud computing era, a natural solution is for the recommendation service provider (referred to as RecSys throughout the paper) to outsource the computations to a cloud server. In the outsourcing, two issues arise. One is the integrity of the computation results. The cloud server may provide some fake results instead of spending its resources in computing the correct ones. The other one is confidentiality and more generally privacy issue. Both integrity and confidentiality are desirable, but we mainly focus on the integrity issue.

With respect to integrity protection in outsourcing collaborative filtering computations, two most relevant works are [9] and [10]. In [9], Sheng et al. relied on some algebraic properties to aggregately verify the correctness (or, integrity) of inner product results computed by the service provider. In [10], Vaidya et al. proposed two approaches to verify the integrity of outsourced computations (precisely, for weighted Slope One and adjusted Cosine-based algorithms). Moreover, they introduced a game-theoretic approach which can serve as a complementary deterring factor against the server's cheating attempt. Besides [9] and [10], there are many other related works. For example, Wong et al. [11] and Dong et al. [4] investigated the integrity issues in outsourcing frequent itemset mining computations, and Liu et al. [6] proposed probabilistic and deterministic methods to verify clustering results for k-means clustering algorithms.

A. Problem Statement and Our Contribution

The problem setting is as follows. The RecSys possesses a rating dataset and wants to compute the predicted ratings for the unrated items. For efficiency reasons, the RecSys will outsource the computations to a cloud server and wants to guarantee the integrity of computation results. In this paper, we focus on the weighted Slope One recommender algorithm from [5].

In Section III, we analyze the cheating and detection strategies in four scenarios from [10], and figure out the best strategies for the RecSys and the cloud server. In Section IV, we analyze the verification via auxiliary data approach and propose a simplified variant. We demonstrate that this variant is more effective than the verification via splitting approach from the perspective of the RecSys. In Section V, we propose a solution to outsource the computations in both stages for the weighted Slope One algorithm in two-server setting, and show the experimental results.

All experimental results are obtained based on Intel(R) Xeon(R) CPU E3-1241 v3 @ 3.50GHz using 15 GB RAM. Both integrity verification approaches can achieve some level of confidentiality, we put the details in the full paper [13] due to space limitation.

II. Weighted Slope One Collaborative Filtering

In a recommender system, the item set is denoted by $\mathbf{B} = \{1, 2, \ldots, M\}$ and the user set is denoted by $\mathbf{U} =$

$\{1, 2, \ldots, N\}$. A user x's ratings are denoted by a vector $\mathbf{R}_x = (r_{x,1}, \ldots, r_{x,M})$. The rating value is often an integer from $\{0, 1, 2, 3, 4, 5\}$. If user x has not rated item i then $r_{x,i}$ is set to be 0. The ratings are often organized in a rating matrix. The functionality of a recommender system is to predict the unrated values. With respect to $\mathbf{R}_x$, a binary vector $\mathbf{Q}_x = (q_{x,1}, \ldots, q_{x,M})$ is defined as follows: $q_{x,b} = 1$ iff $r_{x,b} \neq 0$ for every $1 \leq b \leq M$. Basically, $\mathbf{Q}_x$ indicates which items have been rated by user x. The density of the rating matrix is $d = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} q_{i,j}}{MN}$.

The weighted Slope One recommender algorithm [5] exploits deviation metrics with "popularity differential" notion between items. It predicts the rating of an item for a user from pair-wise deviations of item ratings. The algorithm has two stages.

Computation stage. In this stage, two matrices $\Phi_{M\times M}$ and $\Delta_{M\times M}$ are generated. For every $1 \leq i, j \leq M$, $\phi_{i,j}$ and $\delta_{i,j}$ (i.e. the element located in i-th row and j-th column) are defined as $\phi_{i,j} = \sum_{u=1}^{N} q_{u,i} q_{u,j}$, $\delta_{i,j} = \sum_{u=1}^{N} q_{u,i} q_{u,j} (r_{u,i} - r_{u,j})$ respectively. In more detail, $\phi_{i,j}$ is the number of users who rated both item i and j, while $\delta_{i,j}$ is the *deviation* of the ratings of item i from item j and $\Delta_{M\times M}$ is referred to as the *deviation matrix*.

