Protect both Integrity and Confidentiality in Outsourcing Collaborative Filtering Computations

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

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

Husen Wang
SnT, 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 and privacy for end users, both the integrity of computation results and the confidentiality of original dataset need 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 (with better performances). We then propose a new solution to outsource all computations of the weighted Slope One algorithm in multi-server setting and provide experimental results. We finally discuss the possibility of using homomorphic encryption to achieve both integrity and confidentiality guarantees.**

***Keywords*-Collaborative filtering; Outsourcing; 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. It has been proven to be quite effective in reality. 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. The computational challenge might not be a problem for organizations such as Amazon or Google, but it may be a burden for organizations which either do not have the necessary computation resources or do not want to manage such resources. 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 the cloud. For example, Netflix outsources its collaborative computations to Amazon. When the computations are outsourced, two issues arise. One is the integrity of the computation results. The cloud server may provide some fake results instead of spending its resources to compute the correct ones. Motivations behind such misbehavior could differ, but saving its own cost and deliberately disrupting the recommendation service are the two obvious ones that we foresee. The other one is confidentiality and more generally privacy issue. Many users consider their ratings as sensitive information, which leaks what they have done and their preferences. This is particularly a concern if the recommendation service is for scenarios like healthcare. We argue that both integrity and confidentiality are desirable in practice.

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], Jaideep 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. With respect to confidentiality (or privacy) protection in outsourcing collaborative filtering computations, there have been indirectly-related research results, including cryptographic ones (e.g. [2], [8]) and obfuscation-based ones (e.g. [7], [12]).

A. Problem Statement and Our Contribution

The problem setting is as follows. The recommender service provider RecSys possesses a rating dataset (i.e. a rating matrix shown in Fig. I) 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 and the confidentiality of rating dataset.

In this paper, we focus on the weighted Slope One recommender algorithm from [5] for its nice structure. Nevertheless, our discussions can be adapted to other algorithms. Starting with two integrity verification approaches mentioned by Vaidya et al. [10], our main contribution is the following.

- We first analyze the cheating and detection strategies in four scenarios, and figure out the best strategies for the RecSys and the cloud server. Based on the analysis results, we then demonstrate the performances of the verification via splitting approach for the Movielens MLM (i.e. MovieLens 1M) dataset[1] and the Netflix dataset. In particular, we show a tradeoff between confidentiality and verification efficiency.
- We then analyze the verification via auxiliary approach and propose a simplified variant. Based on the experimental results, we demonstrate that the proposed simplified variant is more effective from the perspective of the RecSys, while it introduces some mild overhead for the cloud server.
- We finally propose a solution to outsource the computations in both stages for the weighted Slope One algorithm in multi-server setting. In this setting, we show that the RecSys can minimize its verification cost and leverage the servers to verify the integrity of outsourced computations.

B. Organization

In Section II, we review the problem of outsourcing weighted Slope One algorithm. In Section III, we present new analysis results for the verification via splitting approach. In Section IV, we analyse the verification via auxiliary data approach, present a variant and show comparison results. In Section V, we present a new oursourcing solution in multi-server setting. In Section VI, we conclude the paper.

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 $\mathbf{R}_x = (r_{x,1}, \ldots, r_{x,b}, \ldots, r_{x,M})$. 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, as shown in Table I. The functionality of a recommender system is to predict the unrated values. With respect to $\mathbf{R}_x$, a binary vector $\mathbf{Q}_x = (q_{x,1}, \ldots, q_{x,b}, \ldots, q_{x,M})$ 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 $d = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M} q_{i,j}}{MN}$.

	Item 1	$\cdots$	Item i	$\cdots$	item M
User 1 ($\mathbf{R}_1$)	$r_{1,1}$	$\cdots$	$r_{1,i}$	$\cdots$	$r_{1,M}$
User 2 ($\mathbf{R}_2$)	$r_{2,1}$	$\cdots$	$r_{2,i}$	$\cdots$	$r_{2,M}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
User N ($\mathbf{R}_N$)	$r_{N,1}$	$\cdots$	$r_{N,i}$	$\cdots$	$r_{N,M}$

Table I
Rating Matrix

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 a 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 elements located in i-th row and j-th column) are defined as follows.

$$\phi_{i,j} = \sum_{u=1}^{N} q_{u,i}q_{u,j}, \quad \delta_{i,j} = \sum_{u=1}^{N} q_{u,i}q_{u,j}(r_{u,i} - r_{u,j})$$

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*. This stage requires $d^2N\frac{M^2-M}{2}$ subtractions and $(d^2N-1)\frac{M^2-M}{2}$ additions. Clearly, the computation of $\Phi_{M\times M}$ matrix is very cheap, so that we always assume RecSys will compute it locally.

