Privacy-Preserving Decentralised Singular Value Decomposition

Bowen Liu and Qiang Tang

Luxembourg Institute of Science and Technology (LIST),
5, Avenue des Hauts-Fourneaux, L-4362, Esch-sur-Alzette, Luxembourg
{bowen.liu, qiang.tang}@list.lu

Abstract. With the proliferation of data and emerging data-driven applications, how to perform data analytical operations while respecting privacy concerns has become a very interesting research topic. With the advancement of communication and computing technologies, e.g. the FoG computing concept and its associated Edge computing technologies, it is now appealing to deploy decentralized data-driven applications. Following this trend, in this paper, we investigate privacy-preserving singular value decomposition (SVD) solutions tailored for these new computing environments. We first analyse a privacy-preserving SVD solution by Chen et al., which is based on the Paillier encryption scheme and some heuristic randomization method. We show that (1) their solution leaks statistical information to an individual player in the system; (2) their solution leaks much more information when more than one players collude. Based on the analysis, we present a new solution, which distributes the SVD results into two different players in a privacy-preserving manner. In comparison, our solution minimizes the information leakage to both individual player and colluded ones, via randomization and threshold homomorphic encryption techniques.

Keywords: Internet of Things · Fog Computing · Edge Computing · Singular Value Decomposition · Paillier Encryption · Threshold Cryptosystem

1 Introduction

Internet of Things (IoT) is increasingly appearing in our lives, which promises to connect everyone with everything from everywhere. In practice, IoT generates a large amount of data that is closely related to the human users (or, owners) of the devices. On the positive side, such data can be used for many useful purposes such as building smart services. However, on the other side, it brings huge privacy risks [13]. In most applications, the root of the privacy issue lies in the fact that data needs to be aggregated to a must-to-be trusted service provider before any service can be provided.

To mitigate the privacy concerns resulted from privacy-invasive data aggregation in general, information security researchers and cryptographers have been

advocating privacy-preserving distributed protocols for decades. Unfortunately, these protocols do not appeal to the real-world applications, which are often designed for the cloud computing paradigm that essentially requires data aggregation to a central third-party server. Recently, with the advancement of communication infrastructure (e.g. 5G) and computing technologies such as the FoG computing concept and its associated Edge computing technologies, it has become a trend to design and deploy decentralised applications, which push computations to the edge so that it avoids data aggregation to some extent. Coined by Numhauser in 2011 [1], there are many (similar) definitions for FoG computing. For example, Cisco [4] defines FoG computing as a paradigm that extends cloud computing [8] and services to the edge of the network, shown in Figure 1. In reality, FoG Computing can process its services at different nodes in the network as opposed to a central server. It significantly decreases the data movement across the network, so as to reduce the congestion, cost, and latency, and it also provides a decentralised infrastructure that naturally facilitates privacy protection. While many FoG-based applications are emerging, it is becoming an interesting research topic to design privacy-preserving solutions for the data exploitation tasks in these applications.

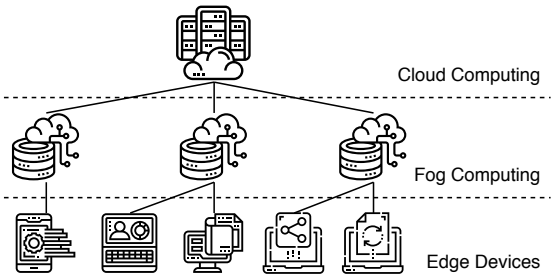

Fig. 1. FoG Computing

1.1 Related Work and Problem Statement

In this paper, we are interested in how to perform Singular Value Decomposition (SVD) based on data from decentralized IoT devices. In machine learning and data mining, SVD is a powerful and fundamental matrix factorization technique. It provides a means to decompose a matrix into a product of three simpler matrices, so that one may discover useful and interesting properties of the original matrix [6]. It finds many applications in reality. One prominent example is recommender systems [9] which are the cornerstone for many personalization services.

Chen et al. [3] proposed a privacy-preserving FoG computing framework for SVD computation. In their solution, the result of SVD is separated into two parts and stored at two different nodes. As a result, if an attacker compromises only one node, then it does not learn everything. Unfortunately, there is no formal analysis in [3]. Han, Ng, and Yu [6] provided a solution for performing

SVD in partitioned dataset, where two players collaborate with each other to perform SVD based on their joint dataset. Their solution is based on a number of cryptographic primitives, and the authors concluded that it is a challenge to reduce the complexity when their solution is used for large dataset. For privacy-preserving matrix factorization (MF) in general, Nikolaenko et al. [10] proposed a garbled circuit-based protocol and Kim et al. [7] proposed a protocol based on homomorphic encryption. In both solutions, the factorization operation is decentralized to two non-colluding servers, one of them controls key materials while the other carries out the factorization operation. When the two servers are compromised at the same time, then everything is leaked. Some further examples include Zhuo et al. [15] utilized full homomorphic encryption to achieve privacy-preserving data statistics on crowd-sourcing data and Zhou et al. [14] utilized one-way trapdoor permutation to realize time series data aggregation for wireless wearable communications.

Despite the literature work, it remains an open problem to design an efficient privacy-preserving SVD (or MF in general) protocol that provides rigorous security even when several players are compromised simultaneously.

1.2 Contribution and Organization

In this paper, we aim at rigorous privacy-preserving SVD solutions in the FoG computing paradigm. Firstly, we review the solution by [3] in Section 3 and demonstrate several privacy risks in Section 4. Our analysis covers a number of scenarios, including individual player/attacker and more colluded players/attackers. Moreover, we also briefly analyse the recommender use case. Based on the analysis, we simplify the FoG architecture from [3] and present a stronger security model, which captures scenarios where several players may collude. We then propose a new solution based on threshold homomorphic encryption and push some of the heavy computation workloads to the edge. We further analyse the security properties and the asymptotic complexities, as well as benchmarking results on a PC. These results appear in Section 5. In addition, we present some preliminary in Section 2 and draw some conclusions in Section 6.

2 Preliminary on Singular Value Decomposition

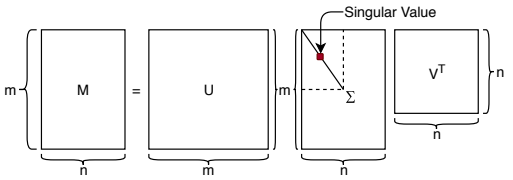

Fig. 2. Singular Value Decomposition

Let $\mathbb{M}$ be a $m \times n$ matrix. As shown in Figure 2, the SVD of $\mathbb{M}$ is a factorization of the form $\mathbb{U}\Sigma\mathbb{V}^T$, where $\mathbb{U}$ is an $m \times m$ left-singular matrix of $\mathbb{M}$, Σ is an

$m \times n$ singular matrix of $\mathbb{M}$, $\mathbb{V}$ is a $n \times n$ right-singular matrix of $\mathbb{M}$, and T means conjugate transpose. In addition, there are also two relations:

$$\mathbb{M} \cdot \mathbb{M}^T = \mathbb{U}\Sigma\mathbb{V}^T \cdot \mathbb{V}\Sigma^T\mathbb{U}^T = \mathbb{U}\Sigma\Sigma^T\mathbb{U}^T;\ \mathbb{M}^T \cdot \mathbb{M} = \mathbb{V}\Sigma^T\mathbb{U}^T \cdot \mathbb{U}\Sigma\mathbb{V}^T = \mathbb{V}\Sigma^T\Sigma\mathbb{V}^T$$

The columns of $\mathbb{U}$ (left-singular vectors) and $\mathbb{V}$ (right-singular vectors) are, respectively, eigenvectors of $\mathbb{M} \cdot \mathbb{M}^T$ and $\mathbb{M}^T \cdot \mathbb{M}$. The non-zero elements of Σ are the square roots of the non-zero eigenvalues of $\mathbb{M} \cdot \mathbb{M}^T$ and $\mathbb{M}^T \cdot \mathbb{M}$.