Prediction stage. This stage uses both $\Phi_{M\times M}$ and $\Delta_{M\times M}$ as well as the original rating matrix to predict the unrated ratings for all users. To compute the predicted rating for item i for user x, the formula of weighted Slope One algorithm is $p_{x,i} = \frac{\sum_{j \in \mathbf{B}/\{i\}} \delta_{i,j} + r_{x,j} \phi_{i,j}}{\sum_{j \in \mathbf{B}/\{i\}} \phi_{i,j}}$.

III. Examining the Verification via Splitting Approach

With the verification via splitting approach, the idea is to *horizontally* split the rating matrix D into r ($r \geq 1$) non-overlapping blocks. These blocks are independently outsourced to the server, who performs the requested computations on the blocks and returns the results. Let $\delta_{i,j}^{(k)}$ be the results based on the k-th block, the server returns the following to the RecSys: intermediate deviation values $\delta_{i,j}^{(k)}$, for all $1 \leq i, j \leq M$ and $1 \leq k \leq r$; final deviation values $\delta_{i,j} = \sum_{k=1}^{r} \delta_{i,j}^{(k)}$, for all $1 \leq i, j \leq M$.

A. Compare four Cheating-Detection Scenarios

In the following, we analyze the four different scenarios cheating-detection scenarios from [10] for verifying the intermediate results $\delta_{i,j}^{(k)}$.

- **Scenario 1** The server randomly picks up α items and sets the deviation values associated with these items to be random numbers in all the blocks. For every block, the RecSys randomly selects θ deviation values to verify by re-computing these values. The verification cost is $r\theta$ deviation computations.
- **Scenario 2** The server randomly picks up in total β users/rows from all blocks and discards their values in the computation. For every block, the RecSys randomly selects θ deviations to verify by recomputing these values. The verification cost is $r\theta$ deviation computations.
- **Scenario 3** The server randomly selects γ blocks and sets the related deviations to be random numbers. The RecSys randomly selects δ blocks to verify by re-computing one deviation value for each of them. The verification cost is δ deviation computations.
- **Scenario 4** For every block, the server cheats with probability ρ and sets the related deviation values to be random numbers. The RecSys randomly selects δ blocks to verify by re-computing one deviation value for each of these blocks. The verification cost is δ deviation computations.

It is clear that, for any given cheating rate for the server, the RecSys would prefer high detection rate and low verification cost. The cheating rate is denoted as ρ, meaning that the server avoids ρ percent of the required computations in the attack, and the detection rate is denoted as P_d, meaning the probability that the RecSys detects cheating. Therefore, we compare these four scenarios in a pairwise manner, by setting a common cheating rate of the server and studying the detection rate and the verification cost for the RecSys. We put the comparison details in the full paper [13], and conclude that the RecSys will prefer Scenario 3 while the server will prefer Scenario 2 or Scenario 1 the most. Informally, for the RecSys, the best strategy is to randomly choose some blocks and randomly choose some deviation values in these blocks to verify. For the server, the best strategy is to randomly choose some blocks and set some deviation values in them to be random values.

With respect to the verification of final deviation values $\delta_{i,j}$ ($1 \leq i, j \leq M$), the RecSys can randomly select a subset and verify them by summing up the related intermediate results.