Prediction stage. The prediction 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 weight Slope One algorithm is

$$p_{x,i} = \frac{\sum_{j\in\mathbf{B}/\{i\}} \delta_{i,j} + r_{x,j}\phi_{i,j}}{\sum_{j\in\mathbf{B}/\{i\}} \phi_{i,j}}$$

The prediction $p_{x,i}$ incurs the following computations: $dM - 1$ multiplications, $(2 + d)M - 5$ additions and one division. In total, $(1 - d)NM$ predictions needs to be calculated for all users.

	Computation Stage	Prediction Stage
+	$(d^2N-1)\frac{M^2-M}{2}$	$(1-d)NM((2+d)M-5)$
−	$d^2N\frac{M^2-M}{2}$	dNM
×		$(1-d)dN(M^2-M)$
/		$(1-d)NM$

Table II
Complexity of Weighted Slope One

Based on the above analysis, the required computations are summarized in Table II. It seems that the computation stage costs the RecSys much less resources

[1] http://grouplens.org/datasets/movielens/

than the prediction stage for both algorithms. To validate this intuition, we carry out an experiment based on the well-known Movielens MLK (i.e. MovieLens 100K) and MLM (i.e. MovieLens 1M) datasets and the Netflix dataset, detailed in Table III. Table IV presents the running time on Intel(R) Xeon(R) CPU E3-1241 v3 @ 3.50GHz using 15 GB RAM.

	Users (N)	Items (M)	Ratings	Density (d)
MLK	943	1.682	100.000	0.063
MLM	6.040	3.952	1.000.209	0.042
Netflix	480.189	17.770	100.480.507	0.012

Table III
Dataset Characteristics

	Computation Stage	Prediction Stage
MLK	17	67
MLM	360	1.620
Netflix	135.847	423.125

Table IV
Implementation Results (seconds)

It confirms that the prediction stage costs much more resources. So, it makes sense to outsource the computations in both stages when the RecSys needs to compute the predications for many of its users.

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. Suppose $\delta_{i,j}^{(k)}$ 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** *cheating strategy.* The server randomly picks up α ($1 \leq \alpha \leq M$) items and sets the deviation values associated with these items to be random numbers in all the blocks. It is easy to check that the cheating will affect $\frac{2\alpha M - \alpha^2 - \alpha}{2}$ deviation values. The cheating rate is $\frac{2\alpha M - \alpha^2 - \alpha}{M(M-1)}$. *verification strategy.* For every block, the RecSys randomly selects θ ($1 \leq \theta \leq \frac{M(M-1)}{2}$) deviation values to verify by re-computing these values. The detection rate is $1 - \left(\binom{\frac{(M-\alpha)(M-\alpha-1)}{2}}{\theta} \Big/ \binom{\frac{M(M-1)}{2}}{\theta}\right)^r$. The verification cost is $r\theta$ deviation computations.
- **Scenario 2** *cheating strategy.* The server randomly picks up in total β ($1 \leq \beta \leq N$) users/rows from all blocks and discards their values in the computation. Suppose the union size of the rated items from these β users is t, then the number of affected deviation values is upper-bounded by $\frac{2tM - t^2 - t}{2}$. We emphasize that this bound is very loose, especially when the density d is small. The cheating rate is $\frac{\beta}{N}$. *verification strategy.* For every block, the RecSys randomly selects θ deviations to verify by recomputing these values. The detection rate is upper-bounded by $1 - \left(\binom{\frac{(M-t)(M-t-1)}{2}}{\theta} \Big/ \binom{\frac{M(M-1)}{2}}{\theta}\right)^r$. As we emphasized above, the actual detection rate can be much smaller than this bound because the unaffected deviation values could be much larger than $\frac{(M-t)(M-t-1)}{2}$. The verification cost is $r\theta$ deviation computations.
- **Scenario 3** *cheating strategy.* The server randomly picks up γ ($1 \leq \gamma \leq r$) blocks and sets the related deviations to be random numbers. The cheating rate is $\frac{\gamma}{r}$. *verification strategy.* The RecSys randomly selects δ ($1 \leq \delta \leq r$) blocks to verify by re-computing one deviation value for each of these blocks. In this case, the detection rate is $1 - \binom{r-\gamma}{\delta} \Big/ \binom{r}{\delta}$. The verification cost is δ deviation computations.
- **Scenario 4** *cheating strategy.* For every block, the server cheats with probability ρ and sets the related deviation values to be random numbers. On average, the cheating rate is ρ. *verification strategy.* The RecSys randomly selects δ ($1 \leq \delta \leq r$) blocks to verify by re-computing one deviation value for each of these blocks. The detection rate is simply $1 - (1-\rho)^\delta$. The verification cost is only δ deviation computations.