A prominent application of SVD is recommender systems. Using the users' rating data, a service provider can recommend other films that they might like. As a toy example, suppose we have 3 users' score records for 4 movies (0 for the case not rated), see Table 1.

	Movie A	*Movie B*	*Movie C*	*Movie D*
User 1	*3*	*0*	*0*	*4*
User 2	*4*	*0*	*1*	*0*
User 3	*0*	*4*	*3*	*0*

Table 1. Rating Matrix

For this toy example, after applying SVD to the rating matrix, we can obtain three singular matrices denoted as $\mathbb{U}$, Σ and $\mathbb{V}^T$.

$$\mathbb{U} = \left(\begin{array}{cc|c} -0.784 & 0.243 & 0.571 \\ -0.588 & 0.000 & -0.809 \\ -0.196 & -0.970 & 0.143 \end{array}\right) \quad \Sigma = \begin{pmatrix} 5.831 & 0.000 & 0.000 & 0.000 \\ 0.000 & 5.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 2.828 & 0.000 \end{pmatrix} \quad \mathbb{V}^T = \left(\begin{array}{cccc} -0.807 & -0.135 & -0.202 & -0.538 \\ 0.146 & -0.776 & -0.582 & 0.194 \\ \hline -0.538 & 0.202 & -0.135 & 0.807 \\ 0.000 & 0.137 & -0.158 & 0.727 \end{array}\right)$$

In practice, a dimensionnality reduction procedure can be applied to generate small-rank feature matrices. Note that the eigenvalue 2.828 is smaller than the other two values, so that we can choose to suppress it and have a truncated variant of Σ, namely

$$\Sigma^* = \begin{pmatrix} 5.831 & 0.000 \\ 0.000 & 5.000 \end{pmatrix}$$

Accordingly, we generate a variant of $\mathbb{U}$ by removing its last column and a variant of $\mathbb{V}^T$ by removing its last two rows. Let the variants be denoted as $\mathbb{U}^*$ and $\mathbb{V}^{*T}$ respectively. By computing $\mathbb{U}^*\Sigma^{*\frac{1}{2}}$ and $\Sigma^{*\frac{1}{2}}\mathbb{V}^{*T}$, we obtain the user and item feature matrices shown in Table 2 and 3.

User 1	-1.893	0.543
User 2	-1.420	0.000
User 3	-0.473	-2.169

Table 2. User Feature Matrix

Movie A	*Movie B*	*Movie C*	*Movie D*
-1.949	-0.326	-0.488	-1.299
0.326	-1.735	-1.301	0.434

Table 3. Movie/Item Feature Matrix

Based on these feature matrices, a recommender service provider can serve users with predictions on the movies they have not rated.

3 Overview of Chen et al.'s Solution

In this section, we introduce the privacy-preserving solution by Chen et al. [3] and also briefly introduce its application to the recommender use case.

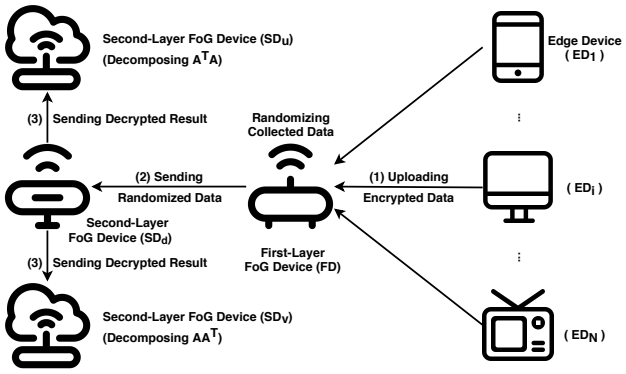

Fig. 3. Fog Computing Architecture

By assuming a FoG architecture, shown in Figure 3, Chen et al. [3] designed a privacy-preserving SVD solution based on some heuristic randomization techniques and the Paillier scheme, described in Appendix A. This FoG computing architecture consists of the following types of entities:

- *Server*: The initialization of the whole system is generated and operated by the server, it is considered as fully trusted node in the FoG architecture. Once, the initialisation is done, the server will not get involved anymore. So, we omit it in Figure 3.
- *Edge Devices ED*: Each edge device is the original collector of the data and represents the human user behind it.
- *First-layer FoG Device FD*: The FoG device is responsible for collecting the data from edge devices and coordinating the SVD operations.
- *Second-layer FoG Devices SD*s: There are three different devices in this category. SD_d decrypts the received information and obtains the randomized data matrix, and prepares data for SD_u and SD_v, who will perform the decomposition.

3.1 Description of the Solution

Parameter	*Propose*
N	*Number of Edge Devices (ED)*
l	*Dimension of data vector*
x	*Range of value in data vector: [0, x]*
k_1	*For generating* $t = 2^{k_1}$
W	*Randomized* $W > max(N, l) \cdot x^2$
S	*Randomized* $S > max(N, l) \cdot (x^2 + 2tWx + t^2W^2)$
k_2	*Bit length of* W
k_3	*Bit length of* S
$\vec{a}$	*For transforming vector into number*
k	*Length of p, q in Paillier cryptosystem*
$(n,\ g)$	*Public key of Paillier cryptosystem*

$(\lambda,\ \mu)$	*Private key of Paillier cryptosystem*

Table 4: Initialization Parameters

Initialisation. The trusted server will setup the parameters for the system, shown in Table 4. In terms of Paillier cryptosystem [11], the server generates the public key $(n = pq, g)$, and the private key (λ, μ). In the meanwhile, the server generates two random coprime numbers W and S respectively. Additionally, a super-increasing vector $\vec{a} = (a_1 = 1, a_2, ..., a_l)^T$ is generated, and each value needs to conform to the following conditions: $\sum_{j=1}^{i-1} a_j \cdot (x + tW + tS) < a_i,\ i \in [2, l]$ and $\sum_{j=1}^{l} a_j \cdot (x + tW + tS) < n$. The Paillier private key (λ, μ) is assigned to SD_d, and the private randomization parameters W and S are assigned to FD, SD_u, SD_v. All other parameters are public.

Privacy-preserving Protocol. Illustrated in Figure 3, the privacy-preserving protocol runs in four steps.