B. Experimental Results

We use MLM and Netflix datasets as examples to study the performances, by assuming the following. The server randomly selects γ blocks, and sets ζ percent of the intermediate deviation values in them to be random numbers. Moreover, the server randomly selects ζ percent of the final deviation values to be random numbers. In every block, the RecSys randomly chooses θ intermediate deviation values to verify. At last, the RecSys randomly chooses 100θ final deviation values to verify. Based on this assumption, the server

can set $\frac{\gamma\zeta}{r} = \rho$ if it wants to achieve the cheating rate ρ, and the RecSys's detection rate is

$$P_d = \left(1 - \left(\binom{\frac{M(M-1)(1-\zeta)}{2}}{\theta} \Big/ \binom{\frac{M(M-1)}{2}}{\theta}\right)^{\gamma}\right) \cdot \left(1 - \left(\binom{\frac{M(M-1)}{2} - 100\theta}{100\theta} \Big/ \binom{\frac{M(M-1)}{2}}{100\theta}\right)\right)$$

It is straightforward to verify that, by using a larger r, the RecSys achieves higher detection rate P_d with the same verification cost. Next, we choose $r = 1$ (i.e. no splitting) and $r = 10$ respectively to have a closer look at the detection rates for the MLM dataset. We skip the details for the Netflix dataset because they are very similar. In the experiment, we fix a number of (ρ, θ) pairs and summarize the P_d values in Tables I and II. In Table II, P_d values are minimized with respect to γ and ζ.

θ \ ρ	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-6}	2^{-9}
10	0.9990	0.9437	0.7369	0.4755	0.1457	0.0194
20	1.0000	0.9968	0.9308	0.7249	0.2702	0.0383
40	1.0000	1.0000	0.9952	0.9243	0.4674	0.0752
60	1.0000	1.0000	0.9997	0.9792	0.6113	0.1107
80	1.0000	1.0000	1.0000	0.9943	0.7163	0.1448
100	1.0000	1.0000	1.0000	0.9984	0.7930	0.1776
200	1.0000	1.0000	1.0000	1.0000	0.9571	0.3236

Table I
P_d Values When $r = 1$

θ \ ρ	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-6}	2^{-9}
1	1.0000	0.9954	0.8594	0.6240	0.1239	0.0035
2	1.0000	1.0000	0.9802	0.8594	0.2757	0.0125
4	1.0000	1.0000	0.9996	0.9802	0.4923	0.0412
6	1.0000	1.0000	1.0000	0.9972	0.6391	0.0771
8	1.0000	1.0000	1.0000	0.9996	0.7431	0.1154
10	1.0000	1.0000	1.0000	0.9999	0.8171	0.1537
20	1.0000	1.0000	1.0000	1.0000	0.9666	0.3194

Table II
P_d Values When $r = 10$

The results show that, to achieve similar detection rates P_d, the required verification cost *decreases* roughly in a linear manner with respect to the number of blocks. We summarize the verification costs in Tables III and IV.

$r = 1$	θ	10	20	40	60	80	100	200
	Total Time.	0.0008	0.0014	0.0026	0.0039	0.0052	0.0067	0.0135
$r = 10$	θ	1	2	4	6	8	10	20
	Total Time.	0.0005	0.0010	0.0020	0.0031	0.0041	0.0051	0.0101

Table III
Total Costs for MLM Dataset (seconds)

$r = 1$	θ	10	20	40	60	80	100	200
	Total Time.	0.02936	0.0546	0.1126	0.1659	0.2220	0.2873	0.5794
$r = 10$	θ	1	2	4	6	8	10	20
	Total Time.	0.0048	0.0071	0.0128	0.0187	0.0240	0.0298	0.0570

Table IV
Total Costs for Netflix Dataset (seconds)

It seems that the larger r the better for the RecSys. However, one concern is that the server may only cheat on a small number of final deviation results instead of ζ percent as we have assumed. In this case, the detection rates will be much smaller than those in Table II.