Below, we compare these four scenarios in a pairwise manner, by considering the following factors: the cheating rate of the server, the detection rate of the RecSys, and the verification cost of the RecSys. For any given cheating rate for the server, the RecSys would prefer high detection rate and low verification cost. Clearly, the server prefers solutions with opposite properties.

- For Scenario 1 and Scenario 2, if we set $\frac{2\alpha M - \alpha^2 - \alpha}{M(M-1)} = \frac{comp(N,M,r) - comp(N-\beta,M,r)}{comp(N,M,r)}$ then the server has the same cheating rate. To this end, $\beta = \frac{(2\alpha M - \alpha^2 - \alpha)N}{M(M-1)}$. Furthermore, for the same verification cost of $r\theta$ deviation

computations, we only need to decide whether $\alpha > t$. If so, then the RecSys has higher detection rate in Scenario 1. Unfortunately, in Scenario 2, we do not have a precise formula and can only upper-bound the detection rate. Nevertheless, we have $t \leq d\beta = \frac{d(2\alpha M - \alpha^2 - \alpha)N}{M(M-1)}$ in this scenario. We can conclude that if $dN < \frac{M}{2}$ then $t < \alpha$ so that the RecSys has higher detection rate in Scenario 1.

- For Scenario 3 and Scenario 4, if we set $\rho = \frac{\gamma}{r}$ then the server has the same average cheating rate. Furthermore, for the same verification cost of δ deviation computations, we have $1 - \binom{r-\gamma}{\delta} \Big/ \binom{r}{\delta} > 1 - (1-\rho)^\delta$. This implies that the RecSys will prefer Scenario 3, while the server will prefer Scenario 4.
- For Scenario 1 and Scenario 3, if we set $\frac{2\alpha M - \alpha^2 - \alpha}{M(M-1)} = \frac{\gamma}{r}$ then the server has the same cheating rate. Furthermore, if we set $r\theta = \delta$ then the RecSys has the same verification cost. For any $\theta \geq 1$, we have $\delta \geq r$ so that the RecSys has detection rate 1 in Scenario 3. This means that the RecSys will always prefer Scenario 3, while the server will prefer the Scenario 1.
- For Scenario 1 and Scenario 4, if we set $\frac{2\alpha M - \alpha^2 - \alpha}{M(M-1)} = \rho$ then the server has the same cheating rate. Furthermore, if we set $r\theta = \delta$ then the RecSys has the same verification cost. Since $\delta \leq r$, then the RecSys has the same verification cost when $\theta = 1$. In this case, if the server has ever cheats, the RecSys has the detection rate 1 in Scenario 4. Note that, in Scenario 4, if the server cheats then the detection rate cannot be computed as $1-(1-\rho)^\delta$ any more. In contrast, the RecSys has a lower detection rate in Scenario 1. This means that the RecSys will prefer Scenario 4, while the server will prefer the Scenario 1.
- For Scenario 2 and Scenario 4, we can use the same reasoning as in the previous comparison to conclude that the RecSys will prefer Scenario 4 while the server will prefer the Scenario 2.

From the comparisons, the RecSys will prefer Scenario 3 while the server will prefer Scenario 2 or Scenario 1 the most. Furthermore, we can draw the following informal conclusions. For the RecSys, the best strategy is to first randomly choose some blocks and then randomly choose some deviation values in these blocks to verify. For the server, the best strategy is to first randomly choose some blocks and then set some deviation values in these blocks to 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

In the following, we use MLM and Netflix datasets as examples to study the performances. Referring to the previous discussions, we assume the RecSys and the server adopt the following strategies respectively.