1. ED_i's data is expressed in vector form $\vec{d_i} = (d_{1i}, d_{2i}, ..., d_{li})^T$. Note that the data from EDs forms a matrix $\mathbb{A}$ as follows.

$$\mathbb{A} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1N} \\ d_{21} & d_{22} & \dots & d_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{l1} & d_{l2} & \dots & d_{lN} \end{bmatrix} \tag{1}$$

   By using the public vector $\vec{a}$, ED_i first converts its vector into an integer.

$$m_i = \vec{d_i}^T \cdot \vec{a} = a_1 d_{1i} + a_2 d_{2i} + ... + a_l d_{li}$$

   It then encrypts this integer with Paillier public key to obtain c_i. At the end of this step, ED_i, for every $1 \le i \le N$, sends c_i to FD.
2. After receiving every c_i, FD chooses two vectors $\vec{z_i} = (z_{1i}, z_{2i}, ..., z_{li})^T$ and $\vec{r_i} = (r_{1i}, r_{2i}, ..., r_{li})^T$, where each element is randomly chosen from $[1, t]$. It then computes the randomization parameter $R_i = \sum_{k=1}^{l} a_k \cdot (z_{ki} \cdot W + r_{ki} \cdot S)$, and sets the randomized ciphertext c_i' as

$$\begin{aligned} c_i' &= c_i \cdot g^{R_i} \ mod\ n^2 \\ &= (g^{m_i} \cdot r_i^n) \cdot g^{R_i} \ mod\ n^2 \\ &= (g^{\sum_{k=1}^{l} a_k \cdot d_{ki}} \cdot r_i^n) \cdot g^{\sum_{k=1}^{l} a_k \cdot (z_{ki} \cdot W + r_{ki} \cdot S)} \ mod\ n^2 \\ &= g^{\sum_{k=1}^{l} a_k \cdot (d_{ki} + z_{ki} \cdot W + r_{ki} \cdot S)} \cdot r_i^n \ mod\ n^2 \end{aligned}$$

   At the end of this step, FD sends c_i' $(1 \le i \le N)$ to SD_d.
3. After receiving every c_i', SD_d decrypts it to obtain

$$m_i' = \sum_{k=1}^{l} a_k \cdot (d_{ki} + z_{ki} \cdot W + r_{ki} \cdot S)\ mod\ n$$

   Next, by applying Algorithm 1, the randomized non-encrypted data in vector form can be computed.

Algorithm 1 Recover Vector from Integer

Input: m'_i and $\vec{a} = (a_1, a_2, ..., a_l)^T$
Output: randomized non-encrypted data $\vec{d'_i}$
1: let $tmp = (t_{a_1}, t_{a_2}, ..., t_{a_l})^T$ within empty value inside
2: $X_l \leftarrow m'_i = \sum_{j=1}^{l} a_k \cdot (d_{ji} + z_{ji} \cdot W + r_{ji} \cdot S)\ mod\ n$
3: **for** $k = l$ *to* 2 **do**
4: $\quad X_{k-1} \leftarrow X_k\ mod\ a_k$
5: $\quad t_{ak} \leftarrow \frac{X_k - X_{k-1}}{a_k} = d_{ki} + z_{ki} \cdot W + r_{ki} \cdot S$
6: **end for**
7: $t_{a_1} \leftarrow X_1 = d_{1i} + z_{1i} \cdot W + r_{1i} \cdot S$
8: **return** $\vec{d_i} = (t_{a_1}, t_{a_2}, ..., t_{a_l})^T$

Now, SD_d has the matrix $\mathbb{A}'$ made up by N different randomized non-encrypted vectors $\vec{d'_i} = (d'_{1i}, d'_{2i}, ..., d'_{li})^T$.

$$\mathbb{A}' = \left[\vec{d'_1}\ \vec{d'_2}\ \dots\ \vec{d'_N}\right] = \begin{bmatrix} d'_{11} & d'_{12} & \dots & d'_{1N} \\ d'_{21} & d'_{22} & \dots & d'_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d'_{l1} & d'_{l2} & \dots & d'_{lN} \end{bmatrix} \tag{2}$$

At last, SD_d sends $res_u = \mathbb{A}' \cdot \mathbb{A}'^T$ and $res_v = \mathbb{A}'^T \cdot \mathbb{A}'$ to SD_u and SD_v respectively.

4. With res_u, SD_u derandomizes every entry e'_{ij} as follows.

$$\begin{aligned} & e'_{ij}\ mod\ S\ mod\ W \\ =& (d_{ij} + z_{ij} \cdot W + r_{ij} \cdot S)\ mod\ S\ mod\ W \\ =& (d_{ij} + z_{ij} \cdot W)\ mod\ W \\ =& d_{ij} \end{aligned}$$

The resulting matrix is $\mathbb{A} \cdot \mathbb{A}^T$. Then, SD_u performs eigenvalue decomposition to obtain $\mathbb{U}$ and Σ of $\mathbb{A} \cdot \mathbb{A}^T$, referring to Section 2. Similarly, SD_v can obtain $\mathbb{V}$ and Σ of $\mathbb{A}^T \cdot \mathbb{A}$. At the end, SD_u and SD_v will store $\mathbb{U}\Sigma^{\frac{1}{2}}$ and $\mathbb{V}\Sigma^{\frac{1}{2}}$ respectively. Note that they might apply the dimension reduction procedure, as mentioned in Section 2 , to store smaller feature matrices.

3.2 Recommender Use Case

Referring to the recommender use case described in Section 2, Chen et al. [3] presented a procedure to extend the privacy-preserving SVD protocol to compute predictions of unrated items for the user of any edge device ED. Assuming this use case, the information that SD_u obtains at the end of privacy-preserving SVD protocol, namely $\mathbb{U}\Sigma^{\frac{1}{2}}$, can be regarded as the user feature matrix, where the i-th row is the feature vector of the user of ED_i. Correspondingly, the information that SD_v obtains at the end of privacy-preserving SVD protocol, namely $\mathbb{V}\Sigma^{\frac{1}{2}}$, can be regarded as the item feature matrix, where the j-th row is the feature vector of the j-th item.

If the user of ED_i wants to retrieve the prediction on the j-th item, the procedure is detailed below.

1. SD_u and SD_v randomize $InfoUi$ and $InfoIj$ with W and S, respectively, and send the randomized data to SD_d. Here, as described above, $InfoUi$ stands for the user feature vector of ED_i, while $InfoIj$ stands for the item feature vector of item j. As an example, the randomization of $InfoUi$ is shown in Algorithm 2.

Algorithm 2 Randomize data with W and S

Input: $InfoUi$, W, S and range of random number $[1, t]$
Output: randomized vector $InfoUi'$
1: **for** every element h_{zi} of $InfoUi$ where $1 \leq z \leq l$ **do**
2: Generate random integer x, y from $[1, t]$
3: $h'_{z1} \leftarrow= h_{z1} + x \cdot W + y \cdot S$
4: **end for**
5: **return** $InfoUi' = (h'_{1i}, h'_{2i}, ..., h'_{li})$

2. After receiving the randomized feature vectors $InfoUi'$ and $InfoIj'$, SD_d computes a randomized score $Score^{'} = InfoUi'\{InfoIj'\}^T$, and sends it to FD.
3. After receiving $Score'$, FD can derandomize it with W and S to obtain the plaintext prediction for ED_i:

$$Score = Score' \bmod S \bmod W = InfoUi \cdot InfoIj^T$$

4 Analysis of Chen et al.'s Solution

In [3], Chen et al. analysed all devices of the system, and claimed that none of them has the ability to learn private data under normal operations. However, we show that their solution has a number of vulnerabilities, it leaks information not only to an individual device but also colluded devices (more seriously).