IV. Examining the Verification via Auxiliary Data Approach

A. Recap the Original Approach

The approach from [10] generates synthetic data to be merged with the original matrix D, as shown in Fig. 1. For efficiency reasons, it is required that $n' \ll N$ and $m' \ll M$. In order to prevent the server from figuring out the synthetic data, it is necessary that the real users "rate" the fake items (Z) as well as the fake users "rate" the real items (Y). Clearly, the fake ratings from Y may lead to inaccurate item-item deviations, while the fake ratings from Z increases the verification computation cost. In [10] the authors come up with the following idea for generating Y and Z: let user u rate items i and j. The impact on the deviation is $r_{u,i} - r_{u,j}$. By setting another user v ratings such that $r_{v,i} - r_{v,j} = -(r_{u,i} - r_{u,j})$, the deviation between i and j is unchanged. E.g. if ratings are in the interval $[min, max]$ then by setting $r_{v,i} = inverse(r_{u,i}) = min + max - r_{u,i}$ all changes in the deviations are eliminated. As seen in Equation (1), as long as the sum of ratings belonging to the two items from the group of users is equal then $\Delta_{i,j} = 0$. This easily generalizes the generation of elements of Y and Z for more items.

$$\begin{aligned} \sum_u r_{u,i} - r_{u,j} = r_{1,i} - r_{1,j} + \cdots + r_{k,i} - r_{k,j} = 0 \\ \Rightarrow r_{1,i} + \cdots + r_{k,i} = r_{1,j} + \cdots + r_{k,j} = \gamma \end{aligned} \tag{1}$$

Formally, this approach can be applied with the following steps: (1) the RecSys generates the synthetic matrices X, Y, Z to be integrated with D; (2) it then randomly permutes the rows and columns of the new matrix and send it to the server; (3) the server computes an intermediate deviation matrix Δ'' for the RecSys; (4) the RecSys derives another deviation matrix Δ' for the matrix shown in Fig. 1 based on the permutations it has done; (5) the RecSys verifies the deviation values for the matrix X; (6) if the verification passes, the RecSys can obtain the deviation matrix Δ for D from Δ'.

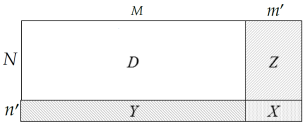

Figure 1. Splitting I

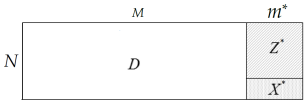

Figure 2. Splitting II

B. A Simplified Variant

We observe that the matrix Y only plays the role of hiding X in Fig. 1 while it increases the server's complexity in computing deviation values. Therefore, we propose to extend D as shown in Fig. 2, where X^* and Z^* are computed in the same way as X and Z. By doing so, everything stays the same except that the incurred complexity by Y is avoided. With the simplified variant and the original approach, the server needs to compute $\frac{(M+m^*)(M+m^*-1)}{2}$ deviation values, among which m^*M deviation values are due to the verification needs while only $\frac{m^*(m^*-1)}{2}$ of them can be used by the RecSys (in step (5) of the procedure). In order to reduce the computational overhead for the server, it is ideal to minimize the value m^* while keeping the $\frac{m^*(m^*-1)}{2}$ deviation values non-zero. Suppose the matrix X^* has n^* rows, then the probability that the inner product of any two columns is nonzero is $\frac{\binom{n^*}{n^*d}-\binom{n^*-n^*d}{n^*d}}{\binom{n^*}{n^*d}}$ where d is the density of D. Tables V and VI shows the minimal n^* values for $\frac{\binom{n^*}{n^*d}-\binom{n^*-n^*d}{n^*d}}{\binom{n^*}{n^*d}}$ to achieve a number of probabilities. Note that due to the lower density, to achieve a similar probability, n^* is larger in the case of the Netflix dataset.

n^*	100	200	300	400	500
Probability	0.1528	0.8413	0.9748	0.9994	0.9999

Table V
VALUES FOR MLM DATASET

n^*	7000	10500	17500	21000	35000
Probability	0.6395	0.7835	0.9220	0.9531	0.9899

Table VI
VALUES FOR NETFLIX DATASET

Let's assume the server will randomly choose ρ percent of deviation values and set them to random values. The detection rate can be computed as follows.