- *cheating strategy*. 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.
- *verification strategy*. In every block, the RecSys randomly chooses θ intermediate deviation values to verify. Moreover, 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

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

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[2]. In the experiment, we fix a number of (ρ, θ) pairs and summarize the P_d values in Tables V and VI. In Table VI, 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 V
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 VI
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

[2]The detection rates for the Netflix dataset are almost the same to those in in Tables V and VI. We skip the details here.

blocks. We summarize the verification costs for the MLM and Netflix datasets in Tables VII and VIII.

$r = 1$	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 VII
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 VIII
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 VI.

C. Verification via Splitting with Confidentiality

The other concern for both $r = 1$ and $r > 1$ is that, after receiving the blocks of D, the server knows that each row of these blocks are the ratings of a user for the list of items. This may make it easy for the server to re-identify the user. To mitigate this risk, the RecSys can randomly and independently permute the columns of every block before sending them to the server. By doing so, it becomes difficult for the server to figure the correspondence between the permuted ratings and the items. The downside is that the server is not able to compute the final deviation values $\delta_{i,j}$ $(1 \le i, j \le M)$ anymore. The RecSys needs to combine these values based on the permutation information. The complexity of combining the intermediate results takes $2(r-1)\frac{M^2-M}{2}$ additions. For the MLM and Netflix datasets, the running time is 36 and 740 seconds respectively. Clearly, this complexity is much higher than the verification costs in Tables VII and VIII.

This implies that for the MLM and Netflix datasets, it is better to set $r = 1$ and permute the columns for the sake of verification efficiency and confidentiality.

IV. Examining the Verification via Auxiliary Data Approach

A. Recap the Original Approach

The original 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'.

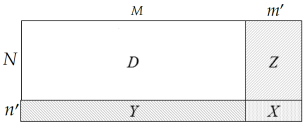

Figure 1. Splitting I

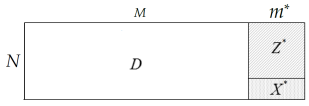

Figure 2. Splitting II

The cost for the RecSys is mainly generating the synthetic data and computing deviation values based on X, and pre-computation can be done.

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 IX and X 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 IX
VALUES FOR MLM DATASET

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

Table X
VALUES FOR NETFLIX DATASET

Next, we fix some cheating and detection rates and figure out the m^* values. 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.

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

With respect to the MLM and Netflix datasets, we fix a number of (ρ, m^*) pairs and summarize the detection rates P_d in Table XI. 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.

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 XI
P_d VALUES FOR MLM AND NETFLIX DATASETS

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 (see Section VI).

Referring to Tables V, VI, and XI, we can roughly say that $m^* = 20$ in the verification via auxiliary 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 XII. With respect to the verification via splitting approach, we take the minimum values from from the last columns in Tables VII and VIII; 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 XII
COST COMPARISON FOR RECSYS (SECONDS)

The downside is that it incurs some overhead for the server, 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 MULTI-SERVER SETTING

The discussions in Section III and IV assume a single-server setting. In this section, we mainly aim at a solution to outsource both stages of the weighted Slope One algorithm in the two-server setting. We also remark on extending the solution further.

A. Outsourcing to Two Servers

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 de-permutation 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 $1 \le i,j \le M$, $\delta_{i,j} = \delta^{(1)}_{i,j} + \delta^{(2)}_{i,j}$.

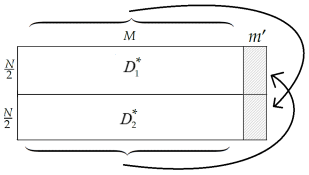

Figure 3. Splitting I

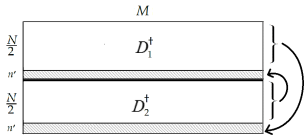

Figure 4. Splitting II

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 only about $\frac{M(M-1)}{2}$ additions. The computational complexity for every server is shown in Table XIII.

	Step 1	Step 2
+	$\frac{(d^2\frac{N}{2}-1)((M+m')^2-(M+m'))}{2}$	$\frac{(1-d)(\frac{N}{2}+n')M((2+d)M-5)}{2}$
−	$\frac{d^2\frac{N}{2}((M+m')^2-(M+m'))}{2}$	
×		$\frac{(1-d)d(\frac{N}{2}+n')(\lvert\mathbf{B}\rvert^2-M)}{2}$
/		$\frac{(1-d)(\frac{N}{2}+n')M}{2}$

Table XIII
Computational Complexity for Individual Server

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 XIV summarizes the results when only one server cheats. Table XV 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.