4.1 Information Leakage to Individual Device

At the end of the privacy-preserving protocol, SD_v obtains $\mathbb{A}^T \cdot \mathbb{A}$ with following notation.

$$\mathbb{A}^T \cdot \mathbb{A} = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1N} \\ v_{21} & v_{22} & \dots & v_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ v_{N1} & v_{N2} & \dots & v_{NN} \end{bmatrix} \tag{3}$$

Leakage 1. We note that the values v_{ii}, for $i \in [1, N]$, are in the following form.

$$\begin{cases} v_{11} = d_{11}^2 + d_{21}^2 + ... + d_{l1}^2 \\ \quad \vdots \\ v_{NN} = d_{1N}^2 + d_{2N}^2 + ... + d_{lN}^2 \end{cases}$$

According to Cauchy-Schwarz inequality [12], based on v_{ii}, SD_v can deduce the upper bound about the average of the elements in $\vec{d_i}$ for each device ED_i, namely $\frac{\sum_{j=1}^{l} d_{ji}}{l} \leq (\frac{v_{ii}}{l})^{\frac{1}{2}}$.

Leakage 2. Taking the elements related to ED_1 as an example, SD_v possesses the following values.

$$\begin{cases} v_{11} = d_{11}^2 + d_{21}^2 + ... + d_{l1}^2 \\ \quad \vdots \\ v_{N1} = d_{1N}d_{11} + d_{2N}d_{21} + ... + d_{lN}d_{l1} \end{cases}$$

Adding all of them, SD_v can obtain $\sum_{j=1}^{N} v_{j1} = \sum_{j=1}^{N} d_{1j} \cdot d_{11} + \sum_{j=1}^{N} d_{2j} \cdot d_{21} + \cdots + \sum_{j=1}^{N} d_{lj} \cdot d_{l1}$. Based on the fact that every $d_{ji} \in [0, x]$, according to the law of large numbers [2], the above equality can be approximately written as

$$\frac{\sum_{j=1}^{N} v_{j1}}{N} = \frac{1+x}{2} \cdot d_{11} + \frac{1+x}{2} \cdot d_{21} + ... + \frac{1+x}{2} \cdot d_{l1}$$

Based on this, SD_v can obtain the approximate average of the elements in $\vec{d_1}$ for device ED_1, namely $\frac{2\sum_{j=1}^{N} v_{j1}}{N \cdot (1+x) \cdot l}$. Clearly, the same analysis applies to ED_i $(2 \leq i \leq N)$.

4.2 When two Devices Collude

If SD_u and SD_v collude, they possess $\mathbb{U}$, Σ and $\mathbb{V}$ which are the singular matrices of $\mathbb{A}$. It means they can restore all private data as $\mathbb{U} \cdot \Sigma \cdot \mathbb{V}^T = \mathbb{A}$. If SD_d and FD collude, they will possess W, S, and the Paillier private key (μ, λ). With this information, they can recover the private data of all EDs. Clearly, if SD_u or SD_v colludes with SD_d, they can also recover everything in the same way.

Next, we investigate the case that SD_v and one ED collude. Generally, let's assume if SD_v and ED_i collude. They possess $\vec{d_i}$ and $\mathbb{A}^T \cdot \mathbb{A}$ as defined in Equation (3).

$$\vec{d_i} = (d_{1i}, d_{2i}, ..., d_{li})^T$$

The elements from the i-th column of $\mathbb{A}^T \cdot \mathbb{A}$ are defined as follows.

$$\begin{cases} v_{1i} = d_{11}d_{1i} + d_{21}d_{2i} + ... + d_{l1}d_{li} \\ \vdots \\ v_{(i-1)i} = d_{1(i-1)}d_{1i} + d_{2(i-1)}d_{2i} + ... + d_{l(i-1)}d_{li} \\ \vdots \\ v_{(i+1)i} = d_{1(i+1)}d_{1i} + d_{2(i+1)}d_{2i} + ... + d_{l(i+1)}d_{li} \\ \vdots \\ v_{Ni} = d_{1N}d_{1i} + d_{2N}d_{2i} + ... + d_{lN}d_{li} \end{cases}$$

Below, we describe two simple active attacks.

Attack 1. Being an active attacker, ED_i can set its vector to be $\vec{d_i} = (1, 1, ..., 1)$. As a result, from the value v_{ji} where $j \in [1, N]$ and $j \neq i$, SD_v and ED_i can learn the average of the elements in the vector of ED_j.

$$\frac{v_{ji}}{l} = \frac{d_{1j} + d_{2j} + ... + d_{lj}}{l}$$

Attack 2. Similarly, ED_i can set its vector to be $\vec{d_i} = (0, ..., 1, ..., 0)$ where the y-th element is 1 and all others are 0. In this case, from the value v_{ji}, SD_v and ED_i can learn the the element d_{yj} in the vector of ED_j.

$$v_{ji} = d_{1j}d_{1i} + d_{2j}d_{2i} + ... + d_{yj}d_{yi} + ... + d_{lj}d_{li} = d_{yj}$$

4.3 When more Participants Collude

Attack 1. Suppose that SD_v and l (or more) EDs collude. If this case, they will possess $\mathbb{A}^T \cdot \mathbb{A}$ defined by Equations (3) and for example the following l $\vec{d}$s.

$$\begin{cases} \vec{d_i} = (d_{1i}, d_{2i}, ..., d_{li})^T \\ \vdots \\ \vec{d}_{(i+l-1)} = (d_{1(i+l-1)}, d_{2(i+l-1)}, ..., d_{l(i+l-1)})^T \end{cases}$$

With the information, SD_v may recover the data of any ED_j, for $j \notin [i, i + l-1]$. Note that the i-th to $i+l-1$-th elements from the j-th column of $\mathbb{A}^T \cdot \mathbb{A}$ are defined as follows. The attack is simply to solve the system of l-variable linear equations.

$$\begin{cases} v_{ij} = d_{1i}d_{1j} + d_{2i}d_{2j} + ... + d_{li}d_{lj} \\ \vdots \\ v_{N(i+l-1)} = d_{1(i+l-1)}d_{1j} + d_{2(i+l-1)}d_{2j} + ... + d_{l(i+l-1)}d_{lj} \end{cases}$$

By solving the system of equation, the attacker can obtain all the value of d_{aj} where $a \in [1, l]$.

Attack 2. Suppose SD_d and h EDs collude, where h is an integer, they will process $\mathbb{A}^{'}$ as defined by Equation (2). For example, consider the following h $\vec{d}$s.

$$\begin{cases} \vec{d_i} = (d_{1i}, d_{2i}, ..., d_{li})^T \\ \quad \vdots \\ \vec{d}_{(i+h-1)} = (d_{1(i+h-1)}, d_{2(i+h-1)}, ..., d_{l(i+h-1)})^T \end{cases}$$

For ED_j, where $j \in [i, i+h-1]$, the relevant elements in $\mathbb{A}^{'}$ lie in the j-th column which can be expressed as:

$$\begin{cases} d_{1j}^{'} = d_{1j} + z_{j1} \cdot W + r_{j1} \cdot S \\ \quad \vdots \\ d_{lj}^{'} = d_{lj} + z_{jl} \cdot W + r_{jl} \cdot S \end{cases}$$

These equations can be transformed into the following form shown in Equations (4), where zs, rs, and W, S are the unknowns.

$$\begin{cases} d_{1j}^{'} - d_{1j} = z_{j1} \cdot W + r_{j1} \cdot S \\ \quad \vdots \\ d_{lj}^{'} - d_{lj} = z_{jl} \cdot W + r_{jl} \cdot S \end{cases} \tag{4}$$

Since rs and zs are random integers chosen from $[1, t]$, according to the law of large numbers [2], we can get an approximated form of the following equation.

$$\frac{1}{l} \cdot \sum_{i=1}^{l} (d_{ij}^{'} - d_{ij}) = \frac{1+t}{2} \cdot W + \frac{1+t}{2} \cdot S$$

Since the computation is based on data from ED_j, we let P_j denote the approximated estimation for $W + S$.

$$P_j = W + S = \frac{2}{(1+t) \cdot l} \cdot \sum_{i=1}^{l} (d_{ij}^{'} - d_{ij})$$

Based on P_j for all $j \in [i, i+h-1]$, SD_d can try to recover W and S by a brute-force attack, shown in Algorithm 3.