$$P_d = 1 - \left(\frac{\frac{(M+m^*)(M+m^*-1)}{2} - \frac{m^*(m^*-1)}{2}}{\frac{\rho(M+m^*)(M+m^*-1)}{2}} \right) \Big/ \left(\frac{\frac{(M+m^*)(M+m^*-1)}{2}}{\frac{\rho(M+m^*)(M+m^*-1)}{2}} \right) \quad (2)$$

We fix a number of (ρ, m^*) pairs and summarize the detection rates P_d in Table VII.

m^* \ ρ	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-6}	2^{-9}
2	0.5000	0.2500	0.1250	0.0625	0.0156	0.0020
4	0.9844	0.8220	0.5512	0.3211	0.0902	0.0117
6	1.0000	0.9866	0.8651	0.6202	0.2104	0.0289
10	1.0000	1.0000	0.9975	0.9452	0.5077	0.0842
20	1.0000	1.0000	1.0000	1.0000	0.9498	0.3103
40	1.0000	1.0000	1.0000	1.0000	1.0000	0.7824
80	1.0000	1.0000	1.0000	1.0000	1.0000	0.9979

Table VII
P_d VALUES FOR MLM AND NETFLIX DATASETS

Interestingly, the P_d values are exactly the same with four-digits precision, even though those of MLM dataset should be slightly larger than those of Netflix dataset. It shows that for reasonable cheating and detection rates the RecSys only needs to add very small-sized X^* from the perspective of column size m^* and compute $\frac{m^*(m^*-1)}{2}$ deviation values based on X^*.

C. Comparison to Verification via Splitting

In comparison, the verification via auxiliary data approach mainly has two advantages.

- It has minimal computational complexity for the RecSys, by minimizing the size of X^*. Furthermore, it allows the RecSys to pre-compute X^* and Y^*.
- It introduces fake data into the dataset and somehow anonymizes the original dataset so that it is more confidentiality-friendly.

Referring to Tables I, II, and VII, we can roughly say that $m^* = 20$ in the verification via auxiliary data approach provides similar detection rates to those of $\theta = 200$ when $r = 1$ and $\theta = 20$ when $r = 10$ in the verification via splitting approach. We present comparison results in Table VIII. With respect to the verification via splitting approach, we take the minimum values from the last columns in Tables III and IV; with respect to the running time of the verification via auxiliary data approach, it is based on a 300×20 matrix X^* for MLM and a 21000×20 matrix X^* for Netflix.

	Verification via Splitting	Verification via Auxiliary Data
MLM	0.0101	0.0010
Netflix	0.0570	0.0054

Table VIII
COST COMPARISON FOR RECSYS (SECONDS)

The downside is that some overhead for the server is incurred, as we have discussed in the beginning of Section IV-B. If m^* is much smaller than M, then the overhead is quite small. Nevertheless, this can be regarded as a tradeoff of the approach.

V. OUTSOURCING IN TWO-SERVER SETTING

Suppose there are two non-colluding servers, named CS_1 and CS_2. The solution outsources the computations of the computation and prediction stages in two steps.

We stress that, in the following solution, the RecSys should randomly permute the matrix D with respect to all the rows and columns before splitting it in both steps. Otherwise, the servers can avoid detection by only honestly computing the to-be-verified values. However, for the simplicity of presentation, we skip this permutation and depermutation steps in our descriptions.

1) In the first step, the players interact as follows.
   a) The RecSys splits the rating matrix D into D_1^* and D_2^* as shown in Fig. 3.

i) It splits D into two $\frac{N}{2} \times M$ sub-matrices D_1 and D_2.
ii) It randomly chooses m' columns from D_1 and put them into D_2. The resulting $\frac{N}{2} \times (M + m')$ matrix is named D_2^*.
iii) It randomly chooses m' columns from D_2 and put them into D_1. The resulting $\frac{N}{2} \times (M + m')$ matrix is named D_1^*.