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

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 XIV
P_d WIR.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 XV
P_d W.R.T. m' AND ρ (MLM AND NETFLIX, TWO CHEAT)

Tables XVI and XVII summarize the results for the MLM dataset and the Netflix Dataset, respectively.

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 XVI
P_d 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 XVII
P_d 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.

B. Outsourcing to More Servers

The proposed solution in the previous subsection can be naturally extended to k-server setting. Based on the experimental results, it is clear that there will be not much gain in step 2 because adding one or two rows have already made the detection rates P_d very big. The only interesting question is on the detection rates in step 1. Below, we take the 3-server setting as an example. We split the matrix D as in Fig. 5 and assume similar outsourcing operations to those in step 1 of the 2-server setting.

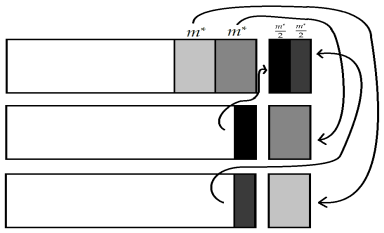

Figure 5. Splitting Matrix D for Three Servers

Table XVIII summarizes the results when only one server cheats, and the P_d values are smaller than those in Table XIV. Table XIX summarizes the results when all servers cheat, and the P_d values are bigger than those in Table XV. This means that 3-server setting is not always better than the 2-sever setting, depending on how many servers cheat.

m' \ ρ	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-6}	2^{-9}
2	0.5833	0.3125	0.1615	0.0820	0.0208	0.0026
4	0.9974	0.9273	0.7195	0.4671	0.1449	0.0193
6	1.0000	0.9984	0.9569	0.7954	0.3320	0.0495
10	1.0000	1.0000	0.9999	0.9897	0.7015	0.1441
16	1.0000	1.0000	1.0000	1.0000	0.9551	0.3405
20	1.0000	1.0000	1.0000	1.0000	0.9917	0.4814
30	1.0000	1.0000	1.0000	1.0000	1.0000	0.7708

Table XVIII
P_d 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	0.9998	0.9818	0.8557	0.3765	0.0570
6	1.0000	1.0000	1.0000	0.9935	0.7072	0.1414
10	1.0000	1.0000	1.0000	1.0000	0.9772	0.3745
16	1.0000	1.0000	1.0000	1.0000	1.0000	0.7183
20	1.0000	1.0000	1.0000	1.0000	1.0000	0.8665
30	1.0000	1.0000	1.0000	1.0000	1.0000	0.9903

Table XIX
P_d W.R.T. m' AND ρ (MLM AND NETFLIX, THREE CHEAT)

VI. PROTECTING CONFIDENTIALITY AND CONCLUDING REMARKS

With the integrity verification approaches, in particular the verification via auxiliary data approach, certain levels of confidentiality can be achieved. Without background knowledge, from the permuted rating matrix it is difficult to figure out the correspondence

between rating values and items. However, if an attacker has some background knowledge, there is less guarantee. To achieve a higher level of confidentiality protection, the RecSys can encrypt the rating matrix and let the server compute the predictions based on the encrypted data. Note that this can even allow users to hide their rating values from the RecSys. For efficiency reasons, the RecSys may encrypt the non-zero rating values, and randomly permute the encrypted values to hide which items have been rated by a certain user. From the implementation in [1], we get the timing cost for homomorphic addition (0.024 ms), homomorphic multiplication (31ms), based on an Intel Core i7-3520M at 2893.484 MHz. For the MLK dataset, the estimated complexity for the server is 548.5 hours. If the server has hardware implementation based on FPGA [3], then it will have an acceleration factor of 24.5 times. For a powerful server with hardware implementations, we may have a relatively optimistic solution for the MLK dataset. However, efficiency will remain be an issue with respect to the MLM and Netflix datasets.

It is an 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 homomorphic encryption schemes. Another open problem is to see how exactly the proposed approaches can be applied to recommender algorithms other than weighted Slope One.

Acknowledgements

Qiang Tang (partially) and Husen Wang are supported by a CORE (junior track) grant from the National Research Fund, Luxembourg. Qiang Tang (partially) and Balázs Pejó are supported by an internal project from University of Luxembourg.

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] 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.