Once W and S are found, the whole matrix $\mathbb{A}$ can be recovered. We emphasize that a brute-force attack has been analyzed in [3], where the attacker tries each possible value of S. The complexity of their attack is $O(2^{k_3})$. In contrast, our attack enumerates the parameter W which is much smaller than S (as shown in Table 4, $S > max(N, l) \cdot (x^2 + 2tWx + t^2W^2)$). The complexity of current brute-force method is only $O(2^{k_2}) + 2lh$.

Algorithm 3 Brute-force the value of W and S

Input: P, Range of W
Output: Value of W and S
1: Let LC denote the set of all results of $d_{ij}^{'} - d_{ij}$ where $i \in [1, l]$ and $j \in [1, N]$ (i.e. Equation (4) for ED_j)
2: **for each** f in range of W **do**
3: $S_{BF} \leftarrow P - f$
4: **for each** LC_k in LC **do**
5: $ModS_k \leftarrow LC_k$ modulo S_{BF}
6: **end for**
7: Let $ModS$ denote the set of all $ModS_k$
8: **if** $GCD(ModS) > 1$ and $coprime(GCD(ModS), S_{BF})$ **then**
9: **return** $W = GCD(ModS)$, $S = S_{BF}$
10: **end if**
11: **end for**

4.4 Analysis of the Recommender Use Case

Regarding the recommender use case from Section 3.2, we observe that it has two privacy vulnerabilities. One is that FD obtains the prediction score for the edge devices. The score indicates the interest of the human user behind the device, so that it may be considered as private information. Disclosing such information may be considered undesirable by many. The other is that SD_d obtains the randomized feature vectors for all prediction queries. Considering the attack from Section 4.3, SD_d may recover W and S, and then recover the plaintext data from all the devices.

5 New Privacy-Preserving SVD Solution

In this section, we first simplify the FoG architecture shown in Figure 3 and propose a stronger security model. Then, we present a new solution and provide detailed security and performance analysis.

5.1 Security Model

Our new FoG architecture is shown in Figure 4. In comparison to that in Figure 3, we get rid of the involvement of the second-layer SD_d device. With this new architecture, the second-layer devices SD_u and SD_v will store the decomposed matrices, while the first-layer device FD is responsible for interacting with the edge devices and coordinating the SVD operations.

The purpose of our solution is to perform SVD based on the private data from all edge devices ED_i $(1 \leq i \leq N)$, where ED_i's input is a data vector $\vec{d_i}$. Note that we assume every data vector is in a column form and all these data vectors form a data matrix $\mathbb{A}$, as defined in Equation (1). As the output, SD_u and SD_v should learn $(\mathbb{U}, \Sigma)$ and $(\mathbb{V}, \Sigma)$ respectively. Any other disclosure about the private information, including ED_i's data, $(\mathbb{U}, \Sigma)$ and $(\mathbb{V}, \Sigma)$, will be

considered as an information leakage. Referring to the description of SVD in Section 2, the legitimate information disclosure is equivalent to disclosing $\mathbb{A} \cdot \mathbb{A}^T$ and $\mathbb{A}^T \cdot \mathbb{A}$ to SD_u and SD_v respectively.

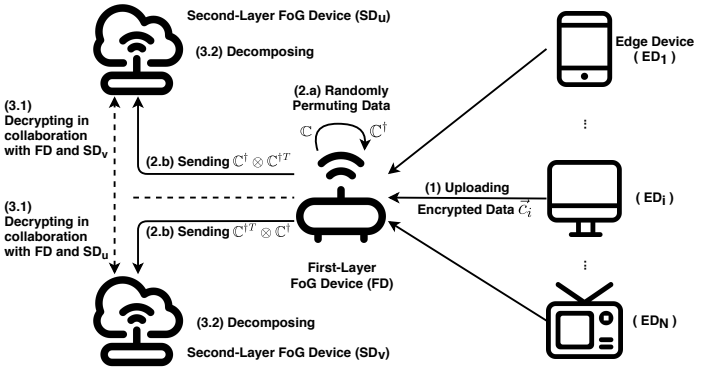

Fig. 4. Simplified FoG Architecture

Comparing with [3] and other distributed machine learning solutions, we will not make a general semi-honest assumption to ask all participants to follow the protocol specification and not to collude with each other. As we have put in our analysis, such an assumption is not realistic in practice, particularly some edge devices can be compromised or forged easily. Instead, we assume some players may collude and try to figure out information that they are not supposed to learn. Next, we enumerate all the attack scenarios and our privacy expectations.

1. When a group of edge devices is regarded as the attacker, it should learn nothing about the private data of other edge devices. This implies that the attacker learns nothing about $(\mathbb{U}, \Sigma)$ and $(\mathbb{V}, \Sigma)$ more than what it can infer from its own data.
2. When SD_u is regarded as the attacker, it only learns $(\mathbb{U}, \Sigma)$. When SD_v is regarded as the attacker, it only learns $(\mathbb{V}, \Sigma)$. When FD is regarded as the attacker, it learns nothing.
3. When SD_u and FD are regarded as the attacker (i.e. they collude), it only learns $(\mathbb{U}, \Sigma)$. When SD_v and FD are regarded as the attacker (i.e. they collude), it only learns $(\mathbb{V}, \Sigma)$. When SD_u and SD_v are regarded as the attacker (i.e. they collude), it learns a randomly permuted data matrix $\mathbb{A}^\dagger$, which is obtained by randomly permuting the rows and columns of $\mathbb{A}$. This means the attacker cannot trivially link a data vector to an edge device and cannot trivially recover the order of the elements in a data vector.
4. When FD and a group of edge devices are regarded as the attacker, it should learn nothing about the private data of other edge devices. This implies that the attacker learns nothing about $(\mathbb{U}, \Sigma)$ and $(\mathbb{V}, \Sigma)$ more than that it can infer from its own data.
5. When SD_u and a group of edge devices are regarded as the attacker, it should learn nothing more than what can be inferred from $\mathbb{A}^\dagger \cdot \mathbb{A}^{\dagger T}$ and the

data vectors of these edge devices. When SD_v and a group of edge devices are regarded as the attacker, it should learn nothing more than what can be inferred from $\mathbb{A}^{\dagger^T} \cdot \mathbb{A}^{\dagger}$ and the data vectors of these edge devices. Recall that $\mathbb{A}^{\dagger}$ is defined in bullet 3.

Note that we do not consider the scenarios, where all edge devices collude or all FoG devices (SD_u, SD_v and FD) collude, because in both cases the attacker will know everything by default in our setting.

In our construction, we will use threshold homomorphic encryption as the main building block, which guarantees that only the legitimate information will be decrypted and delivered to the corresponding parties. As such, when we say the solution does not leak any information about some private data α, it meant that if another piece data β will generate the same output as α then the attacker cannot determine whether α or β has been used as the input. It has the same flavor as the semantic security of the underlying homomorphic encryption scheme.