The RecSys sends D_1^* and D_2^* to CS_1 and CS_2 respectively. Let the user set associated with D_1^* (D_2^*) be denoted as $\mathbb{N}_1$ ($\mathbb{N}_2$). Let's assume the added m' columns corresponding to D_1^* (D_2^*) define a new item set $\mathbf{I}_1$ ($\mathbf{I}_2$).

b) After receiving D_1^*, CS_1 computes a matrix $\Delta^{(1)}_{(M+m')\times(M+m')}$. For every $i, j \in \mathbf{B} \bigcup \mathbf{I}_1$, $\delta^{(1)}_{i,j}$ is computed as $\delta^{(1)}_{i,j} = \sum_{u \in \mathbb{N}_1} q_{u,i} q_{u,j} (r_{u,i} - r_{u,j})$. Similarly, CS_2 computes a matrix $\Delta^{(2)}_{(M+m')\times(M+m')}$. For every $i, j \in \mathbf{B} \bigcup \mathbf{I}_2$, $\delta^{(2)}_{i,j}$ is computed as $\delta^{(2)}_{i,j} = \sum_{u \in \mathbb{N}_2} q_{u,i} q_{u,j} (r_{u,i} - r_{u,j})$.

c) After receiving $\Delta^{(i)}_{M\times M}$ for $i = 1, 2$ from both servers, the RecSys proceeds as follows.
i) It verifies the deviation values for the new item sets $\mathbf{I}_1$ and $\mathbf{I}_2$. Since these values are computed by both CS_1 and CS_2, the verification is just comparison.
ii) If the verification passes, the RecSys computes $\Delta_{M\times M}$. For every $1 \le i, j \le M$, $\delta_{i,j} = \delta^{(1)}_{i,j} + \delta^{(2)}_{i,j}$.

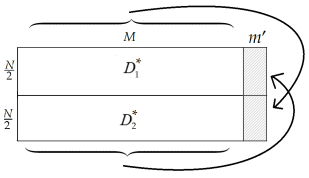

Figure 3. Splitting I

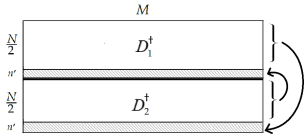

Figure 4. Splitting II

2) In the second step, the RecSys first computes $\Phi_{M\times M}$, and then proceeds as follows.
a) The RecSys splits the rating matrix D into $D_1^\dagger$ and $D_2^\dagger$ as shown in Fig. 4.
i) It splits D into two $\frac{N}{2} \times M$ sub-matrices D_1 and D_2.
ii) It randomly chooses n' rows from D_1 and put them into D_2. The resulting $(\frac{N}{2} + n') \times M$ matrix is named $D_2^\dagger$.
iii) It randomly chooses n' rows from D_2 and put them into D_1. The resulting $(\frac{N}{2} + n') \times M$ matrix is named $D_1^\dagger$.

The RecSys sends $D_1^\dagger$ and $D_2^\dagger$ to CS_1 and CS_2 respectively. In addition, the RecSys sends $\Delta_{M\times M}, \Phi_{M\times M}$ to both servers. Let the user set associated with $D_1^\dagger$ ($D_2^\dagger$) be denoted as $\mathbb{N}_1^\dagger$ ($\mathbb{N}_2^\dagger$). Let's assume the added n' rows corresponding to $D_1^\dagger$ ($D_2^\dagger$) define a new user set $\mathbf{U}_1$ ($\mathbf{U}_2$).

b) After receiving $\Delta_{M\times M}, \Phi_{M\times M}$ and $D_1^\dagger$, CS_1 computes the predictions $p_{x,i}$ for all $x \in \mathbb{N}_1^\dagger$ and $i \in \mathbf{B}$. Similarly, CS_2 computes the predictions $p_{x,i}$ for all $x \in \mathbb{N}_2^\dagger$ and $i \in \mathbf{B}$.

c) After receiving the predictions from both servers, the RecSys first verifies the values form the user sets $\mathbf{U}_1$ and $\mathbf{U}_2$. Since these values are computed by CS_1 and CS_2 simultaneously, the verification is just comparison. If the verification passes, the RecSys can accepts the predictions for user $x \in \mathbf{U}$.

From the description, it is clear that the main cost for the RecSys is to compute $\Delta_{M\times M}$ in the first step. It is about $\frac{M(M-1)}{2}$ additions. The computational overhead for every server is very small, and can be found in [13].

Suppose CS_1 cheats by setting ρ percent values to be random numbers, while CS_2 is honest. The detection rates in the first step and the second step are $1 - f(\rho)$ and $1 - g(\rho)$ respectively. If CS_2 also cheats in the same manner, then the detection rate in the first step and the second step are $1 - f(\rho)^2$ and $1 - g(\rho)^2$ respectively. The functions $f(\rho)$ and $g(\rho)$ are defined as follows. Note that if cheating is detected, the RecSys needs to do extra work to figure out who has/have cheated.

$$f(\rho) = \frac{\binom{\frac{(M+m')(M+m'-1)}{2} - m'(m'-1)}{\rho \frac{(M+m')(M+m'-1)}{2}}}{\binom{\frac{(M+m')(M+m'-1)}{2}}{\rho \frac{(M+m')(M+m'-1)}{2}}}, \quad g(\rho) = \frac{\binom{(1-d)M(\frac{N}{2} - n')}{\rho(1-d)M(\frac{N}{2} + n')}}{\binom{(1-d)M(\frac{N}{2} + n')}{\rho(1-d)M(\frac{N}{2} + n')}}$$

With respect to the MLM and Netflix datasets, we fix a number of (ρ, m') pairs and study the detection rates P_d in step 1. Table IX summarizes the results when only one server cheats. Table X summarizes the results when both servers cheat. Interestingly, the P_d values are exactly the same for both datasets with four-digits precision, even though those of MLM dataset should be slightly larger than those of Netflix dataset.

m' \ ρ	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-6}	2^{-9}
2	0.7500	0.4375	0.2344	0.1211	0.0310	0.0039
4	0.9998	0.9683	0.7986	0.5390	0.1722	0.0232
6	1.0000	0.9998	0.9818	0.8557	0.3765	0.0570
10	1.0000	1.0000	1.0000	0.9970	0.7576	0.1613
16	1.0000	1.0000	1.0000	1.0000	0.9772	0.3745
20	1.0000	1.0000	1.0000	1.0000	0.9975	0.5243
30	1.0000	1.0000	1.0000	1.0000	1.0000	0.8175

Table IX
P_d VALUES W.R.T. m' AND ρ (MLM AND NETFLIX, ONE CHEATS)

m' \ ρ	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-6}	2^{-9}
2	0.9375	0.6836	0.4138	0.2275	0.0611	0.0078
4	1.0000	0.9990	0.9594	0.7875	0.3147	0.0458
6	1.0000	1.0000	0.9997	0.9792	0.6113	0.1107
10	1.0000	1.0000	1.0000	1.0000	0.9413	0.2967
16	1.0000	1.0000	1.0000	1.0000	0.9995	0.6088
20	1.0000	1.0000	1.0000	1.0000	1.0000	0.7737
30	1.0000	1.0000	1.0000	1.0000	1.0000	0.9667

Table X
P_d VALUES W.R.T. m' AND ρ (MLM AND NETFLIX, TWO CHEAT)

Next, we fix a number of (ρ, n') pairs and study the detection rates P_d in step 2. When one server cheats, Tables XI and XII summarize the results.