5.2 Description of the New Solution

Our main tool is a homomorphic encryption scheme, which supports partial homomorphic multiplication (between plaintext and ciphertext) and a polynomial number of homomorphic additions. In addition, we also require the scheme to allow us to support threshold decryption. To this end, the threshold Paillier scheme [5], described in Appendix B, satisfies our needs. This leads to the following initialisation for our new solution.

In the initialization stage, SD_u, SD_v and FD jointly set up the parameters of the threshold homomorphic encryption scheme. We assume the public key is pk, while the private key shares for the FoG devices are denoted as sk_u, sk_v and sk_f respectively. We require a (3, 3) threshold decryption setting, namely all three FoG devices need to collaborate in order to recover a plaintext message.

For the privacy-preserving SVD protocol, we will keep every data vector in encrypted form after leaving the edge devices, and threshold decryption is only carried out to recover the legitimate matrices for SD_u and SD_v respectively. Depicted in Figure 4, the protocol consists of four phases.

1. *Edge Computing* Phase: ED_i $(1 \leq i \leq N)$ uses the public key pk to encrypt its vector $\vec{d_i} = (d_{1i}, d_{2i}, ..., d_{li})^T$ into a ciphertext vector $\vec{c_i} = (c_{1i}, c_{2i}, ..., c_{li})^T$ where $c_{1i} = \mathbf{Enc}(d_{1i}, pk)$ and so on. Then, ED_i sends $\vec{c_i}$, an encrypted inner product $\mathbf{Enc}(\vec{d_i}^T \cdot \vec{d_i}, pk)$, and $\frac{l(l+1)}{2}$ encrypted scalar values $\mathbf{Enc}(d_{xi}d_{yi}, pk)$ $(1 \leq x \leq l, x \leq y \leq l)$ to FD. Note that these encryption operations can be done offline.
   After receiving $\vec{c_i}$ $(1 \leq i \leq N)$, FD will possess a ciphertext matrix $\mathbb{C}$, which is an encrypted counterpart of $\mathbb{A}$ defined in Equation (1).

$$\mathbb{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1N} \\ c_{21} & c_{22} & \cdots & c_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{lN} \end{bmatrix}$$

Next, FD computes the encrypted forms of $\mathbb{A}\mathbb{A}^T$ and $\mathbb{A}^T\mathbb{A}$, generally denoted as $\mathbb{C}\otimes\mathbb{C}^T$ and $\mathbb{C}^T\otimes\mathbb{C}$ respectively. For the sake of notation simplicity, if two ciphertexts encrypt the same plaintext, then we consider them the same.

- We note that, for $1 \leq x, y \leq l$, the element on x-th row and y-th column of $\mathbb{C}\otimes\mathbb{C}^T$ is in the form $\sum_{i=1}^{N} c_{xi}\otimes c_{yi}$, which can be computed by FD based on the encrypted scalar values from all edge devices. For each such element, FD needs to perform $N-1$ homomorphic additions based on the received encrypted scalar values, and it also needs to perform a ciphertext rerandomization for security reasons.
- FD obtains $\mathbb{C}^T\otimes\mathbb{C}$ by pushing the computing task back to the edge devices as follows. For $1 \leq x, y \leq N$, the element on x-th row and y-th column of $\mathbb{C}^T\otimes\mathbb{C}$ is in the form of an encrypted inner products $\vec{c_x}^T\otimes\vec{c_y}$. Note that, if the edge device ED_x is given $\vec{c_y}$, then it can compute $\vec{c_x}^T\otimes\vec{c_y}$ more efficiently by replacing $\vec{c_x}^T$ with the plaintext, namely $\vec{d_x}^T$, and the complexity is l partial homomorphic multiplications, $l-1$ homomorphic additions and a ciphertext rerandomization (to prevent the leakage of the plaintext data). As the values on the diagonal have been sent by the edge devices, there are only $\frac{(N-1)(N-2)}{2}$ encrypted inner products to be computed, every edge device needs to compute $\frac{(N-1)(N-2)}{2N}$ on average.

2. *Randomization* Phase: FD chooses two random permutations. PU randomly permutes the edge device indexes: for any $1 \leq i \leq N$, $\mathsf{PU}(i) \in \{1, 2, \cdots, N\}$. PI permutes the index of elements in data vectors: for any $1 \leq i \leq l$, $\mathsf{PU}(i) \in \{1, 2, \cdots, l\}$. In this phase, based on $\mathbb{C}\otimes\mathbb{C}^T$ and $\mathbb{C}^T\otimes\mathbb{C}$, FD generates their variants corresponding to the following permuted plaintext data matrix.

$$\mathbb{A}^\dagger = \begin{bmatrix} d_{\mathsf{PU}(1)\mathsf{PI}(1)} & d_{\mathsf{PU}(1)\mathsf{PI}(2)} & \cdots & d_{\mathsf{PU}(1)\mathsf{PI}(N)} \\ d_{\mathsf{PU}(2)\mathsf{PI}(1)} & d_{\mathsf{PU}(2)\mathsf{PI}(2)} & \cdots & d_{\mathsf{PU}(2)\mathsf{PI}(N)} \\ \vdots & \vdots & \ddots & \vdots \\ d_{\mathsf{PU}(l)\mathsf{PI}(1)} & d_{\mathsf{PU}(l)\mathsf{PI}(2)} & \cdots & d_{\mathsf{PU}(l)\mathsf{PI}(N)} \end{bmatrix}$$

Let $\mathbb{C}^\dagger$ denote a ciphertext of $\mathbb{A}^\dagger$. Then, FD can generate $\mathbb{C}^\dagger\otimes{\mathbb{C}^\dagger}^T$ and ${\mathbb{C}^\dagger}^T\otimes\mathbb{C}^\dagger$ as follows.

(a) Clearly, PI does not affect $\mathbb{C}^\dagger\otimes{\mathbb{C}^\dagger}^T$, which can be generated based on PU by rearranging the elements in $\mathbb{C}\otimes\mathbb{C}^T$.

(b) Clearly, PU does not affect ${\mathbb{C}^\dagger}^T\otimes\mathbb{C}^\dagger$, which can be generated based on PI by rearranging the elements in $\mathbb{C}^T\otimes\mathbb{C}$.

At the end, FD sends $\mathbb{C}^\dagger\otimes{\mathbb{C}^\dagger}^T$ to SD_u, and sends ${\mathbb{C}^\dagger}^T\otimes\mathbb{C}^\dagger$ to SD_v.

3. *Ephemeral SVD* Phase: After receiving $\mathbb{C}^\dagger\otimes{\mathbb{C}^\dagger}^T$, SD_u can request the help from SD_v and FD to decrypt all the elements to obtain $\mathbb{A}^\dagger{\mathbb{A}^\dagger}^T$. It can then perform decomposition and obtain $\mathbb{U}^\dagger$ and $\Sigma^\dagger$. Similarly, SD_v can obtain $\mathbb{V}^\dagger$ and $\Sigma^\dagger$.

4. *Secure Storage* Phase (optional): Based on some predefined rule, SD_u and SD_v can truncate $\Sigma^\dagger$ in a certain way (see Section 2), and then store an encrypted product instead of the plaintext matrices. For example, if they do not truncate $\Sigma^\dagger$ at all, they store $\mathbf{Enc}(\mathbb{U}^\dagger(\Sigma^\dagger)^{\frac{1}{2}}, pk)$ and $\mathbf{Enc}((\Sigma^\dagger)^{\frac{1}{2}}(\mathbb{V}^\dagger)^T, pk)$ respectively.