n' \ ρ	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}
1	1.0000	0.9994	0.9753	0.8426	0.6033	0.3701
2	1.0000	1.0000	0.9994	0.9753	0.8428	0.6033
3	1.0000	1.0000	1.0000	0.9961	0.9377	0.7506
4	1.0000	1.0000	1.0000	0.9994	0.9754	0.8430
5	1.0000	1.0000	1.0000	0.9994	0.9754	0.8430
6	1.0000	1.0000	1.0000	1.0000	0.9961	0.9378
7	1.0000	1.0000	1.0000	1.0000	0.9985	0.9609

Table XI
P_d VALUES W.R.T. n' AND ρ (MLM, ONE CHEATS)

n' \ ρ	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}
1	1.0000	1.0000	1.0000	0.9998	0.9862	0.8827
2	1.0000	1.0000	1.0000	1.0000	0.9998	0.9862
3	1.0000	1.0000	1.0000	1.0000	1.0000	0.9984
4	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
6	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Table XII
P_d VALUES W.R.T. n' AND ρ (NETFLIX, ONE CHEATS)

It is clear that, for both datasets, even setting a small n' and very low cheating rate, P_d values are very big. Due to its lower density, the P_d values for the Netflix dataset are larger. When both server cheat, the P_d values will even be bigger than those from these tables. Due to space limitation, we skip the details here.

VI. Conclusion

In this paper, we have mainly focused on the integrity protection of weighted Slope One. It is a future work to apply the proposed approaches to other recommender algorithms. It is a widely open problem to investigate lightweight solutions for rigorous protection of confidentiality in outsourcing. A related problem is to achieve confidentiality and integrity simultaneously. With homomorphic encryption, it is straightforward to have integrity because we can add a few dummy records (e.g. with all zeros) to verify integrity. However, it becomes harder if we desire lightweight solutions without completely relying on inefficient cryptographic primitives such as homomorphic encryption schemes.

Acknowledgements

Qiang Tang and Husen Wang are supported by a CORE grant from the National Research Fund, Luxembourg. Qiang Tang and Balázs Pejó are supported by an internal project from University of Luxembourg. We thank Jun Wang for his helpful comments.

References

[1] J. W Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*, pages 45–64. Springer, 2013.

[2] J. F. Canny. Collaborative filtering with privacy. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 45–57. IEEE, 2002.

[3] S. S. Roy and K. Jarvinen and F. Vercauteren and V. Dimitrov and I. Verbauwhede. Modular hardware architecture for somewhat homomorphic function evaluation. In *Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Proceedings*, volume 9293, page 164. Springer, 2015.

[4] B. Dong, R. Liu, and H. W. Wang. Result integrity verification of outsourced frequent itemset mining. In *Data and Applications Security and Privacy XXVII*, pages 258–265. Springer, 2013.

[5] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *SDM*, volume 5, pages 1–5. SIAM, 2005.

[6] R. Liu, H. W. Wang, P. Mordohai, and H. Xiong. Integrity verification of k-means clustering outsourced to infrastructure as a service (iaas) providers. In *SDM*, pages 632–640, 2013.

[7] F. McSherry and I. Mironov. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636. ACM, 2009.

[8] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 801–812. ACM, 2013.

[9] G. Sheng, T. Wen, Q. Guo, and Y. Yin. Verifying correctness of inner product of vectors in cloud computing. In *Proceedings of the 2013 international workshop on Security in cloud computing*, pages 61–68. ACM, 2013.

[10] J. Vaidya, I. Yakut, and A. Basu. Efficient integrity verification for outsourced collaborative filtering. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 560–569. IEEE, 2014.

[11] W. K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis. An audit environment for outsourcing of frequent itemset mining. *Proceedings of the VLDB Endowment*, 2(1):1162–1173, 2009.

[12] I. Yakut and H. Polat. Arbitrarily distributed data-based recommendations with privacy. *Data & Knowledge Engineering*, 72:239–256, 2012.

[13] Q. Tang and B. Pejo. Protect both Integrity and Confidentiality in Outsourcing Collaborative Filtering Computations. Available at http://eprint.iacr.org/2016/079