The permutations, namely PU and PI, only affects the location of device indexes and data elements in the data matrix, so that they do not affect the functionality of SVD in any manner. Referring to the recommender use case, FD can easily determine the feature vectors for a specific user and item in $\mathbb{U}^\dagger(\Sigma^\dagger)^{\frac{1}{2}}$ and $(\Sigma^\dagger)^{\frac{1}{2}}(\mathbb{V}^\dagger)^T$. If the optional *Secure Storage* Phase is adopted, then when an outsider attacker compromises any two of the FoG devices (i.e. SD_u, SD_v and FD) at the end of the solution, it learns nothing due to the threshold decryption requirement.

5.3 Security and Performance Analysis

Security Analysis. We show that the solution satisfies our privacy expectations defined in Section 5.1.

1. When a group of edge devices are regarded as the attacker, it does not learn anything about the private data of other edge devices because it only receives encrypted data and has no access to the decryption oracle.
2. When SD_u (or SD_v) is regarded as the attacker, it only learns $(\mathbb{U}^\dagger, \Sigma^\dagger)$ (or, $(\mathbb{V}^\dagger, \Sigma^\dagger)$) because that is the only information disclosed in the *Ephemeral SVD* Phase. When FD is regarded as the attacker, it clearly learns nothing because of the threshold decryption constraint.
3. When SD_u and FD are regarded as the attacker, it possesses $(\mathbb{U}^\dagger, \Sigma^\dagger)$ and the permutations. As a result, it only learns $(\mathbb{U}, \Sigma)$ because of the threshold decryption constraint. Similarly, when SD_v and FD are regarded as the attacker, it only learns $(\mathbb{V}, \Sigma)$. When SD_u and SD_v are regarded as the attacker, it learns a randomly permuted data matrix $\mathbb{A}^\dagger$, which is equivalent to $(\mathbb{U}^\dagger, \Sigma^\dagger)$ and $(\mathbb{V}^\dagger, \Sigma^\dagger)$.
4. When FD and a group of edge devices are regarded as the attacker, it does not learn anything about the private data of other edge devices because it only receives encrypted data and does not receive any decryption output.
5. Due to the threshold decryption constraint, when SD_u and a group of edge devices are regarded as the attacker, it only learns $\mathbb{A}^\dagger \cdot {\mathbb{A}^\dagger}^T$ and the data vectors of these edge devices. Similarly, when SD_v and a group of edge devices are regarded as the attacker, it only learns ${\mathbb{A}^\dagger}^T \cdot \mathbb{A}^\dagger$ and the data vectors of these edge devices.

Asymptotic Performance Analysis. Regarding the complexity of the new solution, we summarize the number of main cryptographic operations in Table 5. It excludes offline and optional operations. As to notation, Dec, $\oplus$, $\otimes$, rand denote threshold decryption, homomorphic addition, homomorphic multiplication, and

ciphertext rerandomization respectively. Regarding the threshold Paillier scheme described in Appendix B, referring to the cryptographic operations, we make the following note: a partial $\otimes$ is an exponentiation, a $\oplus$ is a modulo multiplication, and a rand is a modulo multiplication.

Player	**Complexity**
ED_i	$\frac{l(N-1)(N-2)}{2N}$ partial $\otimes$, $\frac{(l-1)(N-1)(N-2)}{2N}$ $\oplus$, $\frac{(N-1)(N-2)}{2N}$ rand
FD	$\frac{l(l+1)(N-1)}{2}$ $\oplus$, $\frac{l(l+1)+N(N+1)}{2}$ Dec, $\frac{l(l+1)}{2}$ rand
SD_u	$\frac{l(l+1)+N(N+1)}{2}$ Dec
SD_v	$\frac{l(l+1)+N(N+1)}{2}$ Dec

Table 5. Asymptotic Complexity

Optimised Benchmarking. In order to learn the actual running time of different parties, we implement our solution based on the threshold Paillier scheme. For the benchmarking, we choose a 2048-bit n, set $s = 2$, and split the key into three shares and require the decryption to involve all three key shares. We assume there are 1000 devices and every data vector has 100 elements. In addition, as in the recommender use case, every element of the data vector is a small number from 0 to 5.

To reduce the number of threshold decryption operations, we propose to pack multiple ciphertexts into one and decrypt all of them at once. We have the following:

- Every ciphertext in the matrix $\mathbb{C}^\dagger \otimes {\mathbb{C}^\dagger}^T$ encrypts a number in the range $[0, 2^{15})$. Since the message space for threshold Paillier is $(0, n^2)$, we can pack around 270 ciphertexts C_i $(1 \leq i \leq 270)$ into one as $C_1 \cdot (C_2)^{2^{15}} \cdots (C_{270})^{2^{15\times 269}}$. Note that operations are modulo n^{2+1}. The packing incurs $269 + 15 + 30 + \cdots + 15 \times 269 = 544944$ ciphertext multiplications. Recovering individual plaintext is trivial based on modulo operations with respect to $2^{15\times 269}, 2^{15\times 268}, \cdots, 2^{15}$ sequentially.
- Every ciphertext in the matrix ${\mathbb{C}^\dagger}^T \otimes \mathbb{C}^\dagger$ encrypts a number in the range $[0, 2^{12})$. Similar to the above case, we can pack around 340 ciphertexts C_i $(1 \leq i \leq 340)$ into one as $C_1 \cdot (C_2)^{2^{12}} \cdots (C_{340})^{2^{12\times 339}}$. The packing incurs $339 + 12 + 24 + \cdots + 12 \times 339 = 691899$ ciphertext multiplications.

Player	**Complexity**	**Time**
ED_i	49850 partial $\otimes$, 49850 mul	3s
FD	5050000 mul, 1491 Dec	189s
SD_u	$\mathbb{C}^\dagger \otimes {\mathbb{C}^\dagger}^T$: 19 Dec (comb), 10353936 mul; ${\mathbb{C}^\dagger}^T \otimes \mathbb{C}^\dagger$: 1472 Dec	267s
SD_v	$\mathbb{C}^\dagger \otimes {\mathbb{C}^\dagger}^T$: 19 Dec; ${\mathbb{C}^\dagger}^T \otimes \mathbb{C}^\dagger$: 1472 Dec (comb), 1018475328 mul	15061s

Table 6. Optimised Complexity $(N = 1000, l = 100)$

After the optimisation, based on Table 5, we derive the new asymptotic complexity in Table 6. Note that mul is a modulo multiplication, Dec and Dec (comb) refer to the *Share decryption* and *Combination* algorithms respectively

in Appendix B. Based on our implementation on a PC with 3.40 GHz CPU and 16 GB memory, we obtain the actual running time in the last column.

In order to further reduce the running time for SD_v, there are two more ways to further optimise the computations. The first one is to check the density of the dataset and pack more ciphertexts into one, and the other is to outsource the computations to the edge devices. We leave the investigation of them as future work.

6 Conclusion

In this paper, we analysed the privacy-preserving SVD solution by Chen et al. [3], and demonstrated several privacy vulnerabilities. Based on our analysis, we presented an enhanced solution and provided analysis on both security and efficiency. As an immediate future work, we would like to further optimize its efficiency by exploiting fine-grained packing and computation outsourcing. It is also an interesting work to study the performances with larger datasets and improve the efficiency a step further.

References

1. Bar-Magen Numhauser, J.: Fog computing introduction to a new cloud evolution. University of Alcalá (2012)
2. Barbour, A.D., Luczak, M.J.: A law of large numbers approximation for Markov population processes with countably many types. Probability Theory and Related Fields **153**(3), 727–757 (2012). https://doi.org/10.1007/s00440-011-0359-2
3. Chen, S., Lu, R., Zhang, J.: A flexible privacy-preserving framework for singular value decomposition under internet of things environment. In: IFIP Advances in Information and Communication Technology. vol. 505, pp. 21–37 (2017). https://doi.org/10.1007/978-3-319-59171-1_3
4. Cisco: Fog Computing and the Internet of Things: Extend the Cloud to where the things are (2015)
5. Damgård, I., Jurik, M.: A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System. In: International Workshop on Public Key Cryptography. pp. 119–136. Springer (2001)
6. Han, S., Ng, W.K., Philip, S.Y.: Privacy-preserving singular value decomposition. In: 2009 IEEE 25th International Conference on Data Engineering. pp. 1267–1270. IEEE (2009)
7. Kim, S., Kim, J., Koo, D., Kim, Y., Yoon, H., Shin, J.: Efficient privacy-preserving matrix factorization via fully homomorphic encryption. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. pp. 617–628. ACM (2016)
8. Knorr, E., Gruman, G.: What cloud computing really means. InfoWorld **7**, 20–20 (2008)
9. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)

10. Nikolaenko, V., Ioannidis, S., Weinsberg, U., Joye, M., Taft, N., Boneh, D.: Privacy-preserving matrix factorization. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 801–812. ACM (2013)
11. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238. Springer (1999)
12. Wu, H.H., Wu, S.: Various proofs of the Cauchy-Schwarz inequality. Octogon Mathematical Magazine **17**(1), 221–29 (2009)
13. Yang, Y., Wu, L., Yin, G., Li, L., Zhao, H.: A survey on security and privacy issues in Internet-of-Things. IEEE Internet of Things Journal **4**(5), 1250–1258 (2017)
14. Zhou, J., Cao, Z., Dong, X., Lin, X.: Security and privacy in cloud-assisted wireless wearable communications: Challenges, solutions, and future directions. IEEE wireless Communications **22**(2), 136–144 (2015)
15. Zhuo, G., Jia, Q., Guo, L., Li, M., Li, P.: Privacy-preserving verifiable data aggregation and analysis for cloud-assisted mobile crowdsourcing. In: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. pp. 1–9. IEEE (2016)

A Paillier Cryptosystem

Paillier cryptosystem [11] is a popular public key encryption scheme that possesses useful homomorphic properties. The key generation, encryption and decryption algorithms are as follows.

Key Generation. Choose two large prime numbers p and q which has the same length k. Set the public key to be $n = pq$ and $g \in \mathbb{Z}^*_{n^2}$. The private key is computed by $\lambda = lcm(p-1, q-1)$ and $\mu = (L(g^\lambda \ mod \ n^2))^{-1}$ where $L(x) = \frac{x-1}{n}$.

Encryption. For a message as m, choose a random number $r \in \mathbb{Z}^*_n$, the ciphertext is computed as:

$$c = g^m \cdot r^n \ mod \ n^2$$

Decryption. Given a ciphertext c and private key, the corresponding original message m could be recovered as:

$$m = \frac{L(c^\lambda \ mod \ n^2)}{L(g^\lambda \ mod \ n^2)} \ mod \ n = \frac{c^\lambda - 1 \ mod \ n^2}{g^\lambda - 1 \ mod \ n^2} \ mod \ n$$

Paillier scheme has the following homomorphic properties. Given two ciphertexts c_1, c_2 for messages m_1, m_2, then $c_1 c_2$ is a ciphertext for message $m_1 + m_2$ while $c_1^{m_2}$ is a ciphertext for message $m_1 m_2$. Throughout the paper, we use $\oplus$ to denote homomorphic addition and $\otimes$ to denote homomorphic multiplication.

B The Threshold Damgard-Jurik Scheme

The threshold Damgard-Jurik scheme is a generalization of the Paillier cryptoscheme, but with an increased ciphertext space [5]. The secret key is distributed to v servers, at least t servers of them can proceed decryption efficiently ($1 < t \leq v$).

Key Generation. In addition to the key generation of classic Paillier cryptosystem, we compute $n^{'} = p^{'}q^{'}$ where $p = 2p^{'}+1$ and $q = 2q^{'}+1$. Besides, choose $s > 0$, thus the message space changes to $\mathbb{Z}_{n^s}$ and choose d where $d = 0 \ mod \ m$ and $d = 1 \ mod \ n^s$. Then, make the polynomial $f(X) = \sum_{i=0}^{k-1} a_i X^i \ mod \ n^s n^{'}$. where a_i (for $0 < i < t$) is random value from $0, ..., n^s * n^{'} - 1$ and $a_0 = d$. The sharing secret of i-th decrypting participant is $s_i = f(i)$ (for $0 < i \leq v$) and the public key will be n.

Encryption. To encrypt a plaintext m, choose a random number $r \in \mathbb{Z}^*_{n^{s+1}}$, the ciphertext is computed as:

$$c = g^m \cdot r^{n^s} \ mod \ n^{s+1}$$

Share decryption. The i-th decrypting participant computes $c_i = c^{2\Delta s_i}$ where c is ciphertext and $\Delta = v!$.

Combination. In order to obtain the original message, it requests t shares and combines them as:

$$c^{'} = \prod_{i \in t} c_i^{2\lambda^S_{0,i}} \ mod \ n^{s+1} \ where \ \lambda^S_{0,i} = \Delta \prod_{i^{'} \in t \backslash i} \frac{-i}{i - i^{'}} \in Z$$

Therefore, $c^{'}$ is in the form $c^{'} = c^{4\Delta^2 f(0)} = c^{4\Delta^2 d}$. Note that, $4\Delta^2 d = 0 \ mod \ \lambda$ and $4\Delta^2 d = 4\Delta^2 \ mod \ n^s$, thus, $c^{'} = (1+n)^{4\Delta^2 m} \ mod \ n^{s+1}$. By applying the following Algorithm 4, we can finally restore the plaintext m.

Algorithm 4 Recover Plaintext in Threshold Variant

Input: Combined ciphertext $c^{'} = (1+n)^{4\Delta^2 m} \ mod \ n^{s+1}$
Output: Plaintext m
let $L((1+n)^i \ mod \ n^{s+1}) = (i + \binom{i}{2} n + ... + \binom{i}{s} n^{s-1}) \ mod \ n^s$
thus $i_j = L((1+n)^i \ mod \ n^{j+1}) - (\binom{i_j}{2} n + ... + \binom{i_{j-1}}{j} n^{j-1}) \ mod \ n^j$
$i \leftarrow 0$
for $j = 1$ *to* t **do**
  $t_1 \leftarrow L(a \ mod \ n^{j+1})$
  $t_2 \leftarrow i$
  for $k = 2$ *to* j **do**
    $i \leftarrow i - 1$
    $t_2 \leftarrow t_2 * i \ mod \ n^j$
    $t_1 \leftarrow t_1 - \frac{t_2 * n^{k-1}}{k!} \ mod \ n^j$
  end for
  $i \leftarrow t_1$ (here, $i = 4\Delta^2 m$)
end for
$res \leftarrow i * i^{-1} \ mod \ n^s$
return res